

Theoretical Constraints on Multi-Dimensional Retiming Design Techniques

N. L. Passos, D. C. Defoe, R. J. Bailey, R. Halverson, R. Simpson
Department of Computer Science
Midwestern State University
Wichita Falls, TX 76308

ABSTRACT

Image signal processing depends on computation intensive programs, which include the repetition of sequences of operations coded as nested loops. An effective technique in increasing the computing performance of such applications is the design and use of Application Specific Integrated Circuits using loop transformation techniques, and in particular, multi-dimensional (MD) retiming. The MD-retiming method improves the instruction-level parallelism of uniform loops. While many have written about the multi-dimensional retiming technique, no results have been published on the possible limitations of its application. This paper presents an analysis of that technique and its constraints when applied to nested loops with known index bounds, such as those found in two and three dimensional image processing.

Keywords: multi-dimensional retiming, constraints, filter design, nested loops, instruction-level parallelism, image enhancement

1. INTRODUCTION

Multi-dimensional (MD) applications such as image signal processing usually depend on time-critical sections consisting of loops of instructions. Hence, they demand high computer performance and optimization techniques. As such, a considerable amount of work has been done to improve the efficiency with which the loops are executed. The focus of that work has been directed towards achieving some degree of parallelism within the loops, whether using software (compilers) or hardware (architecture) means. However, most of these studies have focused on the application of the method without a more detailed description on their limitations. In particular, this paper looks at the restrictions on the implementation of nested loops optimized by the multi-dimensional retiming technique.

Recent research has been conducted in the scheduling of multi-dimensional applications, such as the affine-by-statement technique [3], and the index shift method [9]. In areas such as scheduling and parallel processing, algorithms have been developed to speed up the execution time of nested loops by applying loop transformations to the original problem [1, 7, 9, 10, 11, 18]. Most of these techniques require the use of a new schedule vector, i.e., a new execution order of the loop iterations that directly affects the loop bounds and indices. Software pipeline, also known as loop pipelining, is another technique that involves the overlapped execution of consecutive loop iterations in order to minimize the computational execution time [18]. However, the overlapped iterations must come from successive iterations defined by the original loop code. Multi-dimensional retiming produces overlapped iterations just as those found in software pipeline solutions, considering, however, the overlap of iterations through different dimensions of the iteration space [13]. The multi-dimensional retiming technique allows the restructuring of the loop body represented by a general form of multi-dimensional data flow graph (MDFG), while preserving data dependencies [2]. This paper studies the constraints on loop pipelining when applying an MD retiming algorithm for scheduling nested loops. In this study, MD retiming is used to characterize the effect of MD loop pipelining.

Chao and Sha introduced the concept of a restricted MD retiming without resource constraints [2]. This concept was applicable to a specific class of multi-dimensional data flow graphs (MDFGs), where any cycle would have a strictly non-negative total MD delay. In another study, the concept of MD retiming was re-introduced based on the idea of a *schedule-based multi-dimensional retiming* [15]. In that method, a *feasible linear schedule* allows the restructuring of the loop body represented by a general form of MDFG, while preserving data dependencies, and improving the existing parallelism. By associating each execution instance of the loop with an integral index in a Cartesian space, one can compute the multi-dimensional distance between the production and consumption of each data value. Such a distance is then represented as a multi-dimensional delay on the MDFG. The application of multi-dimensional retiming to the MDFG permits the iterations of the original loop body to be naturally overlapped, and makes the existent parallelism explicit.

Under the assumption that each functional unit can execute in one time unit and the memory is local, i.e., it can be accessed in a speed compatible with the optimized processor design, the schedule length associated with the number of control steps required to execute all operations in the loop body is reduced. Such control steps are equivalent to the clock cycles of the circuit design and the reduction on the schedule length improves the overall execution time of the application.

A simple example of improving the performance of a multi-dimensional problem consists of the loop transformation applied to a filter [8], represented by the transfer function:

$$H(z_1, z_2) = \frac{1}{(1 - c_1 * z_1^{-1} - c_2 * z_2^{-1})}$$

A multi-dimensional data flow graph representing this problem is shown in figure 1. Nodes represent operations and edges represent data dependencies. The labels on the edges indicate the multi-dimensional distance between iterations. A computer programming language such as C or Fortran would describe the filter by:

```

for i = 1 to M
  for j = 1 to N
    y(i, j) = c1 * y(i-1, j) + c2 * y(i, j-1) + x(i, j)
  
```

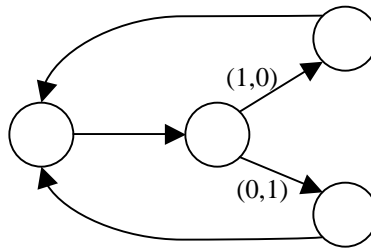


Figure 1. MDGF representing a simple filter

A pre-conceived idea, when implementing the filter design in software, is that the execution order of the nested loop must be row-wise, ignoring all the possibilities of optimization mentioned earlier. In a possible approach used on hardware implementations, the loop could be coded in such a way to describe the execution sequence through a schedule vector parameter, *sch*, as shown below [17]:

```

for (i, j) = (1, 1) to (M, N) sch (0, 1)
  y(i, j) = c1 * y(i-1, j) + c2 * y(i, j-1) + x(i, j)

```

In this new construct, the iterations are doubly indexed from the initial index to the last possible one. The *sch* parameter establishes the execution path along the universe of iterations.

Many papers have been written about multi-dimensional retiming implementation techniques [2, 4, 5, 6, 12, 13, 14, 16], however no results have been published on the possible limitations of their application. In order to evaluate the constraints on the application of multi-dimensional retiming in the design and implementation of the solution to the problem, those implementations are divided into two categories: software implementation and hardware implementation.

For example, the loop shown above is equivalent to the four elementary operations:

```

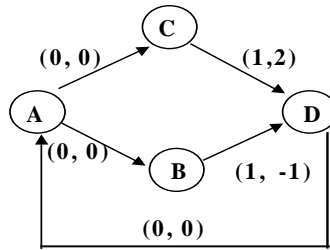
A = x(i, j) + D
B = c1 * y(i-1, j)
C = c2 * y(i, j-1)
D = B + C

```

these operations could be executed concurrently after the application of MD-retiming. However, if M and N, the boundaries of the nested loop, are equal to one, then a full instruction level parallelism (concurrent execution of all instructions in the loop body) is not possible (there are no iterations to be overlapped). The study presented in this paper gives an overview of multi-dimensional retiming and examines the limitations on the software and hardware implementations.

2. BACKGROUND

A nested loop computation is modeled as a cyclic data flow graph, which is called an MD data flow graph (MDFG). A valid MDFG is a directed graph represented by the tuple (V, E, d, t) , where V is the set of operation nodes in the loop body, E is the set of directed edges representing the dependence between the nodes, a function d represents the MD-delay associated to an edge, and a function t returns the time required for computing a node [12]. In this study, t is assumed to be one time unit for all operations. An example of a valid MDFG and its corresponding loop body is presented in Figure 2.



(a)

```

for i = 0 to ...
  for j = 0 to ...
    D(i, j) = B(i-1, j+1) + C(i-1, j-2)
    A(i, j) = 5 + D(i, j)
    B(i, j) = 3*A(i, j)
    C(i, j) = 6*A(i, j)
  end j
end i

```

(b)

Clock cycle	Operations
0	D
1	A
2	B, C

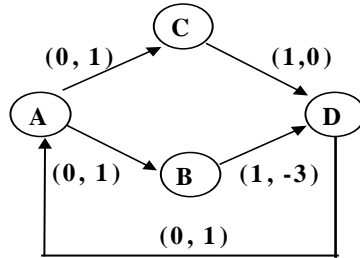
(c)

Figure 2. (a) A valid MDFG. (b) Corresponding loop body. (c) Corresponding schedule table

Multi-dimensional retiming techniques can be used to optimize this MDFG. As it was described earlier, multi-dimensional retiming is an important sequential logic optimization technique for reducing the clock period by repositioning delays (hardware registers) to obtain instruction-level parallelism. A possible retimed MDFG and its corresponding loop body for the example presented in Figure 2 is shown in Figure 3.

To traverse an iteration of the loop in the MDFG represented in Figure 2, a minimum of three clock cycles is required (always based on the assumption that each instruction is executed in one clock cycle). Operation D can be executed in the first clock cycle since its required input data has been computed on previous iterations. Operation A must then be executed in the next clock cycle since it depends on D. Finally, operations B and C can be executed in parallel in the third clock cycle since they both depend on A but do not depend on each other. On the other hand, to traverse one iteration of the retimed loop in the MDFG shown in Figure 3, only one clock cycle is required. The reason for such improvement is that operations A, B, C, and D can now be executed in parallel since there are no intra-iteration dependencies between them.

Multi-dimensional retiming is achieved by pushing an MD-delay from incoming edges of a node to its outgoing edges. If the incoming edge has zero delays, it becomes necessary to apply a retiming function to the corresponding source node to prevent a graph with negative delays in its edges.



(a)

```

for i = 0 to ...
  for j = 0 to ...
    D(i, j+2) = B(i-1, j+3) + C(i-1, j)
    A(i, j+1) = 5 + D(i, j+1)
    B(i, j) = 3 * A(i, j)
    C(i, j) = 6 * A(i, j)
  end j
end i
  
```

(b)

Clock Cycle	Operations
0	D, A, B, C

(c)

Figure 3. (a) A retimed MDFG. (b) Its corresponding loop body. (c) Corresponding schedule table

The retiming of the MDFG in Figure 2 was accomplished by following the procedures outlined below:

- Choose a node to retime (in the example, the chosen node is D)
- Choose an appropriate retiming vector to ensure full parallelization, example $r = (0, 2)$
- Subtract the retiming vector $(0, 2)$ from edges CD and BD (incoming edges of D) and add $(0, 2)$ to D's outgoing edge, DA.
- The code representing the loop is then modified according to the retiming process.

In the example, after retiming D by $(0, 2)$, A was retimed by $(0, 1)$. In the next sections, the limitations on the application of such a method are discussed.

3. SOFTWARE IMPLEMENTATION

Although the software implementation of the MD retiming has been proven to be an effective technique for parallelizing the traversal of nested loops for computationally intensive applications, constraints exist which must be taken into account. Some of these constraints come from the usual point of view that the nested loop is processed according to the sequence of the innermost loop index first, which we call row-wise execution. A simple transformation like swapping the innermost index with the outermost index would produce a sequence of execution that one could call column-wise.

For example, consider the multi-dimensional data flow graph in figure 4. It is easy to verify that if $n = 0$ and $k < 3$, then, retiming D and A by some vector $(0, p)$ would not satisfy the goal of re-distributing the delays among all edges in the graph. The same happens for $m < 3$ and possible retiming vectors of the form $(q, 0)$. Finally, if the loop had only one occurrence, i.e., the loop boundaries were both 1, then no parallelism could be obtained. This last constraint is equally applicable to a software or hardware implementation of the retimed loop.

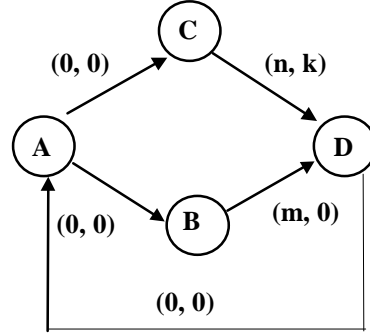


Figure 4. MDFG highlighting the constraints of software retiming

This study begins by evaluating the constraints imposed by the limitation on the number of iterations comprising the loop. Since this limit is directly associated to the size of the iteration space, it is called *spatial constraint* and it is formally defined as follow.

Definition 1: Given a k -level nested loop N , controlled by the set of indices $I = \{i_0, i_1, ..i_k\}$, whose values vary, in unitary increments, in the range $L = \{l_0, l_1, ..l_k\}$ to $U = \{u_0, u_1, ..u_k\}$ where L is the set of lower boundaries for the indices and U is the set of maximum values, such as $l_j \leq i_j \leq u_j$, then the *spatial constraint* Sc of the loop is defined as

$$Sc = [(u_0 - l_0 + 1), (u_1 - l_1 + 1), \dots, (u_k - l_k + 1)]$$

The Sc tuple determines the boundaries of the iteration space as if it had its lower bound equal to $(0, 0, \dots, 0)$, i.e., the tuple establishes the distance between the boundaries of each of the indices required to complete the loop execution and the origin of the iteration space. According to the process of using MD retiming, we can now establish the relation between the maximum retiming operation and the spatial constraint according to the following lemma.

Lemma 1: Given a k -level loop N with spatial constraint $Sc = [s_0, s_1, \dots, s_k]$, the multi-dimensional retiming technique will be able to achieve full parallelism of the loop body instructions if the maximum retiming vector r applied to any node u , $r(u) = (r_0, r_1, ..r_k)$ satisfies the following condition:

$$r(u) < Sc \text{ or } r_j < s_j \quad \forall 0 \leq j \leq k.$$

A second constraint originates from the row-wise (column-wise) execution of the nested loop, required by the software implementation. In this case, the nested loop is supposed to be retimed by some value that maintains the original execution sequence. This optimization is then constrained by the number of one-dimensional delays found in any of the cycles that may exist in the MDFG representing the loop. This number of delays can be easily determined by a linearization function applied to the dependence vector. Such a linearization function is defined by the following expressions:

$$\lambda_r(d) = \min \{ [\prod_1^k (u_h - l_h + 1), \prod_2^k (u_h - l_h + 1), \dots, \prod_k^k (u_h - l_h + 1), 1] \times (w_0, w_1, ..w_k)$$

$$\lambda_c(d) = \min \{ [\prod_{k-1}^0 (u_h - l_h + 1), \prod_{k-2}^0 (u_h - l_h + 1), \dots, \prod_0^0 (u_h - l_h + 1), 1] \times (w_0, w_1, ..w_k)$$

where $d = (w_0, w_1, ..w_k)$ in a row-wise approach, and

$$\lambda(d) \in \{ \lambda_r(d), \lambda_c(d) \}, \text{ depending on the execution approach used (row-wise or column-wise)}$$

In summary, λ amounts to the number of delays found in an equivalent one-dimensional problem, being executed row-wise (λ_r) or column-wise (λ_c), whichever has more delays. However, in order to achieve full parallelism, there is a need to redistribute the dependency delays among all edges in the cycles. This distribution imposes a new limit, called *linear constraint*, which is defined next.

Definition 2: Given an MDFG G , representing the nested loop N , containing a cycle C comprising a set of p edges $\{e_0, e_1, \dots, e_p\}$ and a set of data dependence vectors $D = \{d_0, d_1, \dots, d_m\}$, such that $m \leq p$, then the *linear constraint* on C is the summation of the linear delays found in cycle C divided by the number of edges constituting the cycle $(p+1)$.

This definition can be formulated as

$$L_c = \sum_m \lambda / (p+1)$$

The goal of software retiming is to ensure that there are no zero delays between operations (nodes in the MDFG). This is accomplished by redistributing the MD-delays throughout the MDFG as explained in the previous section. By knowing the linear constraint of the nested loop being optimized, one may infer the possibility of a software implementation of the loop transformed by the application of a MD retiming by verifying the lemma below.

Lemma 2: Given an MDFG G representing the nested loop N , whose minimum linear constraint among all its cycles is L_c , the multi-dimensional retiming technique will be able to achieve full parallelism of the loop body instructions if $L_c \geq 1$.

For example, consider the MDFG in **Figure 4**. The existing cycles are ACDA and ABDA, which implies that they have 3 edges. Now let us assume that the boundary conditions for this loop are 3 and 3. It is clear that we would need to retime node D by (0,2) or (2,0) and node A by (0,1) or (1,0) in order to allow a software implementation of the optimized loop. According to Lemma 1 the *spatial constraint* does not prevent the optimization. However, in order to retime node D, we need to subtract the retiming vector from the dependencies CD and BD and that implies that the summation of the linearized delays must be greater or equal to two. Applying the formulas discussed above, we obtain $3m > 2$ and $3n + k > 2$ in order to retime node D by (0,2). If one had tried to retime D by (2,0), then the minimum number of linearized delays would be 6, unless a loop interchange is applied and a the column-wise execution is used in the software implementation. Figure 5 shows the theoretical retimed values for a fully parallel solution of the example.

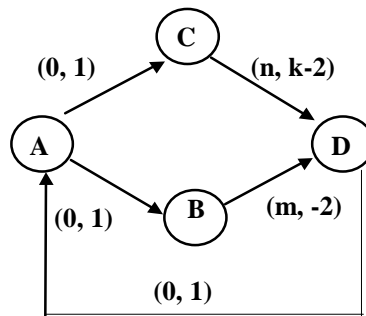


Figure 5. Retimed MDFG for highlighting the constraints of software retiming

4. HARDWARE IMPLEMENTATION

Given the limitations of the software implementation of the MD retiming, the hardware implementation promises to be a more suitable alternative for processing computationally intensive applications. Hardware retiming implementation relaxes the linear constraints found on the software implementation by allowing changes in the execution sequence of the loop according to a new schedule vector. The basic change implies the use of a new vector on the computation of the linearization of the delays, allowing such a value to be raised to numbers larger than the linear constraint. With more flexibility on the linear constraint, the spatial constraint becomes the most significant restriction on the hardware implementation. The results

for software limitations can then be easily extended to the hardware point of view. The linear constraints must still be evaluated. However it has been shown that a well-chosen scheduling vector can eliminate any linear restriction [15].

Figure 6 shows a typical implementation of the filter presented on section 1 [17]. In this implementation, one section of the hardware is responsible for updating the indices of the array according to a pre-defined schedule vector, which may cause the sequence of execution of the iteration space to follow a wavefront approach (diagonal-wise instead of row- or column wise).

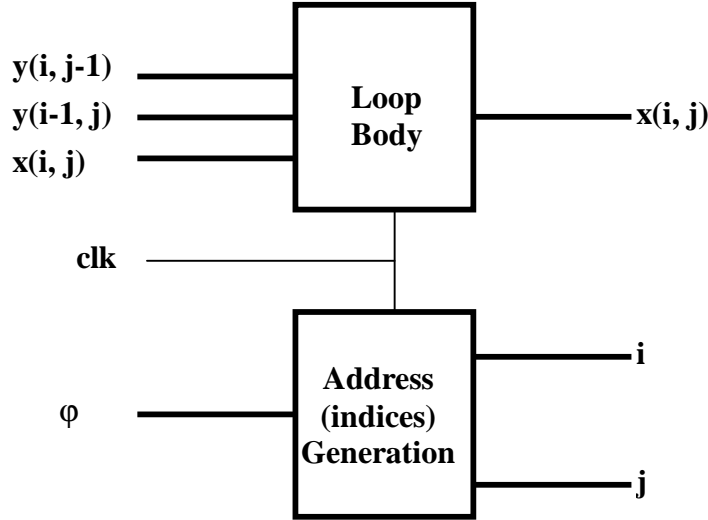


Figure 6. Typical hardware architecture for MD retimed filter.

Under these new characteristics of the problem, while the constraints specified for the software implementation on Lemma 1 do not change, i.e., the same spatial constraints are applicable to the hardware design, the same does not happen to the linearization mechanism. Its new form must include the scheduling vector controlling the execution sequence. Let us assume that a schedule vector $\varphi = (\varphi_0, \varphi_1, \dots, \varphi_k)$ is used to implement a k -level nested loop. The linearization function used in the delay vectors will be defined by the following expression:

$$\lambda(d) = (\varphi_0, \varphi_1, \dots, \varphi_k) \times (w_0, w_1, \dots, w_k)$$

where $d = (w_0, w_1, \dots, w_k)$ in a row-wise approach. By selecting a convenient schedule vector φ , we can guarantee that $\lambda(d)$ will be always greater or equal to one. So our linearized delays would be at least one unit. When applying the multi-dimensional retiming vector to the graph, the new constraint becomes the fact that the linearized retiming can not be larger than the total number of delays in a cycle involving the dependencies being considered. A trivial solution is to choose a retiming vector $r = \times (r_0, r_1, \dots, r_k)$, such that

$$(\varphi_0, \varphi_1, \dots, \varphi_k) \times (r_0, r_1, \dots, r_k) = 0.$$

Under these conditions, there will be a retiming process that, apparently, does not move any linearized delay making the linear constraint non-existent. By knowing these assumptions, the non-existence of a linear constraint for the nested loop being optimized can be verified by the lemma below.

Lemma 3: Given an MDFG G representing a k -level nested loop N , a multi-dimensional retiming $r = \times (r_0, r_1, \dots, r_k)$ and a scheduling vector $\varphi = (\varphi_0, \varphi_1, \dots, \varphi_k)$, if $(\varphi_0, \varphi_1, \dots, \varphi_k) \times (r_0, r_1, \dots, r_k) = 0$, then there is no linear limit to the parallelization of the loop.

This lemma can be proven using the multi-dimensional retiming theory presented in [13].

5. SUMMARY

Image signal processing depends on computation intensive programs, which include the repetition of sequences of operations coded as nested loops. An effective technique in increasing the computing performance of such applications has been the design and use of ASICs using loop transformation techniques, and in particular, multi-dimensional retiming. This method improves the instruction level parallelism of uniform loops. While MD-retiming is an effective method for improving parallelism in loop transformation, this paper has shown that both software and hardware implementations are faced with constraints based on the index bounds and dependencies found in the loop. The estimation of these constraints during the design phase of the implementation allow a fast decision on the process to be used (hardware or software) and also a confirmation of its feasibility.

ACKNOWLEDGEMENTS

This work was partially supported by the National Science Foundation under Grant No. MIP 97-04276.

REFERENCES

1. L.-F. Chao, A. LaPaugh, and E. H.-M. Sha, "Rotation Scheduling: A Loop Pipelining Algorithm," in the *Proc. 30th ACM/IEEE Design Automation Conference*, Dallas, TX, June, 1993, pp. 566-572.
2. L.-F. Chao and E. H.-M. Sha, "Static Scheduling of Uniform Nested Loops," in the *Proceedings of 7th International Parallel Processing Symposium*, Newport Beach, CA, April, 1993, pp. 1421-1424.
3. A. Darte and Y. Robert, "Constructive Methods for Scheduling Uniform Loop Nests," in the *IEEE Transactions on Parallel and Distributed Systems*, August, 1994, Vol. 5, no. 8, pp. 814-822.
4. T. C. Denk, M. Majundar and K. K. Parhi, "Two-Dimensional Retiming with Low Memory Requirements," in the *Proceedings of the 1996 International Conference on Acoustics, Speech, and Signal Processing*, 1996, Vol. 6, pp. 3330-3333.
5. T. C. Denk and K. K. Parhi, "Two-Dimensional Retiming," in the *IEEE Transactions on VLSI*, Vol. 7, No. 2, June, 1999, pp. 198-211.
6. F. Fernandez, and A. Sanchez, "Application of Multidimensional Retiming and Matroid Theory to DSP Algorithm Parallelization," in the *Proceedings of 25th EUROMICRO Conference*, September, 1999, Vol. 1, pp. 511 - 518.
7. L. Lamport, "The Parallel Execution of DO Loops," in the *Communications of the ACM SIGPLAN*, 17(2), February, 1974, pp. 82-93.
8. C. A. Lindley, *Practical Image Processing in C*, New York, NY: John Wiley & Sons, 1991.
9. L.-S. Liu, C.-W. Ho and J.-P. Sheu, "On the Parallelism of Nested For-Loops Using Index Shift Method," in the *Proc. of the 1990 International Conference on Parallel Processing*, 1990, Vol. II, pp. 119-123.
10. L. E. Lucke and K. K. Parhi, "Generalized ILP Scheduling and Allocation for High-Level DSP Synthesis," in the *Proc. Custom Integrated Circuits Conference*, 1993, pp. 5.4.1-5.4.4
11. K. K. Parhi and D. G. Messerschmitt, "Fully-Static Rate-Optimal Scheduling of Iterative Data-Flow Programs Via Optimum Unfolding," in the *Proc. International Conference on Parallel Processing*, 1989, Vol. I, pp. 209-216.
12. N. L. Passos, E. H.-M. Sha, and S. C. Bass, "Loop Pipelining for Scheduling Multi-Dimensional Systems via Rotation," in the *Proceedings of 31st Design Automation Conference*, 1994, pp. 485-490.
13. N. L. Passos and E. H.-M. Sha "Achieving Full Parallelism using Multi-Dimensional Retiming," in the *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 11, November 1996, pp. 1150-1163.
14. N. L. Passos and E. H.-M. Sha, "Scheduling of Uniform Multi-Dimensional Systems under Resource Constraints," in the *IEEE Transactions on VLSI Systems*, December, 1998, Volume 6, Number 4, pages 719-730.
15. N. L. Passos, E. H.-M. Sha, and S. C. Bass, "Schedule-Based Multi-Dimensional Retiming," in *Proceedings of 8th International Parallel Processing Symposium*, Cancun, Mexico, April, 1994, pp. 195-199.
16. N. L. Passos, and E. H.-M. Sha, "Push-up scheduling: optimal polynomial-time resource constrained scheduling for multi-dimensional applications," in the *Proc. of the In. Conf. On Computer Aided Design*, November, 1995, pp. 588-591.
17. N. L. Passos and E. H.-M. Sha, "Synthesis of Multi-Dimensional Applications in VHDL," in the *1996 International Conference on Computer Design*, Austin, TX, October, 1996, pp. 530-535.
18. M. Wolf and M. Lam, "A Loop Transformation Theory and an Algorithm to Maximize Parallelism", *IEEE Trans. On Parallel and Distributed Systems*, vol. 2, n.4, pp. 452-471, 1991.