A SYSTOLIC APPROACH TO LOOP PARTITIONING AND MAPPING INTO FIXED SIZE DISTRIBUTED MEMORY ARCHITECTURES

IOANNIS DROSITIS, NECTARIOS KOZIRIS, NIKOLAOS PAPASPYROU AND PANAYOTIS TSANAKAS

National Technical University of Athens, Department of Electrical and Computer Engineering, Division of Computer Science
Zografou Campus, 157-73 Zografou, Athens, Greece
e-mail: {jdros, nkoziris}@cslab.ece.ntua.gr

This paper presents a new method for the problem of mapping of nested FOR-loops with uniform dependencies, into mesh-connected parallel architectures. This method is based on loop mapping for systolic arrays. The virtual array of cells is derived from the index space, by applying a linear transformation. This array is divided (cut) into a fixed number of clusters, equal to the number of available *real* processors. The basic idea of our method is that the cutting is performed along *properly* selected boundary directions, so as to minimize inter-cluster communication and equilibrate the number of virtual cells for every cluster. Each cluster is then assigned to a different processor, which performs in a roughly independent manner, as the communication requirements are now minimized. This mapping cuts down overall communication delays, while using a fixed number of processors from a (n-1)-dimensional mesh-connected distributed architecture.

Keywords: Loop partitioning, loop mapping, hyperplane method, virtual array of processors, space cuts, distributed architectures.

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. Most methods are based on finding a linear time transformation, since linear time scheduling differs only at a constant from the optimal schedule for "fat" domains, as Darte proved in [3]. Linear scheduling was introduced by Lamport in [8], with the *hyperplane method*[1], which organizes indiced computations into well-defined distinct groups called hyperplanes. Many methods were proposed in literature to find the optimal hyperplane; some of them are based on the solution of diophantine equations [8, 10] and others on the use of integer programming [13, 3] or even linear programming in subspaces [13]. When index spaces with uniform de-

[1] For further studying on this method, see [2] and [13].

pendence vectors are concerned, a polynomial complexity scheduling algorithm is presented in [7].

Once optimal parallel execution is found, an efficient method of mapping the concurrent groups of computations (hyperplanes) into the parallel architecture should be applied. A systematic methodology for mapping into fixed size systolic arrays was presented in [8]. Since the target architecture is synchronously operating, there is no need for communication efficient mapping and the main criterion for optimality is now the *total number of processors*. Other methods dealt with the same problem of mapping, while reducing not only the size but also the resulting dimension of the systolic array (see [4, 6, 11]).

Researchers are trying to minimize the inter-processor communication, by dividing the index space into as much as possible independent groups of computations. Shang and Fortes [14], and Peir [12], 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 is not always feasible. Sheu and Tai have presented a systematic method of partitioning the index space into groups and assign them into different processing elements [15]. Their method first projects the n-dimensional index space onto the zero hyperplane. The resulting projected space is divided into groups of computations, which preserve the initial optimal time schedule. However, their method does not reduce overall communications, since it ignores the maximization of intra-group references. A similar technique was presented by King in [5]. This method produces better results, since it groups together chains of related computations, but requires a greedy (exhaustive) search to define the group, where each computation belongs. Time scheduling is not explicitly defined. They only discuss computational structures on two dimensional index spaces. However, they introduced the idea of grouping related computations together. In addition to this, their target architecture has an unlimited number of processors, which is not realistic for most of actual large index spaces.

In this paper we propose a new method, based on mapping into an unbounded array of systolic cells. The initial index space J^n is linearly transformed into another J'^n, which is used for initial systolic mapping. Transformation matrix T is divided into the hyperplane transformation Π and the space transformation S, which actually maps the initial index space J^n to a (n-1)-dimensional projected virtual space. This space represents a virtual array of cells, which should be further divided to fit the size of the available hardware. Our method analytically derives a partitioning, which minimizes communication requirements and delays. This (n-1)-dimensional array is partitioned into blocks of neighboring virtual nodes, where each block is assigned to a different physical processor. We establish the notion of a *single cut* of the projected index space, along a *boundary* direction. All possible single cuts along these directions are evaluated, by a formula calculating the approximate total communication requirements between the separate parts of each cut.

Once the optimal cut (the one with the minimum communication requirements) has been selected, we perform the same procedure along the rest n-2 dimensions. The resulting set of cuts, called a *mapping*, divides the virtual space into clusters; all points of each cluster are assigned to the same physical processor. The inter-processor communication is reduced, as neighboring virtual processors are merged together into a physical one. In addition, by cutting the virtual space into equal size tiles (except boundary effects), overall computational load is balanced. Thus, the resulting space mapping is efficient, in terms of processor utilization and communication delays.

Basic terminology and notation used throughout this paper is given in *Section 2*. *Sections 3* presents the properties and algorithms used for partitioning; the virtual array of systolic cells and the proposed mapping is elaborated in *Section 4*. An example of our method is presented in *Section 5* and the obtained results are discussed in *Section 6*.

2 Preliminary Concepts and Definitions

We focus on algorithms, which have the form of a nested FOR-loop structure, with uniform data dependencies [11]. The algorithmic model is formally described as follows:

```
for i_0 = l_0 to u_0 do
        …
        for i_{n-1} = l_{n-1} to u_{n-1} do
            S_1
            …
            S_k
        end for
        …
end for
```

where i_i, $0 \le i \le n$-1, are the loop variables, l_i and u_i are integer-valued constants that represent the lower and upper limits for loop variables, and S_j, $1 \le j \le k$, are k assignment statements. For the nested loop given above, the index vector is defined as the vector of dimension n: $\mathbf{i} = [i_0, i_1, \ldots, i_{n-1}]^{\mathrm{T}}$. Furthermore, each statement S_j is an assignment of the form: $v_0 = \mathrm{E}(v_1, v_2, \ldots, v_p)$, where v_0 is an output variable, v_j, $1 \le j \le p$, are input variables and E is an arbitrary expression of the input variables. It should be noted that all variables in the loop statements, may be indexed by the index vector $\mathbf{i}$. Let J^n be the set of indices:

$$J^n = \{[i_0, i_1, ..., i_{n-1}]^{\mathrm{T}} : i_i \in \mathbf{Z} \wedge l_i \le i_i \le u_i, \forall i \text{ where } 0 \le i \le n-1\},$$

where $\mathbf{Z}$ is the set of integer numbers. J^n is an n-dimensional integer space. Each point in this n-dimensional integer space, is a distinct instantiation of the loop body. The instance of statement S_j is represented by $S_j(\mathbf{i})$. Notice that the points of J^n are

ordered *lexicographically*; the usual symbol `<` is used to denote this (linear) ordering.

Dependencies may exist between variables appearing in instances of assignment statements. A (*directed*) dependence between two instances $S_j(\mathbf{i}_1)$ and $S_j(\mathbf{i}_2)$, is characterized by the dependence vector $\mathbf{d} = \mathbf{i}_2 - \mathbf{i}_1$, as in [10]. The dependence matrix *D* contains as columns all existing dependence vectors. A dependence vector is denoted by $\mathbf{d}_i$, $1 \le i \le m$. Such an algorithm will be defined by its index space J^n and its dependence matrix *D* and will be represented by $A(J^n, D)$.

3 Partitioning the Index Space

In this section, we describe our method of partitioning and mapping an *n*-dimensional index space onto a mesh-connected MIMD architecture[2]. This method is based on *systolic loop mapping*.

When bounded number of cells or processors is given, a partitioning methodology should also be applied to fit the virtual array (resulting from mapping phase) into the real one. We follow GPLS method *(Globally Parallel Locally Sequential)*[3]. This means, that since the number of available cells (physical processors) is less than the number of virtual nodes, the virtual array is partitioned into blocks, whose number equals the size of the real (physical) array of processors. All virtual nodes inside the same block are sequentially executed -*by the same processor*.

The cut of the virtual array into the fixed number of blocks is done along directions, which reduces the overall communication requirements and divides the array into equal size partitions. Since everything in each block is executed by the same physical processor there are no communication requirements. The only communication overhead is imposed by inter-block message passing, which is unavoidable, since all blocks are executed in parallel (*Globally Parallel*). Of course, there are now more local memory requirements, but the size of memory in distributed architectures is large enough, and accesses to local memory for the same block are fast. Hereunder, the details of our approach are further analyzed.

3.1 Transformation of the Index Space J^n

We assume that a linear transformation matrix *T* has already been selected. *T* is given as two submatrices, *Π* and *S*, as in [10]:

$$T = \left[\frac{\Pi}{S}\right].$$

[2] In the following lines we will use the notation of [10].
[3] For the inverse approach LPGS see [4].

The first row of the above matrix T is the hyperplane vector Π, which determines the *time execution ordering* for the transformed index space J'^n: $J'^n = T \cdot J^n$. The submatrix S, is the space transformation, which maps onto a (n-1)-dimensional array of systolic cells $S \cdot J^n$, as in [10]. For the needs of our method, we consider that T was *optimally* selected. This means that: Π gives the optimal hyperplane with respect to the least makespan and S produces the optimal space mapping, in terms of the total number of cells and communication links, as shown in [7].

By applying the transformation matrix T to the original index space J, the latter has been transformed to a new index space J'. By ignoring the first dimension of the transformed index space J', we obtain a projected (n-1)-dimensional index space, which will be partitioned and eventually mapped to a (n-1)-dimensional mesh of processing cells, as in [10].

3.2 Cuts and Terminology

In the rest of the paper, the prefix `h-` will be used extensively. It is read as "hyper" and stands for n-dimensional. The terminology after the prefix is taken from the 3-dimensional space. That is:

- ***h-space**:* the n-dimensional space,
- ***h-plane**:* a plane in the h-space, that is a linear subspace of dimension n-1,
- ***h-line**:* a line in the h-space, that is a linear subspace of dimension n-2,
- ***h-mesh**:* a mesh in the (n-1)-dimensional space, that is an array of cells connected in a mesh topology.

With all this in mind, the original and the transformed index spaces J^n and J'^n, are h-spaces. The projected transformed index space $S \cdot J^n$ (the virtual array of systolic cells) is an h-plane, and its bound forms an h-polygon. The h-sides of the h-polygon define a number of h-lines, which we call *binding h-lines*:

- ***binding h-line**:* an h-line satisfying the following two properties:
  a) at least n-1 transformed index points lie on it (line definition);
  b) all remaining transformed index points are located in the same h-semi-plane.

Some more terminology follows:

- ***cut**:* an h-line that is parallel to a pair of binding h-lines; a cut separates the projected transformed index space into two h-semi-planes
- ***multiple cut**:* a non-empty set of cuts, whose h-lines are parallel and equidistant; this implies the balanced partitioning of computational load
- ***mapping**:* a set of multiple cuts, in which there are no two different multiple cuts with parallel h-lines.

4 The Mapping Method

In the course of our method, one must keep in mind the following issues:[4]

- *matrix T has been previously selected so as, to best meet the needs of our problem,*
- *transformed index space is an h-polygon with edges that are parallel in pairs,*
- *real processor space, onto which our transformed space is to be mapped, is an h-mesh and that*
- *optimal cut will be given in terms of continuous space measures (not discrete).*

4.1 Communication Cost Function for Alternative Cuts

The *cost* value of each mapping is computed as the sum of the cost values of its individual cuts. The cost of a cut is defined to be the number of transformed dependence vectors that traverse the cut's h-line. The formula that computes the cost value of a single cut is thus the following (*cost function 1*):

$$c = \sum_{i=1}^{m} \frac{\left|\mathbf{p} \cdot \mathbf{d}'_i\right|}{\left\|\mathbf{p}\right\| \cdot \left\|\mathbf{d}'_i\right\|} \cdot l \cdot \rho \,, \tag{2}$$

where: m is the number of distinct dependence vectors, $\mathbf{p}$ is the vector that is perpendicular to the cut, $\mathbf{d}'_i$ is a single transformed dependence vector, $\|\mathbf{u}\|$ is the Euclidean norm of vector $\mathbf{u}$, l is the h-length of the segment of the h-line that corresponds to the cut and is *within the bounds* of the transformed h-space and ρ is the density of vectors $\mathbf{d}'_i$ along an h-line perpendicular to $\mathbf{d}'_i$ vectors.

The fraction, within the sum of equation 2, is equal to the cosine of the angle between the two vectors $\mathbf{p}$ and $\mathbf{d}'_i$ and expresses the implicit notion that cuts that are as parallel as possible to the transformed dependence vectors, are traversed by fewer such vectors, and thus give better (lower) cost values. The coefficient $\left|\mathbf{p} \cdot \mathbf{d}'_i\right|$ is there to denote the *depth of the calculation* for each dependence vector in the direction of $\mathbf{p}$.

Theorem Parameter ρ is equal to $\left\|\mathbf{d}'_i\right\|$, or: $\rho = \left\|\mathbf{d}'_i\right\|$.

Proof: (*A simple geometric proof for the case that n = 3, is given. A similar proof can be easily derived, for greater dimension.*)

Consider the dependence vector $\mathbf{d}'_i = [a, b]^T$ shown in *Figure 1*, as the length segment AC.

---

[4] As well as the assumptions found in [3].

Assume also that ε is the line perpendicular to vector $\mathbf{d}'_i$. The density ρ of dependence vectors $\mathbf{d}'_i$ along the line ε, equals to the number w of $\mathbf{d}'_i$ vectors that traverse the CE segment, divided by the length δ of CE, i.e.: $\rho = \frac{w}{\delta}$.

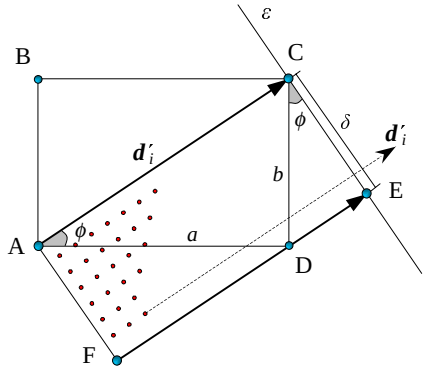

Figure 1. Computing parameter ρ for the 2-dimensional case (n = 3).

In order to compute the number w, of the dependence vectors that traverse CE, one should first notice that these vectors must have their starting point within the parallelogram ACEF. The area of ACEF is equal to the area of ABCD and thus is $a \cdot b$. Since the starting point of each dependence vector has integer coordinates and there are no holes in the projected transformed index space, we conclude that the number of vectors with their starting point within ACEF, is equal to the area of ACEF. Thus $w = a \cdot b$. Also the length of the segment CE is computed by the formula:

$$\delta = b \cdot \cos(\phi) = b \cdot \frac{\mathrm{AD}}{\mathrm{AC}} = b \cdot \frac{a}{\left\| \mathbf{d}'_i \right\|} = \frac{ab}{\left\| \mathbf{d}'_i \right\|}$$

thus, density ρ equals to:

$$\rho = \frac{w}{\delta} = \frac{ab}{\frac{ab}{\left\| \mathbf{d}'_i \right\|}} = \left\| \mathbf{d}'_i \right\|. \qquad (3)$$

Cost function 1 is therefore simplified to (*cost function 2*):

$$c = l \cdot \sum_{i=1}^{m} \frac{\left| \mathbf{p} \cdot \mathbf{d}'_i \right|}{\left\| \mathbf{p} \right\|}. \qquad (5)$$

The cost function 2 that is defined above, gives a heuristic measure, of how good a mapping is. Moreover, when the projected transformed index space is an h-

parallelogram, applying the cost function to all different mappings, leads to the optimal solution with optimal processor utilization.

4.2 Determining Possible Cut Directions

Algorithm 1. *Calculating the binding h-lines*
Step 1.1: Define matrix V of dimension nx2^n, containing as columns all the permutations (2^n) of the coordinates of the index space boundary points:

$$V = \begin{bmatrix} l_0 & l_0 & \cdots & u_0 & u_0 \\ l_1 & l_1 & \cdots & u_1 & u_1 \\ \vdots & \vdots & & \vdots & \vdots \\ l_{n-2} & l_{n-2} & \cdots & u_{n-2} & u_{n-2} \\ l_{n-1} & u_{n-1} & \cdots & l_{n-1} & u_{n-1} \end{bmatrix}.$$

Step 1.2: Calculate the transformation of V: $V' = T \cdot V$.
Step 1.3: Ignore first row of V, which represents time coordinate, and construct matrix W of dimension (n-1)x2^n, containing the other rows. W represents boundary point coordinates in the projected transformed index space.
Step 1.4: Calculate the convex hull of all points contained in W. The result is an h-polygon. The h-sides of the convex hull are the binding h-lines.
Step 1.5: Since the h-sides of the convex hull come in pairs of parallel h-lines[5], they can be represented in pairs by inequalities of the form:

$$\beta \le \alpha_1 x'_1 + \alpha_2 x'_2 + \cdots + \alpha_{n-1} x'_{n-1} + \alpha_n \le \gamma .$$

Let the number of such pairs be b.

4.3 Mapping Algorithm

In order to find all possible mappings, we compute all different ways in which the projected transformed h-space can be allocated to the processor space. We assume that the processor space is denoted by a vector $\boldsymbol{\pi}$, of n-1 elements. Each element shows how many processing cells are available on each direction of the h-mesh, where the transformed h-space is to be mapped[6]. This is equal to the number of cuts that should be made along this direction plus one.

Our mapping algorithm is divided in two phases: *pre-calculation*, where constant coefficients are calculated once and for all, and *calculation*, where the cost of

[5] This can be easily proved, since the original index space is an h-cube and the convex hull is the projection of its linear transformation.
[6] If the actual h-mesh has smaller dimension than n-1, some of its elements will have to be taken equal to 1.

all possible mappings is calculated. In the following, ξ will denote the number of different non-trivial values in vector $\boldsymbol{\pi}$ and ψ_j will denote these values, for $1 \le j \le \xi$.

Algorithm 2. *Pre-calculation of cost for multiple cuts*

Step 2.1: **For all** pairs of binding h-lines:

$$\beta_i \le \alpha_{i,1}x'_1 + \alpha_{i,2}x'_2 + \cdots + \alpha_{i,n-1}x'_{n-1} + \alpha_{i,n} \le \gamma_i,\ 1 \le i \le b\ .$$

Step 2.1.1: **Let** $\mathbf{p}_i$ be the perpendicular vector: $\mathbf{p}_i = [\alpha_{i,1}, \alpha_{i,2}, \cdots, \alpha_{i,n-1}]^{\mathrm{T}}$.

Step 2.1.2: **Calculate** the constant coefficient for cuts in the direction of $\mathbf{p}_i$:

$$depCost_i = \frac{1}{\|\mathbf{p}_i\|}\left(\sum_{j=1}^{m} \left|\mathbf{p}_i \cdot \mathbf{d}'_j\right|\right).$$

Step 2.1.3: **For all** different numbers ψ_j in the vector of processors, $1 \le j \le \xi$:

Step 2.1.3.1: **Calculate** the constant coefficient for multiple cuts in the direction of every $\mathbf{p}_i$, cutting the projected transformed index space in ψ_j segments:

$$mcCost_{i,j} = depCost_i \sum_{k=1}^{\psi_j - 1} cutArea\left(\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_b, \frac{\gamma_i - \beta_i}{\psi_j}\right),$$

where *cutArea()* is a function that returns the h-length of the cut's h-line, according to *Algorithm 4*.

Algorithm 3. *Calculation of the minimal mapping*

Step 3.1: **For all** valid mappings: $\mathbf{m} = [m_1, m_2, \cdots, m_b]^{\mathrm{T}}$:

Step 3.1.1: cost := 0.

Step 3.1.2: **For all** pairs j of binding h-lines, $1 \le j \le b$:

Step 3.1.2.1: **If** $m_i > 1$ **then** cost := cost + $mcCost_{i,j}$, $1 \le j \le \xi$, such that $\psi_j = m_i$.

Step 3.1.3: Keep track of the *lowest cost*.

Algorithm 4. *Calculation of the h-length for a cut*

Parameters: - All $\mathbf{p}_i$ vectors, perpendicular to binding h-lines and

$\mathbf{p} = [p_1, p_2, \cdots, p_{n-1}]^{\mathrm{T}}$ the one that is perpendicular to the cut.

- The constant coordinate for the cut: p_n.

Therefore, the equation of the cut's h-line is: $p_1x'_1 + p_2x'_2 + \cdots + p_{n-1}x'_{n-1} = p_n$.

Step 4.1: **Let** P the set of points that define the cut's h-line segment, initially empty: $P = \varnothing$.

Step 4.2: **For all** combinations of (n-2) binding h-lines not perpendicular to $\mathbf{p}$ (for all $\mathbf{p}_i \ne \mathbf{p}$):

Step 4.2.1: Add the given h-line.

Step 4.2.2: Solve the linear system of dimension (n-1)x(n-1) to compute the point of intersection of the n-1 h-lines.

Step 4.2.3: **If** the solution satisfies all the remaining binding h-lines, **then** add it to P, **else** discard it.

Step 4.3: **Calculate** the h-length of the h-line segment defined by the points in P.

Since all points in P lie on the given h-line, they define an h-line segment[7]. H-line segments are really finite subspaces of dimension n-2, whose h-length must be calculated. The calculation of the h-length of an h-line segment that is defined by a set of k points, where $k \geq n$-1, can be reduced to the same calculation, but for a set of n-1 points. The problem does not apply for the case of $n = 3$, but it is easy to see that in the case of $n = 4$ the h-line segment is a polygon and we have to triangulate it, in order to calculate its area. Thus, it suffices to define the h-length of an h-line segment that is defined by n-1 points, and this can be done inductively:

Algorithm 5. Inductive definition of h-length

Step 5.1: **Base case**, for $n = 3$, use Euclidean distance.

Step 5.2: **Inductive case**, for $n > 3$ do the following:

Step 5.3: Exclude one point arbitrarily: $\mathbf{u}$.

Step 5.4: Use same algorithm to calculate the h-length l' of the h-line segment that is defined by the remaining n-2 points (in an h-space of dimension n-3).

Step 5.5: Find the projection $\mathbf{u}'$ of $\mathbf{u}$ on the h-plane defined by the remaining n-2 points.

Step 5.6: Calculate the Euclidean distance d between $\mathbf{u}$ and $\mathbf{u}'$.

Step 5.7: The result is the product of l and d.

5 EXAMPLE

Consider the following FOR-loop:

for $i_0 = 1$ to 6 do
  for $i_1 = 1$ to 4 do
    for $i_2 = 1$ to 3 do
      $\alpha(i_0, i_1, i_2) = \alpha(i_0, i_1 - 1, i_2) + \alpha(i_0 - 1, i_1, i_2) + \alpha(i_0, i_1, i_2 - 1)$

In this loop, every index point $\alpha(i_0, i_1, i_2)$ is computed relatively to the values of three other points of the index space. The dependence matrix is thus, as long as the transformation assumed:

[7] In the case that $n = 3$, we are certain that only two points will be in P, since all other points will have been discarded. However, for $n > 3$, the number of points in P can be larger.

$$\mathrm{D} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},\ \mathrm{T} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}.$$

The transformation matrix *T* has been selected, so as to best demonstrate the proposed method. At the end of the example, an optimal transformation will be presented. The transformed index vector $\mathbf{i}'$ and data dependence matrix D′ are:

$$\mathbf{i}' = \begin{bmatrix} i_0' \\ i_1' \\ i_2' \end{bmatrix} = \mathrm{T}\cdot\mathbf{i} = \begin{bmatrix} i_0 + i_1 + i_2 \\ i_0 + i_1 \\ i_1 + i_2 \end{bmatrix} \text{ and } \mathrm{D}' = \mathrm{T}\cdot\mathrm{D} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

The first line of D′ denotes *time dependence* of transformed algorithm and is ignored, while the remaining matrix has as columns the *space dependencies* in the transformed index space. In our example, the transformed dependencies are:

$$\mathbf{d}_0' = [1,1]^{\mathrm{T}},\ \mathbf{d}_1' = [1,0]^{\mathrm{T}},\ \mathbf{d}_2' = [0,1]^{\mathrm{T}}.$$

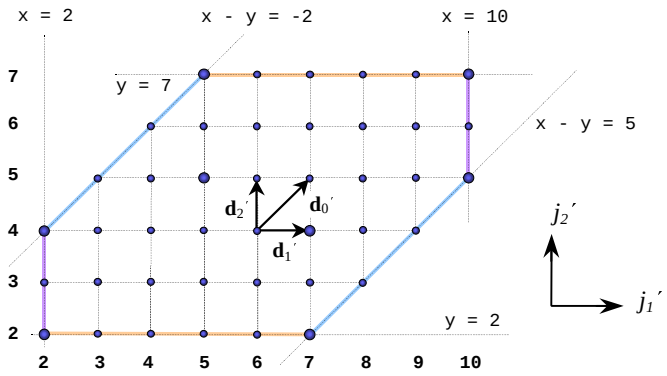

Figure 2. The projected transformed index space.

In *Figure 2* we can see the projected transformed index space, together with the transformed dependencies. In order to calculate the bounds of the projected transformed index space, we apply *Algorithm 1*. The matrices that are calculated in the first three steps of the algorithm are, respectively:

$$\mathrm{V} = \begin{bmatrix} 1 & 1 & 1 & 1 & 6 & 6 & 6 & 6 \\ 1 & 1 & 4 & 4 & 1 & 1 & 4 & 4 \\ 1 & 3 & 1 & 3 & 1 & 3 & 1 & 3 \end{bmatrix},$$

$$\mathrm{V}' = \mathrm{T}\cdot\mathrm{V} = \begin{bmatrix} 3 & 5 & 6 & 8 & 8 & 10 & 11 & 13 \\ 2 & 2 & 5 & 5 & 7 & 7 & 10 & 10 \\ 2 & 4 & 5 & 7 & 2 & 4 & 5 & 7 \end{bmatrix} \Rightarrow$$

$$W = \begin{bmatrix} 2 & 2 & 5 & 5 & 7 & 7 & 10 & 10 \\ 2 & 4 & 5 & 7 & 2 & 4 & 5 & 7 \end{bmatrix}.$$

The convex hull of the points, whose coordinates are given in matrix W, is a polygon whose vertices are the points: (2, 2), (7, 2), (10, 5), (10, 7), (5, 7) and (2, 4). The inequalities that describe the interior of this polygon, in the form of pairs of parallel lines, are the following:

pair 1: $2 \le x \le 10$, *pair 2*: $2 \le y \le 7$, *pair 3*: $-2 \le z \le 5$.

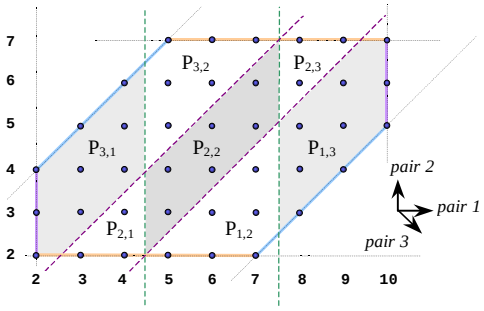

Figure 3. The mapping that was suggested for our example.

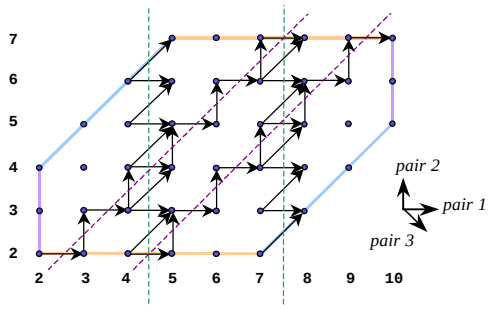

Figure 4. The real communication cost for the suggested mapping.

For our example, we select $\boldsymbol{\pi}$ to be equal to $[3, 3]^T$, which means that the final 2-dimensional mesh will have 3 processors in the first direction and 3 processors in the other, giving a total processor space of 9 cells. Next, we apply *Algorithm 2*, in order to pre-calculate the constant coefficients for the mapping costs. The results of the algorithm are the following:

$$depCost_1 = \frac{1}{1} \cdot (|1| + |1| + |0|) = 2\,,\ depCost_2 = \frac{1}{1} \cdot (|1| + |0| + |1|) = 2\,,$$

$$depCost_3 = \frac{1}{\sqrt{2}} \cdot (|0| + |1| + |-1|) = 1.4142\,,$$

$$mcCost_{1,1} = 2 \cdot (4.6667 + 4.6667) = 18.6668\,,$$

$$mcCost_{2,1} = 2 \cdot (6.6667 + 6.6667) = 26.6668\,,$$

$$mcCost_{3,1} = 1.4142 \cdot (7.07108 + 7.07108) = 20\,,$$

where the length of the cuts have been calculated by using *Algorithms 4* and *5*.

It is not difficult to see that the best mapping indicated by *Algorithm 3* consists of two multiple cuts, one in the direction of *pair 1* and one in the direction of *pair 3*. This mapping is shown in *Figure 3*. Its total cost is:

$$cost = mcCost_{1,1} + mcCost_{3,1} = 38.6668\,.$$

This does not significantly deviate from the real cost, as can be seen in *Figure 4*. We can see that the real cost of the multiple-cut in the direction of *pair 1*, i.e. the

number of dependence vectors traversing the vertical lines, is equal to 20, whereas the estimated cost was 18.6668.[8] Similarly, the real cost of the multiple cut in the direction of *pair 3*, is equal to 22, instead of the estimated 20.

We should note here, that if the projected transformed index space is not an h-parallelogram, there is inevitably a loss of processing cells. This is the case in our 2-dimensional example here, where the underused processors can be seen in *Figure 3*.

If we apply the method proposed in [9], for finding an optimal transformation when mapping to systolic cells, the following results will be produced, among many others, for our example:

$$T_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}, T_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}, T_3 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}, T_4 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

These matrices result in systolic arrays of 42, 24, 12 and 12 cells respectively. They all preserve the optimal hyper-plane $[1, 1, 1]^T$. The optimal transformation for the needs of our problem is T_4. This matrix maps into the least possible number of cells, as shown in [1], and is better than T_3, because it produces an array requiring only two external communication links (two external transformed dependence vectors). In the beginning of the example, we have selected T_1 for presentation, because the resulting structure is big enough to demonstrate the merits of our method.

6 Conclusion

In this paper we have presented a new method for the partitioning and mapping of nested loops onto fixed size, distributed, mesh-connected architectures. This method is based on transforming the initial *n*-dimensional index space J^n into an equivalent J'^n, using a transformation (matrix) *T*, and then divide the projected virtual (*n*-1)-dimensional space, through *S*, into blocks which are assigned to different processors. Interprocessor communication is considerably reduced, by choosing the optimal cut along each dimension. As it was shown, the proposed method is formally presented and easily programmable. Future work includes affine by statement partitioning of the index space, to further reduce the redundant interprocessor communication links.

7 References

1. T. Andronikos, N. Koziris, G. Papakonstantinou, P. Tsanakas, "Optimal Scheduling for UET/UET-UCT Generalized N-Dimensional Grid Task Graphs,"

[8] The deviation is due to the fact that continuous space properties are used. Deviations are quite large in our example, because of its small size. They become insignificant when the size of the index space is larger.

Journal of Parallel and Distributed Computing, vol. 57, no. 2, pp. 140-165, May 1999.
2. A. Darte and Y. Robert, "Constructive Methods for Scheduling Uniform Loop Nests", *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, pp. 814-822, August 1994.
3. A. Darte, L. Khachiyan, and Y. Robert, "Linear Scheduling is Nearly Optimal", *Parallel Processing Letters*, vol. 1.2, pp. 73-81, 1991.
4. A. Darte and Y. Robert, "Mapping Uniform Loop Nests onto Distributed Memory Architectures", *Parallel Computing*, vol. 20, pp. 679-710, 1994.
5. C.-T. King, W.-H. Chou, and L. Ni, "Pipelined Data-Parallel Algorithms: Part II – Design", *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, pp. 486-499, October 1990.
6. N. Koziris, G. Papakonstantinou and P. Tsanakas, "Mapping Nested Loops onto Distributed Memory Multiprocessors" in *Proceedings of the 1997 IEEE International Conference on Parallel and Distributed Systems (ICPADS97)*, IEEE Press, pp. 35-41, Seoul, Korea, Dec. 1997.
7. N. Koziris, G. Papakonstantinou, P. Tsanakas, "Automatic Mapping and Partitioning into Systolic Architectures", *Proceedings of the 5th Panhelenic Conference on Informatics*, pp. 777-790, Dec. 1995, Athens.
8. L. Lamport, "The Parallel Execution of DO Loops", *Communications of the ACM*, vol. 17, pp. 83-93, February 1974.
9. P.-Z. Lee and Z. M. Kedem, "Mapping Nested Loop Algorithms into Multidimensional Systolic Arrays", *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, pp. 64-76, January 1990.
10. D. I. Moldovan and J. A. B. Fortes, "Partitioning and Mapping Algorithms Into Fixed Size Systolic Arrays", *IEEE Transactions on Computers*, vol. C-35, pp. 1-11, January 1986.
11. D. I. Moldovan, "Parallel Processing: From Applications to Systems", *Morgan Kaufmann Publishers*, 1993.
12. J. K. Peir and R. Cytron, "Minimum Distance: A Method for Partitioning Recurrences for Multiprocessors", *IEEE Transactions on Computers*, vol. 38, pp. 1203-1211, August 1989.
13. W. Shang and J. A. B. Fortes, "Time Optimal Linear Schedules for Algorithms with Uniform Dependencies", *IEEE Transactions on Computers*, vol. 40, pp. 723-742, June 1991.
14. W. Shang and J. A. B. Fortes, "Independent Partitioning of Algorithms with Uniform Dependencies", *IEEE Transactions on Computers*, vol. 41, pp. 190-206, February 1992.
15. J.-P. Sheu and T.-H. Tai, "Partitioning and Mapping Nested Loops on Multiprocessor Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, pp. 430-439, October 1991.