DESIGN OF 2-D FILTERS USING A PARALLEL PROCESSOR ARCHITECTURE

Nelson L. Passos *Robert P. Light*
Midwestern State University
Wichita Falls, TX 76308

Virgil Andronache *Edwin H.-M. Sha*
University of Notre Dame
Notre Dame, IN, 46556

ABSTRACT

Two-dimensional filters are usually part of the implementation of digital image processing applications. These filters process recursive sets of instructions and require high computational speed. Optimized implementations of these filters depend on the use of Application Specific Integrated Circuits (ASICs). A system with multiple parallel processing units is a feasible design option able to achieve the required computational performance. In this paper, a loop transformation algorithm, which allows the efficient utilization of a parallel multiprocessor system, is presented. Uniform nested loops representing the image filters and the available processors are modeled as multi-dimensional data flow graphs. A new loop structure is generated so that an arbitrary number of processors available in the system can run in parallel. An example and a description of the algorithm are presented in the paper.

Keywords: retiming, nested loops, filter, multiprocessor, image processing.

1. INTRODUCTION

Image enhancement and edge detection are well-known digital image signal processing applications that may require two-dimensional (2-D) filter-like computational solutions. These applications usually depend on computation intensive code sections, consisting of the repetition of sequences of operations. They are also characterized by the multi-dimensionality of the data involved. An effective technique in improving the computing performance of such applications has been the design and use of Application Specific Integrated Circuits (ASICs). This paper presents a new technique applicable to the design of a 2-D filter system using multiple parallel processors. A multi-dimensional retiming algorithm embedded in this new technique provides the fully parallel utilization of the available processors, thus reducing the overall execution time of the filter function.

Parallel architectures are an important tool in ASIC design. However, these architectures require a careful partitioning of the problem in order to improve the utilization of the parallel processors [2, 17, 24]. During the circuit design phase, nested loop structures can be coded using hardware description languages, such as VHDL constructs, in order to reduce the design time. However, in VHDL, the loop control indices will represent the number of times a section of the circuit will be replicated in the final synthesis under the assumption that there are no physical or cost constraints in the circuit implementation [23]. In this paper, a multi-dimensional retiming technique is used to transform the loop in such a way to produce the parallel solution for the problem for a given number of processing units. Such a solution can then be implemented on a standard multiprocessor architecture.

Retiming was originally proposed by Leiserson – Saxe, focusing on improving the cycle time of one-dimensional problems [13]. Most work done in this area, is subject to limitations imposed by the number of delays (memory elements) existing in a cycle of a data flow graph representing the problem [3, 6, 10, 11, 12, 16, 22, 25]. Other methods focus on multi-processor scheduling and are also applicable to one-dimensional problems [7, 8, 14, 16, 18]. This study focuses on the parallelism inherent to multi-dimensional applications, ignored by the one-dimensional methods.

Retiming and other loop transformations have since been applied in areas such as scheduling and parallel processing, with the main goal of exploiting fine-grain parallelism in the loop body [4, 15]. Due to the different focus in obtaining parallelism, those techniques are not aimed to improve the execution of parallel iterations in multiprocessor systems. Research by Passos and Sha extended the retiming concept to multi-dimensional (MD) applications [19]. The multi-dimensional retiming concept is used in this paper to model the partitioning of the loop among the available processors. Multi-dimensional retiming brings some advantages to the process, since it is readily applicable to the multi-dimensional fields considered, eliminating the need for a loop transformation that converts the original problem to one dimension. Another significant advantage of MD retiming is that there are no restrictions on its applicability, not being constrained by the characteristics of the one-dimensional methods.

As an example, consider a filter with transfer function:

$$H(z1,z2)=\frac{1}{1-\sum_{n1=0}^{1}\sum_{n2=0}^{1} g(n1,n2)*z1^{-n1}*z2^{-n2}}$$

for n1, n2 not simultaneously zero. A multi-dimensional data flow graph (MDFG) can be used to represent this problem as seen in figure 1(a). Figure 1(b) shows a digital circuit design with multiple functional units (multipliers and adders) and memory elements comprising a single processor system designed to solve the filter problem represented in figure 1(a).

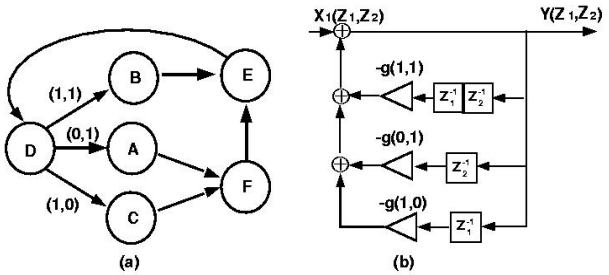

Figure 1. (a) MDFG (b) circuit implementation

The 2-D delay (1,1) in the MDFG is usually implemented by a FIFO structure equivalent to a serial implementation of z_1^{-1} and z_2^{-1} elements. The delay (0,1) is equivalent to z_2^{-1} and the delay (1,0) to z_1^{-1}. The sequential execution time for this design is equivalent to the serial execution of three additions and one multiplication. The nested loop representation of the filter requires a row-wise computation, where the z_2^{-1} element represents only one delay or storage element. Assuming that two identical processors, with the internal design shown in figure 1(b), are available for parallel execution of this loop, an optimization problem arises. The z_2^{-1} delay becomes a direct dependency between two consecutive iterations being executed in the two processors. The same delay also produces a one-delay dependency between two consecutive utilizations of the two-processor system. This implies a non-parallel execution of the two processors, which the 1-D retiming technique cannot change due to the constancy on the number of delays in the cycle involving the two processors and containing the z_2^{-1} delay. Using multi-dimensional retiming on the MDFG representing the parallel implementation of the circuit, that constraint can be eliminated. In this paper, the focus will be on obtaining parallelism between separate iterations of the loop that can be run on different processors, rather than the fine grain parallelism that optimizes the execution on each individual processing element. As a result, the graph to be optimized will be representative of the iterations being executed in parallel.

As an example, Figure 2(a) shows the graph for the filter described earlier, implemented in a two-processor system, however still subject to a sequential execution. After applying the MD retiming operation, the graph will appear as in figure 2(b), where the delays between processors guarantee full parallelism.

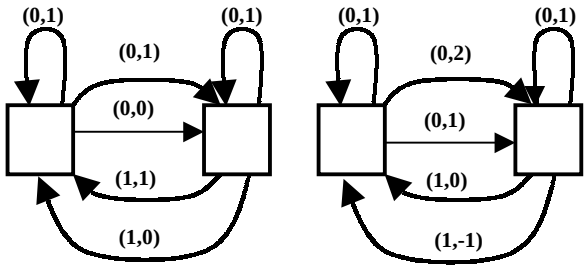

Figure 2. (a) Multiprocessor graph (b) Retimed graph

This paper presents the new loop transformation method, based on the multi-dimensional retiming technique, and its applicability to the design of filters implemented in multiprocessor systems. It starts in the next section with an introduction to the basic principles involved, including an overview of multi-dimensional retiming. Section 3 shows the theoretical aspects of the processor allocation technique, followed by a description of the utilization of the method in a more complex example. A final section summarizes the concepts introduced in the paper.

2. BASIC PRINCIPLES

A multi-dimensional data flow graph (MDFG) *G = (V, E, d, t)* is a node-weighted and edge-weighted directed graph, where *V* is the set of computation nodes, *E* is the set of dependence edges equivalent to the circuit data paths, *d* is a function representing the MD delay between two nodes, and *t* is a function representing the computation time of each node.

An iteration is the execution of each node in *V* exactly once. Iterations are described by a vector *w*, equivalent to a multi-dimensional index, starting at *(0,…,0)*. Iteration dependencies are represented by vector weighted edges. An edge *e* from *u* to *v* with delay vector *d(e)* means that the computation of the node *v* at iteration *w* depends on the execution of node *u* at iteration *w – d(e)*. Thus, an edge with delay *(0,…,0)* represents a data dependence within the same iteration. A legal MDFG

must have no zero-delay cycle, i.e. the summation of the delay vectors along any cycle cannot be *(0,…,0).*

The iteration space of a loop is defined to be the region in the MD discrete Cartesian space whose points correspond one-to-one to the iteration indices. This paper considers loops that present the characteristic of constant dependence vectors, i.e. their data dependencies are at a constant distance in the iteration space. These loops are called uniform loops.

Reexamining the example presented in the previous section, the 2-D filter for an image of N by N pixels can be represented by the nested loop below:

```
for (i=0; i<N; i++)
   for (j=0; j<N; j++)
   {
    y(i,j) = c1*y(i-1,j-1) + c2*y(i,j-1) + c3*y(i-1,j) + x(i,j)
   }
```

In the MDFG shown in figure 1, the nodes A, B and C represent the three required multiplications, while the remaining nodes show the necessary additions.

A multi-dimensional retiming function $r:V \rightarrow Z^n$ applied to a graph $G=(V, E, d, t)$, redistributes the nodes in the original iteration along the iteration space of the MDFG G. The result of this process is a new graph, $G_r=(V, E, d_r, t)$, in which each iteration still contains one instance of every node in G. This transformation is equivalent to a redistribution of the delay elements in the graph. The multi-dimensional retiming function $r(u)$ for a node $u \in$ G represents the offset between the original iteration containing u and the one after retiming. This offset change implies a corresponding change in the delay vectors (edge weights) associated with the MDFG, so that the original dependencies are preserved. Thus, for an edge e between nodes u and v, $d_r(e) = d(e) + r(u) - r(v)$.

3. PROCESSOR ALLOCATION

Given an iteration space representing the execution of a two-dimensional nested loop, the allocation of multiple processors is done along a line parallel to the i-axis (outermost index of the loop). A processor dependency graph (PDG) P, defined below, shows the dependencies among the different processors used in the execution of the nested loop.

Definition For a given *MDFG G= (V, E, d, t)* and k processors, a PDG $P = (\Pi, \varepsilon, \delta)$ is a node weighted and edge weighted direct graph, where Π is the set of processors, with $|\Pi| = k$, ε is the set of dependence edges between processors and δ is a function representing the MD delay between two processors.

An MDFG is transformed to a processor dependency graph (PDG) according to the number of available processors established by the filter designer. The PDG shows the dependency edges among the various processors, assuming they are running consecutive iterations in the original loop code. Figure 3(a) shows the inter-iteration dependencies originated in processor P0 and required in the example seen in figure 1. Figure 3(b) shows the PDG representation of the same graph in the two-processor arrangement.

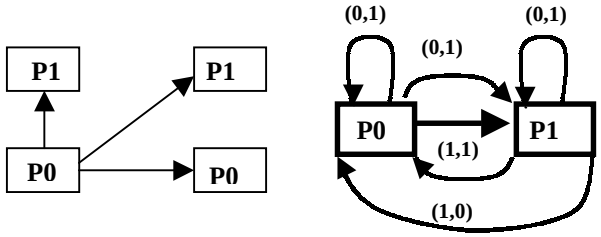

Figure 3: (a) Inter-iteration dependencies (b) PDG representation

In the allocation of processors, and implicitly in the application of the retiming technique, two properties of MDFGs that model loops written with regular constructs need to be noticed [9, 20].

Property 3.1 Given an MDFG $G = (V, E, d, t)$, a retiming function $r = (r_x, r_y)$, a node $u \in V$ and an edge $e \in E$ such that $u \rightarrow^e u$, applying the retiming function r to u, $r(u)$, does not change $d(e)$.

Property 3.2 given a nested loop of the form:

```
for (i = 0;  i < N;  i++)
      for (j=0;  j < N;  j++)
            a[i,j] = f(b[i',j'])
            b[i,j] = g(a[i",j"])
```

with dependence vectors (x,y), where x indicates the iteration distance according to the outermost loop and y represents the innermost loop, then $x \geq 0$.

Property 3.3 Given a nested loop as the one shown in property 3.2, represented by a PDG $P = (\Pi, \varepsilon, \delta)$, and a pair of processors p and $q \in \Pi$ assigned to iteration instances (i, j) and (i+1, j) respectively, then for any edge e = p → q , $\delta(e) > (0,0)$.

As a consequence of property 3.2, a dependence vector such as (-2,3) will not be in the set of dependence vectors associated with the loop. Using these properties, a transformation algorithm can be applied to the PDG for optimal performance. In this algorithm, the memory access time is assumed to be uniform and the loop structures under consideration have constant

dependencies. With these assumptions, the MDFG representing the original problem is translated into a PDG according to the lemma below:

Lemma 3.4 A *processor dependency graph* PDG $P=(\Pi, \varepsilon, \delta)$ is such that, given an MDFG $G=(V, E, d, t)$ and k processors, then $\forall\ e\in E$ connecting nodes $u, v\in V$ with $d(e) = (x,y)$ there exists $e'\in\varepsilon$ connecting processor n, which contains node u, to processor m, which contains node v, with $m, n\in\Pi$, $0\leq m, n < k$, and:

$$m = (n+x) \bmod k$$
$$\delta(e') = (x',y') = (\text{int}((n+x)/k),y)$$

In order to guarantee the parallelism of the multiprocessor system, a multi-dimensional retiming function $r(u) = (0,r_y)$ is applied to the PDG to change the dependence edges eliminating sequential processing among different processors. It can be proven that $(0,r_y)$ is always a valid multi-dimensional retiming vector in a multiprocessor environment and, therefore, a convenient retiming function such as $(0,1)$ can be applied and will result in the elimination of direct dependencies from the PDG as stated in the theorem below:

Theorem 3.5: Given a PDG $P=(\Pi,\varepsilon,\delta)$ with at least one edge $e\in\varepsilon$ such that $\delta(e) = (0,0)$, there exists a retiming function r of the form $(0,r_y(u))$ for all $u\in\Pi$, that applied to P creates $P_r=(\Pi,\varepsilon,\delta_r)$ such that for any $e\in\Pi$, $\delta_r(e) > (0,0)$.

This theorem can be proven by analyzing three possible cases and considering properties 3.1, 3.2 and 3.3. The three cases are:

- $\delta(e) = (0,0)$, which will require the application of the retiming function r, resulting in a dependence $\delta_r(e) = \delta(e) + r > (0,0)$.
- $\delta(e) = (0,y)$, according to properties 3.1, 3.2 and 3.3, which after retiming will result in $\delta_r(e) > (0,0)$,
- $\delta(e) = (x,y)$, with $x > 0$, which would produce $\delta_r(e) = (x, y-r_y)$. In this case, considering that $x > 0$, then $\delta_r(e) > (0,0)$.

Just as in the case of MDFGs, full parallelism is achieved when all edges between any two processors become non-zero delay edges [19]. The loop bodies are then modified according to the retiming function applied to the PDG. Figure 2(b) shows the inter-processor dependencies after the retiming technique was applied to the PDG in figure 3. A synchronization function is now needed to trigger the execution of each processor after an initialization stage, usually called prologue, which is inherent to retiming.

The synchronization function call is controlled by the values of the indices of the loop and the number of the processing unit. When the processor has executed the instructions comprising its mandatory prologue and has already initialized the data required for the parallel execution then the function is activated. The general format of the final code of the loop body for each processor is shown next:

Code for processor #n (k =number of processors)

```
for (i = n;  i < N;  i+k)
  for (j=0;  j<N; j++)
      if ((i==n) && (j==(r_y(p_n) - r_y(p_{n+1}))
                SYNC(p_{n+1})
      y(i,j) = c1*y(i-1,j-1)+c2*y(i,j-1)+c3*y(i-1,j)+x(i,j)
```

The entire process is then summarized in the algorithm MRA, which is a modified version of the OPTIMUS algorithm presented in [21]:

Multiprocessor Retiming Algorithm (MRA):
Input: MDFG G = (V, E, d, t), number of processors k;
Output: retimed G

```
PDG P = (Π,ε,δ) ← transform(G,k);
r_v ← (0,1);
MC(∀u ∈ Π) ← 0
MCmax ← 0
QueueΠ ← ϕ
/* remove original edges with positive delays */
∀e ∈ ε
      ε ← ε - {e , s.t. δ(e) >(0,…,0)}
/* queue non-dependent nodes */
QueueΠ ← QueueΠ ∪ {u ∈ Π, s.t. indegree(u) = 0}
while QueueΠ ≠ ϕ
      get(u, QueueΠ)
      /* check if u needs to be retimed */
      if ∃ v ∈ Π s.t. δ(u → v) = (0,0)
            /* adjust the MC(u) value */
            MC(u) ← MC(u)+1
            MCmax ← max {MC(u),MCmax}
      endif
      /* propagate the values to successor nodes of u */
      repeat ∀ v ∈ Π such that u → v
            indegree(v) ← indegree(v)- 1
            MC(v) ← max{MC(v), MC(u)}
            /* check for new non-dependent nodes */
            if indegree(v) = 0
                  QueueΠ ← QueueΠ ∪ {v}
            endif
      endrepeat
endwhile
/* compute the multi-dimensional retiming */
∀ u ∈ Π
      r(u) = (0,(MCmax - MC(u))*r_v)
End
```

The algorithm MRA will produce the necessary retiming values for each node (processor) represented in the graph, allowing the transformation of the loop into its parallel format.

4. EXAMPLE

In this section, the processor allocation algorithm is applied to the IIR section of a 2D-filter design initially presented in [5]. Its transfer function is given by:

$$H(z_1, z_2) = \frac{\sum_{n1=0}^{2} \sum_{n2=0}^{2} f(i,j) * z_1^{-i} * z_2^{-j}}{\sum_{n1=0}^{2} \sum_{n2=0}^{2} g(i,j) * z_1^{-i} * z_2^{-j}}$$

This filter requires a loop with iterations containing eight multiplications and seven additions. The corresponding PDG for a three-processor system, implementing this filter, is shown in figure 4.

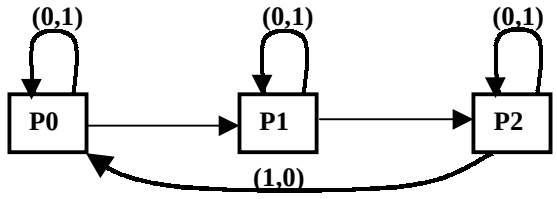

Figure 4. Processor Dependency Graph for the IIR problem

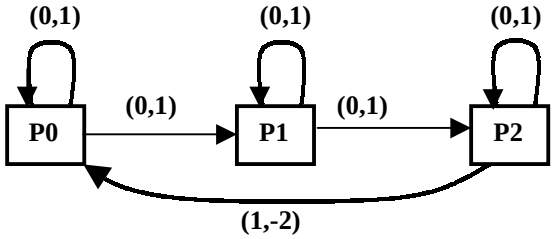

Figure 5. Retimed PDG for the IIR problem.

After applying the multi-dimensional retiming to the processor dependency graph, the PDG changes to the structure presented in figure 5. In this PDG, *P0* has been retimed twice using the retiming function *(0,1),* resulting in an overall retiming value *r(P0) = (0,2). P1* has been retimed once, with the overall retiming *r(P1) = (0,1*). *P2* did not undergo retiming and therefore its retiming value is *r(P2) = (0,0).* These retiming values introduce multi-dimensional delays in all edges representing dependencies between processors. These new delays represent stored data that allow the parallel execution of the tasks assigned to the different processing elements. The code for processors 0 and 1 under the stated conditions becomes:

```
k =  3 (number of processors)
/* processor 0 */
for (i = 0; i < N; i+k)
      for (j=0; j<N;j++)
            if ((i==0) && (j==1)
                  SYNC(p1)
            endif
            filter operations
      endfor
endfor

/* processor 1 */
for (i = 1; i < N; i+k)
      for (j=0; j<N;j++)
            if ((i==1) && (j==1)
                  SYNC(p2)
            endif
            filter operations
      endfor
endfor
```

The SYNC command sends a signal to the named processor, informing that its required data has been computed and allowing it to initiate its execution sequence. Processor P0 triggers processor P1 after a first iteration has been executed, while processor P1 will trigger P2, after 2 iterations of P0 (or one of P1). The non-existence of (0,0) delays in the resulting graph shows that the iterations can be run in parallel on the three-processor system.

5. CONCLUSION

This paper has introduced an algorithm based on multi-dimensional retiming that allows an arbitrary number of processors to work in parallel on applications that make use of nested loop constructs. In particular, this paper demonstrated the application of this new method to the case of a two-dimensional image filter. The loops are modeled in the form of multi-dimensional dependence graphs, which are transformed to multi-processor equivalent versions. Such loops are retimed in order to guarantee fully parallel execution of the nested loop. After retiming, the modified code for each processor is easily generated. A synchronization signal sent between processors guarantees the availability of initial data and correct execution of the iterations distributed in different processors. The theory and basis for the algorithm were presented and an example was shown illustrating the use of the algorithm.

6. ACKNOWLEDGEMENTS

This work was partially supported by the National Science Foundation under Grants No. MIP 95-01006 and MIP 97-04276.

REFERENCES

[1] A. Aiken and A. Nicolau, "Optimal Loop Parallelization," *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation*, pp. 308-317, June, 1988.

[2] W. P. Burleson, "The Partitioning Problem on VLSI Arrays: I/O and Local Memory Complexity," *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pp. 1217-1220, 1991.

[3] L.-F. Chao, A. LaPaugh, and E. H.-M. Sha, "Rotation Scheduling: A Loop Pipelining Algorithm," *Proceedings of the 30th ACM/IEEE Design Automation Conference*, Dallas, TX, pp. 566-572, June, 1993.

[4] A. Darte and Y. Robert, "Constructive Methods for Scheduling Uniform Loop Nests," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, no. 8, pp. 814-822, 1994.

[5] R. Gnanasekaran, "2-D Filter Implementation for real-time Signal Processing," *IEEE Transactions on Circuits and Systems*, v. 35, n. 5, pp. 587-590, 1988.

[6] G. Goosens, J. Wandewalle, and H. de Man, "Loop Optimization in Register Transfer Scheduling for DSP Systems," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 826-831, 1989.

[7] R. Gupta, "Loop Displacement: An Approach for Transforming and Scheduling Loops for Parallel Execution," *Proceedings of the International Conference on Supercomputing*, pp. 388-397, November, 1990.

[8] P. Held, P. Dewilde, E. Deprettere, and P. Wielage, "HIFI: From Parallel Algorithm to Fixed-Size VLSI Processor Array," in *Application-Driven Architecture Synthesis*, ed. F. Catthoor and L. Svensson, Norwell, Massachusetts: Kluwer Academic Publishers, pp. 71-94, 1993.

[9] L. Lamport, "The Parallel Execution of DO Loops," *Communications of the ACM SIGPLAN*, 17(2), pp. 82-93, February, 1974.

[10] M. Lam, "Software Pipelining: An Effective Scheduling for VLIW Machines," *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 318-328, 1988.

[11] T.-F. Lee, A. C.-H. Wu, D. D. Gajski, and Y.-L. Lin, "Performance Optimization of Pipelined Circuits," *Proceedings of the International Conference on Computer Aided Design*, pp. 410-413, November, 1990.

[12] T.-F. Lee, A. C.-H. Wu, D. D. Gajski, and Y.-L. Lin, "An Effective Methodology for Functional Pipelining," *Proceedings of the International Conference on Computer Aided Design*, pp. 230-233, December, 1992.

[13] C. E. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry," *Algorithmica*, 6, pp. 5-35, 1991.

[14] H. Li, S. Tandri, M. Stumm and K. C. Sevcik, "Locality and Loop Scheduling on NUMA Multiprocessors," *Proceedings of the 1993 International Conference on Parallel Processing*, Vol. II, pp. 140-147, 1993.

[15] L. S. Liu, C. W. Ho, and J. P. Sheu, "On the Parallelism of Nested For-Loops Using Index Shift Method," *Proceedings of International Conference on Parallel Processing*, pp. 119-123, 1990.

[16] L. E. Lucke and K. K. Parhi, "Generalized ILP Scheduling and Allocation for High-Level DSP Synthesis," *Proceedings of the Custom Integrated Circuits Conference*, pp. 5.4.1-5.4.4, 1993.

[17] D. I. Moldovan and J. A. B. Fortes, "Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays," *IEEE Transactions on Computers*, Vol. C-35, pp. 1-12, January, 1986.

[18] K. K. Parhi and D. G. Messerschmitt, "Fully-Static Rate-Optimal Scheduling of Iterative Data-Flow Programs Via Optimum Unfolding," *Proceedings of the International Conference on Parallel Processing*, Vol. I, pp. 209-216, 1989.

[19] N. L. Passos and E. H. -M. Sha, "Achieving Full Parallelism Using Multidimensional Retiming," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 11, pp. 1150-1163, November, 1996.

[20] N. L. Passos, A. Leonardi and E. H. -M. Sha, "Nested Loops Optimization for Multiprocessor Architecture Design," in the *Proceedings of the 1998 Midwest Symposium on Circuits and Systems*, Notre Dame, IN, pp. 415-418, August, 1998.

[21] N. L. Passos and E. H. -M. Sha, "Scheduling of Uniform Multi-Dimensional Systems under Resource Constraints, " in the *IEEE Transactions on VLSI Systems*, Volume 6, Number 4, pp. 719-730, December, 1998.

[22] R. Potasman, J. Lis, A. Nicolau, and D. Gajski, "Percolation Based Scheduling," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 444-449, 1990.

[23] D. L. Perry, *VHDL*, McGraw-Hill Inc., New York, New York, 1994.

[24] W. Shang and J. A. B. Fortes, "Independent Partitioning of Algorithms with Uniform Dependencies," *IEEE Transactions on Computers*, Vol. 41, February, pp. 190-206, 1992.

[25] C.-Y. Wang and K. K. Parhi, "High Level DSP Synthesis Using the MARS Design System," *Proceedings of the International Symposium on Circuits and Systems*, pp. 164-167, 1992.