Geometric Scheduling of 2-D UET-UCT
Uniform Dependence Loops

Ioannis Drositis, Theodore Andronikos, George Manis,
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

Email: jdros@cslab.ntua.gr

Abstract

Finding an optimal time schedule is one of the primary tasks in the area of parallelizing uniform dependence loops. Due to the existence of dependence vectors, the index space of such a loop is split into subspaces of points that can be executed at different time instances. The geometric representation of these sets form certain polygonal shapes called patterns, with special attributes and characteristics. In this paper we present a scheduling technique that is based on the geometric attributes of the index space and the dependence vector set. Our strategy can be applied to architectures that consider unit execution-zero communication delay (UET) or unit execution-unit communication (UET-UCT) model, as a new method for transforming UET-UCT problems to UET equivalent ones is presented.

Keywords: UET-UCT loop scheduling.

1 Introduction

One of the primary tasks when dealing with a nested loop is scheduling the computations, while preserving the data dependencies imposed by the initial (lexicographic) loop ordering. Iterations are considered as points in a multidimensional index space, and the dependence vectors are considered as directed arcs between the corresponding indexed points. We are focusing on iteration level parallelism (fine-grain).

Time scheduling of loop-based iterations with uniform dependencies, was introduced by Karp in [12] (for uniform recurrence equations) and elaborated by Lamport in [14], who partitioned the index space into hyperplanes. All iteration-indexed points belonging to the same hyperplane can be concurrently executed. Darte in [7], proved that this method is nearly optimal, while the problem of finding the hyperplane that results in the minimum makespan was solved in [1]. Moldovan, Shang, Darte and others applied the hyperplane method to find a linear optimal execution schedule, using diophantine equations [15], linear programming in subspaces [18], integer programming [7], or graph-theoretic approaches [4]. All these approaches consider unit execution time for each iteration and zero communication for each communication step (UET model).

To cope with a more general approach of the scheduling problem, considering also communication delays, many researchers have focused on task scheduling, which is one of the most important and difficult problems of parallel computing. The algorithm is modeled with a task graph, generally a directed acyclic graph (DAG), where the computation and communication times are depicted as node and edge weights, respectively. Since the general scheduling problem is well known to be NP-complete (see Garey [11]), researchers have given attention to other methods such as heuristics, approximation algorithms etc.

When both computation and communication times are restricted to have unit time length, it is known that scheduling UET-UCT graphs with a bounded number of processors is NP-complete as proved by Rayward-Smith in [17] and by Picouleau in [16]. Even the case of unlimited processors, when task duplication is allowed, is in general polynomially intractable [16]; when the index space is limited another optimal approach in presented in [13]. Restricting to special cases decreases dramatically the complexity, resulting in polynomial time solutions as in [6, 4].

In [5] Chretienne presented an algorithm linear in the cardinality of the vertices of the graph, for optimal makespan on SCT (small communication time, thus including UCT) in-trees and out-trees. Also, there exist polynomial optimal solutions for series-parallel digraphs, bipartite
A nearly optimal scheduling approach for 2-D UET uniform nested loops is presented in [9], based on the geometric properties of the index space and the dependence vectors. The index space of such a loop, can be partitioned into a sequence $ECT_1, ECT_2, \ldots$ of (disjoint) subsets, such that $ECT_t$ contains precisely those points whose earliest computation time is $t$. The geometric representation of these subsets form certain wavefront-like polygonal shapes called patterns (Figure 1). The aforementioned technique, partitions the index space into geometric regions that approximate patterns and have simpler representation, and schedules computations according to the region they belong to.

![Figure 1. Consider an algorithm with dependence vectors $d_1 = (1, 3), d_2 = (2, 2), d_3 = (4, 1)$ and $d_4 = (4, 3). We shown the four initial patterns.](image)

In this work, we present an extension of the above approach, considering communication delays that arise in more general architectures. We do this by introducing a transformation of UET-UCT problems to equivalent UET problems. This is the major contribution of this paper, as no other relative method has been proposed so far. All present UET-UCT scheduling techniques refer to grid or extended grid architectures [2] and to graph and tree algorithms [16, 17]. The UET-UCT polynomial scheduling algorithm is nearly optimal, as it achieves a makespan of constant delay. This algorithm can be generalized for higher dimensions.

Basic terminology and definitions are presented in Section 2, while principal study for the UET case follows in Sections 3 and 4. In Section 5 the UET-UCT case is studied. We conclude in Section 6.

2 Terminology and Definitions

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

$$\text{for } (i_1 = 1; i_1 <= u_1; i_1++) \{$$

$$S_1(i_1);$$

$$\ldots$$

$$S_k(i_1);$$

$$\ldots$$

$$\}$$

where $S_1, \ldots, S_k$ are assignment statements and $J = \{(i_1, \ldots, i_n) | 1 \leq i_r \leq u_r, 1 \leq r \leq n\}$ is the index space, where $J \subset \mathbb{N}^n$. We follow the model that was introduced by Lamport in [14]; the dependence vectors are “uniform”, i.e., constant all over the index space.

Our scheduling policy executes every index point at its earliest computation time (ECT), taking the existing dependence vectors into consideration. This policy guarantees the optimal execution time of the entire loop. We also assume that the execution of each computation requires a single unit of time (Unit Execution Time model). We now give a few definitions.

The index space is partitioned into disjoint time subsets denoted $ECT_t, t \geq 0$, such that $ECT_t$ contains the set of points of $J$, whose earliest computation time is $t$. By definition, $ECT_0$ denotes the boundary (pre-computed) points.

Consider any doubly nested loop along with its index space and its uniform dependence vectors. The geometric shape of $ECT_t$ is called pattern of instance $t$ and is denoted as $Pa_t$ (see Figure 1). For consistency we define $Pa_0 = ECT_0$.

Let $d = (d_x, d_y)$ be a 2-D point; then:

$$J_d = \{j = (j_x, j_y) \in J | d_x \leq j_x, \text{ and } d_y \leq j_y\}.$$ 

d_1, d_2, \ldots, d_m are the dependence vectors of the algorithm.

Their number is denoted $m$. In the rest of this paper, without loss of generality, we assume that the $m$ dependence vectors $d_1, d_2, \ldots, d_m$ are lexicographically sorted, so that $d_1$ is the one with the least $x$-coordinate. We use $DV$ to denote the sequence of them: $DV = (d_1, d_2, \ldots, d_m)$.

The sequence of pattern vectors $PV = (p_1, \ldots, p_k)$, is the minimum subset of the dependence vectors, whose endpoints satisfy the following condition: $J_{d_1} \cup \ldots \cup J_{d_m} = J_{p_1} \cup \ldots \cup J_{p_k}$. Their number is denoted as $k \leq m$.

Given a sequence of points $A = (j_1, j_2, \ldots, j_k)$ and a vector $f$, then $A + f = (j_1 + f, j_2 + f, \ldots, j_k + f)$.

Given a sequence of points $A = (j_1, j_2, \ldots, j_k)$ and a sequence of vectors $B = (f_1, f_2, \ldots, f_k)$, then $A + B = (A + f_1, A + f_2, \ldots, A + f_k)$.

Given the $m$ dependence vectors $d_1, d_2, \ldots, d_m$, we give the following definitions:

The convex subspace that is formed by two vectors $a$ and $b$ (in 2-D) is called a cone and is denoted $Con(a, b)$. Formally, $Con(a, b) = \{j \in \mathbb{N}^2 | j = \lambda_1 \cdot a + \lambda_2 \cdot b, \text{ where } \lambda_1, \lambda_2 \geq 0\}$.

The cones formed by the $x$ and $y$ axes and the dependence vectors with the least $x$ and $y$-coordinate, respectively, are called $Y$-cone and $X$-cone. These vectors are called $d_{min}$ and $d_{min}$ and the cone they defined is called $D$-cone = $Con(d_{min}, d_{min})$ – see Figure 4.

\*\*We assume that the origin of the dependence vectors is the origin of the axes.
have the following property. They form a lexicographically ordered sequence \( CV = \{c_1, c_2, \ldots, c_r\} \), such that:

- \( d = \lambda_1 \cdot c_1 + \lambda_2 \cdot c_{i+1} \), where \( \lambda_1 + \lambda_2 > 1 \), for every \( d \in DV \), \( d \neq c_i , c_{i+1} \) and for exactly one pair of \( (c_i , c_{i+1}) \).

The number of cone vectors is denoted by \( r \). Note that \( d_{emin} \) and \( d_{gmin} \) are always included in \( CV \) (\( c_1 = d_{emin} \) and \( c_r = d_{gmin} \)).

- **X_i-Region** is the set: \( X_i = \{j = (j_x, j_y) \in X \mid (i - 1) \cdot d_{emin_y} < j_y < i \cdot d_{emin_y}, \text{ where } i > 0 \} \).
- **Y_i-Region** is the set: \( Y_i = \{j = (j_x, j_y) \in Y \mid (i - 1) \cdot d_{gmin_y} < j_y < i \cdot d_{gmin_y}, \text{ where } i > 0 \} \).

**Cone Region**: The \( i \)-th region of a cone \( Con(a, b) \) is the set \( Con_i(a, b) = \{j \in Con(a, b) \mid j = \lambda_1 \cdot a + \lambda_2 \cdot b, \text{ where } i < \lambda_1 + \lambda_2 \leq i + 1 \} \) —see Figure 3.

**Region**: Region \( \mathcal{R}_i \) is denoted as the union of \( i \)-regions of all cones:

\[
\mathcal{R}_i = \bigcup_{j=1}^{i-1} Con_i(c_j, c_{j+1}) \cup X_i \cup Y_i.
\]

**Figure 2.** Except \( \mathbf{d}_2 \) and \( \mathbf{d}_4 \), all other vectors belong to the cone vector sequence: \( CV = \{\mathbf{d}_1, \mathbf{d}_3, \mathbf{d}_5\} \). The 5 cones that are formed are colored respectively.

We also define the problem that we are going to study. Given a nested loop algorithm, with index space \( J \) and dependence vector sequence \( DV \):

*Given any index point, compute the time instance that it should be executed; in other words, specify the region that includes this point.*

### 3 Scheduling with Cones & Regions

Our method divides the index space into several cones. Every point located into such a cone, can be written as a linear combination of the vectors that form the cone, with positive coefficients. Intuitively, cone vectors determine the execution order among the points that belong in the cone, by determining the optimal hyperplane for the cone, as proved in [3].

Consider, for example, the cone formed by vectors \( d_{emin} \) and \( d_{gmin} \) in Figure 4 and an index point \( j = (j_x, j_y) \). If \( j = \lambda_1 \cdot d_{emin} + \lambda_2 \cdot d_{gmin} \), and \( d_{emin} \) and \( d_{gmin} \) are lexicographically ordered, then we distinguish the following cases:

- if \( \lambda_1 > 0 \) and \( \lambda_2 > 0 \) \( \Rightarrow \) \( j \in Con(d_{emin}, d_{gmin}) \)
- if \( \lambda_1 > 0 \) and \( \lambda_2 < 0 \) \( \Rightarrow \) \( j \in Y \)-cone
- if \( \lambda_1 < 0 \) and \( \lambda_2 > 0 \) \( \Rightarrow \) \( j \in X \)-cone

For example, considering Figure 4, if we solve equation \( A = \lambda_1 \cdot d_{emin} + \lambda_2 \cdot d_{gmin} \) for \( (\lambda_1, \lambda_2) \), we get \( (\lambda_1, \lambda_2) = (\frac{1}{12}, \frac{1}{12}) \Rightarrow A \in Con(d_{emin}, d_{gmin}) \). Similarly, for point \( B \) we get \( (\lambda_1, \lambda_2) = (\frac{3}{4}, \frac{1}{4}) \Rightarrow B \in Y \)-cone. Finally, for point \( C \) we get \( (\lambda_1, \lambda_2) = (\frac{5}{12}, \frac{7}{12}) \Rightarrow C \in X \)-cone. Note also that \( A \in \mathcal{R}_3 \), as \( \frac{1}{12} + \frac{1}{12} = 3 \), while \( B \in \mathcal{R}_1 \), as \( \frac{3}{4} + \frac{1}{4} = 2 \).

The basic idea behind our algorithm is the following: if point \( j \) belongs in region \( \mathcal{R}_i \), then \( j \) is scheduled for execution at moment \( i \). In order to find out when \( j \) is to be executed, we must first calculate in which region \( j \) belongs. The first step is to determine in which cone \( j \) belongs; e.g., if it belongs to either \( X \) or \( Y \), or \( Con(c_i, c_{i+1}) \) for a pair of cone vectors. This is done by solving the equations \( j = \lambda_1 \cdot c_i + \lambda_2 \cdot c_{i+1}, 1 \leq i \leq r - 1 \), and checking the sign of \( \lambda_1 \) and \( \lambda_2 \) (for details see Algorithm II).

After that, we simply calculate in which specific region \( j \) belongs, using the following expressions, as shown in Figure 4:
The computation of the cone vectors, is the major part of those points that satisfy the following condition:

\[
\mathcal{M}(\mathbf{d}_1, \ldots, \mathbf{d}_m).
\]

Algorithm I (Cone-vector computation)

DivideConeVector \((PV, \mathbf{d}_{x_{\text{min}}}, \mathbf{d}_{y_{\text{min}}})\)

**INPUT:** algorithm pattern-vectors \(PV = (\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k)\)

**OUTPUT:** the \(r\) cone vectors \(CV = (c_1, c_2, \ldots, c_r), r \leq k \leq m\)

1. Let \(CV' = PV - \{\mathbf{d}_{x_{\text{min}}}, \mathbf{d}_{y_{\text{min}}}\}\) (the line that connects the endpoints of \(\mathbf{d}_{x_{\text{min}}}\) and \(\mathbf{d}_{y_{\text{min}}}\)).
2. Discard from \(CV'\) all vectors that their endpoints lie above line \(\mathbf{d}_{x_{\text{min}}}, \mathbf{d}_{y_{\text{min}}}\).
3. If \(CV' = \emptyset\), then return \((\mathbf{d}_{x_{\text{min}}}, \mathbf{d}_{y_{\text{min}}})).\)
4. Else if \(CV' = \{c\}\), then return \((\mathbf{d}_{x_{\text{min}}}, \mathbf{c}, \mathbf{d}_{y_{\text{min}}})).\)
5. Else, let \(s = (d_{y_{\text{min}}}, d_{x_{\text{min}}})/(d_{y_{\text{min}}}, d_{x_{\text{min}}})\); let \(\mathbf{p}_c = (p_{x_c}, p_{y_c}) \in PV\) a vector that minimizes the expression \(p_{y_c} - s \cdot p_{x_c}\) (this point is called furthest point).
6. Let \(CV'_c = \{c \in CV' : c_x < p_{x_c} \land c \in \{\mathbf{d}_{x_{\text{min}}}, \mathbf{p}_c\}\}\) and \(CV'_c = \{c \in CV' : c_x > p_{x_c} \land c \in \{\mathbf{p}_c, \mathbf{d}_{y_{\text{min}}}\}\}.
7. Return the concatenation of 
\[
\text{DivideConeVector}(\{\mathbf{d}_{x_{\text{min}}, CV'_c}, \mathbf{d}_{x_{\text{min}}}, \mathbf{p}_c\}) \quad \text{and} \quad \text{DivideConeVector}(\{CV'_c, \mathbf{d}_{y_{\text{min}}}, \mathbf{p}_c\}).
\]

The above algorithm follows the divide-and-conquer technique. The complexity achieved is \(O(m \cdot \log r)\), where \(m\) is the number of dependence vectors and \(r\) the number of cone vectors (usually quite smaller than \(m\)).

Now that the cone vectors are computed, the \(i\)-step region can be also computed quite efficiently. Starting from the origin, we apply all cone-vectors. By connecting their endpoints, the initial region \(R_1\) is formed. By applying them twice and connecting their endpoints, the step 2 region \(R_2\) is formed and so on. The procedure is shown in Figure 3.

3.2 Scheduling

The index space of a given nested loop algorithm is partitioned into cones and regions, by calculating the cone-vectors of the algorithm. Given an index point \(j\), we first find the cone \(Con(c_k, c_{k+1})\) where \(j\) belongs, and then region \(Con_1(c_k, c_{k+1})\), where \(j\) belongs. Finally, we schedule \(j\) to be executed at moment \(i\). This is done via the following algorithm.

Algorithm II (Scheduling in a Multi-Cone Space)

**INPUT:** \(DV = \langle \mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_m \rangle\) and an index point \(j\).

**OUTPUT:** the time instance \(t_j\) that \(j\) can be executed.

1. **Algorithm I** is applied on \(DV\) and pattern vector sequence \(PV\), as well as \(\mathbf{d}_{x_{\text{min}}}, \mathbf{d}_{y_{\text{min}}}\), are computed.
2. The \(r\) cone vectors are computed by applying **Algorithm I** onto \(PV\). After this step: \(CV = (c_1, c_2, \ldots, c_r)\), where \(c_1 = \mathbf{d}_{x_{\text{min}}}\) and \(c_r = \mathbf{d}_{y_{\text{min}}}\).
3. Find out at which cone our point \(j\) is located: for every cone-vector pair \((c_i, c_{i+1})\) of subsequent vectors \(c_i, c_{i+1} \in CV\), solve the equation: \(j = \lambda_{i+1} \cdot c_i + \lambda_{i+2} \cdot c_{i+1}\) and compute \(\lambda_{i+1}\) and \(\lambda_{i+2}\).
4 Pattern Formation in UCT Problems

Up to this point, we have assumed that any interprocessor communication action needs zero time and only computations take non-zero time (unit execution time problems - UET). Now we consider a more realistic model, where communication between different processors requires one unit of time, the so called UET-UCT model.

The main difference with the UET model is the communication overhead that is imposed by any interprocessor communication. This means that two index points can be computed at successive time instances, only if they are executed at the same processor; in any other case, an extra time instance is needed for communication.

In UET-UCT problems there are certain sequences of points that are computed by the same processor. Each such sequence can be viewed as a chain of computations that is created by a certain dependence vector, as shown in Figure 5. We call this dependence vector the communication vector and denote it as \( \mathbf{d}_c \). A unit time overhead is imposed by communication among all dependence vectors, except \( \mathbf{d}_c \).

In the following sections, the methods presented for the problem of pattern prediction, as well as for scheduling a nested loop, according to the UET model, will be revisited, taking into account the communication delay (UET-UCT model).

4.1 UET-UCT Model Definitions

The main notion in the UET-UCT model, is the communication overhead that is imposed by any interprocessor communication. This means that all dependence vectors, except \( \mathbf{d}_c \), impose a unit time overhead. So, the time subset \( ECT_t \), of points with \( ECT \) \( t \), for the UET-UCT problems, are recursively defined as follows:

- \( ECT_0 = \{ \text{boundary precomputed points} \} \)
- \( ECT_1 = \{ j \in J \mid j - \mathbf{d}_1 \in ECT_0, \forall \mathbf{d}_1 \in DV \} \)
- \( ECT_t = \{ j \in J \mid j - \mathbf{d}_t \in ECT_{t-1}, \text{for every } \mathbf{d}_t \neq \mathbf{d}_c \text{ and } j - \mathbf{d}_c \in ECT_{t-1} \}, \text{where } t \geq 2 \).

4.2 General Pattern Formation

We are searching for a formal expression of the outline of the \( t \)-step pattern. By the definition of \( ECT_1 \), any point \( \mathbf{p} \in pat_1 \) is given from the following expression (quite similar to UET case):

\[ \mathbf{p} = i_1 \cdot \mathbf{p}_1 + i_2 \cdot \mathbf{p}_2 + \ldots + i_k \cdot \mathbf{p}_k \]

instances \((2, 4, \ldots)\).

The notion behind the above definition is that: \( ECT_t \) is generated by \( ECT_{t-1} \) points, according to dependence vector \( \mathbf{d}_c \), and by \( ECT_{t-2} \), according to any other dependence vector. The extra time instance denotes the communication overhead (a unit of time). Some more definitions follow:

Given the sequence \( DV = \{ \mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_m \} \) of dependence vectors, we get pattern vectors by applying \( MC \) operator:

\[ PV = MC(DV) = \{ \mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k \} \]

Consequently, for every vector \( \mathbf{p}_i = (p_{ix}, p_{iy}) \in MC(DV) \) there isn’t any vector \( \mathbf{d}_j = (d_{jx}, d_{jy}) \in DV \) such that \( d_{jx} \leq p_{ix} \) and \( d_{jy} < p_{iy}, 1 \leq i \leq k \) and \( 1 \leq j \leq m \).

\[ PV = MC(DV) = \{ \mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k \} \]

\[ DV = \{ \mathbf{d}_1, \ldots, \mathbf{d}_{t-1}, \mathbf{d}_{t+1}, \ldots, \mathbf{d}_m \} \]

\[ DV_{UET} = \{ \mathbf{d}_1, \ldots, \mathbf{d}_{t-1}, 2\mathbf{d}_c, \mathbf{d}_{t+1}, \ldots, \mathbf{d}_m \} \]

Note that \( PV \) includes all pattern vectors, \( DV \) includes all pattern vectors except \( \mathbf{d}_c \), while \( DV_{UET} = DV \cup 2\mathbf{d}_c \).
Lemma 4.1 can be found in [10].

The above theorem proves that the wavefront of the execution propagation, can be expressed formally, in the UET-UCT case. Furthermore, the pattern outline of the odd time instances, is identical to the outline of a UET algorithm with slightly changed dependence vector set: \( DV_{\text{UET}} = (d_1, \ldots, d_{e}, 2d_l, d_{e+1}, \ldots, d_m) \). Instead of the communication vector \( d_e \), it contains \( 2d_l \).

For a given UET-UCT problem, the pure UET problem with the same \( PV \), except that includes \( 2d_l \), instead of \( d_e \), will be called equivalent UET algorithm.

### 4.3 Scheduling UET-UCT Loops

Due to the previous theorem, we get an equivalence of any UET-UCT problem, with a pure UET one, with slightly changed dependence vector set. This is clearly shown in Figure 7. A UET-UCT algorithm is concerned, and the wavefront of the execution is shown, while considering as \( d_e \), the noted dependence vector. Right besides, we show the execution wavefront of the UET algorithm that contains all dependence vectors except \( d_e \), plus \( 2d_l \).

The equivalence of \( 2i \)-th outline of the UET-UCT execution wavefront, with the \( i \)-th outline of the UET one, where \( i = 1, 2, 3, \ldots \) This equivalence denotes that we can schedule any UET-UCT problem, according to the equivalent UET one. By considering the partitioning of the index space into regions, we schedule each point according to the region it belongs.

The only difference is that the calculated time instance \( t \), of the equivalent UET problem, refers to \( i = 2t \) and \( i' = 2t + 1 \) time instances of the original UET-UCT problem. So, the scheduling will be performed at three faces instead of two:

- specify the region \( t \) that any given point belongs, in the equivalent UET problem,
- specify the actual time instance \( i = 2t \) or \( i' = 2t + 1 \) that the point should be scheduled at, by performing an additional test, and then
- schedule the given point at time \( i \) or \( i' \) according to the previous test.

What we need, is an additional test that distinguishes an index point \( j \) that belongs to the odd time instance \( (2t + 1) \) from another \( j' \) that belongs to the even one \( (2t) \), given the fact that \( j \) and \( j' \) belong to the region of the equivalent UET problem, that defined above. Using the results of Theorem 4.1, we can express \( \text{pat}_{2t+1} \) in relation to \( \text{pat}_{2t} \) and \( \text{pat}_{2t+2} \), only. It is:

\[
\text{pat}_{2t+1} = \mathcal{MC}(\text{pat}_{2t} + PV) = \\
= \mathcal{MC}(\text{pat}_{2t} + DV_{\text{UET}}) \\
= \mathcal{MC}(\text{pat}_{2t} + D V_{\text{UET}} + (d_1, \ldots, d_{e-1}, 2d_l, d_{e+1}, \ldots, d_m)) \\
= \mathcal{MC}(\text{pat}_{2t} + DV_{\text{UET}} + (d_1, \ldots, d_{e-1}, 2d_l, d_{e+1}, \ldots, d_m)) \\
= \mathcal{MC}(\text{pat}_{2t} + DV_{\text{UET}} + (d_1, \ldots, d_{e-1}, 2d_l, d_{e+1}, \ldots, d_m))
\]
This means that if \( j \) belongs to the \( t \) region in the equivalent UET problem, then \( ECT(j) = 2t + 1 \), if and only if \( j - d_i \) belongs to the \( t \) region too. If this is not the case, then \( ECT(j) = 2t \). This fact is illustrated in Figure 8.

The following algorithm is a revision of Algorithm II that works in UET-UCT problems. It implements the procedure that was just described. The index space is partitioned into cones and regions and point location is performed. After that, the additional test is performed, in order to specify the exact execution instance for each point.

Algorithm III (Scheduling a UET-UCT Loop)

**Input:** The dependence vector set \( DV = \{d_1, \ldots, d_m\} \) of the algorithm, communication vector \( d_c \) and an index point \( j \).

**Output:** The time instance \( t_j \) that point \( j \) can be executed.

1. Communication vector \( d_c \) is extracted from \( DV \) and \( 2d_c \) is applied instead; a new dependence set is formed: \( DV_{UET} = \{d_1, \ldots, d_{m-1}, 2d_c, d_{m+1}, \ldots, d_m\} \).
2. Algorithm II is applied on \( DV_{UET} \) and point \( j \); time instance \( t_j \) is computed (\( t_j \) is the execution time in the equivalent UET problem). The candidate execution instances for point \( j \) are \( 2t_j - 1 \) and \( 2t_j \).
3. Algorithm II is applied now on \( DV_{UET} \) and point \( j - d_c \); time instance \( t_{j-d_c} \) is computed.
   - if \( t_{j-d_c} = t_j \Rightarrow \) point \( j \) is executed (at the initial UET-UCT problem) at \( t_j = 2t_j \).
   - if \( t_{j-d_c} \neq t_j \Rightarrow \) point \( j \) is executed at \( t_j = 2t_j - 1 \).

The above algorithm is explained in Figure 8. Regions are named by even numbers only, as they are the boundary of even time instance pattern outlines. Points \( B \) and \( D \) are located, using Algorithm II, into region 6. In other words, the candidate time instances for their execution are 5 and 6. However, point \( A = B - d_c \) resides into previous region, while point \( D = C - d_c \) resides in current region. So, point \( C \) will be executed at time instance \( t_C = 5 \) and point \( B \) at time instance \( t_B = 6 \). The intuition behind that “movement” of \( pat_{2h} \) along \( d_c \) is that, although \( d_c \) imposes no...
4.4 Validity and Optimality

As far as algorithm’s execution optimality is concerned, the points with even execution time are delayed at most by 1 time instance, while the points with odd execution time are delayed at most by 2 (points E and F in Figure 8 correspondingly). Hence, our scheduling algorithm is asymptotically optimal, with a constant delay of at most 2.

4.5 Generalization to Higher Dimensions

Although our method has been described for two dimensional problems, it can be generalized for n dimensions. For the transformation of UET-UCT problems to UET ones, a similar method can be described for the n-dimensional cases, considering the hypersurfaces, formed by the end-points of the dependence vectors, as pattern boundaries, leading to the same conclusion.

For three dimensional algorithms, the dependence vectors that are initially concerned as cone-vectors, are the ones that have the maximum x, y and z-coordinate, respectively. These vector end-points form a plane (a line in 2-D, a hyperplane in n dimensions). Cone-vector computation can continue incrementally, as before, by considering the distances of all pattern vector end-points from the plane mentioned. The procedure will continue incrementally as above.

5 Conclusion

In this paper we presented a nearly optimal scheduling method for uniform dependence nested loop algorithms, considering the general UET-UCT model. We described an exponential procedure for computing the precise k-step pattern boundary, which is the best we can get so far. A nearly optimal geometric scheduling algorithm for the general UET-UCT model is proposed. Finally, we presented a new method for transforming a UET-UCT problem to a UET equivalent one. This last method is a major improvement, as it is the first method presented do far.

Our method has been implemented and tested in numerous examples and can be generalized to handle an arbitrary n-dimensional uniform dependence loop.

References