# Evaluation of Loop Grouping Methods based on Orthogonal Projection Spaces

Ioannis Drositis, Giorgos Goumas, Nectarios Koziris, Panayiotis Tsanakas
and George Papakonstantinou
*National Technical University of Athens*
*Dept. of Electrical and Computer Engineering*
*Computing Systems Laboratory*
*Zografou Campus, Zografou 15773, Greece*
e-mail: {jdros, goumas}@cslab.ece.ntua.gr

## Abstract

*This paper compares three similar loop-grouping methods. All methods are based on projecting the n-dimensional iteration space $J^n$ onto a k-dimensional one, called the projected space, using (n-k) linear independent vectors. The dimension k is selected differently in each method giving various results. The projected space is divided into discrete groups of related iterations, which are assigned to different processors. Two of the methods preserve optimal time completion, by scheduling loop iterations according to the hyperplane method. The theoretical analysis of the experimental results indicates the appropriate method, for specific iteration spaces and target architectures.*

***Index Terms****: loop grouping, orthogonal projection, communication overhead, hyperplane method, mesh-connected architectures.*

## 1. Introduction

Orthogonal projection is the projection of the *n*-dimensional (loop) iteration space $J^n$ onto a *k*-dimensional subspace Null($\Psi$), so that the grouping is done along the *g*-dimensional subspace $\Psi$, where $n = k + g$ [4]. Sheu and Chen in [9] applied the orthogonal projection to project $J^n$ onto an 1-dimensional space (projection line), along an (*n*-1)-dimensional projection space. Groups of neighboring iterations are formed along the (*n*-1)-projection space. Heuristic rules are used to select the projection space, based on the cardinality of the dependence vectors that span space $\Psi$ and their length. The projected space is a line, mapping directly onto a fixed size linear array of processors, thus not exploiting all inherent loop parallelism. In other words, the (*n*-1)-dimensional hyperplane time schedule, which produces the optimal loop execution time, is *not* preserved.

In [10], Sheu and Tai presented a systematic method of partitioning and grouping, based on hyperplane time scheduling. The space partitioning is done along an axis perpendicular to the optimal hyperplane. Unfortunately this, as it will be shown, does not always lead to the best results. Although such a grouping preserves the optimal time execution policy, it does not consider increasing local referencing inside every group. Furthermore, space partitioning is done without taking into consideration the notion of *displacement*, thus giving poor results when this factor is greater than one. Nevertheless, this approach had the advantage of proposing a low complexity linear space scheduling.

A method similar to the above, called Chain Grouping, is presented in [5]. This method considerably enhances the results presented by the previous two papers. It is based on the projection of the iteration space onto a (*n*-1)-dimensional space, using a single projection vector $\boldsymbol{d}_k$ (similarly to [10] an 1-dimensional projection space is used). Grouping is done along a uniform chain of iterations formed by the projection vector $\boldsymbol{d}_k$. The *grouping vector* is chosen so as to maximize referencing inside the same processor, through alternative dependence vector paths. In other words, this method clusters not only directly dependent iterations (via a particular vector) to be executed on the same processor, but also iterations which depend on each other through other non-uniform dependence paths.

In this paper we will present a comparison between these three methods: SC (Sheu and Chen [9]), ST (Sheu and Tai [10]) and CG (Chain Grouping [5]). Although all three use orthogonal projection, they are primarily differentiated by the way they select the projection space $\Psi$ (dimension and orientation), as well as by the way they define their *grouping factor*. Thus, for certain types of problems they result on different communication cost and biased execution time. Method comparison is mainly based on communication cost, as ST and CG methods preserve the optimal hyperplane scheduling, while time scheduling in SC method is not explicitly defined. Several examples are presented, along with comparative results on method behavior, according to certain problem dimension and dependence constrains.

The rest of the paper is organized as follows: basic terminology and definitions used throughout this paper are introduced in *Section-2*. *Section-3* overviews SC, ST and CG partitioning methods. *Section-4* compares the

three methods, while in *Section-5* each method's scoring on certain example types is presented. We conclude in *Section-6*, with some overall remarks on each method's efficiency.

## 2. Preliminary Concepts and Definitions

### 2.1. Model of the Algorithms

We will use the computational *uniform* data dependence model of perfectly nested FOR-loop algorithms, widely used in many similar papers (e.g. see [1], [2], [3], [6], [7], [8]). So, our algorithms are of the form:

```
FOR  i1=l1 TO u1 DO
     ...
     FOR  in=ln TO un DO
        AS1(i)
        ...
        ASk(i)
     ENDFOR
     ...
ENDFOR
```

**Figure 1.** The algorithm model.

where $l_i$ and $u_i$ are integer-valued constants (boundary values of the *i*-th inner loop); instance vector is denoted as $\boldsymbol{i} = (i_1, ..., i_n)$ and $AS_1, ..., AS_k$ are assignment statements of the form: $V_0 = E(V_1, V_2, ..., V_k)$, $k \in \mathbf{N}$, where $V_0$ is an output variable indexed by $\boldsymbol{i}$ and produced by expression E operating on input variables $V_1, V_2, ..., V_k$, also indexed by $\boldsymbol{i}$.

### 2.2. Notation

The sets of naturals, integers and rational numbers are denoted by **N**, **Z** and **Q** respectively. *n* is the number of nested FOR-loops and *m* is the number of dependence vectors of the algorithm. $J^n \subset \mathbf{Z}^n$ is the *set of indices*: $J^n = \{(j_1, ..., j_n) \mid j_i \in \mathbf{Z} \wedge l_i \le j_i \le u_i, 1 \le i \le n\}$. Each point in this *n*-dimensional integer space is a distinct instantiation of the loop body. A *dependence vector* is denoted as $\boldsymbol{d}_i = (d_{i1}, ..., d_{in})$, $1 \le i \le m$. The *dependence set D* of an algorithm A is the set of all dependence vectors of this algorithm: $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, ..., \boldsymbol{d}_m\}$. Note that all dependence vectors are considered *uniform*, i.e. independent of the indices of iterations. In the rest of the paper, $J^n$ will be interchangeably called *index space* or *iteration space* and $\boldsymbol{i}(i_1,\ldots i_n)$ are the *index* or *iteration points*.

### 2.3. Structural Properties of the Index Space

Every single iteration is depicted as a unique point $\boldsymbol{i}(i_1, i_2, \ldots, i_n)$ of the index space $J^n \subset \mathbf{Z}^n$. Supposing that variable *a* is generated in iteration $\boldsymbol{i} = (i_1, i_2, \ldots, i_n)$ and may be used in iteration $\boldsymbol{j} = (j_1, j_2, \ldots, j_n)$, there is a dependence vector for variable *a* between these two iterations. So, index points are linked through dependence vectors, creating *chains* of iterations (for more analytical definitions see [10], [5] and [1]). Since there are different dependence vectors, there may exist *alternative dependence paths* that link various index points. Consequently, there may be different ways of dividing the index space into chains of iterations.

On the other hand, dependence vectors impose *precedence constraints* between successive index point execution, thus partitioning the index space into sets, chains and anti-chains, which must be obtained through the establishment of a partial ordering among iterations (see also [1]). For every dependence $\boldsymbol{d}_k$, several distinct chains, of index points linked through $\boldsymbol{d}_k$, are formed. Every chain has a "*starting" point*, which has no "*ancestors*", thus it is the first in this chain. All such "starting" points form a *base set* relevant to this $\boldsymbol{d}_k$ vector (see also [5]):

***Definition 2.1*** *(base set)*
*Given an algorithm $A(J^n, D)$, a base set $B_{\boldsymbol{d}_k}$ with respect to a specific $\boldsymbol{d}_k \in D$, is defined as: $B_{\boldsymbol{d}_k} = \{\boldsymbol{b}_{\boldsymbol{d}_k} \in J^n \mid \boldsymbol{b}_{\boldsymbol{d}_k} - \boldsymbol{d}_k \notin J^n\}$.* ■

Note that the member points $\boldsymbol{b}_{\boldsymbol{d}_k}$ of $B_{\boldsymbol{d}_k}$ are called *base points*, with respect to the specific $\boldsymbol{d}_k \in D$. As shown in [5], $B_{\boldsymbol{d}_k}$ can be computed efficiently.

A chain that is generated by a dependence vector $\boldsymbol{d}$ and a base point $\boldsymbol{j}_0 \in B_{\boldsymbol{d}}$ is called a *uniform chain* with respect to $\boldsymbol{d} \in D$ (for chains and *antichains* see also [5]):

***Definition 2.2*** *(uniform chain)*
*A chain $C_{\boldsymbol{d}} = \boldsymbol{j}_0, ..., \boldsymbol{j}_k$ of $J^n$ is a uniform chain with respect to $\boldsymbol{d}$ iff:*

- $\boldsymbol{j}_{i+1} = \boldsymbol{j}_i + \boldsymbol{d}$, $0 \le i \le k-1$ *and*
- $\boldsymbol{j}_0 - \boldsymbol{d} \notin J^n$ *and* $\boldsymbol{j}_k + \boldsymbol{d} \notin J^n$.

*$C_{\boldsymbol{d}}$ is completely determined by its minimum element $\boldsymbol{j}_0$ because $\boldsymbol{j}_i = \boldsymbol{j}_0 + i \cdot \boldsymbol{d}$, where $\boldsymbol{j}_0$ is called the basis of $C_{\boldsymbol{d}}$. Clearly $\boldsymbol{j}_0 \in B_{\boldsymbol{d}}$.* ■

A *time schedule* divides the index space into antichains, where each antichain contains non-related iterations, scheduled at the same time instance. On the other hand, *space schedule*, divides the index space into chains, where each chain contains related iterations assigned to the same processor. Grouping methods try to cluster neighboring chains together, while keeping track of the following criteria: minimizing the *overall completion time* and minimizing *communication* between iterations which belong to different groups (processors).

## 3. Projection Grouping Methods

Partitioning methods based on *orthogonal projection* are following similar techniques, and only the projection

and grouping parameters are specifically defined and optimized (for method details see [9] and [4]). Neighboring chains of index points are grouped together, according to different in each method criteria. In order, for a certain method, to show up the close vicinity of the index points that are clustered together, the index space is *projected* onto subspaces. Then, certain sets of linked points, inside these subspaces, are grouped together and assigned to the same processor.

The procedure followed by all methods includes the following steps. The $n$-dimensional iteration space $J^n$ is projected along a $g$-dimensional subspace $\Psi \subset \mathbf{Q}^g$ and it is transformed into a $k$-dimensional space (where $k \leq n$). The grouping is done along this $k$-dimensional (projected) subspace $J_P{}^k = \text{Null}(\Psi)$ so that $k = n - g$ or $n = k + g$ (recall that $\Psi$ and $J_P$ are perpendicular). Every group inside the subspace $J_P{}^k$ is assigned to a different processor. Obviously, the dimension $k = \dim(J_P{}^k) = \dim(\text{Null}(\Psi))$ determines the dimension of the required processor array.

In general, subspace $\Psi$ is generated by one or more dependence vectors, in order to minimize communication between the generated groups. Each of the above methods proposes a different procedure to reduce communication overhead between different groups, which means different criteria of selecting the optimal projection space.

## 3.1. SC Method

Sheu and Chen in [9] project $J^n$ onto an 1-dimensional space (projection line) along a ($n$-1)-dimensional projection space $\Psi$. Groups of neighboring iterations are formed along the ($n$-1)-projection space $\Psi$. Heuristic rules are used to select the projection space, based on the cardinality of dependence vectors that span the space $\Psi$ and their length. The projection space is $\Psi = \text{span}(P)$, where $P$ is selected so as to meet the following rules:

***Rule 1:*** *As much dependence vectors as possible are chosen from set D to form set P, where: dim(span(P)) = n-1.* (In other words set $P$ must satisfy the condition that rank(mat($P$)) = $n$-1, such that $|P|$ is maximal.)

***Rule 2:*** *If more than one set satisfies Rule-1, the set with the minimum value of* $\sum_{\boldsymbol{d}_i \in P} len(\boldsymbol{d}_i)$ *is chosen to produce the projection space* $\Psi$, *where* $len(\boldsymbol{d}) = \sqrt{\sum_{1 \leq j \leq n} (d_j)^2}$ .

Once set $P$ is selected, the index space is projected to the 1-dimensional space that is perpendicular to span($P$) space. Thus, it can be directly mapped onto a linear array of processors. Points with the same projection are assigned to the same processor.

**Example 1.**

Consider the following algorithm A($J^2$, $D$):

```
for j1 = 0 to 4 do
 for j2 = 0 to 5 do
  a(j1, j2) = a(j1, j2-1) + a(j1-1, j2) +
              a(j1-1, j2-1) + a(j1-1, j2-2)
 end
end
```

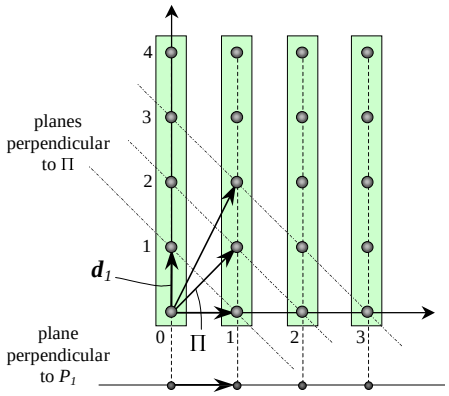


**Figure 2.** Projection using $\boldsymbol{d}_1 = (0, 1)$ as the projection vector.

The index space is $J^2 = \{(j_1, j_2) \mid 0 \leq j_1 \leq 4, 0 \leq j_2 \leq 5\}$ and the set of dependence vectors is $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \boldsymbol{d}_3, \boldsymbol{d}_4\}$, where $\boldsymbol{d}_1 = (0, 1)$, $\boldsymbol{d}_2 = (1, 0)$, $\boldsymbol{d}_3 = (1, 1)$, and $\boldsymbol{d}_4 = (1, 2)$. By applying *Rule-1*, four sets $P_1 = \{\boldsymbol{d}_1\}$, $P_2 = \{\boldsymbol{d}_2\}$, $P_3 = \{\boldsymbol{d}_3\}$ and $P_4 = \{\boldsymbol{d}_4\}$ can be obtained, since rank(mat($P_1$)) = rank(mat($P_2$)) = rank(mat($P_3$)) = rank(mat($P_4$)) = 1 and $|P_1| = |P_2| = |P_3| = |P_4| = 1$. By applying *Rule-2* we find out that $len(\boldsymbol{d}_1) = len(\boldsymbol{d}_2) = 1$, $len(\boldsymbol{d}_3) = \sqrt{2}$ and $len(\boldsymbol{d}_4) = \sqrt{5}$. Consequently, we can choose either $P_1$ or $P_2$ as set $P$. In *Figure-2* we show the projection and mapping on a line perpendicular to set $P_1$. ■

## 3.2. ST Method

Sheu and Tai in [10], proposed the projection of the whole index space $J^n$, onto a plane perpendicular to the hypreplane vector $\Pi$, in order to preserve the optimal hyperplane schedule. In other words, $\Psi$ is 1-dimensional, since $\Psi = \text{span}(\Pi)$ and the perpendicular subspace $\text{Null}(\Psi) \equiv \text{Null}(\text{span}(\Pi))$ is ($n$-1)-dimensional, thus resulting in mapping on ($n$-1)-dimensional processor arrays. After the projection has taken place, a grouping is performed along the so called *grouping* and *auxiliary grouping vectors* that are defined below:

***Definition 3.1*** *(grouping vector)*

*Let* $r_i$ *be the smallest positive integer such that:* $r_i \cdot \boldsymbol{d}_i^p \in \boldsymbol{Z}^n$. *The projected dependence vector* $\boldsymbol{d}_l^p$ *is selected as grouping vector iff:* $r_l = max_{\boldsymbol{d}_i^p \in D^p}\{r_i\}$, *where* $D^p$ *represents the set of the projected dependence vectors.* ■

Supposing that rank(mat($D^p$)) = $\beta$, $\beta$-1 more dependence vectors are selected from $D^p$ - { $\boldsymbol{d}_l^p$ }, such that they

are linearly independent with $\boldsymbol{d}_l^p$. These $\beta$-1 additional dependence vectors are called *auxiliary grouping vectors*, as together with the grouping vector, give the ability of grouping along the $n$-1 dimensional projected space (in other words: $n$-1 = $\beta$). The following definitions are necessary for process formal presentation.

***Definition 3.2*** *(neighbouring groups)*

*Suppose* $u_0^p$, $v_0^p$ *and* $w_0^p$ *are the base vertices of groups* $G_i$, $G_j$ *and* $G_k$. *If* $u_0^p = v_0^p - r \cdot \boldsymbol{d}_l^p$ $(= v_0^p - \boldsymbol{d}_j^p)$, *and* $u_0^p = w_0^p + r \cdot \boldsymbol{d}_l^p$ $(= w_0^p + \boldsymbol{d}_j^p)$, *then* $G_j$ *and* $G_k$ *are the forward and backward neighbouring groups of* $G_i$, *along the grouping vector* $\boldsymbol{d}_l^p$ *(along the auxiliary grouping vector* $\boldsymbol{d}_j^p$*), respectively.* ■

SC method can now fully-described by the following steps:

***Step 1:*** *Select* $\Pi$ *as the projection vector* $\boldsymbol{d_k}$ $(\Psi = span(\boldsymbol{d_k}) = span(\Pi))$.

***Step 2:*** *Project* $J^n$ *and* $D$ *onto the (n-1)-dimensional Null(span(*$\Pi$*)). Projected* $J_p^{n-1}$ *and* $D^p$ *are obtained.*

***Step 3:*** *Select grouping vector* $\boldsymbol{d}_l^p$ *as the one that maximizes grouping factor r:*

$$r = r_l = max_{\boldsymbol{d}_i^p \in D^p} \left\{ r_i : r_i = min\{z \in \mathbf{N}\} : r_i \cdot \boldsymbol{d}_i^p \in \mathbf{Z}^n \right\}.$$

***Step 4:*** *Select* $\beta$*-1 auxiliary grouping vectors from* $D^p$ - $\{ \boldsymbol{d}_l^p \}$, *so as:*

$$span(\{ \boldsymbol{d}_l^p, \boldsymbol{d}_{aux_1}^p, \ldots, \boldsymbol{d}_{aux_{\beta-1}}^p \}) = dim(Null(\Psi)) = n\text{-}1.$$

*[After Step 4, the projected structure can be split into parallel lines along the direction of* $\boldsymbol{d}_l^p$ *.]*

***Step 5:*** *Select a line arbitrarily and choose a projected point as the base vertex. Starting from that vertex, group every r points along* $\boldsymbol{d}_l^p$ *into a group. Let these groups be the initial seed groups.*

***Step 6:*** *From the seed groups, find all backward and forward neighbouring groups and use these as seed groups. Repeat step until there exists no neighbouring group of the seed groups.*

***Step 7:*** *If there are some lines whose projected points are not grouped, go to Step 5.*

Note that, for every group $G_i = \{ v_0^p, v_1^p, \ldots, v_{r-1}^p \}$ of the projected structure, the corresponding partitioned block $B_i$ is the set:

$$\bigcup_{v_k^p \in G_i} \{ \boldsymbol{j} \in J^n \wedge \boldsymbol{j} = v_k^p + t \cdot \Pi, t \in \mathbf{R} \}.$$

**Example 2.**

Consider algorithm A($J^2$, $D$) of *Example-1*. For the dependence vectors mentioned before, the optimal hyperplane $\Pi$ is [1, 1], which gives Null(span($\Pi$)) = span({(-1, 1)}). These give $D^p$ = {1, -1, 0, 1} and $r$ = 1, taken from grouping vector $\boldsymbol{d}_1^p = 1$.

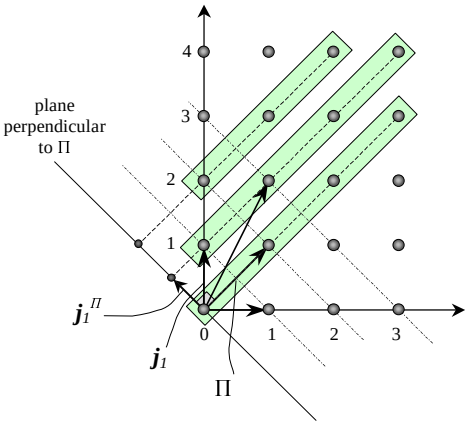


**Figure 3.** Projection using $\Pi$ as the projection vector.

In this example there is no need of auxiliary grouping vectors, as the projected space is linear. In *Figure-3* we show the projection and mapping on a line perpendicular to $\Pi$. ■

## 3.3. CG Method

Chain Grouping, presented in [5], applied the orthogonal projection to map onto a ($n$-1)-dimensional array. This method is quite similar to ST and also preserves the optimal hyperplane time schedule. A dependence vector is chosen from set $D$ and the index space is projected to a ($n$-1)-dimensional space that is perpendicular to the chosen dependence vector. This vector is called *projection vector* $\boldsymbol{d}_k$. $\Psi$ is 1-dimensional since $\Psi$ = span($\boldsymbol{d}_k$) and the perpendicular subspace is consequently ($n$-1)-dimensional. Chain Grouping clusters into the same processor uniform chains of iterations, formed by the projection vector $\boldsymbol{d}_k$. The number of uniform chains that are clustered in the same processor, is defined by a *grouping factor*, given by

$$r = \left\lfloor \frac{\Pi \cdot \boldsymbol{d}_k / \gcd(d_{ki})}{\mathrm{disp}\Pi} \right\rfloor.$$

***Definition 3.3*** *(projection vector)*

*Given an algorithm A($J^n$, D) and the corresponding optimal hyperplane* $\Pi$, *the projection vector* $\boldsymbol{d}_k$ *is selected as:*

$$\boldsymbol{d_k}: \left\lfloor \frac{\Pi \cdot \boldsymbol{d}_k / \gcd(d_{ki})}{\mathrm{disp}\Pi} \right\rfloor = \max_{\boldsymbol{d_j} \in D,\ i = 1, \ldots, n} \left\{ \left\lfloor \frac{\Pi \cdot \boldsymbol{d}_j / \gcd(d_{ji})}{\mathrm{disp}\Pi} \right\rfloor \right\},$$

*where gcd(.) is the greatest common divisor of the coordinates* $d_{ji}$ *of* $\boldsymbol{d}_j$. ■

Once the projection vector $\boldsymbol{d}_k$ is selected from set $D$, the method projects all points of $J^n$ to the plane perpendicular to $\boldsymbol{d}_k$. The following definition determines the

projection of a point $\boldsymbol{j}(j_1, \ldots, j_n)$, with respect to the projection vector $\boldsymbol{d}_k$.

***Definition 4.4*** *(projected point)*
*Given a projection vector $\boldsymbol{d}_k(d_{k1},\ldots,d_{kn})$, the projected point $\boldsymbol{j}^{\boldsymbol{d}_k}$ is defined as the projection of the index point $\boldsymbol{j}(j_1,\ldots, j_n)$ onto a plane perpendicular to $\boldsymbol{d}_k$. Mathematically:* $\boldsymbol{j}^{\boldsymbol{d}_k} = \boldsymbol{j} - \frac{\boldsymbol{j}\cdot\boldsymbol{d}_k}{\boldsymbol{d}_k\cdot\boldsymbol{d}_k}\boldsymbol{d}_k$ *(see [10] for details).* ■

Consequently, the projection of all points of $J^n$ produces the *Projected Index Space*. The vector that is used to group the projected points of $J^n$ into groups is called *grouping vector*.

***Definition 4.5*** *(grouping vector)*
*Given an algorithm A($J^n$, D) and the corresponding optimal hyperplane Π, the grouping vector is selected as:*
$\boldsymbol{d}_g$: $\Pi\boldsymbol{d}_g = min\{\Pi\boldsymbol{d}_i, i = 1, ..., m\}$, *where* $\boldsymbol{d}_i \in \{D - \boldsymbol{d}_k\}$. ■

Once the projection vector $\boldsymbol{d}_k$ is selected, the method finds set $B_{\boldsymbol{d}_k}$, the base set with respect to $\boldsymbol{d}_k$.

CG method can be fully-described by the following steps:

***Step 1****: Selection of projection vector $\boldsymbol{d}_k$ and of grouping vector $\boldsymbol{d}_g$.*
*(Hint: If there exist more than one $\boldsymbol{d}_k$ with minimum r, repeat steps 2-4 for all candidate $\boldsymbol{d}_k$'s and select the one that creates the projected base set of step 2, with less cardinality.)*

***Step 2****: Finding set $B_{\boldsymbol{d}_k}$ and the projected base set $B^{\boldsymbol{d}_k}_{\boldsymbol{d}_k}$.*

***Step 3****: Partitioning the projected base set $B^{\boldsymbol{d}_k}_{\boldsymbol{d}_k}$ into subsets $B^{\boldsymbol{d}_k}_{m,\boldsymbol{d}_k}$ using the grouping factor r.*

***Step 4****: Finding the uniform chains $C_{b_i^{\boldsymbol{d}_k}}$*

***Step 5****: Grouping together the uniform chains $C_{b_i^{\boldsymbol{d}_k}}$ into the groups $G_m^{\boldsymbol{d}_k}$ and assigning each group to a different processor.*

**Example 3.**

Consider the algorithm of *Example-1*. The optimal hyperplane time schedule Π is [1, 1]. The grouping factor *r* is: $r = max\left\lfloor \frac{\Pi\cdot\boldsymbol{d}_j / \gcd(d_{ji})}{\mathrm{disp}\Pi} \right\rfloor$, $j$ = 1, ..., 4 and it is $r$ = 3 for $\boldsymbol{d}_4$ vector, which is chosen to be the projection vector.

The grouping vector is one out of $\boldsymbol{d}_1$, $\boldsymbol{d}_2$. We arbitrarily choose $\boldsymbol{d}_1$ = (0, 1). The base set of the index space is: $B_{\boldsymbol{d}_4}$ = {(0, 5), (0, 4), …, (3, 1), (4, 0), (4, 1)}. The whole projected basis set is: {(-2, 1), (-8/5, 4/5), …, (12/5, -6/5), (14/5, -7/5), (16/5, -8/5)}. If we group each 3 points along the direction of $\boldsymbol{d}_1^{\boldsymbol{d}_4}$ we have the following partitioning:

$B^{\boldsymbol{d}_4}_{1,\boldsymbol{d}_4}$ = {(-2, 1), (-8/5, 4/5), (-6/5, 3/5)} →

$B_{1,\boldsymbol{d}_4}$ = {(0, 3), (0, 4), (0, 5)},

…

$B^{\boldsymbol{d}_4}_{5,\boldsymbol{d}_4}$ = {(14/5, -7/5), (16/5, -8/5)} →

$B_{5,\boldsymbol{d}_4}$ = {(4, 0), (4, 1)}.

*Figure-4* shows partitioning and mapping according to Chain Grouping method.

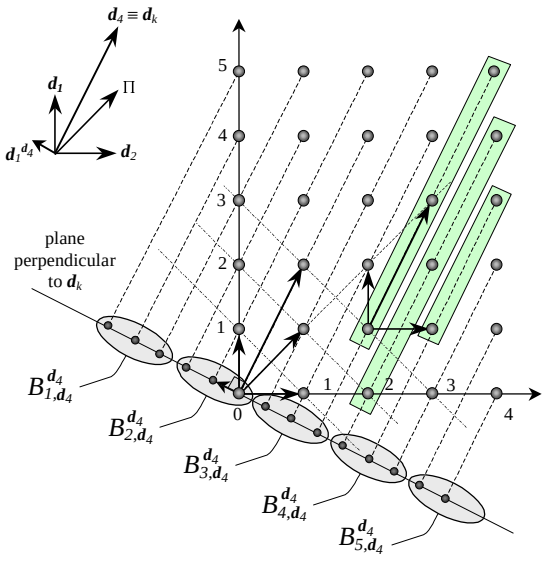


**Figure 4.** The partitioning of the projected base set $B^{\boldsymbol{d}_k}_{\boldsymbol{d}_k}$ along the projected grouping vector $\boldsymbol{d}_1$. ■

## 4. Method Comparison

In this section we compare the three methods in terms of performance on certain criteria, considering various sizes of iteration spaces and dependence sets. We focus on evaluating *communication cost*, *total execution time* and *algorithm complexity*. It is important to clarify that an objective comparison requires 2-dimensional problems, where all three methods project to a 1-dimensional space (mapping onto a linear processor array). Nevertheless, for the sake of completeness, we further elaborate on methods' behavior in iteration spaces with dimension greater than two.

### 4.1. SC Method

As above analyzed in method presentation, the *n*-dimensional index space is projected onto a line to form a 1-dimensional projected structure. A projection space $\Psi$ = span(*P*) is chosen, such that the projected line will lead to

a partitioning with low communication overhead. For this reason set $P$ contains as many dependence vectors as possible (*Rule-1*). If possible, these vectors should be of small length (*Rule-2*), since the amount of data communication caused by small vectors is larger.

This method, generally, performs well in terms of communication minimization, especially when $P$ containing more than $n$-1 dependence vectors ($|P| > n$-1) and $n > 2$. However, if the dependence set contains the unitary vectors (a fact that is quite common in nested loops) and $|P| = n$-1, method's performance decreases dramatically (compare *examples 5.1-5.3* with *5.6-5.8*). In addition, this method does not take into consideration the optimal execution of iterations on time, thus it does not exploit all inherent parallelism. However, one can claim that clustering together iterations, which belong to the same time hyperplane, may increase the grain of parallelism and be efficient for multiprocessors. What really happens is that, the different groups are closely related and have continuous data exchange (fine graining).

Consequently, SC neither achieves the optimal parallelization time nor does improve the communication cost, than the other two algorithms as we will see further on (see example in CG method presentation).

As far as the algorithm projection complexity is concerned, it doesn't perform so well, due to the calculation of set $P$ and the traversing of the whole index space on projection (the other two methods avoid traversing whole space by defining certain base sets). The method involves calculating all candidate $\binom{m}{k}$ $P$ sets, where $k = m$-1. If rank(mat($P$)) $\neq n$-1, calculation continues for $k = m$-2, etc., until k = $n$-1, thus $O(m^3 \cdot n)$. Space traversing complexity is $O(|J^n|)$.

An indubitable advantage of this method is that, no matter what the initial dimension is, it always performs a mapping onto a linear array of processors. In other words, we have processor dependent mapping (linear array), no matter how big the loop's dimension is.

### 4.2. ST Method

In this method, the $n$-dimensional index space is projected onto a ($n$-1)-dimensional space that is perpendicular to the hyperplane vector $\Pi$. In other words, the performed projection *does not* take into consideration the dependence set.

Communication reduction is achieved by grouping neighboring chains of iterations. The grouping factor is derived from the coordinates of the projected dependence vectors. The projected space is Null($\Pi$) and grouping is performed along this space. As far as execution time is concerned, this method behaves quite well, as it preserves the optimal hyperplane time schedule along $\Pi$.

Method's projection involves grouping and auxiliary grouping vector selection, as well as base set computation. Grouping vector selection is of $O(m \cdot n)$ complexity, while selection of auxiliary grouping vectors is of $O(m \cdot n^2)$ complexity. Base set can be computed efficiently, as shown in [5].

As far as the algorithmic model is concerned, this method introduces the idea of the *base set*. This technique avoids whole space traversing on index space projection, which means that $O(|J^n|)$ complexity is avoided. When projecting, it suffices to project only the base set and all other points are derived from it.

A disadvantage of this method (as well as of CG method too) is that it requires a ($n$-1)-dimensional processor array to apply the mapping onto, which means that in large dimension problems ($n > 4$) this method is of no practical interest.

### 4.3. CG Method

Chain grouping resembles ST method; it projects onto a ($n$-1)-dimensional space, by choosing a projection vector from the dependence matrix $D$. The criterion used to select the projection vector from set $D$, is maximizing the so called, grouping factor $r$. As one can see from its definition, this method takes into consideration both the algorithm time flow and the dependence vectors. In other words, mapping is both time and space dependent.

This method performs very well in terms of low communication especially when $r > 1$. In 2-dimensional problems, where all three methods can be fairly compared, this method has clearly the best performance. In larger dimensions, is still indubitably better than ST method, but cannot follow the performance of SC in terms of overall communication overhead. This is normal, since SC maps to quite fewer processors than the other two methods. This means that the iterations that are clustered together by SC are more than in CG and ST. However, the need for communication with the other processors is as often as in CG, thus also requiring for frequent synchronization among neighboring processors. This increases dramatically the overall completion time and makes the overall communication reduction an impractical asset.

This method preserves the optimal time schedule, given by vector $\Pi$ (see [5] for proof details). As far as complexity is concerned, this method has the same, relatively low complexity, as ST method.

## 5. Examples

In this section, we compare the performance of three methods, in terms of communication cost, using examples with specific multi-dimensional iteration spaces. We have developed a tool, called LOOPDEP, which takes as input

either concrete iteration spaces or randomly generated ones. For both ST and CG methods, the optimal hyperplane vector Π is computed according to method presented in [8]. Random dependence set generator produces different vectors according to several constrains, like number of vectors and maximum coordinate value. We focused mainly on 2-dimensional iteration spaces since, as we have mentioned before, all three methods map onto a linear processor array. The example algorithms have been selected to be as realistic as possible, with dependence vectors similar to those actually met in practice.

To calculate the communication cost, we apply all three methods and count all crossings of group boundaries, for all dependence vectors, thus representing communication requirements.

**Example 5.1.**

Consider again the algorithm A($J^2$, $D$) used in the examples of *Section-4*.

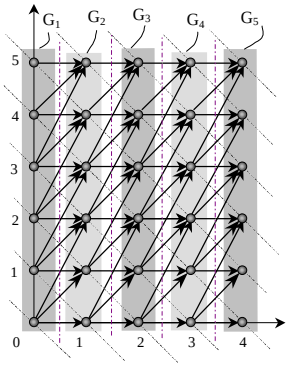


**Figure 5.** SC along $\boldsymbol{d}_1$.

The index space is $J^2 = \{(j_1, j_2) \mid 0 \le j_1 \le 4, 0 \le j_2 \le 5\}$, and the set of dependence vectors is $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \boldsymbol{d}_3, \boldsymbol{d}_4\}$, where $\boldsymbol{d}_1 = (0, 1)$, $\boldsymbol{d}_2 = (1, 0)$, $\boldsymbol{d}_3 = (1, 1)$, and $\boldsymbol{d}_4 = (1, 2)$.

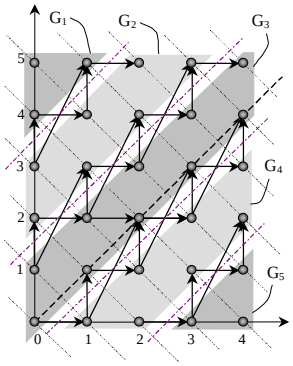


**Figure 6.** ST along Π.

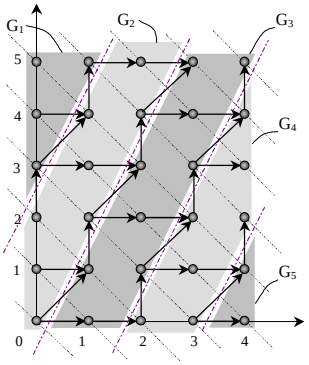


**Figure 7.** CG along $\boldsymbol{d}_4$.

The communication cost of the three methods is:

**SC**: 60, **ST**: 32, **CG**: 30.

In *Figures 5-7* we show the mapping according to the three methods, as well as the calculation of the total communication cost, by counting the number of dependence vectors that cross the boundaries. ■

**Table 1.** Communication cost in specific 2-dimensional problems with size 10×10.

| No | set *D* | Π | Comm. Cost | | |
|---|---|---|---|---|---|
| | | | SC | ST | CG |
| **5.2** | (0, 1) (1, 0) (1, 1) | [1, 1] | 171 | 90 | 90 |
| **5.3** | (0, 1) (1, 0) (1, 1) (1, 3) | [1, 1] | 234 | 153 | 130 |
| **5.4** | (0, 1) (1, 0) (1, 2) | [1, 1] | 162 | 126 | 90 |
| **5.5** | (0, 1) (1, 0) (1, 3) | [1, 1] | 153 | 153 | 90 |
| **5.6** | (1, 1) (1, 2) (1, 3) | [1, 0] | 135 | 216 | 135 |
| **5.7** | (1, 1) (1, 2) (2, 1) | [1, 0] | 144 | 225 | 117 |
| **5.8** | (1, 1) (1, 3) (3, 1) | [1, 0] | 126 | 226 | 117 |
| **5.9** | (1, 2) (1, 3) (2, 1) (3, 1) | [1, 0] | 198 | 270 | 162 |
| **5.10** | (1, 1) (1, 3) (1, 2) (3, 1) | [0, 1] | 198 | 279 | 144 |
| **5.11** | (1, 0) (0, 1) (1, 2) (2, 4) (1, 3) | [1, 1] | 243 | 237 | 132 |

We first applied LOOPDEP in several specific 2-dimensional iteration spaces, presented in the following *Table-1*.

**Table 2.** Average communication cost for CG, ST and SC for 2-dimensional iteration spaces with size 10×10.

| *m* | max cord | CG | ST | SC |
|---|---|---|---|---|
| 2 | 3 | 60 | 109 | 74 |
| 2 | 4 | 60 | 89 | 73 |
| 2 | 5 | 50 | 78 | 64 |
| 3 | 3 | 118 | 130 | 137 |
| 3 | 4 | 123 | 130 | 123 |
| 3 | 5 | 102 | 110 | 102 |
| 4 | 3 | 173 | 192 | 174 |
| 4 | 4 | 143 | 166 | 176 |
| 4 | 5 | 139 | 161 | 146 |

**Table 3.** Communication cost in 3, 4 & 5-dimensional problems.

| *n* | *m* | Average Communication Cost | | |
|---|---|---|---|---|
| | | SC | ST | CG |
| 3 | 3 | 73 | 144 | 104 |
| 3 | 4 | 106 | 210 | 150 |
| 4 | 4 | 371 | 880 | 388 |
| 4 | 5 | 547 | 1119 | 894 |
| 5 | 5 | 3271 | 8498 | 6750 |
| 5 | 6 | 2897 | 10138 | 8679 |

For a general overview on higher dimension problems, we applied LOOPDEP to various, randomly generated, 3, 4, 5-dimensional iteration spaces with different sizes, number and size of dependence vectors and calculated the communication cost for them in terms of intergroup links.

*Table-2* summarizes the results. The values below are the average of 10 repetitions for different instances of the iteration spaces, having the specifications depicted on first and second columns of the table. *Table-3* summarizes results for iteration spaces with dimension larger than 2.

### Comments on the Results

**1.** In two-dimensional algorithms (*examples 5.1-11*) CG method performs clearly better than the other two. This is due to the fact that this method not only chooses the projection vector that results to minimum communication, but also performs a time optimal grouping along the grouping vector. Notice the results of *example-4*. CG projects perpendicular to (1, 2), ST perpendicular to Π[1, 1] and SC perpendicular to (1, 0). Apparently, the choice of CG leads to mapping with the lowest communication cost. Notice also the results of *example-5.11*. Both CG and SC project perpendicular to (1, 2). CG performs a grouping along (1, 0), which nearly halves the communication cost without extending the optimal execution time.
**2.** Notice the large communication cost of SC in *examples 5.1-5* and *5.11*. SC performs badly because of the presence of the two unitary vectors, which due to their size prevail in the selection phase as the possible projection vectors. Applying *Rule-1* and *2* of the method we project to a space perpendicular to (1, 0), only avoiding the communication cost caused by this particular vector. We thus ignore the other unitary vector (0, 1) and all other vectors, which cause this large communication cost.
**3.** Communication minimization on ST depends on the deviation of the hyperplane vector Π from the direction of the dependence vectors (compare *example 5.2* with *5.7* & *5.10*). Zero deviation (if dependence vectors are parallel to Π) results in independent groups, while large deviation results in no communication reduction at all.
**4.** In larger dimensional problems it is obvious that SC method outperforms the other two in terms of communication minimization. Low communication is achieved by projecting perpendicular to a large number of vectors (totally $|P|$), thus eliminating the effect of those vectors to total communication cost. CG and ST project perpendicular to one vector, eliminating the effect of this particular vector only.
**5.** CG outperforms ST in larger dimensional problems, as well, when only communication overhead is considered. Note that if $\Pi = \boldsymbol{d}_k$ and $r = 1$, the two methods perform identical mappings.

## 6. Conclusion

In this paper we presented and compared three methods for grouping loops based on orthogonal projection. Examining every method's steps, together with specific results taken from experimental examples, we come to the following conclusions. Concerning 2-dimensional algorithms, CG method is undoubtedly the most efficient one. It scores lower communication cost, optimal execution time and relatively low complexity. In larger dimensional problems, SC method indeed outperforms the other two in terms of communication cost, without promising better results in the total execution time (which is certainly at least as much as the optimal execution time of the other two methods). This is normal because, although overall communication overhead is reduced, the frequency of need for communication remains the same.

Another fact that should be taken into consideration, when applying these methods in practice, is the underlying processor architecture. SC maps onto a linear processor array whereas CG and ST map onto a ($n$-1)-dimensional one.

## 7. Acknowledgments

This work was partially funded by the Ministry of Development, General Secretariat for Research and Technology, project PENED 99ΕΔ521.

## 8. References

[1] T. Andronikos, N. Koziris, Z. Tsiatsoulis, G. Papakonstantinou and P. Tsanakas, "Lower Time and Processor Bounds for Efficient Mapping of Uniform Dependence Algorithms into Systolic Arrays," *Journal of Parallel Algorithms and Applications*, vol. 10, 3-4, pp. 177-194, 1997.
[2] A. Darte, Y. Robert, "Mapping Uniform Loop Nests onto Distributed Memory Architectures," *Parallel Computing*, vol. 20, pp. 679-710, 1994.
[3] A. Darte and Y. Robert, "Constructive Methods for Scheduling Uniform Loop Nests," *IEEE Trans. Parallel Distrib. Syst*., vol. 5, no. 8, pp. 814-822, Aug. 1994.
[4] S. H. Friedberg, A. J. Insel, L.E. Spence, Linear Algebra. *Prentice-Hall, Englewood Cliffs. N.J. 1979.*
[5] N. Koziris, G. Papakonstantinou and P. Tsanakas, "Mapping Nested Loops onto Distributed Memory Multiprocessors," *Proceedings of the 1997 IEEE Intern. Conf. on Parallel and Distributed Systems (ICPADS97)*, IEEE Press, pp. 35-41, Seoul, Dec. 1997.
[6] D.I. Moldovan and J.A.B. Fortes, "Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays," *IEEE Trans. Comput*., vol C-35, no 1, pp. 1-11, Jan. 1986.
[7] J.K. Peir and R. Cytron, "Minimum Distance: A Method for Partitioning Recurrences for Multiprocessors," *IEEE Trans. Comput*., vol. 38, no. 8, pp.1203-1211, Aug. 1989.
[8] W. Shang and J.A.B. Fortes, "Time Optimal Linear Schedules for Algorithms with Uniform Dependencies," *IEEE Trans. Comput*., vol. 40, no. 6, pp. 723-742, June 1991.
[9] J.-P. Sheu and T.-S. Chen, "Partitioning and Mapping Nested Loops for Linear Array Multicomputers," *Journal of Supercomputing*, vol. 9, pp. 183-202, 1995.
[10] J.-P. Sheu and T.-H. Tai, "Partitioning and Mapping Nested Loops on Multiprocessor Systems," *IEEE Trans. on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 430-439, Oct. 1991.