An Efficient Algorithm for the Optimal Linear Schedule of Uniform Dependence Algorithms

T. Andronikos, N. Koziris, Z. Tsiatsoulis,
G. Papakonstantinou[1] and P. Tsanakas

Digital Systems and Computers Laboratory,
Computer Science Division,
Department of Electrical and Computer Engineering
National Technical University of Athens
Zographou Campus, 15773, Athens, Greece

Abstract

Real-time dynamic control of robot manipulators incurs significant computational load, that can be tackled by parallel processing techniques. One of the most promising areas of research in this field is the area of automatic parallelization of sequential algorithms. In this paper we present an efficient algorithm for the parallelization of nested FOR(DO)-loops with uniform data dependencies. Our work is based on the hyperplane concept, i.e. a set of computations which can be executed simultaneously and which lie on a group of parallel hyperplanes. The resulting hyperplane is guaranteed to be optimal. The primary task when dealing with serial algorithms which will be executed in parallel architectures, is finding a time mapping that defines a new execution ordering and enables parallelism. Our objective is to schedule computations in time, for a specific class of algorithms, the uniform dependence algorithms. The method presented here finds the optimal linear mapping, and outperforms all the previously presented methods in terms of time complexity.

1. Introduction

One of the most tedious tasks for a lot of sequencial algorithms is the execution of nested FOR(DO)-loops with uniform data dependencies. Therefore, a significant number of papers has been devoted to the efficient parallelization of these loops [1], [3], [8]. Most of the well-known signal and image processing algorithms, such as LU decomposition, convolution, etc., belong in this category. Furthermore, a large class of algorithms with non-uniform dependencies, fall into this class after applying methods such as the one presented in [17].

The first step in parallelizing a FOR(DO)-loop is finding its optimal parallel, execution time, ignoring whatever restriction, the available hardware would impose on. Once optimal parallel execution is found, other criteria focusing upon hardware specifications can afterwards be established [4], [5], [9]. In the past, many methods dealt with this problem, [13], [14], [16]. All of them are based on the solution of diofantine equations [7], [10], or on the use of linear programing approaches [13].

This paper presents a new procedure for finding the optimal Π. Our algorithm presents an alternative approach to the determination of the optimal linear scedule. All previously presented methods have the disadvantage of not exploiting the regularity of the index space. Our method is based on the schematical representation of the dependence graph, and on the possible relations among the various dependence vectors. The uniformity of the index space, imposed from constant dependence vectors, is a virtue exploited in order to attain $O(n^3m^3)$ time complexity, where n is the dimension of the index space and m is

---

[1]To whom all correspondence should be addressed

the number of the dependence vectors. We partition the index space into subsets, called cones. Inside every cone we define an optimal linear scedule, called hyperplane. The proposed algorithm results in a set of candidate optimal hyperplanes, from which the best is chosen, with respect to the specific algorithm's structure.

2. Basic Concepts and Definitions

2.1. The Model of the Algorithms

In this paper we are only interested in algorithms that exhibit regular computation pattern. These algorithms are nested FOR(DO)-loops with *uniform* data dependencies:

```
FOR I_1=l_1 TO u_1 DO
...
FOR I_n=l_n TO u_n DO
   AS_1(I)
   ...
   AS_K(I)
ENDFOR
...
ENDFOR
```

Fig. 1: The Algorithmic model

In Fig. 1: (1) l_i and u_i are integer-valued constants, (2) $\mathbf{I}=(I_1, ..., I_n)$ and (3) $AS_1, ..., AS_K$ are assignment statements of the form $V_0=E(V_1, ..., V_l)$, where V_0 is an output variable indexed by **I** and E is an expression of the input variables $V_1, ..., V_l$, also indexed by **I**.

2.2. Notation

We use the following notation:

- N (Z) is the set of naturals (integers).
- n is the number of nested loops and m is the number of dependence vectors.
- $J^n \subset Z^n$ is the set of indices, i.e. $J^n = \{(I_1, ..., I_n) \mid I_i \in Z \wedge l_i \le I_i \le u_i, 1 \le i \le n\}$, also called *index space*. Each point in this n-dimensional integer space represents a distinct instantiation of the loop body. The geometrical form of the index space is that of a *hyperparallelepiped*.
- $\mathbf{d}_i=(d_{i1}, ..., d_{in})$, $1 \le i \le m$, is a *dependence vector*.

We shall not give here the analytical definitions of the various kinds of dependencies. We assume that the reader is already familiar with these concepts, and if this is not the case, we refer him to the references [10], [11]. The following two things should be emphasized:

1) by definition, dependence vectors are always > than **0**, where **0**=(0, ..., 0) and > is the *lexicographic* ordering

2) in our model all dependence vectors are *uniform*, i.e. their elements are constants and not functions defined over index sets. The theory we present and apply here is based on this assumption.

- The *dependence set* DS of an algorithm A is the set of all dependence vectors of this algorithm: $DS=\{\mathbf{d}_1, \mathbf{d}_2, ..., \mathbf{d}_m\}$.

Example 2.1. Consider the following doubly nested loop:

```
FOR I=1 TO 3 DO
   FOR J=1 TO 3 DO
   AS1: A[I, J]=A[I-1, J-1]+A[I, J-1]+1
   AS2: B[I, J]=A[I-1, J]+2
   ENDFOR
ENDFOR
```

Fig. 2: Algorithm A_1

Here, on iteration (i, j), statement AS_1 depends on the results of the same statement on iterations (i-1, j-1) and (i, j-1) and statement AS_2 depends on the results of statement AS_1 on iteration (i-1, j). This is expressed by the three dependence vectors: $\mathbf{d}_1=(i, j)-(i, j-1)=(0, 1)^T$, $\mathbf{d}_2=(i, j)-(i-1, j)=(1, 0)^T$ and $\mathbf{d}_3=(i, j)-(i-1, j-1)=(1, 1)^T$. Thus, the dependence is the need for values computed at earlier iterations.

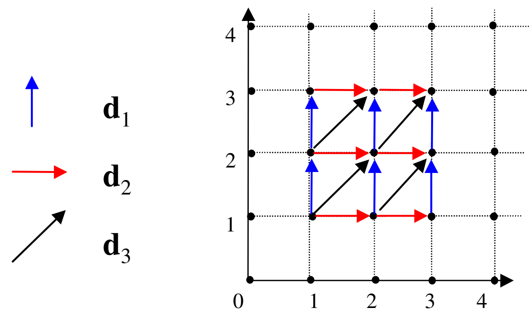

Fig. 3: The index space J^2

The index space represents the iterations of the loop body. The dependence vectors among assignment statements relate points of the index space, so we speak of dependence vectors among index points.

2.3. Basic Definitions

In the rest of the paper we shall abstract a sequential algorithm A by the ordered pair (J^n, DS), where J^n is the index space and DS is the dependence set of the algorithm. Each element **j** in the index space is a n-tuple (j_1, j_2, ..., j_n) which corresponds to one instance, i.e. one iteration of the nested loops. All the k statements which are computed in the **j** loop instance are represented by the **j** point in the index space.

Let TIME=$\{t_0, t_1, ..., t_i, ...\}$ be the infinite set of time instants. For notational convenience we agree that:

(1) The computation of the algorithm always commences at time t_0, and

(2) t_0 is identified with 0, t_i with i and TIME with N [5].

Assumption 2.1. The execution of any assignment statement is completed in one discrete time instant t_i. ■

All parallelization techniques regarding the correct parallel execution of an algorithm of this class rely on the following fundamental assumption:

Assumption 2.2. Assignment statement AS is correctly executed at instant t iff all assignment statements AS_1, AS_2, ..., AS_n on which AS depends, have been correctly executed at previous instants. We now express the above assumption in a formal manner. ■

Axiom t is the *least* natural at which $\mathbf{j}\in J^n$ can be correctly executed iff $\forall \mathbf{i}\in J^n$ for which $\exists \mathbf{d}\in DS$ such that $\mathbf{i}+\mathbf{d}=\mathbf{j}$, the least natural at which $\mathbf{i}\in J^n$ can be correctly executed is t_i with $t_i<t$. ■

Definition 2.1. A finite sequence **S**=<S_0, ..., S_k> of nonempty subsets of J^n is a *schedule* of A(J^n, DS) iff:

(1) $S_i \cap S_j=\varnothing$, whenever $i\neq j$,

(2) $\cup_0^k S_i=J^n$ and

(3) S_i (i.e. all l the index points of S_i) are correctly executed (in parallel) at instant i, $0\leq i\leq k$. ■

In practice this means that a schedule for A(J^n, DS) is a function $\mathbf{s} : J^n\rightarrow N$, such that $\forall \mathbf{i}, \mathbf{j}\in J^n$, $\mathbf{j}=\mathbf{i}+\mathbf{d} \Rightarrow \mathbf{s}(\mathbf{i})<\mathbf{s}(\mathbf{j})$, where $\mathbf{d}\in DS$.

Definition 2.2. Given A(J^n, DS), a *free schedule* (see also [9] for another definition of the same concept) is a function $\mathbf{s_f} : J^n\rightarrow N$:

1) $\mathbf{s_f}$ is a schedule.

2) $\mathbf{s_f}(\mathbf{j})=0$, $\forall \mathbf{j}\in J^n$ such that $\mathbf{j}-\mathbf{d}\notin J^n$.

3) $\mathbf{s_f}(\mathbf{j})=1+\max\{\mathbf{s_f}(\mathbf{j}-\mathbf{d}) \mid \mathbf{d}\in DS\}$.■

Obviously, a free schedule is a schedule with the additional property that computations are executed as soon as their operands are available. Thus, this schedule achieves the lower execution time bound.

Definition 2.3. Given A(J^n, DS), a *linear schedule* (or *linear time transformation*) Π is a mapping with the following properties:

1) Π is linear

2) Π respects the dependecies; if $\mathbf{j}_1$, $\mathbf{j}_2\in J^n$ and $\mathbf{j}_2=\mathbf{j}_1+\mathbf{d}$ then $\Pi\mathbf{j}_2>\Pi\mathbf{j}_1$ or equivalently $\Pi\mathbf{d}>0$ for every $\mathbf{d}\in DS$.

3) the time function $\mathbf{s}(\mathbf{j})$ which shows at what time instance the **j** index point is executed is given by $\mathbf{s}(\mathbf{j})=\left\lfloor \dfrac{\Pi\mathbf{j}-t_0}{\mathrm{disp}\Pi} \right\rfloor$, where $\mathrm{disp}\Pi=\min\{\Pi\mathbf{d}_i \mid \mathbf{d}_i\in DS\}$. ■

Definition 2.4. Let t_Π be the total time associated to linear transformation Π. Π_{OT} is *optimal* iff $t_{\Pi_{OT}}=\min\{t_{\Pi_j} \mid \Pi_j\in LS\}$, where *LS* is the class of linear schedules. ■

In definition 2.4. Π was defined as a transformation $J^n\rightarrow N$, thus allowing multiple index points to correspond to the same integer $\Pi\mathbf{j}$, i.e. to the same instant $\mathbf{s}(\mathbf{j})=\left\lfloor \dfrac{\Pi\mathbf{j}-t_0}{\mathrm{disp}\Pi} \right\rfloor$.

The hyperplane Π partitions the index set into groups of computations which can be executed in parallel. All **j** for which **s**(**j**) is the same can be executed simultaneously. The geometrical view of Π is a hyperplane which sweeps index points. All index points which belong to the same hyperplane (or group of hyperplanes) are executed concurrently.

Definition 2.5. Let x_1, ..., x_n be the orthocanonic axes of a n-dimensional vector space. Then $\mathbf{i}_1$, ..., $\mathbf{i}_n$ are the unitary vectors along these axes. ■

Definition 2.6. $\Pi(\mathbf{v}_1, ..., \mathbf{v}_n)$ is the (n-1)-dimensional hyperplane that passes through the end points of the n vectors $\mathbf{v}_1$, ..., $\mathbf{v}_n$, with common starting point the origin. ■

Definition 2.7. The **cone** of the n vectors $\mathbf{v}_1$, ..., $\mathbf{v}_n$, denoted $\mathbf{C}(\mathbf{v}_1, ..., \mathbf{v}_n)$, is $\{\mathbf{j}\in R^n \mid \mathbf{j}=\lambda_1\mathbf{v}_1+\lambda_2\mathbf{v}_2+...+\lambda_n\mathbf{v}_n$, with $\lambda_i\geq 0$, $1\leq i\leq n\}$.■

3. Main Results

The *sequential* execution of the algorithm induces the lexicographic ordering among the index points of J^n because the index points are executed in this way, i.e. $\mathbf{j}_1$ instance is executed before $\mathbf{j}_2$ iff $\mathbf{j}_1<\mathbf{j}_2$. In order to minimize the computation time of the algorithm this ordering must be altered, in such a way that the correctness of the algorithm is maintained. According to the Axiom of Section 2, this can be accomplished via the preservation of the dependencies among index points. The introduction of a linear Π aims to change this serial execution ordering, while not disturbing the dependencies among different indexed computations. In order to achieve the *optimal parallelization* of a sequential algorithm A with respect to linear time scheduling, we must determine the optimal linear parallel schedule.

Theorem 3.1. The parallel execution time of $A(J^n, DS)$, when a Π is applied is given by:

(1) $$t=\left\lceil \frac{\max \Pi\, \mathbf{j}_1-\min \Pi\, \mathbf{j}_2+1}{\text{disp}\Pi} \right\rceil,$$ or

(2) $$t=\left\lfloor \frac{\max \Pi\, \mathbf{j}_1-\min \Pi\, \mathbf{j}_2}{\text{disp}\Pi} \right\rfloor+1$$

(equivalence can be easily proved). This theorem is proven in [14].

To simplify the discussion we have considered, without loss of generality, 2-D index spaces. Our method can be extended in n>2 dimensions. The intuition behind our method is to divide the index space into disjoint cones, which are formed by *some* of the dependence vectors plus $\mathbf{i}_1$ (and perhaps $\mathbf{i}_2$). The vectors that define those cones *also define all the possible optimal hyperplanes* [2]. Each possible optimal hyperplane leads to a time schedule. Such a hyperplane divides the line segment from the origin point of the index space to the greatest lexicographic point (ending point) j_f in t_Π subsegments. Obviously, the optimal hyperplane Π is the one which leads to the least t_Π. It can be proved that the cone which includes the j_f defines the optimal Π [2].

The proposed algorithm finds these dependence vectors which form the possible optimal hyperplanes. It is based on the following property of linear schedules: if $\mathbf{j}$, $\mathbf{i}\in J^n$ with $\mathbf{i}+\mathbf{d}=\mathbf{j}$ then a valid schedule ensures $\Pi\mathbf{i}<\Pi\mathbf{j} \Rightarrow \Pi\mathbf{d}>0$, $\forall \mathbf{d}\in DS$. An optimal Π is one such: $\mathbf{s}(\mathbf{j})=\mathbf{s}(\mathbf{i})+1$ for as many as possible points $\mathbf{j}$: $\mathbf{i}+\mathbf{d}_k=\mathbf{j}$, for $\mathbf{d}_k\in DS$. This means that an optimal linear schedule Π organises computations to execute as fast as the correct and linear execution of the algorithm permits. Thus an optimal Π, ensures that as many as possible consecutive index points are executed in consecutive time instances. It is obvious that the scheduling algorithm should search for such Π's: $\mathbf{j}=\mathbf{i}+\mathbf{d} \wedge \mathbf{s}(\mathbf{j})=\mathbf{s}(\mathbf{i})+1$ thus $\Pi\mathbf{d}=\text{disp}\Pi$, for as many as possible $\mathbf{d}\in DS$, $\mathbf{j}$, $\mathbf{i}\in J^2$ and for all the other vectors $\mathbf{d}_i\in DS$, $\Pi\mathbf{d}_i>0$ holds. Since Π is in 2-D, it has the form of $\Pi=[a\ b]$. So, we should search for two dependence vectors $\mathbf{d_a}$, $\mathbf{d_b}$ which form the Π plane.

For every other $\mathbf{d}$ we ensure that $\mathbf{d}=\lambda_1\mathbf{d}_a+\lambda_2\mathbf{d}_b$, $\lambda_1+\lambda_2>1$, thus if: $\mathbf{i}+\mathbf{d}=\mathbf{j}$, then for those $\mathbf{i}$, $\mathbf{j}$ it holds: $\mathbf{s}(\mathbf{j})>\mathbf{s}(\mathbf{i})+1$, which is valid. Nevertheless, for $\mathbf{i}'$, $\mathbf{j}'$ with $\mathbf{i}'+\mathbf{d}_a=\mathbf{j}'$ or $\mathbf{i}'+\mathbf{d}_b=\mathbf{j}'$, we have $\mathbf{s}(\mathbf{j}')=\mathbf{s}(\mathbf{i}')+1$, thus conscecutive time execution for $\mathbf{i}'$, $\mathbf{j}'$.

The following algorithm calculates a set of optimal Π's from which the optimal with respect to the specific index space is chosen. It considers pairs of dependence vectors and detects which of them forms possible optimal hyperplanes. This algorithm outperforms all previously presented algorithms since it has lower computational time complexity, and specifically $O(n^3m^3)$. This is because it works on DS and not on the index space or the solution space [10], [13].

The 2-D Algorithm for finding the optimal Π

Input:

DS={$\mathbf{d}_1$, ..., $\mathbf{d}_m$}

Output:

PLANE={$\Pi_i(\mathbf{v}_1, ..., \mathbf{v}_n)$}

Step 0: INITIALIZATION

DS=DS;

ODS=<>;

PLANE=∅;

Step 1:

Find $\mathbf{d}_x$ such that $\mathbf{d}_{x1}=\min\{d_{i1} \mid i=1, ..., m\}$

DS=DS-{$\mathbf{d} \mid \mathbf{d}\in \mathbf{C}(\mathbf{i}_2, \mathbf{d}_x)$}

PLANE=PLANE∪{$\Pi=[d_{x2}\ 0]$}

If $\forall \mathbf{d}_i\in DS$, $d_{i2}\geq 0$ then

find $\mathbf{d}_y$ such that $d_{y2}=\min\{d_{i2} \mid i=1, ..., m\}$

DS=DS-{$\mathbf{d}$ | $\mathbf{d}\in\mathbf{C}(\mathbf{i}_1, \mathbf{d}_y)$}
PLANE=PLANE∪{Π=[0 d_{y1}]}
else
find $\mathbf{d}_y$ such that Angle($\mathbf{d}_x$, $\mathbf{d}_y$)=max{Angle($\mathbf{d}_x$, $\mathbf{d}_i$) | $\mathbf{d}_i\in$DS}
EndIf
ODS=<ODS, $\mathbf{d}_x$>
DS=DS-{$\mathbf{d}_x$}

Step 2:
Find $\mathbf{d}_j$ such that: $\forall \mathbf{d}_i\in$DS-{$\mathbf{d}_j$}, $\mathbf{d}_i=\lambda_{i1}\mathbf{d}_x+\lambda_{i2}\mathbf{d}_j$, with $\lambda_{i1}+\lambda_{i2}>1$
PLANE=PLANE∪{Π($\mathbf{d}_x$, $\mathbf{d}_j$)}
ODS=<ODS, $\mathbf{d}_j$>
DS=DS-{$\mathbf{d}_j$}

Step 3:
$\mathbf{d}_x=\mathbf{d}_j$
If $\mathbf{d}_j=\mathbf{d}_y$ then
Goto Step 4
else
Goto Step 2
EndIf

Step 4:
All possible optimal hyperplanes for A(J^n, DS) are in the PLANE set. The optimal Π for the specific J^n is Π($\mathbf{v}_a$, $\mathbf{v}_b$), where $\mathbf{C}(\mathbf{v}_a, \mathbf{v}_b)$ is the cone that contains $\mathbf{j}_f$ ($\mathbf{j}_f$ is the greatest lexicographicaly index point).

STOP

Fig. 4: The Algorithm for the optimal Π

ODS is a list containing *all dependence vectors* that participate in the formation of optimal linear hyperplanes. The PLANE set contains *all possible optimal hyperplanes* Π, which may be *more than one*.

4. Illustrative Example

Example 4.1. Consider the following doubly nested loop:

```
FOR I=0 TO N1 DO
   FOR J=0 TO N2 DO
      A[I, J]=A[I-1, J-4]+A[I-4, J-1]+A[I-2, J-2]*3
      B[I, J]=A[I-3, J-5]*B[I-5, J-3]+5
   ENDFOR
ENDFOR
```

Fig. 5: Algorithm A_2

Here, there are five dependence vectors: $\mathbf{d}_1$=(i, j)-(i-1, j-4)=$(1, 4)^T$, $\mathbf{d}_2$=(i, j)-(i-4, j-1)=$(4, 1)^T$, $\mathbf{d}_3$=(i, j)-(i-2, j-2)=$(2, 2)^T$, $\mathbf{d}_4$=(i, j)-(i-3, j-5)=$(3, 5)^T$ and $\mathbf{d}_5$=(i, j)-(i-5, j-3)=$(5, 3)^T$.
The steps of the Algorithm are the following:

Step 0:
DS={$\mathbf{d}_1$, $\mathbf{d}_2$, $\mathbf{d}_3$, $\mathbf{d}_4$, $\mathbf{d}_5$}
ODS=<>
PLANE=∅

Step 1:
$\mathbf{d}_x=\mathbf{d}_1$
$\mathbf{d}_y=\mathbf{d}_2$
PLANE={[1 0]}
ODS=<ODS, $\mathbf{d}_1$>
DS={$\mathbf{d}_2$, $\mathbf{d}_3$, $\mathbf{d}_4$, $\mathbf{d}_5$}
PLANE=PLANE∪{[0 1]}

Step 2:
$\mathbf{d}_j=\mathbf{d}_3$ because $\mathbf{d}_4=2/3\mathbf{d}_1+7/6\mathbf{d}_3$ and $\mathbf{d}_5=-2/3\mathbf{d}_1+17/6\mathbf{d}_3$ and $\mathbf{d}_2=-\mathbf{d}_1+5/2\mathbf{d}_3$
ODS=<ODS, $\mathbf{d}_3$>
DS={$\mathbf{d}_2$, $\mathbf{d}_4$, $\mathbf{d}_5$}
PLANE={Π($\mathbf{d}_1$, $\mathbf{d}_3$)}

Step 3:
$\mathbf{d}_x=\mathbf{d}_3$

Step 2:
$\mathbf{d}_j=\mathbf{d}_2$ because $\mathbf{d}_4=17/6\mathbf{d}_3-2/3\mathbf{d}_2$ and $\mathbf{d}_5=7/6\mathbf{d}_3+2/3\mathbf{d}_2$
ODS=<$\mathbf{d}_1$, $\mathbf{d}_3$, $\mathbf{d}_2$>
DS={$\mathbf{d}_4$, $\mathbf{d}_5$}
PLANE={Π($\mathbf{d}_1$, $\mathbf{d}_3$), Π($\mathbf{d}_2$, $\mathbf{d}_3$)}

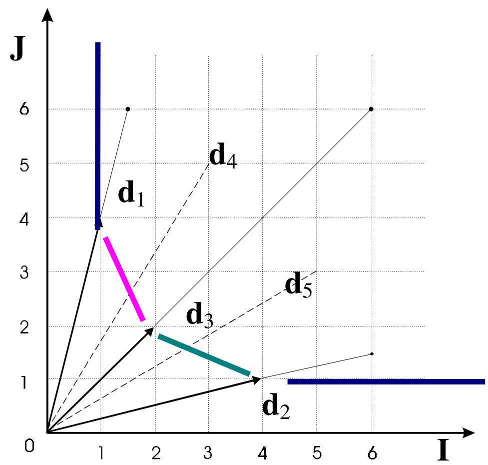

Fig. 6: The index space and the hyperplanes

Step 3:
$\mathbf{d}_x=\mathbf{d}_2=\mathbf{d}_y$

Step 4:
$\mathbf{j}_f=(N_1, N_2)$. Therefore, when $N_1/N_2>4$, the optimal Π is [0 1], when $1\le N_1/N_2\le 4$ the optimal is Π($\mathbf{d}_2$, $\mathbf{d}_3$), when $1\le N_2/N_1\le 4$ the optimal is Π($\mathbf{d}_1$, $\mathbf{d}_2$) and finally when $N_2/N_1>4$, the optimal is [1 0].

STOP

5. Conclusion

In this paper, we have presented an efficient algorithm, which finds the optimal time hyperplane (linear time schedule) for the class of uniform dependence algorithms. The method presented here guarantees the extraction of the optimal linear transformation from a two-dimensional nested loop with uniform dependencies, resulting in the optimal time execution among all linear schedules, and outperforms all the previously presented methods in terms of time complexity.

Work in progress includes the establishment of new criteria concerning optimal loop execution in multi-processors systems based on optimal linear schedules.

6. References

[1] A. Aiken, A. Nicolau, "Optimal Loop Parallelization," *Proc. SIGPLAN*, 1988.

[2] T. Andronikos, N. Koziris, Z. Tsiatsoulis, G. Papakonstantinou and P. Tsanakas, "Low Complexity Methods for Optimal Linear Time Scheduling of Uniform Dependence Algorithms," Digital Systems and Computers Laboratory, NTUA, Technical Report 03/95.

[3] U. Banerjee, S.-C. Chen, D. J. Kuck and R. A. Towle, "Time and Parallel Processor Bounds for Fortran-Like Loops," *IEEE Trans. Comput*., vol. C-28, no. 9, pp. 660-670, Sept. 1979.

[4] T.-S. Chen and J.-P. Sheu, "Communication-Free Data Allocation Techniques for Parallelizing Compilers on Multicomputers," IEEE Trans. Parallel Distrib. Syst., vol. 5, no. 9, pp. 924-937, Sept. 1994.

[5] O.H. Ibarra and S.M. Sohn, "On Mapping Systolic Algorithms onto the Hypercube," *IEEE Trans. Parallel Distrib. Syst*., vol. 1, no. 1, pp. 48-63, Jan. 1990.

[6] R. Karp, R. Miller, S. Winograd, "The Organization of Computations for Uniform Recurrence Equations," *J. ACM*, vol. 14, no. 3, pp. 563-590, July 1967.

[7] L. Lamport, "The parallel execution of DO loops," *Commun. ACM*, vol.17, no.2, pp. 83-93, Feb. 1974.

[8] P.-Z. Lee and Z.M. Kedem, "Synthesizing Linear Array Algorithms from Nested For Loop Algorithms," *IEEE Trans. Comp*., vol. 37, no. 12, pp. 1578-1598, Dec. 1988.

[9] P.-Z. Lee and Z.M. Kedem, "Mapping nested loop algorithms into multidimensional systolic arrays," *IEEE Trans. Parallel Distrib. Syst*., vol. 1, no. 1, pp. 64-76, Jan. 1990.

[10] D.I. Moldovan and J.A.B. Fortes, "Partitioning and mapping algorithms into fixed size systolic arrays," *IEEE Trans. Comput*., vol C-35, no 1, pp. 1-11, Jan. 1986.

[11] D.I. Moldovan, *Parallel Processing: From Applications to Systems*, California: Morgan Kaufmann, 1993.

[12] J.K. Peir and R. Cytron, "Minimum Distance: A method for partitioning recurrences for multiprocessors," *IEEE Trans. Comput*., vol. 38, no. 8, pp.1203-1211, Aug. 1989.

[13] W. Shang and J.A.B. Fortes, "Time optimal linear schedules for algorithms with uniform dependencies," *IEEE Trans. Comput*., vol. 40, no. 6, pp. 723-742, June 1991.

[14] W. Shang and J.A.B. Fortes, "Independent partitioning of algorithms with uniform dependencies," *IEEE Trans. Comput*., vol. 41, no. 2, pp. 190-206, Feb. 1992.

[15] W. Shang and J.A.B. Fortes, "On time mapping of uniform dependence algoritms into lower dimensional processor arrays," *IEEE Trans. Parallel Distrib. Syst*., vol. 3, no. 3, pp. , May 1992.

[16] W. Shang, M.T. O'Keefe and J.A.B. Fortes, "On Loop Transformations for Generalized Cycle Shrinking", I*EEE Trans. Parallel Distrib. Syst*., vol. 5, no. 2, pp. 193-204, Feb. 1994.

[17] T.H. Tzen and L.M. Ni, "Dependence Uniformization: A Loop Parallelization Technique", *IEEE Trans. Parallel Distrib. Syst*., vol. 4, no. 5, pp. 547-558, May 1993.