Optimal Scheduling for UET-UCT Generalized n-Dimensional Grid Task Graphs†

Theodore Andronikos, Nectarios Koziris,
George Papakonstantinou and Panayiotis Tsanakas

National Technical University of Athens
Dept. of Electrical and Computer Engineering
Computer Science Division
Zografou Campus, Zografou 15773, Greece

e-mail: {tedandr, nkoziris, papakon}@dsclab.ece.ntua.gr

Abstract

The n-dimensional grid is one of the most representative patterns of data flow in parallel computation. The most frequently used scheduling models for grids is the unit execution - unit communication time (UET-UCT). In this paper we enhance the model of n-dimensional grid by adding extra diagonal edges. First, we calculate the optimal makespan for the generalized UET-UCT grid topology and, then, we establish the minimum number of processors required, to achieve the optimal makespan. Furthermore, we solve the scheduling problem for generalized n-dimensional grids by proposing an optimal time and space scheduling strategy. We thus prove that UET-UCT scheduling of generalized n-dimensional grids is low complexity tractable.

1. Introduction

Task scheduling is one of the most important and difficult problems in parallel systems. Since the general scheduling problem is known to be NP-complete (see Ullman [13]), researchers have given attention to other methods such as heuristics, approximation algorithms etc. In their paper Papadimitriou and Yannakakis [10] proved the intractability of the general scheduling problem of a task graph with arbitrary communication and computation times and proposed a clever heuristic with guaranteed worst performance twice the optimum makespan. In addition to this, Gerasoulis and Yang have proposed in [9], [14] the Dominant Sequence Clustering, a low complexity heuristic for general task graph scheduling, which is based on the critical path of tasks. On the other hand, by restricting the general scheduling problem to instances with simple properties, we may come up with tractable solutions. For example, Jung, Kirousis et al. in [7] have presented a polynomial algorithm which finds the optimal makespan when the communication cost is constant and task duplication is allowed.

When considering UET cases, Andronikos et al. in [1], have given a polynomial time optimal scheduling and mapping into systolic architectures, where, due to the special hardware, communication need not be taken into consideration. In addition to this, in UET scheduling of arbitrary task graphs, which are produced by general nested loops, Koziris et al in [8] have given a polynomial time schedule and polynomial time efficient mapping onto a clique of processors, based on PERT techniques.

When both computation and communication times are restricted to have unit time length, it is known that scheduling UET-UCT graphs with bounded number of processors is NP-complete as Rayward-Smith proved in [12] or Picouleau in [11] by reduction from the unbounded UET-UCT instance. Even the case of unlimited processors, when no task duplication is allowed, is in general polynomially intractable [10]. On the other hand, using task duplication, Colin et Chretienne in [6] have presented a polynomial optimal schedule for arbitrary task graphs with UET and SCT (Small Communication Times, thus including UCT). Since the arbitrary task graph scheduling with UET- UCT and no duplication with unlimited processors is NP-complete, researchers have focused on special cases of DAGs. In [4] Chretienne presented an algorithm linear in the cardinality

† Partially supported by the Greek Secretariat of Research and Technology (GSRT) under the PENED/1405 project.

of the vertices of the graph, for optimal makespan on SCT in-trees and out-trees (thus covering UCT). In addition to this, there exist polynomial optimal solutions for Series-Parallel digraphs, bipartite graphs and trees with UCT as surveyed in [5].

This paper solves the problem of UET-UCT scheduling for task graphs having the form of a grid on unbounded number of processors, assuming no duplication. Grids and particularly generalized grids are typical task graphs, which model most of the signal processing algorithms and linear algebra methods such as matrix multiplication, LU decomposition etc. We extend the simple grid model of [2] by considering generalized n-dimensional grids. We prove that the time and space scheduling problem for generalized grid is low complexity tractable. We calculate the *optimal makespan* for UET-UCT grids. Having established the optimal makespan, we calculate the *optimal number of processors*, i.e., the minimum number of processors required to achieve the optimal makespan. We present an optimal time and space scheduling policy for UET-UCT grids. Our schedule partitions the vertices of the grid into disjoint sets that lie on a family of parallel hyperplanes. Each hyperplane contains vertices, which are executed on different processors at the same time. The hyperplanes are defined differently in each case due to the fact that in the UET/UCT case we must also consider the communication overhead. When communication is taken into account, the maximal coordinate of the grid determines the form of the optimal hyperplane. In addition to the above, we provide the reader with two scheduling algorithms which calculate the exact time instant and processor number, where an arbitrary vertex of the grid is to be executed under an optimal schedule. The time complexity of these algorithms is independent of the grid size and depends only on the dimension n of the grid. Thus they outperform all previously known PERT or CPM techniques for UET/UET-UCT scheduling of general task graphs [5]. Since we exploit the regularity of the grid task graph, we need not navigate through the graph in polynomial time to calculate the properties of a task node. We calculate for any task, in constant time, given its coordinates, the exact execution time, under an optimal schedule, and the total number of adequate processors.

The paper is organized as follows: In Section 2 we give the notation and some definitions and in Section 3 we present the UET-UCT optimal scheduling strategy for n-dimensional generalized grids. Finally in Section 4 we establish the minimum number of processors adequate for scheduling UET-UCT grids and we present an illustrative example of a 3-D grid.

2. Generalized Grids

In this section we introduce the concept of generalized grid, which is an enhancement of the usual grid model.

2.1. Notation

In the rest of the paper the following notation is used:

- N is the set of naturals.
- n is the dimension of the grid.
- G_{P_n} is the n-dimensional grid with *terminal point* $P_n=(u_1, \ldots, u_n)$.
- GVS is the *grid vector set*, i.e., the set of the vectors of the grid.

2.2. Basic Concepts

Definition 2.1. The *initial segment* of N^n with *terminal point* $P_n=(u_1, \ldots, u_n)\in N^n$, denoted $N_0(P_n)$, is the set $\{(k_1, \ldots, k_n)\in N^n \mid 0\le k_i\le u_i,\ 1\le i\le n\}$. ■

Definition 2.2. Let $\mathbf{e}_i$ be $(\underbrace{0,\ldots,0}_{i-1},1,\underbrace{0,\ldots,0}_{n-i})$, $1\le i\le n$. The *grid vector set*, denoted GVS, is the set $\{\mathbf{d}=(d_1, \ldots, d_n)\in N^n \mid \mathbf{d}=\lambda_1\mathbf{e}_1+ \ldots +\lambda_n\mathbf{e}_n$, where $\sum_{i=1}^{n}\lambda_i>0$ and $\lambda_i\in\{0, 1\}, 1\le i\le n\}$. Given a $\mathbf{d}\in$ GVS, *support*($\mathbf{d}$) is the number of non zero coordinates of $\mathbf{d}$. ■

Definition 2.3. The generalized n-dimensional grid with terminal point P_n, denoted G_{P_n}, is the DAG with vertices the set $N_0(P_n)$ and directed edges the set $\{(\mathbf{i}, \mathbf{j})\in (N_0(P_n))^2 \mid \mathbf{j}=\mathbf{i}+\mathbf{d}, \mathbf{d}\in \text{GVS}\}$. ■

The following properties hold:

(1) All the n coordinates of the vectors of the GVS are either 0 or 1. Naturally, the $\mathbf{0}$ vector has been excluded, which means that support($\mathbf{d}$)$\ge$1 $\forall\mathbf{d}\in$ GVS. The vectors that have exactly one coordinate 1 and all other 0 are the n unitary vectors $\mathbf{e}_i$, $1\le i\le n$.

(2) It is trivial to see that $|\text{GVS}|=\sum_{i=1}^{n}\binom{n}{i}=2^n-1$.

Definition 2.4. For every vertex $\mathbf{j}$ of a grid G_{P_n}, we define the following sets:

(1) IN($\mathbf{j}$) = $\{\mathbf{i}\in N_0(P_n) \mid \mathbf{j}=\mathbf{i}+\mathbf{d}$, where $\mathbf{d}\in$ GVS$\}$, and

(2) OUT($\mathbf{j}$) = $\{\mathbf{i}\in N_0(P_n) \mid \mathbf{i}=\mathbf{j}+\mathbf{d}$, where $\mathbf{d}\in$ GVS$\}$. ■

The directed edges of a grid induce a partial ordering over the vertices in a natural way. If $\mathbf{i}$ and $\mathbf{j}$ are two vertices of a grid, we write $\mathbf{i}<\mathbf{j}$ iff $\exists\mathbf{d}_1\ldots\exists\mathbf{d}_k\in$ GVS such that $\mathbf{j}=\mathbf{i}+\mathbf{d}_1+ \ldots +\mathbf{d}_k$.

The intuition behind the partial ordering notion is that the edges represent *precedence constraints* that have to be satisfied in order to correctly complete the tasks represented by the vertices. The formal definition of the

schedule must reflect our intuition that a vertex **j** correctly begins its execution at instant k iff *all the vertices* **i**∈IN(**j**) *have completed their execution and communicated their results (if needed) to* **j** *by that instant.*

Definition 2.5.

- A schedule for the grid G_{P_n}, denoted $S(G_{P_n})$, is an ordered couple (S_{TIME}, S_{PROC}), where S_{TIME} and S_{PROC} are the time and processor schedules, respectively, defined as follows:

(1) S_{PROC}: $N_0(P_n)\rightarrow\{0, \ldots, m\}$, m∈N, such that task **j** is assigned to processor S_{PROC}(**j**), and

(2) S_{TIME}: $N_0(P_n)\rightarrow N$ such that $\forall \mathbf{j}\in N_0(P_n)\forall \mathbf{i}\in IN(\mathbf{j})$ $S_{TIME}(\mathbf{j})-S_{TIME}(\mathbf{i}) \geq \begin{cases} p, & \text{if } S_{PROC}(\mathbf{i}) = S_{PROC}(\mathbf{j}) \\ p+c, & \text{if } S_{PROC}(\mathbf{i}) \neq S_{PROC}(\mathbf{j}) \end{cases}$, where p is the processing time and c the communication delay.

- The *makespan* of a time schedule S_{TIME} for the grid G_{P_n}, denoted $M(S_{TIME})$, is $\max\{S_{TIME}(\mathbf{j})+p \mid \mathbf{j}\in N_0(P_n)\}$, where p is the processing time.
- Given the schedule $S(G_{P_n})=(S_{TIME}, S_{PROC})$, $N_{PROC} = \max\{|\{\mathbf{j}\in N_0(P_n) : S_{TIME}(\mathbf{j})=k\}| : 0\leq k\leq M(S_{TIME})\}$. ■

The makespan gives the completion time of the last task and, therefore, determines the time required for the completion of the whole grid. N_{PROC} gives the maximum number of processors required by the specific schedule.

In case of UET we assume p=1, c=0, and in case of UET/UCT we assume p=c=1.

In this paper, our objectives are:

(1) To find an *optimal time schedule* $S_{TIME_{OPT}}$, i.e., a schedule whose makespan is *minimum*.

(2) To establish the *optimal number of processors* $N_{PROC_{OPT}}$, i.e., the minimum number of processors required to execute an optimal time schedule.

(3) To find an *optimal space schedule* $S_{PROC_{OPT}}$ that realizes $S_{TIME_{OPT}}$ using $N_{PROC_{OPT}}$ processors.

A schedule ($S_{TIME_{OPT}}$, $S_{PROC_{OPT}}$) is called *optimal* and is denoted $S_{OPT}(G_{P_n})$.

Theorem 2.1. For every grid G_{P_n} and every time schedule S_{TIME} we have:

$M(S_{TIME}) = S_{TIME}(P_n)+p$, where p is the processing time.

Proof

First, notice that $S_{TIME}(P_n)>S_{TIME}(\mathbf{j})$, $\forall \mathbf{j}\in N_0(P_n)-\{P_n\}$ for every time schedule S_{TIME}. To prove that, let us assume to the contrary that for some $\mathbf{j}\neq P_n$ we have $S_{TIME}(\mathbf{j})\geq S_{TIME}(P_n)$. However, if $\mathbf{j}\neq P_n$ then $\mathbf{j}<P_n$ and from Definition 2.5 we derive $S_{TIME}(P_n)>S_{TIME}(\mathbf{j})$, which is a contradiction. It must, therefore, be the case that $S_{TIME}(P_n) =\max\{S_{TIME}(\mathbf{j}) \mid \mathbf{j}\in N_0(P_n)\}$, i.e., $M(S_{TIME})=S_{TIME}(P_n)+p$, where p is the processing time. ❒

By Definition 2.5, an optimal time schedule $S_{TIME_{OPT}}$ achieves the minimum makespan. Theorem 2.1 asserts that the makespan is the execution time of the terminal point P_n. Thus, an optimal time schedule $S_{TIME_{OPT}}$ schedules P_n to be executed at the least possible time. In what follows, we first establish the least possible time at which P_n can be executed and then, the scheduling policy, which organizes the execution of the other nodes so as to achieve the optimal execution time for P_n.

3. Optimal Parallel Time for UET-UCT

In this section, we shall study the generalized UET-UCT grids. We partition the vertices of a grid into parallel hyperplanes. However, the presence of communication delays imposes further difficulties, which differentiate the equation of the optimal hyperplane from the simple UET case. We prove that under an optimal scheduling policy, all vertices belonging to the same hyperplane are executed at the same time instant.

Example 3.1. Given an arbitrary **j** which is executed at time k, consider the set OUT(**j**). Under any optimal scheduling strategy, *at most one* **i**∈OUT(**j**) will be executed at time k+1 and all others at later time instants. Obviously, the issue here is the selection of the **i** that will lead to the optimal makespan. It will be shown that the maximal coordinate of the grid determines this selection.

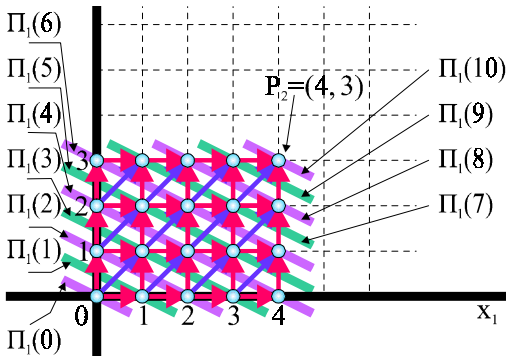

Fig. 1: An optimal schedule for G_{P_2}.

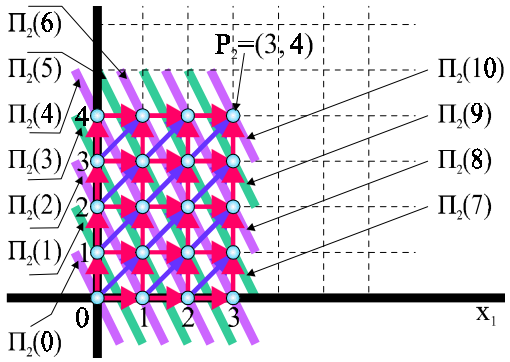

Fig. 2: An optimal schedule for G_{P_2}.

In the UET-UCT case, due to the presence of communication delays, there is no unique family of hyperplanes that are optimal for every grid. Instead, the family of optimal hyperplanes depends on the maximal coordinate of the terminal point. For the grid G_{P_2} (see Fig. 1) the optimal hyperplanes are $x_1+2x_2=k$, $0\le k\le 10$, whereas for the grid $G_{P_2'}$ (see Fig. 2) the optimal hyperplanes are $2x_1+x_2=k$, $0\le k\le 10$. ❒

Definition 3.1. Given the grid G_{P_n}, $\Pi_i(k)$, $1\le i\le n$, is the set $\{(k_1, \ldots, k_n)\in N_0(P_n) \mid 2(x_1+ \ldots +x_{i-1}+x_{i+1}+ \ldots +x_n)+x_i=k,\ k\in N\}$. ■

Geometrically, $\Pi_i(k)$ consists of the common points of the grid $G(P_n)$ and the n-1 dimensional hyperplane $2(x_1+ \ldots +x_{i-1}+x_{i+1}+ \ldots +x_n)+x_i=k$.

Lemma 3.1. For every grid G_{P_n}, with $P_n=(u_1, \ldots, u_n)$, the following hold:

(1) $\Pi_i(0) = \{\mathbf{0}\}$ and $\Pi_i(2u_1+ \ldots +2u_{i-1}+u_i+2u_{i+1}+ \ldots +2u_n) = \{P_n\}$,

(2) $\Pi_i(k) \neq \emptyset$, $0\le k\le 2(u_1+\ldots+u_{i-1}+u_{i+1}+\ldots+u_n)+u_i$,

(3) $\Pi_i(k) = \emptyset$, when $k>2(u_1+\ldots+u_{i-1}+u_{i+1}+\ldots+u_n)+u_i$,

(4) $\forall \mathbf{j}\in\Pi_i(k)\exists r(1\le r\le n)$ $\mathbf{j}+\mathbf{e}_r\in N_0(P_n)$, $0\le k<2(u_1+ \ldots +u_{i-1}+u_{i+1}+ \ldots +u_n)+u_i$. ❒

Lemma 3.2 presents the relation between vertices belonging to successive hyperplanes $\Pi_i(k)$, $\Pi_i(k+1)$ and $\Pi_i(k+2)$. From any vertex **j** belonging to $\Pi_i(k+2)$, if we backtrack using the $\mathbf{e}_i$ vector, the resulting vertex will definitely belong to the previous $\Pi_i(k+1)$ plane. This property reflects the fact that all points connected with vector $\mathbf{e}_i$ will be executed on *successive* time steps by the *same* processor. Thus, the communication delay due to $\mathbf{e}_i$ is zero. On the other hand if we go back using any other $\mathbf{e}_r\neq\mathbf{e}_i$, the resulting vertex will be on the $\Pi_i(k)$ plane. In this case, the $\mathbf{e}_r$ edge imposes a unit communication delay. Finally, if we backtrack from any vertex **j** belonging to $\Pi_i(k+2)$, using an arbitrary vector $\mathbf{d}\neq\mathbf{e}_r$, $1\le r\le n$, the resulting vertex will belong to a previous hyperplane $\Pi_i(r)$, $r<k$.

Lemma 3.2. For every grid G_{P_n}, with $P_n=(u_1, \ldots, u_n)$, the following hold:

(1) If $\mathbf{j}\in\Pi_i(k+1)$ and $\mathbf{j}-\mathbf{e}_i\in N_0(P_n)$, then $\mathbf{j}-\mathbf{e}_i\in\Pi_i(k)$, $0\le k<2(u_1+ \ldots +u_{i-1}+u_{i+1}+ \ldots +u_n)+u_i$,

(2) If $\mathbf{j}\in\Pi_i(k+2)$ and $\mathbf{j}-\mathbf{e}_r\in N_0(P_n)$, then $\mathbf{j}-\mathbf{e}_r\in\Pi_i(k)$, $1\le r\neq i\le n$, $0\le k<2(u_1+ \ldots +u_{i-1}+u_{i+1}+ \ldots +u_n)+u_i-1$,

(3) $N_0(P_n)\cap(\bigcup_{\substack{r=1\\ r\neq i}}^{n}(\Pi_i(k+2)-\mathbf{e}_r)\cup(\Pi_i(k+1)-\mathbf{e}_i)) = \Pi_i(k)$,

$0\le k<2(u_1+ \ldots +u_{i-1}+u_{i+1}+ \ldots +u_n)+u_i-1$,

(4) $N_0(P_n)\cap \bigcup_{\substack{\mathbf{d}\in GVS\\ \mathbf{d}\neq\mathbf{e}_r}} (\Pi_i(k+2)-\mathbf{d}) \subseteq \bigcup_{r=k+3-2n}^{k-1} \Pi_i(r)$, $0\le k<2(u_1+ \ldots +u_{i-1}+u_{i+1}+ \ldots +u_n)+u_i-1$. ❒

Now that we have partitioned the grid into hyperplanes and established the relation among vertices belonging to successive hyperplanes, we can present the optimal scheduling policy based on this partitioning. The following lemma gives the least possible execution time for every vertex of the grid. Every vertex $\mathbf{j}=(k_1, \ldots, k_n)$ has a *maximal coordinate*, i.e., a coordinate k_i for which $k_i\ge k_r$, $1\le r\le n$. It can be proved that the earliest execution time of **j** is $2(k_1+ \ldots +k_{i-1}+k_{i+1}+ \ldots +k_n)+k_i$, where k_i is a maximal coordinate of **j**.

Lemma 3.3. Let G_{P_n} be a UET-UCT grid and let k_i be a maximal coordinate of vertex $\mathbf{j}=(k_1, \ldots, k_n)$. Then the earliest execution time of **j** is k if $\mathbf{j}\in\Pi_i(k)$. ❒

Now, we can establish the optimal execution time for any UET-UCT grid G_{P_n}.

Theorem 3.1. Let u_i be a maximal coordinate of the terminal point $P_n=(u_1, \ldots, u_n)$ of the UET-UCT grid G_{P_n}. Then $M(S_{TIME_{OPT}})=2(u_1+ \ldots +u_{i-1}+u_{i+1}+ \ldots +u_n)+u_i+1$.

Proof

We know that $M(S_{TIME_{OPT}})=S_{TIME_{OPT}}(P_n)+1$ (Theorem 2.1); consequently, Lemma 3.3 implies that $S_{TIME_{OPT}}(P_n) = 2(u_1+ \ldots +u_{i-1}+u_{i+1}+ \ldots +u_n)+u_i+1$. ❒

3.1. **Optimal Time Scheduling Policy**

In the UET-UCT case we can not execute all vertices of the grid at the earliest possible time if we want to achieve the optimal makespan. As a result, *the number of different optimal time schedules equals the number of maximal coordinates of the terminal point of the grid*. If u_i is a maximal coordinate of P_n, the following time schedule is optimal: $S_{TIME_{OPT}}(\mathbf{j})=2(k_1+ \ldots +k_{i-1}+k_{i+1}+ \ldots +k_n)+k_i$, where $\mathbf{j}=(k_1, \ldots, k_n)\in N_0(P_n)$.

Example 3.2. The optimal schedule for the grids G_{P_2} and G_{P_3} are depicted in Fig. 1 and Fig. 4, respectively. In order to achieve the optimal time schedule for the terminal point, we have to execute certain points at a later time. In G_{P_3} these points are of the form $k_2\mathbf{e}_2+k_3\mathbf{e}_3$, where $0\le k_2\le 2$ and $0\le k_3\le 1$ (see Fig. 3). The vertices with the same y and z coordinates (Fig. 3) must be executed on the same processor, which means that the optimal time schedule must execute the vertices of $\Pi_1(k)$ at instant k, $0\le k\le 9$ (see Fig. 4), i.e., we have $S_{TIME_{OPT}}(P_3)=9$. The optimal time schedule is given in Table 1; the row headings are the processors and the column headings are the execution times of the vertices. Note that this schedule is not optimal in terms of processors. ❒

4. **Optimal Number of Processors with UCT**

In this section we shall establish the optimal number of processors for UET-UCT grids, using the optimal time

scheduling policy of Section 3, which determines the least possible number of processors required at every instance k. In what follows, $\Pi_{i_{MAX}}$, $1 \le i \le n$, denotes the maximum value of $|\{\Pi_i(k)\}|$, $0 \le k \le 2(u_1 + \ldots + u_{i-1} + u_{i+1} + \ldots + u_n) + u_i$.

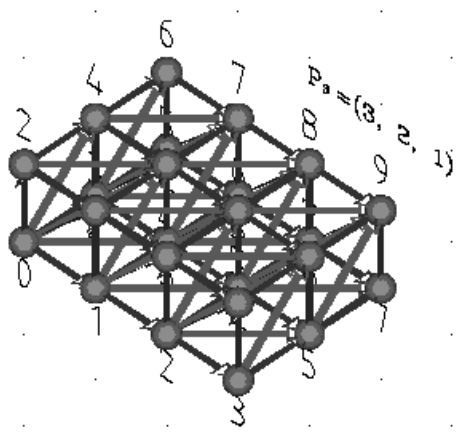

Fig. 3: Execution Time for G_{P_3}.

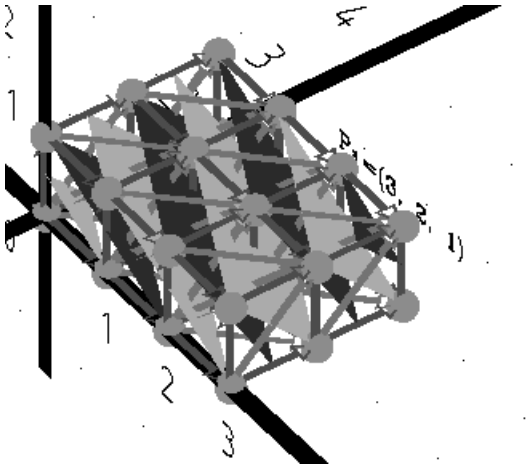

Fig. 4: An optimal scheduling of G_{P_3}.

Table 1: The optimal time schedule for G_{P_3}.

	0	1	2	3	4	5	6	7	8	9
P_0	(0,0,0)	(1,0,0)	(2,0,0)	(3,0,0)						
P_1			(0,0,1)	(1,0,1)	(2,0,1)	(3,0,1)				
P_2			(0,1,0)	(1,1,0)	(2,1,0)	(3,1,0)				
P_3					(0,1,1)	(1,1,1)	(2,1,1)	(3,1,1)		
P_4					(0,2,0)	(1,2,0)	(2,2,0)	(3,2,0)		
P_5							(0,2,1)	(1,2,1)	(2,2,1)	(3,2,1)

Definition 4.1. Given an optimal time schedule for the grid G_{P_n}, the *k-th concurrent antichain* of G_{P_n}, $0 \le k \le u_1 + \ldots + u_n$, denoted CA_k, is the set $\{\mathbf{j} \mid S_{TIME_{OPT}}(\mathbf{j}) = k\}$. A concurrent antichain is *maximal*, denoted CA_M, iff $|CA_M| \ge |CA_k|$, $0 \le k \le u_1 + \ldots + u_n$. ■

Lemma 4.1. For every UET-UCT grid G_{P_n}, with terminal point $P_n=(u_1, \ldots, u_n)$, $|\Pi_i(k)| = \binom{n + \left\lfloor \frac{k}{2} \right\rfloor - 1}{n-1} + \sum_{r=1}^{n} (-1)^r \sum \binom{n + \left\lfloor \frac{k}{2} \right\rfloor - (u_1' + 1) - \ldots - (u_r' + 1) - 1}{n-1}$, where $u_i' = \left\lfloor \frac{u_i}{2} \right\rfloor$ and $u_r' = u_r$, $1 \le r \ne i \le n$, $0 \le k \le 2(u_1 + \ldots + u_{i-1} + u_{i+1} + \ldots + u_n) + u_i$.

Proof

The cardinality of $\Pi_i(k)$ is equal to the number of integer solutions of the equation $2x_1 + \ldots + 2x_{i-1} + 2x_{i+1} + \ldots + 2x_n + x_i = k$, $0 \le k \le 2(u_1 + \ldots + u_{i-1} + u_{i+1} + \ldots + u_n) + u_i$ (1), such that $0 \le x_r < u_r + 1$. One can verify that this number is equal to the number of integer solutions of the equation $x_1' + \ldots + x_i' + \ldots + x_n' = \left\lfloor \frac{k}{2} \right\rfloor$, where $0 \le x_i < \left\lfloor \frac{u_i}{2} \right\rfloor + 1$ and $0 \le x_r < u_r + 1$, $1 \le r \ne i \le n$. This number is given by the following formula: $\binom{n + \left\lfloor \frac{k}{2} \right\rfloor - 1}{n-1} + \sum_{r=1}^{n} (-1)^r \sum \binom{n + \left\lfloor \frac{k}{2} \right\rfloor - (u_1' + 1) - \ldots - (u_r' + 1) - 1}{n-1}$. In the above formula, n is the dimension of the grid and the inner summation ranges over all r-combinations of n (see [3]). ❒

The following two corollaries follow immediately from Lemma 4.1.

Corollary 4.1. Let $P_n=(u_1, \ldots, u_n)$ be the terminal point of the grid G_{P_n} and let $u_i = u_r$, $1 \le r \ne i \le n$. Then $|\Pi_i(k)| = |\Pi_r(k)|$, $0 \le k \le 2(u_1 + \ldots + u_{i-1} + u_{i+1} + \ldots + u_n) + u_i$. ❒

Corollary 4.2. For every UET-UCT grid G_{P_n}, with terminal point $P_n=(u_1, \ldots, u_n)$, we have $\Pi_{i_{MAX}} = |\Pi_i(\left\lfloor \frac{2(u_1 + \ldots + u_{i-1} + u_{i+1} + \ldots + u_n) + u_i}{2} \right\rfloor)|$. ❒

Lemma 4.2. Let u_i be a maximal coordinate of the terminal point $P_n=(u_1, \ldots, u_n)$ of the grid G_{P_n}. Then $\Pi_i(k) = CA_k$, $0 \le k \le 2(u_1 + \ldots + u_{i-1} + u_{i+1} + \ldots + u_n) + u_i$.

From Section 3 we know that different maximal coordinates of the terminal point correspond to different optimal time schedules. If we combine this fact with Corollary 4.1, we deduce that any two CA_k and CA_k' corresponding to different optimal time schedules have the same cardinality. We can, therefore, proceed to state the following lemma.

Lemma 4.3. Let CA_{MAX} be a maximal concurrent antichain corresponding to an optimal time schedule for the grid G_{P_n}, with $P_n=(u_1, \ldots, u_n)$. Then $|CA_{MAX}| \le N_{PROC_{OPT}}$. ❒

Theorem 4.1. Let u_i be a maximal coordinate of the terminal point $P_n=(u_1, \ldots, u_n)$ of the UET-UCT grid G_{P_n}. Then $N_{PROC_{OPT}} = \Pi_{i_{MAX}}$. ❒

4.1. **Optimal Scheduling Policy**

Lemma 4.3 in conjunction with Lemma 4.1 gives a straightforward method for scheduling UET-UCT grids. From Section 3 we know that any optimal scheduling policy demands that $S_{TIME_{OPT}}(\mathbf{j})=k$ for every point $\mathbf{j}=(k_1, \ldots, k_n)\in \Pi_i(k)$, where $k=2(k_1+ \ldots +k_{i-1}+k_{i+1}+ \ldots +k_n)+k_i$ and u_i is a maximal coordinate of P_n. If $k_i\geq 1$, then **j** must be executed by the *same* processor that completed vertices $(k_1, \ldots, k_{i-1}, 0, k_{i+1}, \ldots, k_n)$, …, $(k_1, \ldots, k_{i-1}, k_i-1, k_{i+1}, \ldots, k_n)$. If $k_i=0$, then **j** can be executed by any of the available processors for the k time instant. The exact processor where **j** will be executed can be easily calculated. This gives us a size independent scheduling strategy for UET-UCT grids with unbounded number of processors. Obviously, the calculation of the optimal schedule for the whole grid is, inevitably, linear in the size of the grid.

Example 4.1. For G_{P_3}, $P_3=(3, 2, 1)$, we get $|\Pi_i(k)| = \left(\left\lfloor\frac{k}{2}\right\rfloor+2\right)-\left(2\left(\left\lfloor\frac{k}{2}\right\rfloor\right)+\left(\left\lfloor\frac{k}{2}\right\rfloor-1\right)\right)+\left(2\left(\left\lfloor\frac{k}{2}\right\rfloor-3\right)+\left(\left\lfloor\frac{k}{2}\right\rfloor-2\right)\right)-\left(\left\lfloor\frac{k}{2}\right\rfloor-5\right)$, $0\leq k\leq 9$ (see Table 2). Thus, $N_{PROC_{OPT}}=\Pi_{MAX}=4$. In Table 3, we present the optimal schedule; the row headings are the processors and the column headings are the execution times of the vertices. It is important to note that we reuse processing elements 0 and 1. ❒

Table 2: The cardinality of $\Pi_i(k)$ for G_{P_3}.

k	0	1	2	3	4	5	6	7	8	9
num of procs:	1	1	3	3	4	4	3	3	1	1

Table 3: The optimal UET-UCT schedule for G_{P_3}.

	0	1	2	3	4	5	6	7	8	9
P_0	(0,0,0)	(1,0,0)	(2,0,0)	(3,0,0)						
P_1			(0,0,1)	(1,0,1)	(2,0,1)	(3,0,1)				
P_2			(0,1,0)	(1,1,0)	(2,1,0)	(3,1,0)				
P_3					(0,1,1)	(1,1,1)	(2,1,1)	(3,1,1)		
P_0					(0,2,0)	(1,2,0)	(2,2,0)	(3,2,0)		
P_1							(0,2,1)	(1,2,1)	(2,2,1)	(3,2,1)

5. **Conclusion**

In this paper we have proved that UET-UCT scheduling of task graphs having the form of n-dimensional generalized grids is not only tractable but it also has a constant time solution. We have presented an optimal strategy for both time and space scheduling of these grids with and without communication delays. The proposed time strategy for UET-UCT graphs is proved to achieve the minimum thus optimal makespan. In addition to this, the exact lowest number of processors needed to execute a grid task graph with UET and unit communication delays is calculated. Our scheduling achieves the optimal makespan and the minimum number of processors in constant time. While the case of unbounded number of processors is solved, the case of bounded number of available processors remains open.

References

1. Andronikos, T., Koziris, N., Tsiatsoulis, Z., Papakonstantinou, G., and Tsanakas, P. Lower Time and Processor Bounds for Efficient Mapping of Uniform Dependence Algorithms into Systolic Arrays. To appear in *Journal of Parallel Algorithms and Applications*.
2. Bampis, E.,. Delorme, C., and Konig, J.C. Optimal Schedules for d-D Grid Graphs with Communication Delays. *Symposium on Theoretical Aspects of Computer Science (STACS96*). Grenoble France 1996.
3. Berman, G., and Fryer, K.D. *Introduction to Combinatorics*. Academic Press, New York, 1972.
4. Chretienne, P. A Polynomial algorithm to Optimally Schedule Tasks over a Virtual Distributed System under Tree-like Precedence Constraints. *Eur. J. Oper. Res.* 43, pp. 225-230, 1989.
5. Chretienne, P. and Picouleau C. Scheduling with Communication Delays: A Survey, *in* Scheduling Theory and its Applications *pp 65-90. John Wiley & Sons 1995*.
6. Colin, J. Y., and Chretienne, P. CPM Scheduling with Small Communication Delays and Task Duplication. *Oper. Res.* 39, pp. 680-684.
7. Jung, H., Kirousis, L., and Spirakis, P. Lower Bounds and Efficient Algorithms for multiprocessor Scheduling of DAGS with communication Delays. *Proceedings of 1st ACM SPAA 1989, pp. 254-264,* and *Information and Computation 105, pp 94-104, 1993.*
8. Koziris, N., Papakonstantinou, G., and Tsanakas, P. Optimal Time and Efficient Space Free Scheduling For Nested Loops. *The Computer Journal*. 39, 5, pp. 439-448, 1996.
9. Gerasoulis, A., and Yang, T. On the Granularity and Clustering of Directed Acyclic Task Graphs. *IEEE Trans. Parallel Distrib. Syst*. 4, 6, pp. 686-701, 1993.
10. Papadimitriou, C., and Yannakakis, M. Toward an Architecture-Independent Analysis of Parallel Algorithms. *SIAM J. Comput*. 19, pp. 322-328, 1990. *Extended Abstract in Proceedings STOC 1988.*
11. Picouleau, C. Etude de Problems d' Optimization dans les Systemes Distribues. These, Universite Pierre et Marie Curie, 1992.
12. Rayward-Smith, V.J. UET Scheduling with Unit Interprocessor Communication Delays and Unlimited Number of Processors. *Discrete Applied Mathematics*. 18, pp. 55-71, 1987.
13. Ullman, J. NP-Complete Scheduling problems. *Journal of Computer and Syst. Sciences*. 10, pp. 384-393, 1975.
14. T. Yang, T., and Gerasoulis, A. DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors. *IEEE Trans. Parallel Distrib. Syst*. 5, 9, pp. 951-967, 1994.