Geometric Pattern Prediction and Scheduling of Uniform Dependence Loops*

Ioannis Drositis, Theodore Andronikos, Aneta Alexandridi,
George Papakonstantinou and Nectarios Koziris

Computing Systems Laboratory,
Computer Science Division,
Department of Electrical and Computer Engineering,
National Technical University of Athens,
Zographou Campus, 157-73, Athens, Greece

e-mail: `jdros@cslab.ntua.gr`

Abstract

One of the primary tasks in the area of boundary value problems, including uniform dependence loops, is predicting the execution propagation, as well as finding an optimal time schedule. The index space of a uniform dependence loop, due to the existence of dependence vectors, is split into subspaces of points that can be executed at a certain time instance. The geometric representation of these sets form certain polygonal shapes called patterns, with special attributes and characteristics. In this work, we present a new geometric methodology for predicting the execution propagation of the pattern boundaries. A simple low-cost scheduling algorithm is based on the presented (pattern) analysis.

***Keywords**: Execution pattern prediction, wavefront, uniform dependence loop, nested loop scheduling.*

1 Introduction

Predicting the execution wavefront in the class of *uniform loop nests* is quite promising, as it can lead up to new scheduling prospectives.

Many methods were proposed for time scheduling, originating from the *hyperplane* method, presented by Lamport [10]. The main notion behind such scheduling is the introduction of a linear tranformation, which organizes indiced computations into well defined distinct groups, called *hyperplanes*. As the hyperplane method was proved to be nearly optimal from Darte [3], many researchers have focused on it. As a result, it was extended by several researchers such as Shang [12] and Darte [4], by allowing more computations to be executed in parallel.

On the other hand, general scheduling approaches were never used in loop parallelization. General multiprocessor scheduling with precedence constrains was proved to be NP-complete [8], even when the number of available processors is unbounded [11]. However, in our case, things are much more simple, as the index space is characterized by regularity, due to the presence of uniform dependence vectors.

Our paper presents a new geometric methodology of predicting the execution propagation of a uniform nested loop index space. Due to the existence of dependence vectors, only a certain set of points can be executed at every moment. The geometric border of this set, forms a polygonal shape called *pattern*. We shall describe how these patterns are formed, the rules that they follow and whether they can be computed non-exhaustively; their special attributes and characteristics will be studied; whether we can predict their shape and at what complexity cost.

Under a different perspective, the index space can be partitioned into a sequence $T_1, T_2, \ldots$ of (disjoint) subsets, such that T_n contains precisely those points whose earliest computation time is n. By definition, the (pre-computed) boundary points form T_0. We prove that the aforementioned wavefront-like polygonal patterns are precisely the geometric representation of the above subsets (see Figure 1). Our goal is to predict the pattern outline of the k-th time step. The index space will be partitioned into patterns and then each point can be scheduled according to the pattern it belong to. For example, if $\mathbf{j} = (j_x, j_y)$ is in pattern k, then its earliest computation time is k (see Figure 1).

Conditions for the exact determination of the pattern at the computation instance k are examined; hence, optimal scheduling can be performed. A nearly optimal scheduling algorithm, presented in detail in [6], is based on the idea of patterns.

*This work was partially supported by the General Secretariat of Research and Development.

Basic terminology and definitions are presented in Section 2, while principal study follows in Sections 3 and 4. Conclusion and future work resides in Section 5.

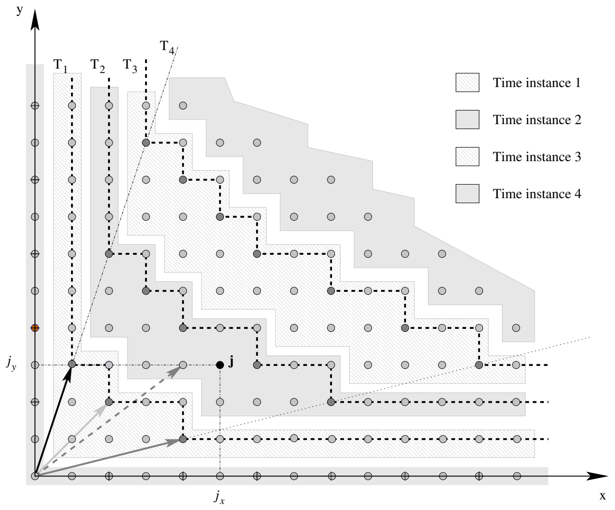

Figure 1. *Consider an algorithm with three dependence vectors* $\mathbf{d}_1 = (1,3)$*,* $\mathbf{d}_2 = (2,2)$*,* $\mathbf{d}_3 = (4,1)$ *and* $\mathbf{d}_4 = (4,3)$*. Each shaded polygon depicts the subset of* J *that can be executed at a certain moment. This is what we call pattern; the first four patterns are shown (for moments 1-4). The dashed lines mark the border of each pattern, called pattern outline. On the axes lie the pre-computed at moment* $t = 0$ *(boundary) points. Point* $\mathbf{j} = (j_x, j_y)$ *is included in* Pat_2*. Note that* $\mathbf{d}_4$ *doesn't affect pattern border, as it is not a pattern vector.*

2 Terminology and Definitions

Uniform dependence loops, in the general n-dimesional case, have the following form:

```
for (i1=0; i1 <= u1; i1++)
     ...
     for (in=0; in <= un; in++) {
          S1(I);
          S2(I);
          ...
     }
     ...
}
```

where $S_1, \ldots, S_k$ are assignment statements and $J = \{(i_1, \ldots, i_n) \mid 0 \leq i_r \leq u_r, 1 \leq r \leq n\}$ is the index space, where $J \subset \mathbf{N}^n$. We follow the model that was introduced by Lamport in [10]; the dependence vectors are "uniform", that is constant allover the index space.

This work is focused on 2-D nested loops, where $J \subset \mathbf{N}^2$. We intend to generalize our results on higher dimensions in a subsequent paper.

Due to the existence of dependence vectors, two instances of any of the above statements S_i cannot be executed independently. So, the ideal parallel execution which partitions the index space into independent point sets and executes all sets in parallel (in different processors), cannot be applied in the general case.

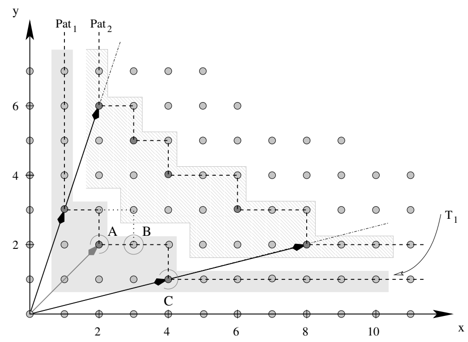

Figure 2. *First two patterns of an example algorithm. Subset* T_1 *of* J *that can be executed at moment* $t = 1$ *is shown.* T_2 *is the set of points that can be executed at moment* $t = 2$*; one can note that* Pat_1 *and* T_1 *are identical.*

Our scheduling policy executes every index point at its *earliest computation time* (ECT), forced by the existing dependence vectors. This policy guarantees the optimal execution time of the entire loop. We consider also, that each computation takes a single time instance for execution (*Unit Execution Time* model).

We proceed now to define the most important concepts of our work:

- The index space is partitioned into disjoint **time subsets** denoted T_k, $k \geq 0$, such that T_k contains the set of points of J, whose earliest computation time is k. By definition, T_0 denotes the boundary (precomputed) points.

- Consider any doubly nested loop along with its index space and its uniform dependence vectors. The geometric shape of the subset of index points that can be computed initially (T_1) is called **initial pattern** (see Figure 2) and is denoted as Pat_1.

- Using the algorithms described in [7], starting from the initial pattern, we can compute successive similar wavefront-like polygons which we denote by $Pat_k, k \geq 1$. We call Pat_k the **pattern corresponding to moment k**. The intuition behind the pattern notion is that Pat_k is the geometric representation of T_k. By definition, the initial pattern is simply Pat_1. In Figure 1, we depict the patterns that correspond to moments $1, 2, 3$ and 4. For completion reasons, we define $Pat_0 = T_0$.

- **Pattern-outline** is the upper boundary of each pattern Pat_k and is denoted as $pat_k, k \geq 1$. In this work, the points of the pattern outlines are given and computed in clockwise order, unless noted. So, pat_k is a *list* of points, not a simple set. As above: $pat_0 = \langle(0,0)\rangle$.

- **Pattern points** are those points that are necessary in order to define the polygon shape of the pattern. The rest of the points just complete the polygon area. Consider for example points A, B and C in Figure 2. Points A and C are pattern points; B is not.

- **Pattern vectors** are those dependence vectors whose end-points are the pattern points of the initial pattern. Their number is usually denoted as m.

- **Dependence difference vectors** are the vectors that are the difference of the m pattern vectors. They are at most $m-1$ and we define two types of them: the *absolute* difference vectors: $\mathbf{a}_1 = \mathbf{d}_2 - \mathbf{d}_1, \mathbf{a}_2 = \mathbf{d}_3 - \mathbf{d}_1, \ldots, \mathbf{a}_{m-1} = \mathbf{d}_m - \mathbf{d}_1$, and the *relative* difference vectors: $\mathbf{b}_1 = \mathbf{d}_2 - \mathbf{d}_1, \mathbf{b}_2 = \mathbf{d}_3 - \mathbf{d}_2, \ldots, \mathbf{b}_{m-1} = \mathbf{d}_m - \mathbf{d}_{m-1}$ (see Figure 3). Without loss of generality, we consider that the m pattern vectors $\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_m$, are already lexicographically sorted, so that $\mathbf{d}_1$ is the the one with the least x-coordinate and so on.

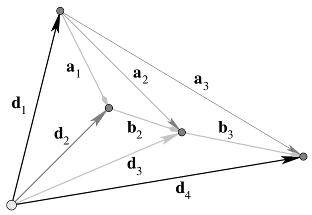

Figure 3. *Absolute and relative dependence difference vectors for an example algorithm with 4 dependence vectors.*

In the rest of this work, by the term *pattern boundary* we usually refer to pattern points only, which are painted in darker coloration in all figures. Also, *time instance* is sometimes referred to as *step* or *moment*.

The reader should also note, that pattern points and vectors are represented by lists, instead of sets, as they are considered lexicographically sorted, unless noted.

3 Analytical Pattern Computation

In this section the geometric procedure of the pattern formation will be presented. Certain computational geometry algorithms are used, while the interested reader may refer to [7] for more details. We mention the two problems that we are concerned onto:

- **Wavefront Problem**: *compute the arbitrary pattern boundary* pat_k *that corresponds to time instance* k.

- **Scheduling Problem**: *given any index-space point, locate the two consecutive patterns that include this point.*

The crucial feature is that the general pattern (that of an arbitrary moment) depends *only* on the initial pattern form.

3.1 Initial Pattern Formation

Initial pattern contains all points that can be computed at $t = 1$. Each dependence vector imposes a restriction to the points that can be computed. So, only a subspace of J can be executed initially, due to each vector. By intersecting the subspaces for all dependence vectors, initial pattern is formed, as shown in Figure 4.

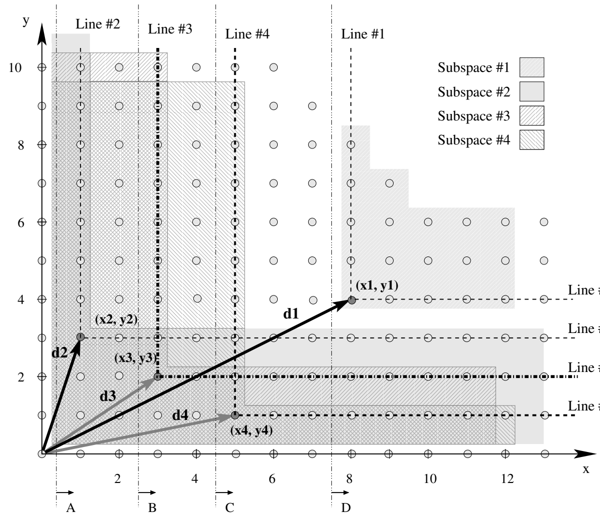

Figure 4. *Formation of the initial pattern through subspace intersection. Note that only pattern vectors* $\mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4$ *affect initial pattern boundary.*

We present a computational geometry plane-sweeping algorithm that efficiently solves this problem (for more on plane-sweeping algorithms see [5]).

Algorithm I *(Initial pattern-outline formation)*

INPUT: The m dependence vectors $\mathbf{d}_1, \ldots, \mathbf{d}_m$ of the algorithm.

OUTPUT: Initial pattern outline $pat_1 = \langle \mathbf{p1}_1, \mathbf{p1}_2, \ldots, \mathbf{p0}_l \rangle$, $l \leq m$.

1. Position all dependence vectors at the origin of the axes. The end-points of the dependence vectors are the candidate pattern points (Figure 2). The initial pattern-point list is currently empty: $pat_1 = \emptyset$.

2. From the origin of the x-axis, start sweeping with a vertical *scan-line* to the right.

3. While moving along x-axis, the first dependence vector end-point that the scan-line comes across, is added to the pattern-point list (in our example this is point **A**): $pat_1 = \langle \mathbf{p0}_1 \rangle$.

4. Continue moving the scan-line to the right. For every dependence vector end-point that is reached, it's y-coordinate is checked; if it is less than the y-coordinate of the point last added, the current end-point is also added to the list.

5. Repeat Step 4, until the endpoints of all dependence vectors are crossed.

6. The resulting point-list pat_1 contains the initial pattern outline.

By the above algorithm the pattern vectors can also be computed.

3.2 General Pattern Formation

Before continuing, we present *Algorithm II*, which will be used further on. Given a set of points, it computes the pattern outline that these points form; in other words it discards the points that are not contributing to the outline. One can note that it is a generalization of *Algorithm I*.

Algorithm II *(Pattern formation out of a set of points)*
INPUT: A set of points that may contribute to the pattern outline.
OUTPUT: The list of points that form the pattern outline.

1. All points are sorted by their x-coordinate and a sorted list is created, so that the point with the smallest x-coordinate would be first at this candidate point list.

2. Pattern point list is now empty. First point (the one with the smallest x-coordinate) is inserted into that list and next (second) point is now the *current* one.

3. Compare current point's y-coordinate with that of last inserted point. If smaller, insert current point into list, else current point is omitted. For points with equal x-coordinate, *only* the one with the smallest y-coordinate is put on the pattern list. Any other is removed from the list.

4. If there exist any points that have not been checked yet, make current point the next in the candidate point list and repeat Step 3. Otherwise, current time instance pattern is ready.

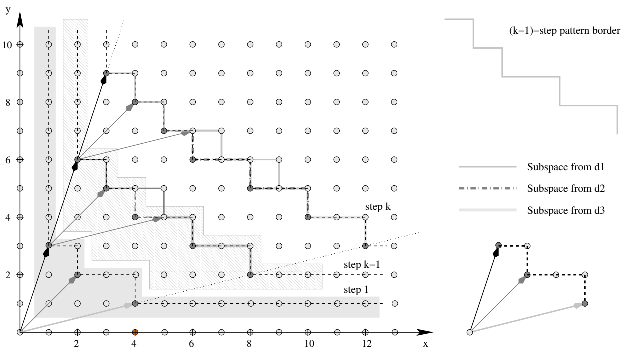

Figure 5. *Formation of k-step pattern in an example algorithm with 3 pattern vectors: the intersection of all transpositions of the $(k-1)$-step pattern according to all pattern vectors. The three images of the k-step border are drawn with different shades, as shown on the right of the figure. Initial pattern is also shown.*

In the previous sections we have described the procedure that the initial pattern is formed: as an *intersection of* the *subspaces* formed by the pattern vectors.

A similar, subspace-intersection procedure, can be followed for the general k-step pattern outline by considering all subspaces that contribute to the formation of this pattern. These subspaces are the m transpositions applied to $(k-1)$-step pattern outline by each pattern dependence vector, as in Figures 5 and 6.

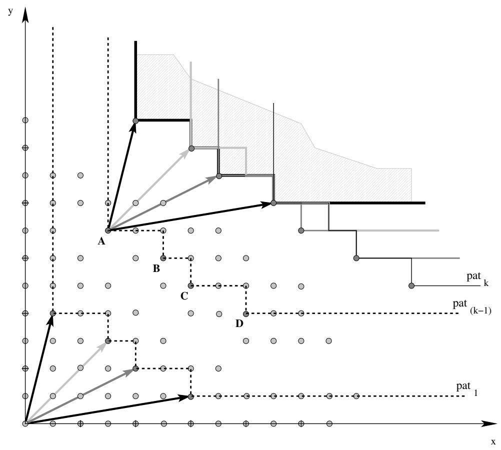

Figure 6. *Computing k-step pattern from subspaces formed by the previous one: pat_{k-1} is moved by each dependence vector $\mathbf{d}_1, \ldots, \mathbf{d}_4$, starting from all points on pat_{k-1}. The intersection of the resulted subspaces, form the k-step pattern outline pat_k. For simplicity, only the subspace along $\mathbf{d}_1$ (inside darker-line border) has been shaded and only the 4 subspaces formed by point **A**, are shown.*

The algorithm that computes the general k-step pattern, is based on the two algorithms that we presented before and is formally described as follows:

Algorithm III *(Formation of k-step pattern outline)*
INPUT: Dependence vector list and time-instance number $k \in \mathbf{N}$, $k > 0$.
OUTPUT: The point lists of the oulines of all patterns until k.

1. Apply *Algorithm I* to input list: pat_1 is found (the m pattern points, as well as the m pattern vectors are calculated).

2. If $k = 1$, return pat_1.

3. If $k > 1$, for every point $\mathbf{p}_i \in pat_{k-1}$, find all m points that are spawned by all pattern vectors. These points form set S_k, of candidate points of pat_k; note that $|S_k| \leq m \cdot |pat_{k-1}|$.

4. Apply *Algorithm II* to candidate point set S_k: a new list pat_k is generated, which contains k-step pattern outline points.

Although it is the first algorithm that computes the exact k-step pattern outline, one can note that it works recursively, which means that for k-step pattern outline computation, all $1, 2, \ldots, (k-1)$-step outlines should be computed too.

Also, several k-step points are computed from more than one points of the previous pattern, which indicates procedure's inefficiency. However, it is the best we can get so far, for the general dependence vector case.

Only in certain special cases, its outline can be computed quite efficiently, as it will described further on (*Uniformity Condition*).

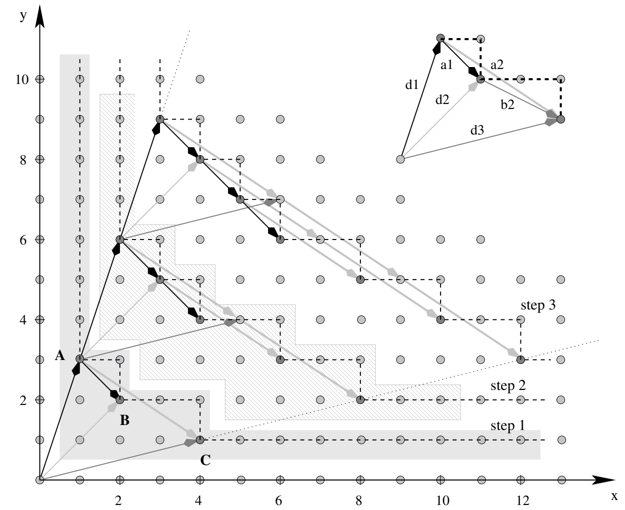

Figure 7. *Formation of the 3 first pattern boundary points by the difference vectors* $\mathbf{a}_i$, *i = 1, 2, ..., m-1 and* $\mathbf{d}_{xmin} \equiv \mathbf{d}_1$. *Every point* $\mathbf{j} \in pat_1$ *is given from* $\mathbf{j} = \mathbf{d}_{xmin} + k_1 \cdot \mathbf{a}_1 + k_2 \cdot \mathbf{a}_2$, *where* $k_1, k_2 \in \mathbf{N}$ *and* $k_1 + k_2 \leq 1$. *For example, point* ***A*** *is given for* k_1 *= 0 and* k_2 *= 0, point* ***B*** *is given for* k_1 *= 1 and* k_2 *= 0, while point* ***C*** *is given for* k_1 *= 0 and* k_2 *= 1.*

3.3 General Pattern Analytical Description

The above procedure implies the obvious method of finding the pattern exhaustively, by spanning the pattern vectors from the beginning of the axes. In other words, the pattern vectors are the ones that produce the *innermost concave wavefront* of points that are ready for execution. Step k pattern is formed by following the pattern vectors, starting from $(k-1)$-step pattern.

It is obvious that Pat_1 consists of all points $\mathbf{j} = (j_x, j_y)$, such that $\mathbf{j} - \mathbf{d}_i \in T_0$ for any dependence vector $\mathbf{d}_i, 1 \leq i \leq m$. Hence, the points of pat_1 will be a subset of the end-points of the dependence vectors: $pat_1 \subseteq \{\mathbf{d}_i\}, 1 \leq i \leq m$.

Let $\mathbf{d}_{xmin}$ be the pattern vector with the minimum x-coordinate, as shown in Figure 7. Every dependence vector can be expressed as $\mathbf{d}_i = \mathbf{d}_{xmin} + \mathbf{a}_i$, and equivalently, pat_1 can be written as:

$$pat_1 \subseteq \{\mathbf{d}_{xmin} + k_1 \cdot \mathbf{a}_1 + \ldots + k_{m-1} \cdot \mathbf{a}_{m-1}\},$$

where:

$k_i \in \mathbf{N}$ and $k_1 + k_2 + \ldots + k_{m-1} \leq 1, 1 \leq i \leq m$.

The initial pattern is formed by the intersection of the subspaces formed by the dependence vectors and more precisely, by the pattern vectors. Hence pat_1 includes the points $\mathbf{j} = (j_x, j_y)$ such that:

- $k \cdot d_{xmin_x} + k_1 \cdot a_{1_x} + \ldots + k_{m-1} \cdot a_{m-1_x} \leq j_x$
- $k \cdot d_{xmin_y} + k_1 \cdot a_{1_y} + \ldots + k_{m-1} \cdot a_{m-1_y} = j_y$
- $j_y = minimum$ *(for a certain* j_x *may be multiple* j_y*)*
- $k_1 + k_2 + \ldots + k_{m-1} \leq 1 = k$.

As any pattern outline can be derived from the previous one, by taking the transpositions mentioned before, the above expression can recursively be generalized as follows. The general pattern outline pat_k includes all points $\mathbf{j} = (j_x, j_y)$ such that:

- $k \cdot d_{xmin_x} + k_1 \cdot a_{1_x} + \ldots + k_{m-1} \cdot a_{m-1_x} \leq j_x$
- $k \cdot d_{xmin_y} + k_1 \cdot a_{1_y} + \ldots + k_{m-1} \cdot a_{m-1_y} = j_y = min$

and

- $k_1 + k_2 + \ldots + k_{m-1} \leq k$, with $k_i \in \mathbf{N}, 1 \leq i \leq m-1$.

According to the above, the formal description of the two problems can be generally stated as:

Wavefront Problem: *given the number of the time instance k, we would like to find all index-space points that form k-step pattern outline* pat_k. *The formal restrictions that give point* $\mathbf{j} = (j_x, j_y) \in pat_k$ *are:*

- $k \cdot d_{xmin_x} + k_1 \cdot a_{1_x} + \ldots + k_{m-1} \cdot a_{m-1_x} \leq j_x$
- $k \cdot d_{xmin_y} + k_1 \cdot a_{1_y} + \ldots + k_{m-1} \cdot a_{m-1_y} = j_y = min$
- $k_1 + \ldots + k_{m-1} \leq k$, with $k_i \in \mathbf{N}$, for $1 \leq i \leq m-1$

Scheduling Problem: *given any index-space point* $\mathbf{j} = (j_x, j_y)$, *we would like to find the time-instance k that it can be executed. The restrictions for k are:*

- $k \cdot d_{xmin_x} + k_1 \cdot a_{1_x} + \ldots + k_{m-1} \cdot a_{m-1_x} \leq j_x$
- $k \cdot d_{xmin_y} + k_1 \cdot a_{1_y} + \ldots + k_{m-1} \cdot a_{m-1_y} \geq j_y$
- $k_1 + k_2 + \ldots + k_{m-1} \leq k, k_i \in \mathbf{N}, 1 \leq i \leq m-1$, and
- $k = minimum$

From the description of the two problems, one can note that they can be solved analytically using *linear programming* techniques. However, the above expressions are the first analytical ones that give any point of k-step pattern outline for a general uniform dependence loop algorithm.

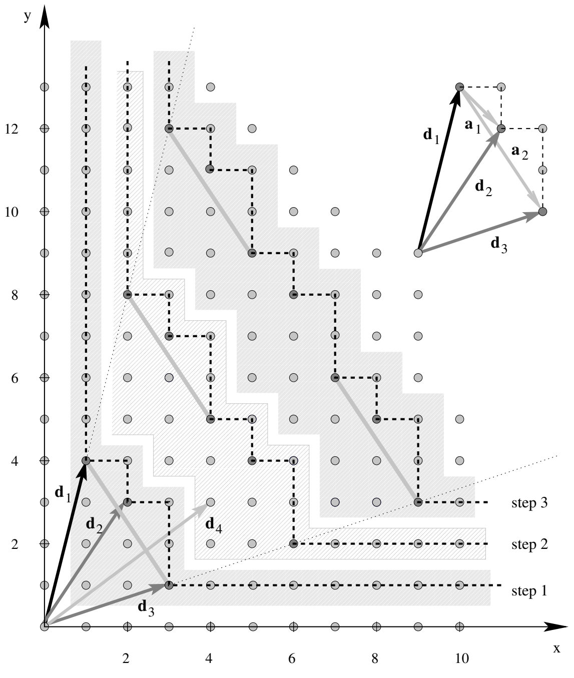

Figure 8. *An algorithm with 4 dependence vectors $\mathbf{d}_1 = (1,4)$, $\mathbf{d}_2 = (2,3)$, $\mathbf{d}_3 = (3,1)$ and $\mathbf{d}_4 = (4,3)$, which is not a pattern vector. The pattern outline is perfectly scalable: general k-step pattern outline is computed by considering the outline of the initial pattern k times.*

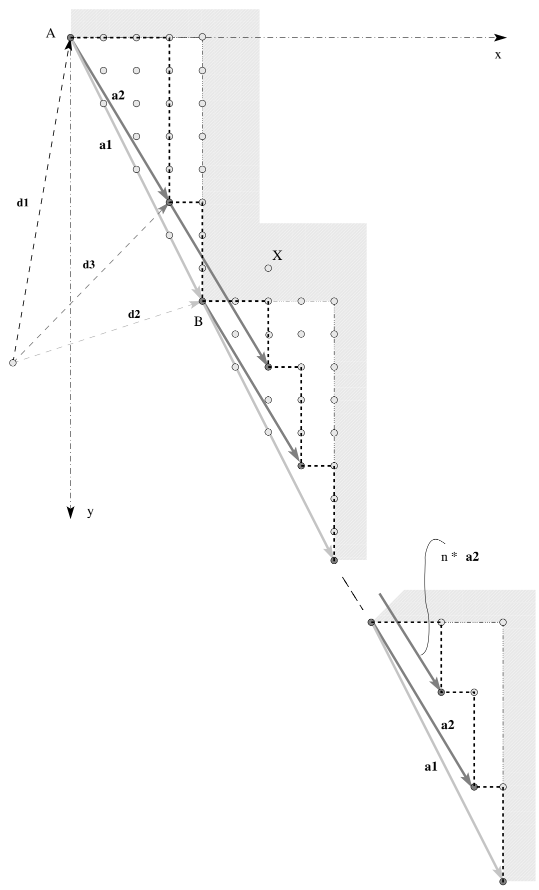

Figure 9. *Pattern outline depends on the slopes of the difference vectors. It is comprised by distinct parts, whose outline is the triangle formed by $\mathbf{d}_{xmin}$ and $\mathbf{d}_{ymin}$, as shown. The pattern points are determined by the endpoints of a linear combination of the algorithm dependence difference vectors.*

3.4 Pattern Outline Uniformity Condition

The interesting point is that the above expression, can be quite simplified in certain cases. In these cases the pattern outline is *perfectly* scalable and k-step outline is more than obvious, as in Figure 8.

We will describe a condition that makes the pattern precisely normal and uniform. If the given condition holds, the k-step pattern can be computed by –simply– considering the initial pattern k times.

Lemma 3.1 *Consider a nested loop algorithm with two dependence difference vectors $\mathbf{a}'_1$ and $\mathbf{a}'_2$ and their slopes* g_1 and g_2, respectively. If $g_1 \geq g_2$ and $a'_{1y} \geq a'_{2y}$ holds, then the points that contribute to pattern outline formation are (as in Figure 9):*

- *the endpoints of $k \cdot \mathbf{a}'_1$, where $0 \leq k \leq n$,*
- *potentially, the endpoints of $k \cdot \mathbf{a}'_2$, that reach the triangles formed by $\mathbf{a}'_1$.*

Proof†
Obviously, as difference vector $\mathbf{a}'_1$ is the one with the greatest slope, the end points of $k \cdot \mathbf{a}'_1$ will all be pattern outline points. All extra (potential) points will be end points of vectors given by the expression: $k_1 \cdot \mathbf{a}'_1 + k_2 \cdot \mathbf{a}'_2$, where $0 \leq k_1, k_2 \leq k$ and $k_1 + k_2 = n$ (for the n time instance). Also, if a point X belongs to the shaded region shown, it doesn't affect the pattern, as there is a point with less x and y-coordinate which eliminates its influence (point B). ⊣

Lemma 3.2 *Consider m dependence difference vectors $\mathbf{a}_i$ and their slopes g_i, such that $g_1 \geq g_i$ for $2 \leq i \leq m-1$. Without loss of generality $a_{1y} \geq a_{iy}$, $2 \leq i \leq m-1$. Then the previous Lemma holds for $\mathbf{a}'_1 = \mathbf{a}_1$ and $\mathbf{a}'_2$ any linear combination of the form $k_2 \cdot \mathbf{a}_2 + k_3 \cdot \mathbf{a}_3 + \ldots + k_m \cdot \mathbf{a}_{m-1}$, where $0 \leq k_i \leq m-1$ and $k_2 + k_3 + \ldots + k_{m-1} = n$.*

Proof
Obvious from previous Lemma and $\mathbf{a}_i$ definitions. ⊣

Theorem 3.1 *Consider a nested loop algorithm with m pattern vectors $\mathbf{d}_{p_1}, \mathbf{d}_{p_2}, \ldots, \mathbf{d}_{p_m}$. Let $\mathbf{a}_i = \mathbf{d}_{p_i} - \mathbf{d}_{p_{i+1}}$*

* Using the coordinate system of Figures 9 and 10.
† Based on the formulas of §3.3.

be the $m-1$ difference vectors, with slope values g_i. If the following hold, for $2 \le i \le m-1$:

1. *$g_1 \ge 2 \cdot g_i$,*
2. *$a_{1y} \ge a_{iy}$ and*
3. *$a_{ix} \ge \frac{a_{1x}}{2}$*

then the pattern outline is recurrent: it can be computed by replicating every pattern part by the first part (which is the pattern outline of the first time instance).

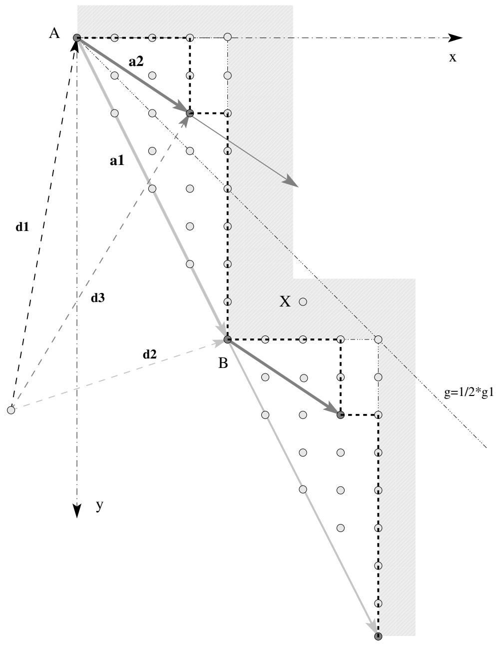

Figure 10. *Theorem explanation; consider the difference vector $\mathbf{a}_{gmax} \equiv \mathbf{a}_1$, as the one with the largest slope value g_{max}. If the slope values of all other difference vectors are less than the half of g_{max}, the outline of each pattern can be produced by a repeated "subpattern".*

Proof
It is obvious that every linear combination of the difference vectors $\mathbf{a}_i$, is located at the shaded area, because for the resultant slope g_{sum} it is: $2 \cdot g_{sum} \le g_1$ and $a_{sum_x} \ge a_{1_x}$, because of (3). So, for the pattern outline formation only vectors $\mathbf{a}_i$ contribute. Every vector $k_i \cdot \mathbf{a}_i$, due to (1), (2) and (3), creates only one point at every triangle, at the same (relative) position. According to the above and from the two previously presented lemmata, the pattern outline is recurrent and it is formed by the pattern of the first time instance. ⊣

Many other conditions of uniformity can be proved, based on the same approach. Figures 5 and 7 give another such uniformity case: the initial pattern is split into certain subpatterns. By taking each subpattern k-times, the k-step pattern is precisely formed.

It can be proved that the conditions for this case are:

- $g_1 > g_2 > \ldots > g_{m-1}$ and
- $a_{1y} < a_{2y} < \ldots < a_{(m-1)y}$

3.5 Pattern Outline Computation Complexity

In this section we study the complexity of the above procedure for pattern computation (complexity of *Algorithms I, II* and *III*).

For *Algorithm I*, we have at most m dependence vector end-points that form the pattern outline. These points are sorted, by ascending x-coordinate and then checked for descending y-coordinate order, in other words, algorithm's complexity is:

$$O(m \cdot \log m) + O(m) = O(m \cdot \log m).$$

On *Algorithm III*, complexity will be found recursively, as for calculating k pattern, $(k-1)$-pattern must have been calculated also. Supposing that $Cx(p_k)$ is the complexity of step-k pattern-outline computation, it is for step 1:

$$Cx(p_1) = O(m \cdot \log m),$$

as only *Algorithm I* is applied for this step. For the outline of the arbitrary step k it is $|pat_k| = m^k$, in the worst case. For computing next step outline pat_{k+1}, we get all points that can be reached by every point $\mathbf{p}_k \in pat_k$, along any pattern vector (at most m). All these $m^k \cdot m$ points need to be sorted. So:

$$Cx(p_{k+1}) = Cx(p_k) + m^k \cdot m + (m^{k+1} \cdot \log(m^{k+1})) \Rightarrow$$
$$Cx(p_{k+1}) = Cx(p_k) + m^{k+1} \cdot (1 + \log(m^{k+1})) \Rightarrow$$
$$Cx(p_{k+1}) = Cx(p_k) + O(m^{k+1} \cdot \log(m^{k+1})).$$

In other words:

$$Cx(p_2) = Cx(p_1) + m^2 \cdot \log(m^2) = O(m^2 \cdot \log(m^2)),$$
$$Cx(p_3) = Cx(p_2) + m^3 \cdot \log(m^3) = O(m^3 \cdot \log(m^3)),$$
$$\vdots$$
$$Cx(p_k) = O(m^k \cdot \log(m^k)).$$

So, the complexity of computing the k-step pattern outline pat_k, for a nested loop algorithm with m dependence vector is:

$$Cx(pat_k) = O(m^k \cdot \log(m^k))$$

This result proves the pattern-outline exponential computational cost that mentioned before.

4 Pattern Computation and Scheduling

In [6] we have presented an efficient scheduling algorithm, based on the pattern theory that we presened above. The index space is partitioned into *time regions*, which correspond to *pattern borders*, in our method. For each time instance, the borders of the corresponding pattern and the time region slightly differ, which in turn, improves the scheduling complexity. The exact border of the pattern is not always calculated precisely.

It is proven in [7] that this scheduling technique is asymptotically optimal, in the sense that the execution time assigned to each point is *at most one* more than its earliest computation time. In Figure 11, the described pattern-outline methodology that was previously described, as well as the one presented in [6] for a given doubly nested loop, are shown.

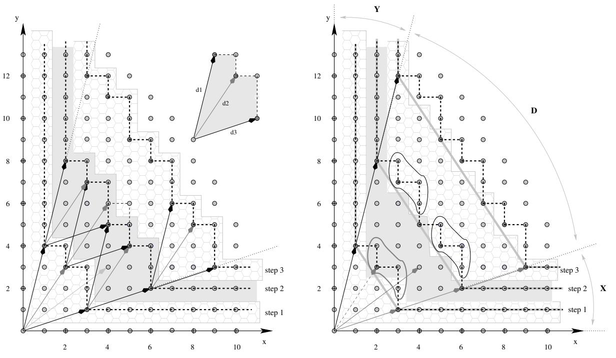

Figure 11. *Pattern formation for an example algorithm with 4 dependence vectors.* ***a)*** *By the analytical method (left) the accurate pattern is computed, but an efficient scheduling algorithm cannot be easily achieved.* ***b)*** *A variation of the analytical method is also shown (right); although it computes a trapezoid (less accurate) k-step pattern, it leads to an efficient scheduling. The points mentioned are the ones omitted for the next time instance.*

As it is shown in Figure 11, the index space is split into several conic subspaces (which are angles in 2-D). Although we are not following a recursive procedure, we can approximately predict the *outline* of the k-step pattern. Certain geometric sub-patterns, called *regions*, are formed. These regions have simpler representation than the accurate patterns, which simplifies point-location, but give suboptimal scheduling results.

5 Conclusion

In this paper we presented a geometric method of pattern-outline prediction for uniform dependence nested loop algorithms. We described the exhaustive procedure for computing the precise k-step pattern outline, which is the best we can get so far. Then we presented certain expressions that describe analytically the general pattern formation. Finally, certain conditions that ensure pattern uniformity are presented. These conditions lead to to a low complexity scheduling method, which is a major improvement, as far as scheduling complexity is concerned.

We believe that our methodology can be generalized to handle an arbitrary n-dimensional uniform dependence loop and we are currently working towards this goal.

References

[1] T. Andronikos, N. Koziris, G. Papakonstantinou, and P. Tsanakas. An efficient algorithm for the optimal linear schedule of uniform dependence algorithms. In *Proceedings of the First ECPD International Conference on Advanced Robotics and Intelligent Automation*, pages 140–145, Athens, Greece, 1995.

[2] T. Andronikos, N. Koziris, Z. Tsiatsoulis, G. Papakonstantinou, and P. Tsanakas. Lower bounds for parallel time and number of processors for the class of uniform dependence algorithms. Technical Report DSCLAB-TR-95-03, Digital Systems and Computers Laboratory - N.T.U.A., 1995.

[3] A. Darte, L. Khachiyan, and Y. Robert. Linear scheduling is nearly optimal. *Parallel Processing Letters*, 1.2:73–81, 1991.

[4] A. Darte and Y. Robert. Constructive methods for scheduling uniform loop nests. *IEEE Transactions on Parallel and Distributed Systems*, 5(8):814–822, 1994.

[5] M. de Berg, M. van Kreveld, M. Overmans, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 1997.

[6] I. Drositis, T. Andronikos, A. Kokorogiannis, G. Papakonstantinou, and N. Koziris. Geometric scheduling of 2-d uniform dependence loops. In *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS 2001*, Korea, 2001.

[7] I. Drositis, T. Andronikos, and G. Papakonstantinou. 2-d pattern prediction and scheduling. Technical Report CSLAB-TR-00-01, Computing Systems Laboratory - N.T.U.A., 2000.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-completeness*. W. H. Freeman and Company, New York, 1979.

[9] N. Koziris, G. Papakonstantinou, and P. Tsanakas. Optimal time and efficient space free scheduling for nested loops. *The Computer Journal*, 39(5):439–448, 1996.

[10] L. Lamport. The parallel execution of do loops. *Communications of the ACM*, 37(2):83–93, February 1974.

[11] C. Papadimitriou and M. Yannakakis. Toward an architecture-independent analysis of parallel algorithms. *SIAM Journal of computers, Extended Abstract in Proceedings STOC 1988*, 19:322–328, 1990.

[12] W. Shang and J. Fortes. Time optimal linear schedules for algorithms with uniform dependencies. *IEEE Transactions on Computers*, 40(6):723–742, 1991.

[13] J. Sheu and T. Tai. Partitioning and mapping nested loops on multiprocessors systems. *IEEE Transactions on Parallel and Distributed Systems*, 24:430–439, 1991.