CENTRE D'AUTOMATIQUE ET INFORMATIQUE

SECTION INFORMATIQUE

CODE GENERATION FOR DATA MOVEMENTS IN HIERARCHICAL MEMORY MACHINES

Corinne Ancourt

Workshop on Compilers for Parallel Machines

Document EMP-CAI-I A/196

Code Generation for Data Movements in Hierarchical Memory Machines

Corinne Ancourt [*]
Ecole des Mines de Paris
Centre d'Automatique et Informatique
France

Abstract

Parallel tasks generated by automatic parallelizers do not take advantage of supercomputer memory hierarchies. This paper presents algorithms to transform a parallel task into an equivalent one that uses data with fast access memory. This new task moves the used data from shared to local memory, performs computations with copies in local memory, and writes back modified data.

Array elements are often used in parallel tasks. Our goal is to generate codes automatically in order to move array elements that are accessed in a set of loops during an execution of the task. This set of array elements is characterized by a set of integer points in $\mathcal{Z}^p$ that is not necessarily a convex polyhedron. Thus, algorithms previously known often generate codes that have complex loop bounds and/or that may induce multiple copies of the same element.

In the case of data transfers from global memory to local memory, it is possible to copy a superset of accessed elements, for instance its convex hull. A trade-off has to be made between local memory space, transfer volume and loop bound complexity (i.e. control overhead).

To copy data back from local memory to global memory is more difficult because global memory consistency must be preserved. Each processor should only copy its own results, to avoid errors and, secondarily, to decrease global memory traffic.

The input of our main algorithm is an integer convex polyhedron defining the computation space and an affine function representing the index expressions. Its output is a set of nested loops containing data transfer codes and computation codes with local array references. Each element is copied only once. Loop bound expressions use integer divisions to generate non-convex sets.

For most practical programs, this algorithm provides optimal code in number of data movements and control overhead.

Introduction

The performance of shared memory multiprocessors is often limited by shared memory access time. Introduction of local and cache memories allows for the reduction of memory access time, but creates two problems:

- management of data transfers between memory levels,
- maintenance of coherency in the shared memory.

Many methods [CeFe78], [BrDu83], [OwAg89], [ChVe89], [Karl89] have been developed to resolve these difficulties for cache memories, managed by the system. On the other hand, few methods [GJGa88], [GJGa88b] have been proposed for the management of local memories, which must be done by programmers.

Programs produced by automatic parallelizers do not generally take advantage of local memories. The

[*]E-mail: <ancourt@ensmp.fr>

goal of our study is to add one phase to parallelizers in order to transform a parallel task into an equivalent one that uses these fast access memories. This phase transforms one parallel task into a sub-program containing: code of transfer that moves needed data from shared memory to local memories, computation code that will execute with data residing in local memory, code of transfer that moves data modified during execution from local memories to shared memory.

Characterization of used and modified data raises problems when arrays are used. Their elements are often referenced in sets of loops contained in a parallel task, and their transfer into local memory is important. Thus, transfer code generation of data referenced by array in a set of loops is the subject of this paper.

In the first section, an example introduces the type of sub-program that we want to generate at the end of the transformation phase. After specification of notations used in this paper in the section 2, the third section details the different steps of the transformation phase. The different problems that we have met and the solutions that we propose are presented. The next section describes more precisely, with a theoretical approach, the generation phase of the transfer codes and specifies how we propose to compute the data set that has to be transferred. Section 5 presents why the proposed algorithms could be used to improve some phases of code generation for methods used in another context: the tasks parallelization on distributed memory multiprocessors. Finally, a comparison of our method with others is presented in the section 6, before the conclusion.

1 Example

In the first step, the parallel task is assumed to contain only assignment statements (no CALL or IO statements). Loop bounds and subscript expressions are assumed to be linear expressions.

The following example is designed to demonstrate problems that may arise:

<u>The initial task:</u>

```
DO  A=1,4*P
 DO  B=1,N
  DO  C=1,M
   DO  D=1,L
    TA(2*B+D,C+D,A)  =  TB(B+D,C+D,A)
   ENDDO
  ENDDO
 ENDDO
ENDDO
```

<u>The parallel task:</u>

```
DOALL IT=0,3
 DO   A=1+IT*P,(IT+1)*P
  DO   B=1,N
   DO   C=1,M
    DO   D=1,L
      TA(2*B+D,C+D,A)=TB(B+D,C+D,A)
       Ref.1                Ref.2
   ENDDO
  ENDDO
 ENDDO
ENDDO
```

The parallelizer detects that first loop A can be executed in parallel on a 4-processor. The sub-program that we want to generate is in figure 1

TA and TB are arrays accessed in shared memory. TLA and TLB are arrays needed to store data in local memories. The first part of the generated code contains declarations of local array dimensions. All computations will be executed in local memory and will only reference these local arrays. The second part is the code for transferring data used by the task from shared memory to local memories; TB array elements are written in local array TLB. The third part is the computation code; references to arrays TA and TB have been transformed in local array references TLA and TLB. The last part is the code for transferring the computed results from local memory to shared memory. TLA array elements are written back to shared memory in array TA. Loop bounds contain integer divisions, introduced by our algorithms, to translate the non-convexity of the set of referenced elements. These elements are represented in figure 2.

```
TASK Paral-T(IT,P)

GLOBAL  REAL TA(3:2*N+L,2:M+L,1+IT*P:(IT+1)*P)
GLOBAL  REAL TB(2:N+L,2:M+L,1+IT*P:(IT+1)*P)

LOCAL   REAL TLA(3:2*N+L,2:M+L,1+IT*P:(IT+1)*P)
LOCAL   REAL TLB(2:N+L,2:M+L,1+IT*P:(IT+1)*P)
C
C Copy  from  shared  memory  into  local  memory (Ref.2)
C
DO I=1+IT*P,(IT+1)*P
   DO J=2,M+L
      DO K=MAX (2,J-M+1),MIN(L+N, J-1+N)
          TLB(K,J,I) ⟵  TB(K,J,I)
      ENDDO
   ENDDO
ENDDO
C
C Computation Code
C
DO A=1+IT*P,(IT+1)*P
  DO B=1,N
    DO C=1,M
        DO D=1,L
            TLA(2*B+D,C+D,A) = TLB (B+D,C+D,A)
           Ref.1                 Ref.2
        ENDDO
    ENDDO
  ENDDO
ENDDO
C
C Copy  from  local  memory  back  to  shared  memory (Ref.1)
C
DO I=1+IT*P,(IT+1)*P
  DO J=1-2*N,M-2
    DO K=MAX(3,1+2*((2-J)/2)),MIN(L+2*N,L+2*((M-J)/2))
        TA(K,J+K,I) ⟵  TLA(K,J+K,I)
    ENDDO
  ENDDO
ENDDO

RETURN
END
```

Figure 1: Example 1

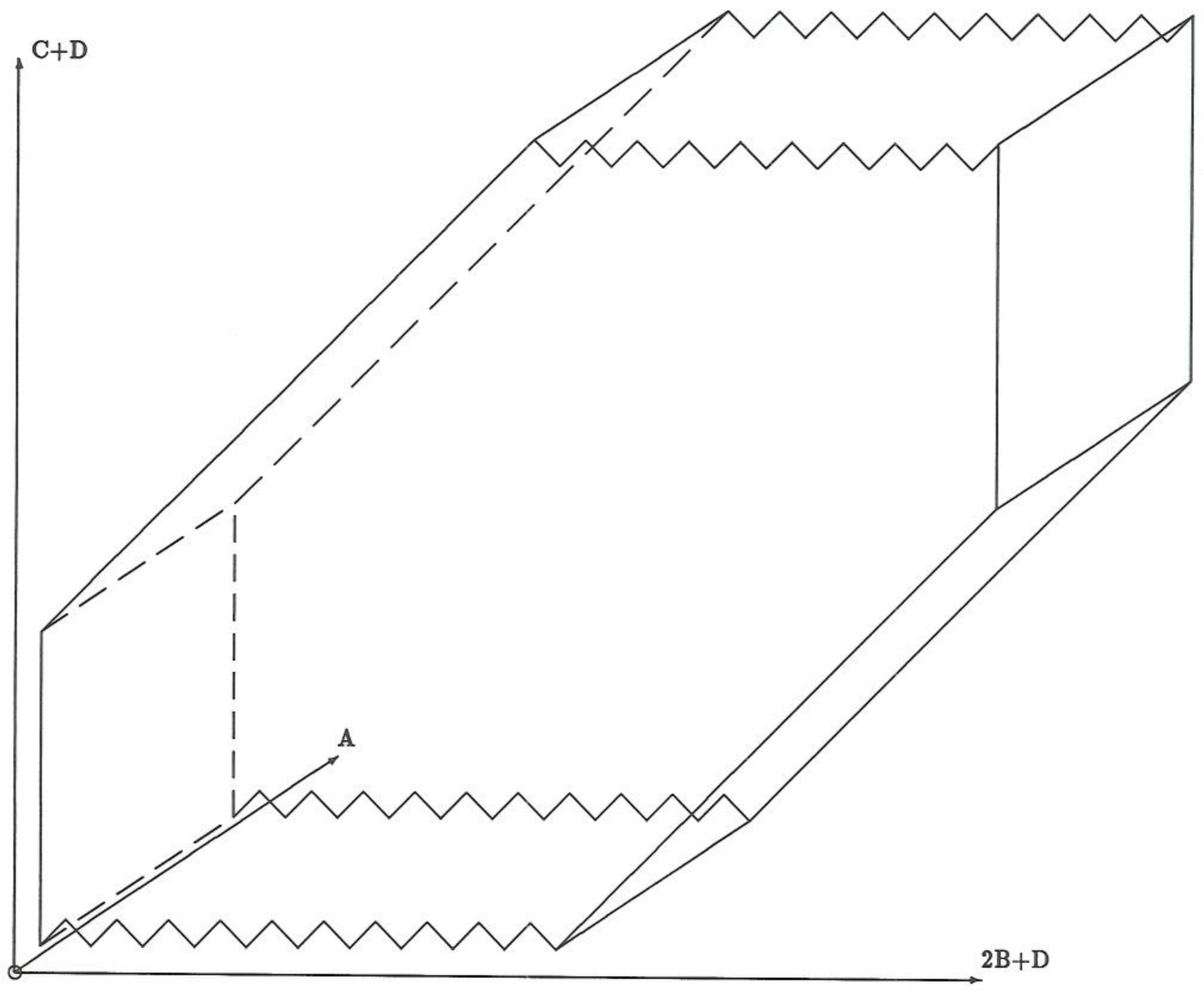


Figure 2: The set of modified data

2 Notations

Let us consider the computation code of the previous program. It contains four loops on A,B,C and D which define an 4-Dimension iteration space I. Each loop body iteration can be specified by an iteration vector $\vec{\imath}$ such that $\vec{\imath}^T$ = (A,B,C,D). Linear bounds are assumed to remain in the linear algebra framework and the iteration set is a polyhedron defined by a matrix α and a symbolic vector β whose coordinates are linear expressions built on numerical values, symbolic constants and variables constant over the loop: $\alpha\vec{\imath} \leq \beta$.

In this figure, the iteration set is defined

$$by: \quad \alpha = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$and\,\beta = \begin{pmatrix} -1 - IT * P \\ (1 + IT) * P \\ -1 \\ N \\ -1 \\ M \\ -1 \\ L \end{pmatrix}$$

such that

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} \leq \begin{pmatrix} -1 - IT * P \\ (1 + IT) * P \\ -1 \\ N \\ -1 \\ M \\ -1 \\ L \end{pmatrix}$$

This iteration set is formally defined by a system of linear inequalities S, also called constraints, which are built from a row of α and a coefficient of β.

The floor operator $\lfloor S \rfloor$ will denote the subset of integer points contained in the system of inequalities S.

3 Steps in the transformation phase

This section describes different steps in the transformation phase. It demonstrates:

- how to compute the dimensions of local arrays,
- how to characterize the set of used and modified data in parallel tasks,
- a solution to transfer efficiently the set of used data from shared memory into local memories,
- a solution to copy efficiently the set of modified data back from local memory into shared memory.

All these phases must take into consideration the two main objects: maintenance of memory coherency and minimization of transfer costs.

3.1 Local arrays dimensions

The number of referenced array elements in a parallel task is often less than the number of elements in the whole array. Allocation of local array memory must correspond to this subset. In a Fortran context, dimensions of local arrays must be linear expressions of integers and constants. To generate them, lower and upper bounds of values that are referenced by array elements in the parallel task must be precisely computed. Because symbolic constants may occur in bound expressions of the iteration space, classical algorithms of integer programming cannot always be used. Thus, the algorithm of parametric integer programming proposed in [Feau88b] will be used.

3.2 Affine image

Characterization of used and modified data is problematic when arrays are referenced. As a matter of fact, it is the set of all referenced array elements in a set of loops contained in a parallel task that must be characterized. This set correspond to a set of integer points. It is the major cause of problems.

The characterization of a set of loops [Irig88] that cover the elements of a polyhedron is simpler than for a set of loops that examine the elements of any set of integer points. Thus, our goal is to try to describe the set of referenced elements by a set of polyhedra in order to generate transfer code easily.

However, two principal problems arise: the set of referenced array elements is neither necessarily a convex set nor a whole polyhedron.

3.2.1 Non-convex set

The set of referenced array elements is the affine image of the iteration space by the access function T. We also call this set the *image domain.*

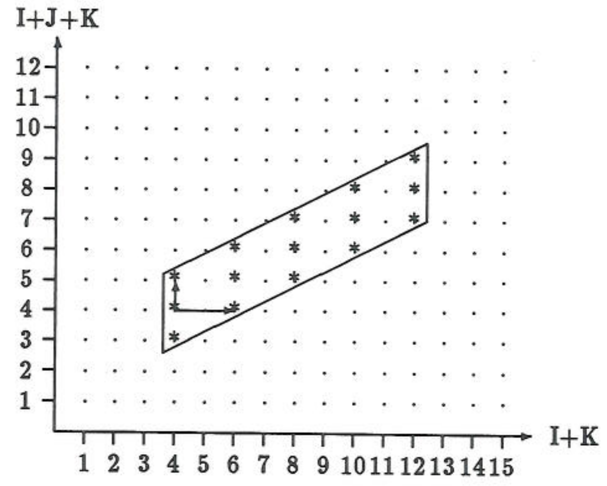
Example 2 shows that this set is not necessarily a convex polyhedron. Indeed, the image of a set of integer points (the iteration space) by an affine function do not referenced necessarily a convex set. Sometimes, the domain borders are non convex.

To solve this problem, section 4 describes how it is possible to introduce integer divisions in loop bound expressions in order to translate the non convexity of the image domain borders into a set of loops scanning it.

3.2.2 Hole polyhedron

The following example presents a set of referenced array elements, represented by $*$ on the figure, which references an hole polyhedron.

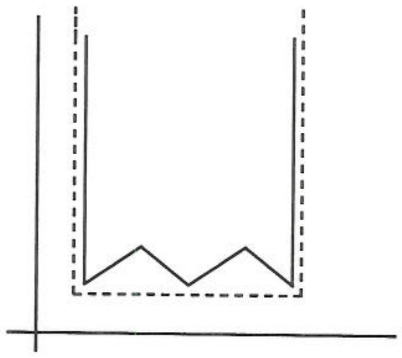
```
 DO $I = 1,3$
  DO $J = 1,3$
    DO $K = 1,3$
      T($2*I+2*K, I+J+K$)=...
    ENDDO
  ENDDO
ENDDO
```



In this example, on the axis $I+K$, only elements that are multiple of 2 are referenced. To minimize the number of transfers, the whole polyhedron defining these elements must not be copied. A new basis must be found in order to only scan the referenced elements. Here, the new basis $\begin{pmatrix} 2I+2K & 0 \\ 0 & I+J+K \end{pmatrix}$ will be used to express the referenced array elements, thereby reducing the new iteration space (in this new basis) to a whole polyhedron. This new basis can be found by computing the Hermite reduced form [NeWo88] associated with the access function T.

3.3 Copy in

For transfers from shared memory to local memories, it is not necessary to copy the set of exactly modified data. We can copy a larger set because these data will be used only by local parallel tasks. Thus, no conflicts between different sets of local copies can arise.

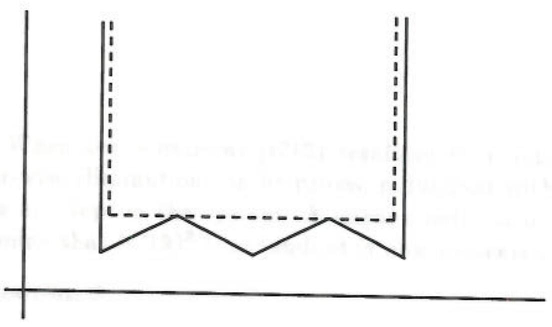


If the set of used array elements is not a convex polyhedron, a slightly larger set will be transferred: the smallest convex polyhedron including the set of actually used data.

3.4 Copy back

For transfers from local memory to shared memory, it is not correct to copy back one element that has not been modified by the local task. In fact, this data might have been modified by another task, and the two different values of one data cannot reside in a coherent shared memory. Thus, copy back code of exact modified data must be generated.

When the image domain is not a convex set, integer divisions will be inserted in bound expressions of loops of transfer code in order to translate the non convexity of the affine domain borders, in all cases where it is possible.



In other cases, the non-convex set will be cut into a set of convex polyhedra, and the end phase will generate as many transfer codes as convex sets contained in the non-convex image domain.

The following section describes more precisely how the domain image is characterized and details the generation code phase.

4 Computation of Image polyhedra

The set of referenced array elements is the affine image of the iteration set by the access function T. The first step of image domain computation is to find one basis $\vec{t}$ able to scan it. Hermite reduced form associated to the access function T permits us to find one of these bases [NeWo88].

The image polyhedron S is then computed from the index set I by a change of basis from index basis $\vec{i}$ to image basis $\vec{t}$. If the dimension d [1] of the affine image is less than the dimension n of the iteration space then only d loops are necessary to scan the image polyhedron. Thus, $n-d$ variables can be eliminated from the image polyhedron constraints defined by the system S.

One method of eliminating the $n-d$ useless variables (i.e. to compute the projection of the image polyhedron on the first d basis vectors; also called basis variables) follows. The key is to eliminate non array basis variables from the constraints without introducing new array elements.

4.1 Exact integer projection

4.1.1 Legal Integer Pairwise Elimination

Great care must be taken not to modify the set of affine image integer points when one non basis variable from the image polyhedra is eliminated. To preserve these image points, integer divisions must be introduced.

$$Let\ E_j = \alpha_j + \sum_{l=1, l\neq k}^{n} a_{jl}\, i_l$$

be an integer linear expression, R a set of constraints, a_{jl} and α_j integers, a_{pk} and a_{qk} positives integers and i_k a variable.

Theorem 1

$$Let\ S = \left\{ \begin{array}{lcl} E_p + a_{pk}\, i_k & \leq 0 & (C1) \\ E_q - a_{qk}\, i_k & \leq 0 & (C2) \\ R & & \end{array} \right.$$

$$and\quad S_{/k} = \left\{ \begin{array}{ll} \frac{E_q + a_{qk} - 1}{a_{qk}} \leq \frac{-E_p}{a_{pk}} & (C12) \\ R & \end{array} \right.$$

where $S_{/k}$ is derived from S using integer divisions[2] *to eliminate i_k. Then:*

$$proj(\lfloor S \rfloor, i_k) = proj(\lfloor S_{/k} \rfloor, i_k)$$

This system $S_{/k}$ does not necessarily define a polyhedron since integer divisions may introduce *holes* in a convex polyhedron. Therefore, this elimination operation is not an internal operation. We present some conditions that allow variable elimination by simple pairwise elimination [Four24] without modifying the projection. They will be used to eliminate as many variables as possible while preserving the projection and the system linearity. These conditions follow:

Theorem 2

$$Let\ S = \left\{ \begin{array}{ll} E_p + a_{pk}\, i_k & \leq 0 \\ E_q - a_{qk}\, i_k & \leq 0 \\ R & \end{array} \right.$$

$$and\quad S' = \left\{ \begin{array}{l} a_{pk}\, E_q \leq -a_{qk}\, E_p \\ R \end{array} \right.$$

where S' is obtained from S using the pair-wise elimination method and R is any system.

$$a_{pk} = 1 \vee a_{qk} = 1 \Longrightarrow$$

$$proj(\lfloor S \rfloor, i_k) = \lfloor proj(S, i_k) \rfloor = proj(\lfloor S' \rfloor, i_k)$$

[1] equals to the rank of the Hermite form associated to the access function T

[2] Different definitions exist for non-positive integers. Here the remainder is always assumed to be positive.

4.2 Computation of Image Polyhedra Constraints

The image polyhedron S is computed from the index set I by a change of basis from index basis $\vec{i}$ to image basis $\vec{t}$. If the dimension d [4] of the affine image is less than the dimension n of the iteration space then only d variables (called *basis variables*) are necessary to express the constraints defining the image polyhedron. Thus, $n - d$ variables (called *non-basis variables*) can be eliminated from the image polyhedron constraints defined by the system S.

The first step of the algorithm consists in projecting as many useless variables (non-basis variables) as possible using the pair-wise elimination method for constraints satisfying the conditions of theorem 2 and 3.

The second step is to eliminate redundant constraints. All redundant constraints on useless variables can be eliminated if the variable does not appear in a constraint of superior level. Two constraints on useful variables must be conserved.

Finally, integer divisions are introduced in constraint expressions. All remaining useless variables are eliminated from S by combination of constraints pairs and introduction of integer divisions, if the variable does not appear in a constraint of superior level.

The system obtained may still contain useless variables. In this case, constraints on these useless variables also appear in the set of loops scanning the image polyhedra. As with integer divisions, occurrence of these variables in constraints express the non convexity of the affine image.

This algorithm is described in figure 3.

4.3 Generation of the loops scanning the image domain

Let SI be the set of constraints computed by the previous algorithm for the image polyhedra. Let SI_1 be the subset of SI made of the inequalities with no integer divisions, and SI_2 its complement wrt. SI.

The method proposed in [Irig88] is used to generate automatically the set of d [5] loops scanning the elements of the polyhedron S_1. The d different steps follow:

- the bounds of the most internal loop in the final generated code are computed. The whole system S_1 gives these bounds in function of other loop variables.

For example : $\begin{cases} -t_1 + t_2 \le -1 \\ t_1 - t_2 \le 10 \\ -t_2 \le -1 \\ t_2 \le 10 \end{cases}$

bounds of loop variable t_1 are linear function of other variables:

$$\begin{cases} -t_1 \le -1 - t_2 \\ t_1 \le 10 + t_2 \end{cases}$$

- the bounds of the $(d-1)-th$ loop are computed next. The system S_1 is projected onto variables $t_2, t_3, ..., t_d$ and t_2 is expressed in function of the $d-2$ first loop variables,
- afterwards, the bounds of the $(d-i)-th$ loop $(2 \le i \le d-2)$ are computed. The system S_1 is projected on variables $t_{i+1}, t_{i+2}, ..., t_d$ and t_{i+1} is expressed in function of the $d-i-1$ first loop variables,
- finally, the successive projections of the first $d-1$ loop variables allow the computation of the numerical loop bounds of the most external loop.

  At each step of the algorithm, redundant constraints are eliminated using specific rules.

To generate the nested loops set defining the image polyhedra, the previously described algorithm is applied to SI_1. Inequalities of SI_2 are added as loops bounds or used in a guard if a variable of higher level in the inequality appears on two sides of the inequality.

5 Code Generation for Distributed Memory Multiprocessors

Management of distributed memories is more complex than that of shared memory multiprocessors because data partitioning in local memories must be taken into account. A lot of methods [CaKe88], [GJGa88], [RoPi89], [Tsen89], [APTh90], [Gern90], [KoMe90] assume that this data partitioning is given by the programmer and is "simple". It must be easily exploited by methods that generate data transfers and assign execution of a set of instructions to a processor. Thus, data partitioning often results in a set of polyhedra.

When data partitioning is done, the set of instructions to be executed on each processor must be evaluated. Many methods [CaKe88], [RoPi89], [APTh90], [Gern90] have chosen to execute on a processor the set of instructions that correspond to a write operation of a

[4] equals to the rank of the Hermite form associated to the access function T

[5] the dimension of the affine image

```
EXACT-PROJECTION:

1. For each non basis variable $i_k$ $(d+1 \le k \le n)$ of $S$ do:
    • Divide $S$ in two sets: set $S_{i_k}$ of inequalities containing $i_k$,
                              and set $R$ of other inequalities,
    • Let $S_{/k}$ be $R$,
    • Let POS be the set of inequalities $\{E_p + a_{pk} i_k \le 0\}$ of $S_{i_k}$ where $a_{pk}$ is a positive integer and NEG
      the set of inequalities $\{E_q - a_{qk} i_k \le 0\}$ where it is negative.
      For each pair of inequalities $(pos \in POS, neg \in NEG)$ do:
        – for each pair of inequalities such that $a_{pk} = 1$ or $a_{qk} = 1$ do:
                $S_{/k} = S_{/k} \bigcup \{a_{pk} E_q \le -a_{qk} E_p\}$
        – for each pair of inequalities such that ( $a_{pk} E_q + a_{pk} a_{qk} - a_{pk} \le -a_{qk} E_p$ is redundant in $R$), do
          nothing
        – for each pair of inequalities such that $a_{pk} > 1$ and $a_{qk} > 1$
          and ( $a_{pk} E_q + a_{pk} a_{qk} - a_{pk} \le -a_{qk} E_p$ is not redundant) do:
                $S_{/k} = S_{/k} \bigcup \{a_{pk} E_q \le -a_{qk} E_p\} \bigcup \{pos, neg\}$
    • Let $S = S_{/k}$
2. Associate with each constraint $C_j$ the rank $R_j = max\{ i \mid a_{ji} \neq 0\}$
3. Eliminate redundant inequalities of $S$ using the following rules:
    • First, eliminate redundant inequalities associated with the largest rank $R_i$.
    • keep at least two constraints associated with a basis variable $i_l$ $(1 \le l \le d)$: one constraint
      where the $i_l$ coefficient is positive, and one other where it is negative,
    • Eliminate all redundant constraints $C_r$ associated with a rank $R_r$ if $(\forall C_s \in S, R_s > R_r$ then $a_{sr} = 0)$
4. Eliminate non basis variables $i_k$ $(d+1 \le k \le n)$ of constraints by introducing integer divisions:
    • for each pair of inequalities $(E_p + a_{pk} i_k \le 0 \, , E_q - a_{qk} i_k \le 0) \mid (\forall C_s \in S, R_s > k$ then $a_{sk} = 0)$ do:
        – Eliminate $i_k$ by introducing integer divisions
        – associate with the new constraint $C_t$:
            * the rank $R_t = max\{i \mid a_{ti} \neq 0\}$ if $i_t$ appears only on one side of the inequality,
            * no number otherwise
```

Figure 3: Algorithm to compute the exact image polyhedra

The data transfer codes generated by our algorithms are practically always optimal, i.e. each data that must be transferred is copied only once. More precisely the number of transfers is minimal when the data set that must be transferred is a convex set or the union of data sets referenced by functions with uniform dependences.

The use of polyhedra to characterize all domains (iteration and image) that we need to manipulate permits us to decrease some initial application hypotheses (linear test statements can be translated in the form of linear constraints) and to extend simply our first results, described in this paper, to the case where there are several references to the same array.

Parallelization methods of tasks for distributed memory multiprocessors also need to compute data sets: the set of used data and the set of iterations that must be executed on one processor. When data partitioning in local memories is a union of polyhedra, our algorithms can be used to evaluate them. In many cases, the generated codes allow for the scanning of data sets without guards or masks (integer divisions are sufficient). Thereby they decrease the control overhead at task execution time, especially for non-convex data sets.

Acknowledgements

I wish to give special thanks to F. Irigoin for his help and constructive remarks at various stages of this work. I also wish to thank P. Jouvelot and A. Frey for their critical reading of this paper.

References

[Anco90] C. Ancourt, *Génération de code pour multiprocesseurs à mémoires locales*, Thèse de l'Université Pierre et Marie Curie, in progress

[APTh90] Andre F., Pazat J-L., Thomas H., " PANDORE: a system to manage data distribution ", *International Conference on Supercomputing*, pp 380-388, June 1990

[BrDu83] Briggs F.A., Dubois M., "Effectiveness of private caches in multiprocessor systems with parallel-pipelined memories," *IEEE Transactions on computers* , Vol. 32, No 1, Jan. 1983.

[CaKe88] Callahan D., Kennedy K., " Compiling programs for distributed-memory multiprocessors ", *Journal of supercomputing* , pp 151-169, 1988

[CeFe78] Censier L., Feautrier P., "A new solution to coherence problems in multicache systems," *IEEE transactions on computers*, Vol. c-27, No 12, December 1978.

[Chen87] Cheng M. C., "General criteria for redundant and nonredundant linear [214z inequalities," *Journal of optimization theory and applications* Vol. 53, No 1, April 1987.

[ChVe89] Cheong H., Veidenbaum A., "Compiler-directed cache management for multiprocessors" *Tech. Report C.S.R.D. Univ. of Illinois at Urbana-Champaign* , No 937, 1989

[EJWB90] Eisenbeis C., Jalby W., Windheiser D., Bodin F., " A Strategy for array management in local memory," *3rd Workshop on Languages and Compilers for Parallel Computing*, Irvine, 1990.

[Feau88] Feautrier P., "Semantical analysis and mathematical programming," *Int. Workshop on Parallel and Distributed Algorithms*, Bonas, 3-6 Octobre 1988.

[Feau88b] Feautrier P., "Parametric integer programming," *RAIRO Recherche opérationnelle* pp:243-268, Sept. 1988.

[Four24] Fourier J.B.J., "Analyse de travaux de l'Académie Royale des Sciences, pendant l'année 1824, partie mathématique," *Histoire de l'Académie Royale des Sciences de l'Institut de France* , 1827.

[Gern90] Gerndt H. M., " Automatic parallelization for distributed-memory multiprocessing systems" *Dissertation zur Erlangung der Doktorwurde der Mathematisch-Naturwissenschaftlichen Fakultat der Rheinischen Friedrich-Wilhelms-Universitat* Bonn 1989.

[GJGa88] Gallivan K., Jalby W., and Gannon D., "On the problem of optimizing data transfers for complex memory systems," *Proceeding of the ACM Int. Conf. on supercomputing*, St-Malo, 1988.

[GJGa88b] Gannon D., Jalby W., Gallivan K., " Strategies for cache and local memory management by global program transformation ", *Journal of Parallel and Distributed Computing* , Vol. 5, pp 587-616, 1988.

[Irig88] Irigoin F., "Code generation for the hyperplane method and for loop interchange," *Rapport ENSMP-CAI-88-E/102/CAI/I*, 1988

[Karl89] Karlovsky S.R., "Automatic management of programmable caches: Algorithms and experience." *Thesis Univ. of Illinois at Urbana-Champain* , 1989

[KoMe90] Koelbel C., Mehrotra P, " Supporting shared data structures on distributed memory architectures " *Second ACM SIGPLAN symposium on Principles and Practice of Parallel Programming* , Vol. 25, No 3, March 1990

[NeWo88] Nemhauser G.L., Wolsey L.A., "Integer and combinatorial optimization," *Ed. Wiley* , 1988.

[OwAg89] Owcki S., Agarwal A., " Evaluating the performance of sotfware cache coherence " *Tech. Report Digital* , No 41, 1989

[RoPi89] Rogers A., Pingali K., " Process decomposition through locality of reference ", *ACM Conference on Programming Langage Design and Implementation* , pp 69-80, June 1989

[SLYe88] Shen Z., Li Z., Yew P.C., "An empirical study on arrays subscripts and data dependences," *International Conference on Parallel processing* , August 1989, pp. 145-153.

[Tsen89] Tseng P.S., " A parallelizing compiler for distributed-memory parallel computers ", *PhD Thesis - Canergie Mellon University* , 1989