Mapping Nested Loops onto Distributed Memory Multiprocessors

Nectarios Koziris, George Papakonstantinou and Panayotis Tsanakas

National Technical University of Athens
Dept. of Electrical and Computer Engineering
Computer Science Division
Zografou Campus, Zografou 15773, Greece
e-mail:{nkoziris, papakon}@dsclab.ece.ntua.gr

Abstract

This paper presents Chain grouping; a new low complexity method for the problem of partitioning the index space into groups with little intercommunication requirements, for mapping onto distributed mesh-connected architectures. First the loop iterations are scheduled in time, according to the hyperplane method, taking into consideration the minimum time displacement. Then, the index space is divided into discrete groups of related computations, which are assigned to different processors, while preserving the optimal makespan. The Chain grouping method is based on grouping along a uniform chain of computations, formed by a particular dependence vector. This vector will be proved as the best to reduce the total communication requirements. Inside every group, the optimal hyperplane scheduling is preserved, and the references to intragroup computations are considerably increased. The partitioned groups are afterwards assigned to meshes of processors. The resulting space mapping maximises processor utilisation and cuts down overall communication delays while preserving the optimal hyperplane time schedule.

Index Terms --- *Loop partitioning, hyperplane method, distributed architectures, uniform chains of computations***.**

1. INTRODUCTION

The primary task in parallelizing a FOR-loop is finding a schedule of computations into time, while preserving the data dependencies of the initial lexicographic loop order. Many methods were proposed for finding the dependencies between different statements, and how these affect the loop's inherent parallelism. Dependence analysis methods, such as these evaluated in [12], are used to trace the actual dependence set. Since dependence vectors describe the flow of computations they are used to find the optimal parallel execution time. The lower bound on the parallel execution time and on the number of processors for nested loops with uniform dependencies were presented in [1]. As far as linear schedules are concerned, the widely used method is the hyperplane method, presented by Lamport in [9]. This method was further extended by other researchers such as Shang in [14] with the introduction of displacement, or Darte [3] by affine by statement schedules, thus allowing more computations to be executed in parallel. The main notion behind such scheduling is the introduction of a linear transformation, which organizes indexed computations into well-defined distinct groups called hyperplanes. Mathematical conditions which guarantee validity for a linear schedule were well defined and presented, and algorithms which find the optimal linear schedule were extensively elaborated [4], [10], [14].

Linear schedules in loop parallelization were used in almost every attempt to map a nested loop into systolic architectures. Linear transformations affect all indexed computations uniformly, thus producing time schedules which fit into regularly interconnected parallel architectures. Typical examples for linear transformations are mapping transformations for systolic arrays [10], which due to their linearity, impose no artificial communication overhead, or job management requirements which would increase circuitry complexity, or schedules for grid task graphs with zero or unit communication delays [2], [8].

Once optimal parallel execution is found, an efficient method of mapping the concurrent groups of computations into a parallel architecture should be applied. In the past, many proposed mapping methodologies were focused upon special purpose parallel architectures such as systolic arrays. Since systolic cells are synchronously operating, there is no need for communication efficient mapping. Fewer papers dealt with the problem of mapping nested loops into MIMD architectures. In message passing multiprocessor systems, communication delays play the most important role, since they can increase dramatically the total execution time. Researchers are trying to minimise the interprocessor communication, thus increasing the intraprocessor activity, by dividing the index space into as much as possible independent groups of computations. Shang in [13], Hollander in [5] and Peir in [11], have presented methods for dividing the index space into independent sets of computations which are assigned to different processors, thus zeroing the communication cost. Unfortunately the independent partitioning of the index space in not always feasible. Sheu and Tai, in [15], presented a systematic method of partitioning the index space into groups and assign them into different processing elements. Their method is 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 lead to the best results. Although such a grouping preserves the optimal time execution policy, it does not consider increasing the local references inside every group. Furthermore, space partitioning is done without taking into consideration the notion of displacement, thus giving poor results when displacement is greater than one. Nevertheless, this approach had the advantage of proposing a linear space scheduling, with low complexity. In addition to this, King et al in [6], have presented a method, which groups related computations, together, following one or more dependence vectors. However, their method exhaustively transverses the whole index space, to define the group where each computation belongs. Time schedule is not explicitly defined. Furthermore they only discuss computational structures on two dimensional index spaces. However, they introduced the idea of grouping along dependence vectors, which reduces overall communication requirements.

In this paper we will present Chain Grouping, a method which considerably enhances the results presented by Sheu and Tai, in [15]. We propose a different index space partitioning. Grouping is done along a uniform chain of computations formed by a specific dependence vector. Grouping along a dependence vector ensures that, for every index point, references to previous computation through this dependence vector, will be satisfied inside the same processor. Furthermore, the grouping vector is chosen so as to maximise the references inside the same processor, through other dependence vector paths. In other words, chain grouping not only includes directly dependent computations to be executed on the same processor, but also computations which depend on each other through additional non-uniform dependence paths. In addition to this, Chain grouping preserves the optimal time schedule determined by the hyperplane method of [9]. The outperforming results of our proposed Chain Grouping are shown through a complete example, together with a comparison with the method presented in [15].

The rest of the paper is organised as follows: Basic terminology and definitions used throughout this paper are introduced in Section 2. In Section 3 the structural properties of the index space are presented. Finally, Section 4 presents the Chain grouping, the proposed method for dividing the index space into communication reduced groups while preserving efficient communication and processor utilisation, together with an illustrative example.

2. Preliminary Concepts and Definitions

2.1. The Model of the Algorithms

In this paper we will use as a computational model nested FOR-loop algorithms with *uniform* data dependencies. That is, our algorithms are of the form:

FOR $I_1=l_1$ TO u_1 DO
...
FOR $I_n=l_n$ TO u_n DO
$AS_1(I)$
...
$AS_k(I)$
ENDFOR

where: (1) l_i and u_i are integer-valued constants, (2) $\mathbf{I}=(I_1, ..., I_n)$ and (3) $AS_1, ..., AS_k$ are assignment statements of the form $V_0=E(V_1, ..., V_l)$, where V_0 is an output variable indexed by $\mathbf{I}$ and produced by expression E operating on input variables $V_1, ..., V_l$, also indexed by $\mathbf{I}$.

2.2. Notation

Throughout this paper the following notation is used: N is the set of naturals, Z is the set of integers, n is the number of nested FOR-loops of the algorithm and m is the number of dependence vectors of the algorithm. $J^n \subset Z^n$ is the set of indices: $J^n = \{(I_1, ..., I_n) \mid I_i \in Z \wedge l_i \le I_i \le u_i, 1 \le i \le n\}$. Each point in this n-dimensional integer space is a distinct instantiation of the loop body. A *dependence vector* is denoted $\mathbf{d}_i=(d_{i1}, ..., d_{in})$, $1 \le i \le m$. The *dependence set* D of an algorithm A is the set of all dependence vectors of this algorithm: $D=\{\mathbf{d}_1, \mathbf{d}_2, ..., \mathbf{d}_m\}$. Notice that all dependence vectors are considered uniform, i.e. independent of the indices of computations.

3. Structural Properties of the Index Space

In this section we shall present the partial ordering introduced by the dependence vectors. Dependencies impose precedence constraints, and partition the index space into sets, chains and antichains through the establishment of a partial ordering among computations.

We now proceed to define the following sequences of sets and binary relations over J^n:

Definition 3.1:
(i) $S_0 = \{\mathbf{j} \in J^n \mid \forall \mathbf{d} \in D, \mathbf{j}-\mathbf{d} \notin J^n\}$ and $F_0 = \varnothing$
(ii) $S_{i+1} = \{\mathbf{j} \in J^n \mid \exists \mathbf{d} \in D, \exists \mathbf{i} \in S_i$ such that $\mathbf{i}+\mathbf{d}=\mathbf{j}\}$ and
$F_{i+1} = F_i \cup \{(\mathbf{i}, \mathbf{j}) \mid \mathbf{j} \in J^n$ and $\exists \mathbf{d} \in D\ \exists \mathbf{i} \in S_i$: $\mathbf{i}+\mathbf{d}=\mathbf{j}\}$ ■

Definition 3.2: The binary **dependence** relation $D^* \subseteq (J^n)^2$ is the reflexive transitive closure of the binary relation F_r :

$$D^* = \cup_0^{r-1} D_i, \quad \text{where}$$
$$D_0(\mathbf{j}_1, \mathbf{j}_2) \Leftrightarrow \mathbf{j}_1 = \mathbf{j}_2$$
$$D_1(\mathbf{j}_1, \mathbf{j}_2) \Leftrightarrow F_r(\mathbf{j}_1, \mathbf{j}_2)$$
$$D_{i+1}(\mathbf{j}_1, \mathbf{j}_2) \Leftrightarrow \exists \mathbf{j}_3\ (D_i(\mathbf{j}_1, \mathbf{j}_3) \wedge F_r(\mathbf{j}_3, \mathbf{j}_2))$$ ■

In the rest of the paper we will simply write $\mathbf{j}_1 <_D \mathbf{j}_2$ instead of the more awkward $D^*(\mathbf{j}_1, \mathbf{j}_2)$.

The J^n set equipped with the binary relation $<_D$ is a partially ordered set, since it can be easily proved that $<_D$ is reflexive, antisymmetric and transitive. Abbreviating, $(J^n, <_D)$ is a poset, and this term will be used next.

The main property of a poset is its partitioning in chains and antichains. These terms are defined as follows:

Definition 3.3: Consider the poset $(J^n, <_D)$:

- A subset $C=\{\mathbf{j}_1, ..., \mathbf{j}_r\}$ of J^n is called a chain iff every two elements of this set are related with $<_D$. Formally, there exists an order $\mathbf{j}_1 <_D .... <_D \mathbf{j}_i <_D .... <_D \mathbf{j}_r$
- A subset $A=\{\mathbf{j}_1, .., \mathbf{j}_r\}$ of J^n is called antichain iff no two distinct elements are related with $<_D$. ■

Definition 3.4: Given an algorithm $A(J^n, D)$, a base set B_{d_k} with respect to a specific $\mathbf{d_k}\in D$, is defined as $B_{d_k}=\{\mathbf{b_{d_k}}\in J^n \mid \forall \mathbf{i}\in J^n$ we have that $\mathbf{i}+\mathbf{d_k}\neq \mathbf{b_{d_k}}\}$. ■

In Appendix I we give a low complexity algorithm for finding the base set B_{d_k}.

Definition 3.5: A chain $C_d=\mathbf{j}_0, ..., \mathbf{j}_k$ of J^n is a *uniform* chain with respect to $\mathbf{d}\in D$ iff:

(1) $\mathbf{j}_{i+1}=\mathbf{j}_i+\mathbf{d}$, $0\leq i\leq k-1$,

(2) $\mathbf{j}_0-\mathbf{d}\notin J^n$ and $\mathbf{j}_k+\mathbf{d}\notin J^n$.

C_d is completely determined by its minimum element $\mathbf{j}_0$ because $\mathbf{j}_i=\mathbf{j}_0+i\mathbf{d}$. We shall call $\mathbf{j}_0$ the *basis* of C_d. Clearly $\mathbf{j}_0\in B_d$. ■

Recall that a linear transformation Π organizes a $\mathbf{j}$ computation to be executed at $t_j=\left\lfloor \frac{\Pi\mathbf{j}+t_0}{\mathrm{disp}\Pi} \right\rfloor$, $\mathbf{j}\in J^n$ where t_0 is the initial time instant. From now on, we will assume $t_0=0$ for simplicity reasons.

Given an n-dimensional nested loop, every single iteration is depicted as a unique point $\mathbf{j}(j_1,\ldots,j_n)$ of the index space. According to the algorithmic model defined in Section 2, the dependence vectors link different points of the index space. This dependence relation between the loop iterations, introduces a partial ordering inside the index space. This ordering creates chains of computations, which are closely related one to another by a dependence. Definition 3.3 gives this partial ordering relation between points of the index space. Since there exist different dependence vectors, there exist many alternative paths which link various nodes of the J^n. Consequently there exist many ways of dividing the index space into chains of computations.

Applying a schedule of computations into time is equivalent to dividing the index space into antichains, where each antichain contains computations, which should be scheduled at the same time instance. Once the partitioning into time antichains using a time schedule is done, we should group computations together using a grouping method. Grouping of computations is done keeping track of the following criteria: minimising the overall makespan (completion time of the last point) and minimizing the communication between computations, which belong to different processing elements. Our objective is to reduce the interprocessor communication while preserving the minimum makespan.

In this paper, we consider as time schedule the hyperplane method introduced by Lamport in [9]. Since the time antichains are uniquely defined by the hyperplane Π, the minimization of communication is done by organizing computations into chains which have the maximum possible intrachain activity. Our method for efficient communication schedules is based on the fact that reducing the overall communication is equivalent to maximizing the intraprocessor communication. Our space schedule is based on assigning chains of computations to the same processor. Thus, the dependence vector which incurs the partial ordering inside every chain is satisfied inside the processor. On the other hand we try to use compound chains which are formed by dependence vectors, which are linear combinations of other simpler dependence vectors.

4. THE CHAIN GROUPING METHOD

Given a finite index space J^n and the dependence set D, the first step is to find the hyperplane vector Π, which determines the optimal execution time. Once Π is found by applying one of the methods presented in [4], or in [14], we should partition the index space into disjoint groups of related computations. An efficient grouping method should reduce the amount of intergroup communication while preserving the execution ordering imposed by the time scheduling step. As far as the grouping problem is concerned, Sheu in [15] proposed the projection of the whole index space V^p onto a plane perpendicular to the projection vector Π. This method does not lead to efficient grouping, since it ignores the clustering of related computations inside the same group. Furthermore it needs the transversal of the whole index space, since every point of J^n is projected to the plane perpendicular to Π. Consider for example the part of the index space shown in figure 1. The dependence vector set is $D=\{(1, 2), (1, 0)\}$. The optimal hyperplane is $\Pi[1, 1]$. Consider two points $\mathbf{j_1}(0, 1)$ and $\mathbf{j_2}(1, 2)$. Projecting them on the perpendicular to Π plane, we have the same projection vector $\mathbf{j}_1^{\Pi}$. Obviously $\mathbf{j_1}$, $\mathbf{j_2}$ are grouped inside the same group, but there are no direct dependence relations between them. Of course $\mathbf{j_1}$, $\mathbf{j_2}$ are linked through the path $(0,1)>_D(0,2)>_D(1,2)$, but this path contains points which belong to different groups according to the mapping proposed in [15], thus imposing communication overhead. In example 4.1, we will apply both this method and our proposed Chain grouping and the comparative results will show how much the communication is further reduced with Chain grouping.

Chain grouping proposes a slight different procedure. Given an algorithm $A(J^n, D)$, the first step is to find the optimal time linear schedule given by the hyperplane method. Once Π is found, the partitioning phase of the index space into groups is starting. According to the criterion of reducing the overall communication cost, we should try to group related computations together. Recall, that from the structural properties of the index space, two computations $\mathbf{i}$, $\mathbf{j}$ are related if there exists a partial ordering $\mathbf{i}<_D\mathbf{j}$ or vice versa. Obviously, when this partial order relation is determined only by the same dependence vector d_k, then the chain of computations which is formed is called *uniform.* Chain scheduling tries to put into the same processor, uniform chains of computations formed by one selected dependence vector. This vector is called the projection vector $\mathbf{d_k}$ and it is selected from the dependence set by satisfying the following criterion: Grouping as many as possible similar uniform chains of computations together, while preserving the hyperplane time execution order. This means that all index points belonging to different chains which are mapped to the same processor, are executed at different time instances. In addition to this, the fact that as many as possible uniform chains along $\mathbf{d_k}$ are grouped together, means that we try to encapsulate into the same processor-group as many as possible dependence relations. Obviously, since every chain formed by $\mathbf{d_k}$ is inside the same processor, there are no communication overheads for this particular vector. Also, since the grouping factor r is the biggest possible, this means that most of references from one point $\mathbf{i}$ to another $\mathbf{j}$ through the $d \in D$-{$\mathbf{d_k}$} are kept inside the same processor, thus not contributing to the overall communication overheads. Theorem 4.1 proves that when the projection vector is selected as the one with the maximum grouping factor r, then the time execution order given by the optimal Π is preserved. In addition to this, since r is the maximum possible, there exist index points belonging to the same group which are linked through dependence vectors other than $\mathbf{d_k}$. The following definition gives the properties of the projection dependence vector $\mathbf{d_k}$:

Definition 4.1 (*Projection vector*): Given an algorithm $A(J^n, D)$ and the corresponding optimal hyperplane Π, the projection vector d_k is selected as

$$\mathbf{d_k}: \quad \left\lfloor \frac{\Pi \mathbf{d_k} / \gcd(d_{ki})}{\mathrm{disp}\Pi} \right\rfloor = \max \left\lfloor \frac{\Pi \mathbf{d_j} / \gcd(d_{ji})}{\mathrm{disp}\Pi} \right\rfloor, \quad d_j \in D,$$

$i=1..,n$ where gcd() is the greatest common divisor of the coordinates d_{ji} of $\mathbf{d_j}$. ■

In addition to the above definition, we call $r = \left\lfloor \frac{\Pi \mathbf{d_k} / \gcd(d_{ki})}{\mathrm{disp}\Pi} \right\rfloor$ the grouping factor with respect to d_k projection vector.

Once the projection vector $\mathbf{d_k}$ is selected from the set D, we should project all points of J^n to the plane perpendicular to $\mathbf{d_k}$. Since the index space J^n is n-dimensional, the projection of all points to the n-1 dimensional plane perpendicular to $\mathbf{d_k}$, gives n-1 dimensional projection points. The following definition determines the projection of a point $\mathbf{j}$ $(j_1,\ldots,j_n)$ with respect to the projection vector $\mathbf{d_k}$.

Definition 4.2 (*Projected Point*): Given a projection vector $\mathbf{d_k}(d_{k1},\ldots,d_{kn})$, we define the projected point $\mathbf{j^{d_k}}$ as the projection of the index point $\mathbf{j}(j_1,\ldots,j_n)$ onto a plane perpendicular to $\mathbf{d_k}$. ■

Mathematically, $\mathbf{j^{d_k}} = \mathbf{j} - \frac{\mathbf{j} \cdot \mathbf{d_k}}{\mathbf{d_k} \cdot \mathbf{d_k}} \mathbf{d_k}$, as shown in figure 2.

Consequently, the projection of all points of J^n produces the Projected Index Space:

Definition 4.3 (*Projected Index Space*): Given an index space V and a projection vector $\mathbf{d_k}$, we define the Projected Index Space V^{d_k} as the set V^{d_k}= { $\mathbf{j^{d_k}}$: $\mathbf{j^{d_k}} = \mathbf{j} - \frac{\mathbf{j} \cdot \mathbf{d_k}}{\mathbf{d_k} \cdot \mathbf{d_k}} \mathbf{d_k}$, $\mathbf{j} \in J^n$}. ■

All projected points of the index space are linked through the corresponding projected dependence vectors. If $\mathbf{j_1}$, $\mathbf{j_2}$ are two points of J^n, where $\mathbf{j_1} = \mathbf{j_2} + \mathbf{d}$, $\mathbf{d} \in D$-{$\mathbf{d_k}$}, then the corresponding projected points $\mathbf{j_1^{d_k}}$, $\mathbf{j_2^{d_k}}$ are linked through $\mathbf{d^{d_k}}: \mathbf{j_1^{d_k}} = \mathbf{j_2^{d_k}} + \mathbf{d^{d_k}}$ where $\mathbf{d^{d_k}}$ is the projection of the d on the plane perpendicular to $\mathbf{d_k}$.

Definition 4.4 (*Projected Dependence Vector*): Given a projection vector $\mathbf{d_k}$ and a dependence vector $\mathbf{d} \in D$-{$\mathbf{d_k}$}, the projected dependence vector $\mathbf{d^{d_k}}$ is defined as:

$$\mathbf{d^{d_k}} = \mathbf{d} - \frac{\mathbf{d} \cdot \mathbf{d_k}}{\mathbf{d_k} \cdot \mathbf{d_k}} \mathbf{d_k}. \qquad ■$$

The vector that is used to group the projected points of J^n into groups is called the *grouping* vector. The following formally defines the grouping vector $\mathbf{d_g}$:

Definition 4.5 (*Grouping vector*): Given an algorithm $A(J^n, D)$ and the corresponding optimal hyperplane Π, the grouping vector is selected as $\mathbf{d_g}$: $\Pi\mathbf{d_g}$= min{$\Pi\mathbf{d_i}$, i=1,...,m, $\mathbf{d_i} \in D$}

Once the projection vector $\mathbf{d_k}$ is selected, we find the set B_{d_k} (definition 3.5), the base set with respect to $\mathbf{d_k}$. The algorithm of Appendix I gives a straightforward way of finding B_{d_k}. We do not need to project all points of J^n to the plane perpendicular to $\mathbf{d_k}$, but only the points of B_{d_k} as lemma 4.3 proves. The following definition presents the properties of the projected Base Set with respect to $\mathbf{d_k}$:

Definition 4.6 (*Projected Base Set*): Given a base set B_{d_k} with respect to a specific $\mathbf{d_k} \in D$ and a projection vector $\mathbf{p}$, we define the projected base set $B_{d_k}^{d_k}$ as the set

$$B_{d_k}^{d_k} = \{\mathbf{j^p}: \mathbf{j^p} = \mathbf{b_{d_k}} - \frac{\mathbf{j} \cdot \mathbf{p}}{\mathbf{p} \cdot \mathbf{p}} \mathbf{p}, \mathbf{b_{d_k}} \in B_{d_k}\} \qquad ■$$

The projected base set $B_{d_k}^{d_k}$ contains all projected points of B_{d_k} onto the plane perpendicular to $\mathbf{d_k}$. Since every $\mathbf{b_{d_k}} \in B_{d_k}$ is the base element of every chain C_{d_k} formed by a $\mathbf{d_k}$ vector, the set $B_{d_k}^{d_k}$ contains the projected *basis* (see definition 3.6) of all possible C_{d_k}. Our method groups together subsequent chains C_{d_k}, whose basis are linked with the projected grouping vector $\mathbf{d_g^{d_k}}$. Once the projected base set $B_{d_k}^{d_k}$ is found, we cluster together r projected points $b_o^{d_k}$, ..., $b_{r-1}^{d_k}$ along the direction of $\mathbf{d_g^{d_k}}$ into a $B_{1,d_k}^{d_k}$ subset. Since a $B_{1,d_k}^{d_k}$ subset contains r *basis* points of r uniform C_{d_k} chains, every group G_l includes r C_{d_k} chains. Each group G_l is formed by the base subset $B_{1,d_k}^{d_k}$ following every chain of computations C_{d_k} along the grouping vector d_k. The partitioning in such G_i groups is the partitioning of the index space with the Chain Grouping method.

Each subset G_i is called a partitioned group of J^n with respect to $\mathbf{d_k}$. Obviously, depending on the selection of $\mathbf{d_k}$, we can split the index space J^n into various different G_i's.

The following gives the formal description of a Chain grouping:

Definition 4.7: Given an algorithm $A(J^n, D)$, we define as Chain Grouping $G(J^n, \Pi, d_k)$ the partitioning of the index set J^n into disjoint groups G_i such that:

- $\bigcup_{i=0}^{a} G_i = J^n$
- $G_i=\{\mathbf{j}\in J^n: \mathbf{j}=\mathbf{b}_{\mathbf{d_k}} + \lambda\mathbf{d_k},\ \mathbf{b}_{\mathbf{d_k}} \in B^{d_k}_{l,d_k} \wedge \lambda\in Q\}$

Where $B^{d_k}_{l,d_k}$ is a subset of the projected base set $B^{d_k}_{d_k}$ containing r basis grouped by d_g.

Finally, in order to evaluate the different proposed partitioning schemes of the index space J^n, in terms of total communication requirements, we count the total number of edges between computations belonging to different group pairs.

Algorithm (*Chain Grouping*)

Input: An algorithm A (J^n, D) and the corresponding time hyperplane $\Pi(a_1,\dots a_n)$
Output: A set of partitioned groups $G(J^n, \Pi, \mathbf{d_k}) = \{G_0,\dots,G_l\}$

Step 1: *Selection of* $\mathbf{d_k}$ *projection vector.* Choose $\mathbf{d_k}$:

$$\left\lfloor \frac{\Pi\mathbf{d_k} / \gcd(d_{ki})}{\mathrm{disp}\Pi} \right\rfloor = \max\left\lfloor \frac{\Pi\mathbf{d_j} / \gcd(d_{ji})}{\mathrm{disp}\Pi} \right\rfloor,\ \mathbf{d_j}\in D$$

Step2: *Selection of* $\boldsymbol{d_g}$ *grouping vector*

Choose $\mathbf{d_g}$: $\Pi\mathbf{d_g}= \min\{\Pi d_j\}$, $\mathbf{d_j}\in D-\{\mathbf{d_k}\}$

Step 3: *Finding the set* B_{d_k}

Apply the algorithm of Appendix I, to find the set B_{d_k} of base elements for all uniform chains C formed by $\mathbf{d_k}$ vector.

Step 4: *Finding the projected base set* $B^{d_k}_{d_k}$

Given the set B_{d_k} from Step 3, find the projected points $\{ b^{d_k}_{d_k} : b^{d_k}_{d_k} = \mathbf{b}_{\mathbf{d_k}} - \frac{\mathbf{b}_{\mathbf{d_k}} \cdot \mathbf{d_k}}{\mathbf{d_k} \cdot \mathbf{d_k}} \mathbf{d_k},\ \mathbf{b}_{\mathbf{d_k}} \in B_{d_k} \}$

Step 5: *Partitioning the projected base set* $B^{d_k}_{d_k}$ *into subsets* $B^{d_k}_{l,d_k}$ *using the grouping factor r*

Step 6: *Finding the uniform chains* $C_{b^{d_k}_i}$

Step 7: *Grouping together the uniform chains* $C_{b^{d_k}_i}$ *into the groups* $G^{d_k}_i$

Step 8: *Assigning each group to a different processor*

Lemma 4.1: Consider $V\equiv J^n$. The projected index space V^{d_k} is the same as $B^{d_k}_{d_k}$ the projection of the base set B_{d_k} using $\mathbf{d_k}$ as projection vector. (Proof: See [7]) ❒

Lemma 4.2: Let the projected points $b^{d_k}_0,\dots,b^{d_k}_{r-1}$ be grouped into the same projected base subset $B^{d_k}_{l,d_k}$ where $r=\left\lfloor \frac{\Pi\mathbf{d_k} / \gcd(d_{ki})}{\mathrm{disp}\Pi} \right\rfloor$ is the grouping factor, $r \in N^*_+$. Then all points $\mathbf{j}\in J^n$ projected to $b^{d_k}_j$ will be executed at different time. (*Proof:* See [7]) ❒

Lemma 4.3: If we group k projected points $b^{d_k}_o,\dots,b^{d_k}_k$ where k>r, then there exist two points inside the same group which are executed at the same time. *(Proof:* Similar to previous lemma, see [7]) ❒

Theorem 4.1: Given an algorithm $A(J^n,D)$, Chain grouping $G(J^n, \Pi, d_k)$ preserves the optimal time scheduling defined by the hyperplane Π. (*Proof:* Derived from Lemmata 4.1 and 4.2, see [7]) ❒

Example 4.1:
Consider the following algorithm $A(J^2,D)$

```
For j0 = 0 to 4 do
   For j1 = 0 to 4 do
        a(j0, j1) = a(j0-1, j1-2) + a(j0, j1-1) + a(j0-1, j1)
    end
end
```

Clearly the index space is $J^2=\{(j_0,j_1) \mid 0\le j_o,j_1\le 4\}$, and the set of dependence vectors is $D=\{d_1,d_2,d_3\}$ where $d_1=(0,1)$, $d_2=(1,2)$, $d_3=(1,0)$
Applying the hyperplane method on $A(J^2,D)$ we find that the optimal in terms of total execution time hyperplane is Π[1,1]. We will apply both Chain Grouping and the grouping presented in [15] to see the outperforming results of our proposed method.

a) Chain grouping:

The grouping factor r is $r=\max\left\lfloor \frac{\Pi\mathbf{d_j} / \gcd(d_{ji})}{\mathrm{disp}\Pi} \right\rfloor$, j=1,..,3 and it is $r=3$ for $\mathbf{d_2}$ vector which is chosen to be the projection vector. The grouping vector is one of the $\mathbf{d_1}$, $\mathbf{d_3}$. We arbitrarily choose $\mathbf{d_1}=(0,1)$.
Consider the point $\mathbf{j_1}(0,2)$. From definition we have the projected point $\mathbf{j_1^{d_2}}=(-4/5,2/5)$. In similar way, we find the projections of the other points belonging to B_{d_2}. The projection of the grouping vector $\mathbf{d_1}$ with respect to $\mathbf{d_2}$ is: $\mathbf{d_1^{d_2}}$ (-2/5,1/5).

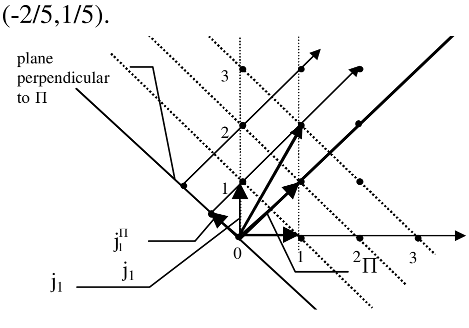

Fig. 1: Projection using Π as the projection vector

If we group each 3 points along the direction of $\mathbf{d_1^{d_2}}$ we have the following partitioning as figure 2 shows:

$B^{d_3}_{1,d_3}=\{(-2,1),\ (-8/5,4/5),\ (-6/5,3/5)\}\rightarrow\ B_{1,d_3}=\{0,3),\ (0,4),\ (0,5)\}$, $B^{d_3}_{2,d_3}=(-4/5,2/5),\ (-2/5,1/5),\ (0,0\rightarrow B_{2,d_3}=\{0,0),\ (0,1),\ (0,2)\}$, $B^{d_3}_{3,d_3}=\{(2/5,-1/5),\ (4/5,2/5),\ (6/5,-3/5)\}\rightarrow B_{3,d_3}=\{(1,1),\ (1,0),\ (2,1)\}$, $B^{d_3}_{4,d_3}=\{(8/5,-4/5),\ (2,-1),\ (12/5,-6/5)\}\ \rightarrow\ B_{4,d_3}=\{(2,0),\ (3,1),\ (3,0)\}$, $B^{d_3}_{5,d_3}=\{(14/5,-7/5),\ (16/5,-8/5)\}\rightarrow B_{5,d_3}=\{(4,0),\ (4,1)\}$

Clearly $B^{d_3}_{d_3}=\bigcup_{i=1}^{5}B^{d_3}_{i,d_3}=V^{d_3}$

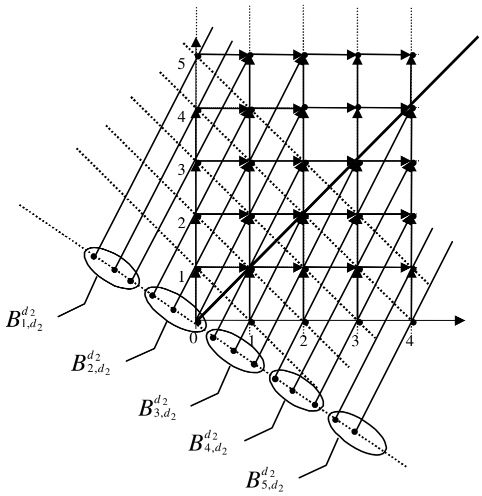

Fig 2: The partitioning of the projected base set $B^{d_k}_{d_k}$ along the projected grouping vector d_1

On the other hand, if we choose as projection vector the Π as Sheu in [15], the grouping factor *r* is *r*=max $\{r_i\}$, where r_i is the smallest positive integer such that $r_i d_i \in Z^n$, and in this example *r*=2

$B_{d_3}=\{(0,1),\ (0,2),\ (0,3),\ (0,4),\ (0,5),\ (1,0),\ (1,0),\ (2,0),\ (3,0),\ (4,0)\}$

$V^{d_3}=\ \{(-2,1),\ (-8/5,4/5),\ (-6/5,3/5),\ (-4/5,2/5),\ (-2/5,1/5),\ (0,0),\ (2/5,-1/5),\ (4/5,2/5),\ (6/5,-3/5),\ (8/5,-4/5),\ (2,-1),\ (12/5,-6/5),\ (14/5,-7/5),\ (16/5,-8/5)\}$

t_i	G_1	G_2	G_3	G_4	G_5
0		(0,0)			
1		(0,1)	(1,0)		
2		(0,2)	(1,1)	(2,0)	
3	(0,3)	(1,2)	(2,1)	(3,0)	
4	(0,4)	(1,3)	(2,2)	(3,1)	(4,0)
5	(0,5)	(1,4)	(2,3)	(3,2)	(4,1)
6	(1,5)	(2,4)	(3,3)	(4,2)	
7		(2,5)	(3,4)	(4,3)	
8			(3,5)	(4,4)	
9			(4,5)		

Table 1: Schedule produced by Chain Grouping

b) Sheu grouping:

The grouping factor r is r=max $\{r_i\}$, where r_i is the smallest positive integer such that $r_i d_i \in Z^n$, and in this example r=2.

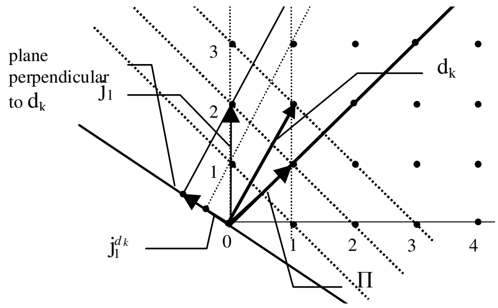

Fig. 3: Projection using Π as the projection vector

t_i	G_1	G_2	G_3	G_4	G_5
0			(0,0)		
1			(0,1)	(1,0)	
2		(0,2)	(1,1)	(2,0)	
3		(0,3)	(1,2)	(2,1)	(3,0)
4	(0,4)	(1,3)	(2,2)	(3,1)	(4,0)
5	(0,5)	(1,4)	(2,3)	(3,2)	(4,1)
6	(1,5)	(2,4)	(3,3)	(4,2)	
7		(2,5)	(3,4)	(4,3)	
8		(3,5)	(4,4)		
9			(4,5)		

Table 2: Schedule produced by Sheu Grouping

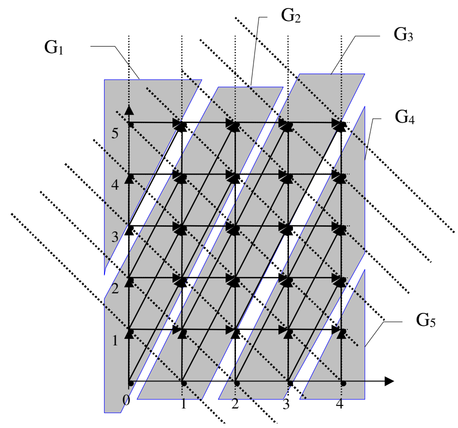

Fig. 4: Chain Grouping along d_3

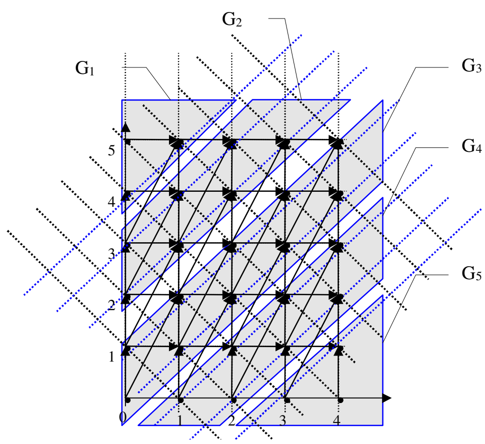

Fig. 5: Sheu Grouping along Π

	G_1	G_2	G_3	G_4	G_5
G_1		5	-	-	-
G_2	5		11	-	-
G_3	-	11		11	-
G_4		-	11		5
G_5	-	-	-	5	

Table 3: Communication cost for grouping of table 3.

	G_1	G_2	G_3	G_4	G_5
G_1		5	-	-	-
G_2	5		8	-	-
G_3	-	8		8	-
G_4		-	8		3
G_5	-	-	-	3	

Table 4: Communication cost for grouping of table 1

Both grouping methods preserve the time schedule given by $\Pi[1,1]$ (see fig. 4 and 5). If we compare the results of tables 3 and 4, we will see that Chain Grouping has a total of 24 links whereas Sheu [15] grouping has 32 interprocessor links. Thus there has been a 25% improvement when the grouping was performed along d_3 dependence vector.

5. CONCLUSION

In this paper we presented Chain grouping, a new method for partitioning the index space into groups of computations which have little intercommunication requirements. Grouping is performed by projecting the index space onto a plane perpendicular to the projection dependence vector. The projection vector was proved to be the best in terms of reducing the overall intergroup communication. Chain grouping reduces the overall communication costs and maximises processor utilization, while preserving the optimal time schedule introduced by the optimal hyperplane Π. It has also been shown that the suggested method outperforms similar methods presented in [6] or [15], in terms of total communication requirements, while having lower complexity.

APPENDIX

I. Algorithm for generating B_{d_k}

begin
for i=1 to n
    if $d_{ki}>0$ then boundary(i)=$d_{ki}+l_i-1$
        else if $d_{ki}<0$ then boundary(i)=$d_{ki}+u_i-1$
    generate $j=(j_1, ..., j_n)\in J^n$ such that
    if $d_{ki}>0$ then $l_i\le j_i\le$boundary(i)
        else if $d_{ki}<0$ then boundary(i)$\le j_i\le u_i$
            else $j_i=0$
end

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, no. 3-4, pp. 177-194, 1997.
2. T. Andronikos, N. Koziris, G. Papakonstantinou and P. Tsanakas, "Optimal Scheduling for UET-UCT Generalized n-Dimensional Grid Task Graphs," *Proceedings of the 11th IEEE/ACM International Parallel Processing Symposium (IPPS97), pp. 146-151, Geneva, Switzerland.*
3. A. Darte, Y. Robert, "Mapping Uniform Loop Nests onto Distributed Memory Architectures," *Parallel Computing,* vol. 20, pp 679-710, 1994.
4. 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.
5. E. H. Hollander, "Partitioning and Labeling Loops by Unimodular Transformations", *IEEE Trans. on Parallel Distributed Systems*, vol. 3, no. 4, pp. 465-476, Jul. 1992.
6. Chung-Ta King, W-H Chou and L. Ni, "Pipelined Data-Parallel Algorithms: Part II Design", *IEEE Trans. on Parallel Distributed Systems*, vol. 2, no. 4, pp. 430-439, Oct. 1991.
7. N. Koziris, G. Papakonstantinou and P. Tsanakas, "Mapping Nested Loops onto Distributed Memory Multiprocessors," *Technical Report CS-DSCLAB-TR-2-97, Digital Systems and Computers Laboratory*, NTUA, 1997.
8. N. Koziris, T. Andronikos, G. Economakos, G. Papakonstantinou, and P. Tsanakas, "Automatic Hardware Synthesis of Nested Loops using UET Grids and VHDL," *Proceedings of the International Conference on High Performance Computing and Networking (HPCN97), Lecture Notes in Computer Science, Springer-Verlag, pp. 888-897,* Vienna, Austria 1997.
9. L. Lamport, "The Parallel Execution of DO loops," *Commun. ACM*, vol.17, no.2, pp. 83-93, Feb. 1974.
10. 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.
11. 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.
12. K. Psarris, "On Exact Data Dependence Analysis," *Proceedings of the 1992 International Conference on Supercomputing*, ACM Press, pp. 303-312, Washington, 1992.
13. W. Shang and J.A.B. Fortes, "Independent Partitioning of Algorithms with Uniform Dependencies", *IEEE Trans. Comput*., vol. 41, no. 2, pp. 190-206, Feb. 1992.
14. 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.
15. J.-P. Sheu and T.-H. Tai, "Partitioning and Mapping Nested Loops on Multiprocessor Systems", *IEEE Trans. on Parallel Distributed Systems*, vol. 2, no. 4, pp. 430-439, Oct. 1991.