Optimal Loop Parallelization in n-Dimensional Index Spaces

Ioannis Drositis, Theodore Andronikos, Marios Kalathas,
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, mkalath, papakon}@cslab.ntua.gr

Abstract

This paper deals with the problem of scheduling uniform nested loops. The majority of the scheduling methods use linear transformations, while they consider no communication cost. The presented method is based on the geometric properties of the index space. It leads to a nearly optimal schedule and outperforms the existing scheduling techniques. It is the first loop-scheduling method that considers communication cost.

Keywords*: optimal scheduling, communication cost, UET, UET-UCT.*

1 Introduction

Scheduling nested loops with uniform dependencies was studied by Lamport, who partitioned the index space into hyperplanes [13]. The idea behind this is that all points that belong to the same hyperplane can be executed in parallel. Darte proved that this method is nearly optimal [7], while the problem of finding the hyperlane 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 [14], linear programming in subspaces [18], integer programming [7], or graph-theoretic approaches [3]. All these approaches consider unit execution time for each iteration and zero communication for each communication step (UET model).

One way to tackle the scheduling problem in the presence of communication delay, is to model the algorithm by a directed acyclic task 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 ([21], [10]), researchers have given attention to other methods such heuristics, approximation algorithms etc. Papadimitriou and Yanakakis proposed a clever heuristic which guaranteed worst performance twice the optimum makespan, for a task graph with arbitrary communication and computation times [15]. When there is unit time computation and communication, the model is known as UET-UCT.

In this paper we present a unified approach that solves both the UET and UET-UCT paradigm of uniform dependence loops. Our method uses concepts of computational geometry in order to partition the index space into regions and schedules computations according to the region they belong to. Furthermore, we reduce any UET-UCT problem to a UET equivalent one. This is one of the major contributions of our work, as no other relative method has been proposed so far.

The proposed UET-UCT scheduling algorithm is nearly optimal, as it achieves a makespan of constant delay from the earliest possible.

2 Terminology & Definitions

The uniform dependence loops have the following form:

```
for (i1=1; i1<=u1; i1++) {
    ...
    for (in=1; in<=un; in++) {
        S1(I);
        S2(I);
        ...
    }
```

. . .
}

where $S_1, \ldots, S_k$ are assignment statements and $J = \{(i_1, \ldots, i_n) \mid 1 \leq i_r \leq u_r, 1 \leq r \leq n\}$ is the index space ($J \subset \mathbb{N}^n$). $\mathbf{U} = (u_1, \ldots, u_n)$ is the terminal point of the algorithm. The dependence vectors are "uniform", i.e. constant all over the index space. Given the m dependence vector set $D = \{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_m\}$, we give the following definitions:

$ECT(\mathbf{j})$ is the *earliest computation time* of index point $\mathbf{j}$. The index space is partitioned into disjoint **time subsets** denoted ECT_i, $i \geq 0$, such that ECT_i contains the set of points of J whose earliest computation time (ECT) is i.

$\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$ are the unitary vectors along axes $x_1, \ldots, x_n$, respectively.

Cone: The convex subspace formed from q vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_q \in \mathbb{N}^n$ is called a *cone* and is denoted $Con(\mathbf{a}_1, \ldots, \mathbf{a}_q)$. Formally it is defined as: $Con(\mathbf{a}_1, \ldots, \mathbf{a}_q) = \{\mathbf{j} \in \mathbb{N}^n \mid \mathbf{j} = \lambda_1 \mathbf{a}_1 + \ldots + \lambda_q \mathbf{a}_q$, where $\lambda_1, \ldots, \lambda_q \geq 0\}$. Sometimes, we will just use Con_i for simplicity to denote cone i. We will consider two types of cones: the *non-trivial cones*, or just cones, and the *trivial cones*.

Non-Trivial Cone: A cone that is defined exclusively by dependence vectors is called non-trivial cone.

Trivial Cone: A cone that is defined by dependence vectors and at least one unitary axis vector is called trivial and is denoted $TrCon(\mathbf{a}_1, \ldots, \mathbf{a}_q, \mathbf{e}_s, \ldots, \mathbf{e}_t)$. Likewise, we use the notation $TrCon_j$ to denote trivial cone j.

Cone Set: The non-trivial and trivial cones together comprise the cone set CS. The dependence vectors and the unitary vectors along the axes may form different sets of non-trivial and trivial cones. The following instrumental properties capture the physical meaning of the partitioning we choose:

1. every index point $\mathbf{j} \in J$ belongs to a non-trivial or to a trivial cone
2. if $\mathbf{j} \in Con_i = Con(\mathbf{c}_p, \ldots, \mathbf{c}_q)$, then $\mathbf{j} = \lambda_p \mathbf{c}_p + \ldots + \lambda_q \mathbf{c}_q$, where $\lambda_p + \ldots + \lambda_q = maximum$.

Of course it is possible to express $\mathbf{j}$ as a linear combination of more than one subset of dependence vectors. However, the sum of the coefficients in any other combination will not exceed the aforementioned maximum.

Cone Vectors: The cone vectors $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_r$ are precisely those dependence vectors that form the cones of the cone set. Each dependence vector is not always a cone vector.

Facets: A polygon with $k \geq n$ vertices $\mathbf{a}_1, \ldots, \mathbf{a}_k$, with the extra property that any n of its vertices are coplanar but not colinear, is a facet f with k vertices, denoted $f = \langle \mathbf{a}_1, \ldots, \mathbf{a}_k \rangle$.

Inner Facets: We consider the set that consists of the dependence vectors plus the terminal point $\mathbf{U}$. As *inner facets* we define the facets of the convex hull of this set which don't contain the terminal point $\mathbf{U}$. The sequence of the inner facets is defined as $\mathcal{IF} = \langle if_1, if_2, \ldots \rangle$ (see Figure 1).

Outer Facets: We consider the set that consists of the cone vectors plus the origin of the axes $\mathbf{O}$. As *outer facets* we define the facets of the convex hull of this set that contain $\mathbf{O}$. The sequence of outer facets is defined as $\mathcal{OF} = \langle of_1, of_2, \ldots \rangle$.

Plane Normal Vector: Given n linearly independent points $\mathbf{p}_1, \ldots, \mathbf{p}_n \in J^n$, the normal vector to the facet that is defined by these points is denoted $\mathbf{n}(\mathbf{p}_1, \ldots, \mathbf{p}_n)$.

Cone Region: The i-th region of $Con(\mathbf{a}_1, \ldots, \mathbf{a}_q)$ is the set $Reg_i(\mathbf{a}_1, \ldots, \mathbf{a}_q) = \{\mathbf{j} \in Con(\mathbf{a}_1, \ldots, \mathbf{a}_q) \mid \mathbf{j} = \lambda_1 \mathbf{a}_1 + \ldots + \lambda_q \mathbf{a}_q$, where $i \leq \lambda_1 + \ldots + \lambda_q < i+1\}$.

Trivial Cone Region: Consider the trivial cone $TrCon_\ell = TrCon(\mathbf{a}_p, \ldots, \mathbf{a}_q, \mathbf{e}_s, \ldots, \mathbf{e}_t)$ and let $\mathbf{e} = \mathbf{e}_s + \ldots + \mathbf{e}_t$. The i-th region of $TrCon_\ell$ is the set $TrReg_i(\mathbf{a}_p, \ldots, \mathbf{a}_q, \mathbf{e}_s, \ldots, \mathbf{e}_t) = \{\mathbf{j} \in TrCon_\ell \mid \lfloor \mathbf{j} \cdot \mathbf{n}(\mathbf{a}_p, \ldots, \mathbf{a}_q, \mathbf{a}_p + \mathbf{e}, \ldots, \mathbf{a}_q + \mathbf{e}) \rfloor = i\}$.

Region: Region $\mathcal{R}_i$ denotes the union of the i-regions of all trivial and non-trivial cones.

The intuition behind the definition of the cone vectors is quite simple. Let $\mathbf{j}$ be an arbitrary index point and assume that the longest path[1] (comprised of dependence vectors) from the axes to $\mathbf{j}$ is k. Any valid schedule must respect the dependencies and thus cannot execute $\mathbf{j}$ prior to moment k because $ECT(\mathbf{j}) = k$. The properties of the cone vectors guarantee that if $\mathbf{j} \in Con(\mathbf{a}_1, \ldots, \mathbf{a}_q)$, i.e. $\mathbf{j} = \lambda_1 \mathbf{a}_1 + \ldots + \lambda_q \mathbf{a}_q$, then $\lambda_1 + \ldots + \lambda_q \geq k$.

The inner and outer facets are used to determine the scheduling properties of the algorithm. In general, there is an inner facet per non-trivial cone. Its vertices are the dependence vectors that define the cone. This facet defines the optimal scheduling hyperplane for the cone in question. The

[1] The *length* of the path is the number of dependence vectors that form it.

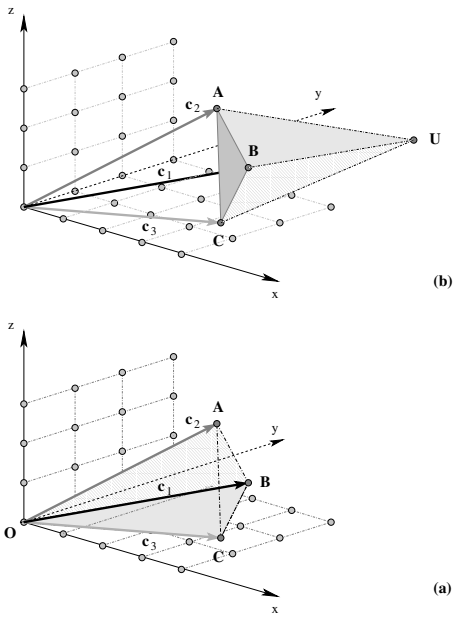

Figure 1. Inner and outer facets for a 3-dimensional algorithm with 3 dependence vectors.

outer facets contribute to the determination of the scheduling hyperplanes of the trivial cones.

The scheduling policy that executes every index point $\mathbf{j}$ at its earliest computation time $ECT(\mathbf{j})$, leads to optimal parallel time. Our scheduling policy guarantees that:

- In the UET case, every index point $\mathbf{j}$ is executed at moment $ECT(\mathbf{j})$ or $ECT(\mathbf{j}) + 1$.
- In the UET-UCT case, every index point $\mathbf{j}$ is executed at moment $ECT(\mathbf{j})$ or $ECT(\mathbf{j}) + 1$ or $ECT(\mathbf{j}) + 2$.

Hence, our policy guarantees the asymptotically optimal execution time of the entire loop.

3 Relation to Convex Hull

We must emphasize the equivalence of the problem of finding the cone vectors with that of computing the convex hull of the dependence vectors and of terminal point $\mathbf{U}$.

If we compute the convex hull (and ignore the artificial terminal point), the remaining points are precisely the cone vectors. Any dependence vector that belongs to the interior of the convex hull, also belongs to a cone defined by certain cone vectors.

According to the above, we present an algorithm that computes the cone vectors by computing the convex hull of the dependence vectors. This algorithm is a revision of `QuickHull` algorithm [4] for convex hulls.[2]

4 Partitioning the Index Space

The basic idea of our method is the partitioning of the index space into several cones and each cone into regions such that:

- given an arbitrary index point $\mathbf{j}$, we can find out a cone Con_i or a trivial cone $TrCon_i$ such that $\mathbf{j} \in Con_i$ or $\mathbf{j} \in TrCon_i$
- if $\mathbf{j} \in Con_i$ ($\mathbf{j} \in TrCon_i$), we can find a region $Reg_k \subseteq Con_i$ ($TrReg_k \subseteq TrCon_i$) such that $\mathbf{j} \in Reg_k$ ($\mathbf{j} \in TrReg_k$)
- $\mathbf{j}$ is executed at moment k.

4.1 Calculation of the Non-Trivial Cones

In order to determine the non-trivial cones, the inner facets must be computed. This is done by considering an algorithm quite similar to `QuickHull` algorithm mentioned above. Every inner facet is formed by a sequence of cone vectors. These cone vectors also define a corresponding non-trivial cone. Thus, for every inner facet, we keep track of the corresponding cone and cone vectors.

The following algorithm determines the inner facets, the cones and the cone vectors.

Algorithm 1 *inner facet computation*

INPUT: $D = \{\mathbf{d}_1, \ldots, \mathbf{d}_m\}$ and terminal point $\mathbf{U}$.

OUTPUT: Inner facet sequence $\mathcal{IF} = \langle if_1, if_2, \ldots \rangle$.

1. Let $D' = D \cup \mathbf{U}$.
2. The convex hull $\mathcal{CH}_{in} = conv(D')$ of D' is determined. Let $\mathcal{F} = \langle f_1, f_2, \ldots \rangle$ be the sequence of the facets of the convex hull $\mathcal{CH}_{in}$.
3. The sequence of the inner facets follows from the sequence $\mathcal{F}$ if the facets containing the terminal point $\mathbf{U}$ are removed: $\mathcal{IF} = \langle if_1, if_2, \ldots \rangle = \langle f_i \in \mathcal{F} \mid \mathbf{U} \notin f_i \rangle$.
4. The vertices of each inner facet define a cone. The set of (non-trivial) cones is determined as $NonTrCS = \{Con_i(\mathbf{a}_{i1}, \mathbf{a}_{i2}, \ldots) \mid \mathbf{a}_{ij} \in if_i\}$.
5. The cone vectors are the dependence vectors that form the set of non-trivial cones $CV = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_r\}$.

In effect, the above algorithm determines the optimal scheduling hyperplanes of all non-trivial cones: they coincide with the inner facets.

[2] For more on this computational geometry method the interested reader may refer to [5] and [11].

4.2 Calculation of the Trivial Cones

For the calculation of the trivial cones, the computation of the outer facets suffices. To each outer facet corresponds a trivial cone. The number of the vectors that form the trivial cone depends on the specific facet. The following algorithm determines the outer facets and then the set of the trivial cones.

Algorithm 2 *outer facet computation*

INPUT: Cone vector set $CV = \{\mathbf{c}_1, \ldots, \mathbf{c}_r\}$ and axes origin $\mathbf{O}$.

OUTPUT: Outer facet sequence $\mathcal{OF} = \langle of_1, of_2, \ldots \rangle$.

1. Let $CV' = CV \cup \mathbf{O}$.
2. The convex hull $\mathcal{CH}_{out} = conv(CV')$ of CV' is determined. Let $\mathcal{F} = \langle f_1, f_2, \ldots \rangle$ be the sequence of the facets of $\mathcal{CH}_{out}$.
3. The outer facets result from $\mathcal{CH}_{out}$; they are the facets that contain $\mathbf{O}$: $\mathcal{OF} = \langle of_1, of_2, \ldots \rangle = \langle f_i \in \mathcal{F} \mid \mathbf{O} \in f_i \rangle$.
4. Let $\mathbf{U}_1, \ldots, \mathbf{U}_n$ be the projections of $\mathbf{U}$ on the axes $x_1, \ldots, x_n$.
   - For every outer facet $of_i = \langle \mathbf{a}_{i1}, \ldots, \mathbf{a}_{iq} \rangle$, consider those points $\mathbf{U}_j$ that belong to the negative subspace relative to the hyperplane defined by of_i. Call those points $\mathbf{U}_s, \ldots, \mathbf{U}_t$.
   - Let $\mathbf{e}_s, \ldots, \mathbf{e}_t$ be the corresponding unitary vectors; the trivial cone $TrCon_i$ corresponding to the outer facet of_i is $TrCon_i = Con(\mathbf{a}_{i1}, \ldots, \mathbf{a}_{iq}, \mathbf{e}_s, \ldots, \mathbf{e}_t)$.
5. The procedure is repeated for all outer facets and the set of trivial cones is determined as $TrCS = \{TrCon_1, TrCon_2, \ldots\}$.

In the case of a trivial cone, the optimal scheduling hyperplane is defined in a different way. Let the trivial cone $TrCon_i = Con(\mathbf{a}_{i1}, \ldots, \mathbf{a}_{ip}, \mathbf{e}_s, \ldots, \mathbf{e}_t)$ and $\mathbf{e} = \mathbf{e}_s + \ldots + \mathbf{e}_t$. The scheduling hyperplane is the one defined by the points $\mathbf{a}_{i1}, \ldots, \mathbf{a}_{iq}, \mathbf{a}_{i1} + \mathbf{e}, \ldots, \mathbf{a}_{iq} + \mathbf{e}$.

5 Scheduling in n Dimensions

After the calculation of the non-trivial and trivial cones, it is possible to accomplish the scheduling of each point of the index space. A point that belongs to a non-trivial cone is expressed as a linear combination of the vectors that define the cone; we determine the scheduling time of this point by adding the coefficients of the linear combination and then taking the floor.

For a point that belongs to a trivial cone, the scheduling time is computed by calculating the inner product of this point and the normal vector that corresponds to this trivial cone and then taking the floor again. The following algorithm implements the above scheduling in the absence of communication cost.

Algorithm 3 *geometric scheduling in n dimensions*

INPUT: $D = \{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_m\}$, point $\mathbf{U}$ and a point $\mathbf{j} \in J$.

OUTPUT: The time instance t_j when j will be scheduled.

1. *Algorithm 1* is applied to D; the inner facets $\mathcal{IF} = \langle if_1, if_2, \ldots \rangle$, the non-trivial cones $NonTrCS = \{Con_1, Con_2, \ldots\}$ and the cone vectors $CV = \{\mathbf{c}_1, \ldots, \mathbf{c}_r\}$ are determined.
2. *Algorithm 2* is applied to CV; the outer facets $\mathcal{OF} = \langle of_1, of_2, \ldots \rangle$ are determined.
3. Given an arbitrary point $\mathbf{j} \in J$, we determine whether it is located inside a non-trivial cone or inside a trivial cone.
4. If $\mathbf{j}$ belongs to a non-trivial cone, $\mathbf{j}$ is expressed as a linear combination of the vectors that define this cone; the resulting coefficients are computed. Their sum gives the scheduling time: $t_j = \lfloor \lambda_1 + \lambda_2 + \ldots + \lambda_q \rfloor$.
5. Otherwise, $\mathbf{j}$ belongs to a trivial cone $TrCon_i$, which corresponds to the outer facet of_i. Let $\mathbf{n}_i$ be the vector normal to of_i. The scheduling time then is $t_j = \lfloor \mathbf{j} \cdot \mathbf{n}_i \rfloor$.

6 Scheduling with Communication Cost

Any UET-UCT algorithm can be transformed in an equivalent UET algorithm, with slightly changed dependence vector set, as proven in [16]. The 2D geometric scheduling technique for UET/UET-UCT algorithms is presented in [9]. With this paper we generalize the geometric scheduling technique for n-dimensional spaces.

The general idea is the following: let $D = \{\mathbf{d}_1, \ldots, \mathbf{d}_m\}$ be the dependence set of the initial UET-UCT algorithm and assume that $\mathbf{d}_c$, one of the above dependence vectors, is chosen as the communication vector, i.e. index points connected by $\mathbf{d}_c$ are assigned to the same processor. The equivalent UET algorithm has the following dependence vectors: $\{\mathbf{d}_1, \ldots, \mathbf{d}_{c-1}, 2\mathbf{d}_c, \mathbf{d}_{c+1}, \ldots, \mathbf{d}_m\}$.

7 Conclusion

In this paper we presented an asymptotically optimal scheduling method for uniform nested loops. We have also developed a tool that evaluates this scheduling method in comparison with other methods, such as the hyperplane method.

In Figure 2 we present an indicative comparison of scheduling randomly generated loops with 11-15 dependence vectors and sizes up to 1.000 repetitions per dimension. Our scheduling method and the hyperplane one are

compared with the optimal execution procedure, in both UET and UET-UCT algorithms. During the optimal execution, we consider that each point is executed at its earliest computation time.

Our method imposes a delay to the 0.2% of the index points, while the hyperplane method to 9% of them. All other points are executed at their earliest computation time in both methods.

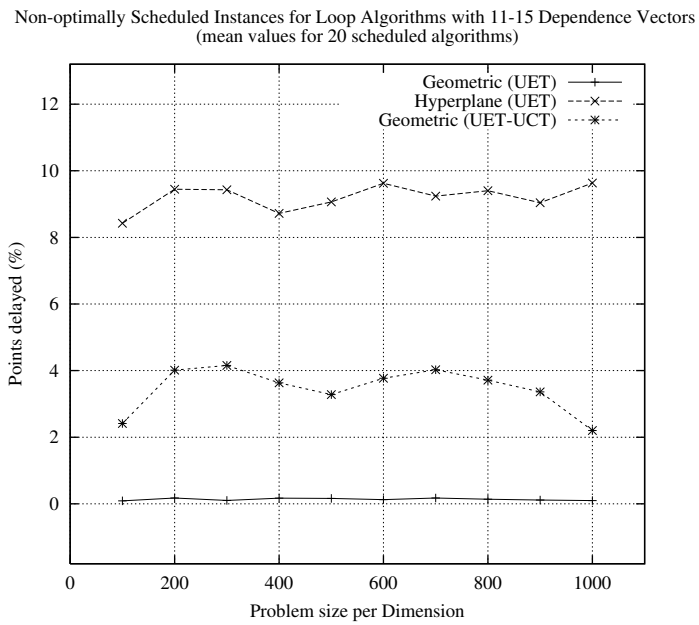

Figure 2. Comparison of geometric and hyperplane methods with the optimal one, for algorithms with 11-15 dependence vectors. The percentages of the delayed index points are displayed, for algorithms with and without communication cost (UET-UCT and UET models respectively).

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, G. Papakonstantinou, and P. Tsanakas. Optimal scheduling for UET/UET-UCT generalized n-dimensional grid task graphs. *Journal of Parallel and Distributed Computing*, 57(2):140–165, May 1999.

[3] T. Andronikos, N. Koziris, Z. Tsiatsoulis, G. Papakonstantinou, and P. Tsanakas. Lower time and processor bounds for efficient mapping of uniform dependence algorithms into systolic arrays. *Journal of Parallel Algorithms and Applications*, 10(3-4):177–194, 1997.

[4] C. B. Barder, D. P. Dobkin, and H. Hundanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, December 1996.

[5] T. M. Chan. *Output-Sensitive Construction of Convex Hulls*. PhD thesis, The University of British Columbia, October 1995.

[6] P. Chretienne. Task scheduling with interprocessor communication delays. *European Journal of Operational Research*, 57:348–354, 1992.

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

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

[9] I. Drositis, T. Andronikos, G. Manis, G. Papakonstantinou, and N. Koziris. Geometric scheduling of 2-D UET/UET-UCT uniform dependence loops. In *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing - PDP 2002*, pages 343–350, Gran Canaria Island, Spain, 2002. IEEE Press.

[10] 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.

[11] D. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15:287–299, 1986.

[12] 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.

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

[14] D. Moldovan and J. Fortes. Partitioning and mapping algorithms into fixed size systolic arrays. *IEEE Transactions on Computers*, C-35(1):1–11, 1986.

[15] 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.

[16] G. Papakonstantinou, T. Andronikos, and I. Drositis. On parallelization of UET/UET-UCT loops. *Neural, Parallel and Scientific Computations*, 9(3-4):279–318, September & December 2001.

[17] V. J. Rayward-Smith. UET scheduling with unit interprocessor communication delays and unlimited number of processors. *Discrete Applied Mathematics*, 18:55–71, 1987.

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

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

[20] P. Tsanakas, N. Koziris, and G. Papakonstantinou. Chain Grouping: A method for partitioning loops onto mesh-connected processor arrays. *IEEE Transactions on Parallel and Distributed Systems*, 11(9):941–955, September 2000.

[21] J. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10:384–393, 1975.