

Rotation Scheduling on Synchronous Data Flow Graphs

Rama Krishna Pullaguntla¹, Samer F. Khasawneh², and Timothy W. O'Neil³

¹Hyland Software Inc., Westlake Ohio USA

²Dept. of Computer Science, Kent State University, Kent Ohio USA

³Dept. of Computer Science, The University of Akron, Akron Ohio USA

Abstract - Scheduling loops optimally is one of the important steps in parallel processing, since many applications are made up of iterative processes. There are few iterative processes, which can produce or consume more than one unit of data. These processes can be best described using synchronous data flow graphs (SDFG) or multi-rate graphs. A great deal of research has been done to optimize SDFGs using techniques such as retiming. In this research, we apply a technique called "rotation scheduling" to reduce the execution times of SDFGs. Finally we demonstrate the contributions of our research using suitable examples.

Keywords: Rotation Scheduling, Synchronous Data Flow Graphs.

1 Introduction

Parallelism is one of the most important aspects in any application that is being developed today. Many researchers are constantly working to improve the degree of parallelism that can be explored in any type of application irrespective of whether its hardware or software. One such technique which extracts a good amount of parallelism from the iterative processes is called "Rotation scheduling on synchronous data flow graphs".

Synchronous data flow graphs, also called multirate graphs are used to represent the data flow in multiprocessor applications. They are similar to data flow graphs but the applications that are represented using SDFGs are capable of producing and consuming multiple units of data. The nodes of a SDFG represent functional elements and the edges between them denote the connections between the functional elements. These nodes produce or consume a fixed number of data tokens.

Rotation scheduling is a technique that is used to schedule iterative processes that can be represented using data flow graphs. The goal of this technique is to optimize the length of the schedule for a particular process. The output of this technique is a static schedule that indicates the start time of each node in the data flow graph. The basic idea behind rotation scheduling is to reduce the idle time of the hardware

by moving the operations that are scheduled at the beginning to the end of the schedule.

Existing research has contributed a lot towards scheduling data flow graphs with the help of several techniques. We are going to briefly discuss some of these contributions that are related to our research interests.

The basis for rotation scheduling is the retiming technique. This technique has been clearly demonstrated in [1]. In retiming, the delays on the edges of the circuit are redistributed without affecting the dependencies between the operations. This technique pulls some of the delays from the incoming edge and pushes them onto the outgoing edge thereby reducing the length of the longest non-zero delay path. In most cases, each time the retiming is applied, it produces a better schedule for the process.

Scheduling data flow graphs using the retiming technique has been clearly explained in [2]. This article gives a detailed description of finding the optimal schedule for data flow graphs by combining two techniques, retiming and unfolding.

The authors in [3] have proposed a method which applies retiming to optimize the schedule of synchronous data flow graphs with the help of another type of graph called an equivalent homogeneous graph (EHG). This graph is used to represent the SDFG in the form of a data flow graph where each operation is capable of producing and consuming a single unit of data.

The application of the rotation scheduling technique on iterative processes was discussed in [4] where the iterative process is represented using a data flow graph. This article emphasized the methodology used in the implementation of rotation scheduling. It proposed new heuristics such as rotation span, best span and half rotation. These methods help in finding all possible schedules that can be obtained from the initial setup.

The behavior and intricacies of synchronous data flow graphs have been clearly explained in [7]. This paper describes the basic features of a synchronous data flow graphs such as consistency, liveness, etc. This article also provided an

algorithm to find the static schedule for a synchronous data flow graph.

The authors in [6] describe another facet of rotation scheduling, using it as a technique to reduce the size of the code. This is used in the applications which are pipelined. The loop generally requires a piece of code that needs to be executed before it gets started. This is called the prologue and similarly, it needs a piece of code after it is executed which is called epilogue. This technique uses rotation scheduling to integrate the prologue and epilogue into the pipeline which significantly reduces the amount of code that needs to be written.

The authors in [5] discuss some of the important features like boundedness and liveness of an SDFG. They used Petri nets to explain these characteristics. *Liveness* of an SDFG indicates whether the process can be executed infinitely and *boundedness* describes whether the process can be implemented with limited amount of memory. The article provides in depth information on different facets of boundedness such as strict boundedness and self timed boundedness.

The authors in [8] propose an algorithm that gives the minimum achievable latency for concurrent real time applications. This article used synchronous data flow graphs to analyze streaming applications such as video conferencing, telephony and gaming. Latency is one of the important metrics to measure the performance of a real time application. The research paper also proposed a heuristic that optimizes the latency under a given throughput.

The authors in [9] discuss an algorithm called dynamic scheduling which reduces the cost of data exchanges between the processors once the tasks are assigned to them. Many scheduling algorithms tend to neglect the cost of data exchange once the tasks are scheduled. The main objective of this algorithm is to minimize the cost by effectively mapping synchronous data flow graphs onto processor networks.

It is always a difficult proposition to generate an optimal schedule for the processes without knowing the computation times of the tasks involved. The authors in [10] address this situation by introducing a new facet of rotation scheduling. It proposes two new algorithms called probabilistic retiming and probabilistic rotation scheduling that generate good schedules for these kinds of processes. These algorithms use data flow graphs where each node denotes the task with probabilistic computation time.

As we observe, most of the existing research has emphasized optimizing homogeneous data flow graphs using methods such as retiming, rotation scheduling in combination with several other techniques. In this research, we mainly focus on the application of the rotation scheduling technique

on the synchronous data flow graph to find an optimal schedule for the iterative process represented by such a graph.

In this research, we develop a `Down_Rotate ()` algorithm that applies rotation scheduling technique on the synchronous data flow graphs.

The rest of this article is organized as follows. Section II will give detailed information on the concepts of retiming, rotation scheduling, data flow graphs and synchronous data flow graphs. It also describes the properties of data flow graphs such as schedule length, iteration bound, clock period, liveness, consistency, etc. Section III will describe the implementation of algorithms that are used to apply rotation scheduling on synchronous data flow graphs. It also describes some of the heuristics developed to find the optimum schedules and discusses their limitations. Section IV will provide the conclusions that are inferred from this research and provides information on enhancements that can be done to this research.

2 Background

In this chapter, we present a detailed description on different forms of representations that we used in this research. As the focus of this research is on the iterative processes, we use data flow graphs to represent the data flow between the operations that exist in the process. We also describe several other parameters of these graphs that measure the efficiency of the process in terms of time taken.

2.1 Synchronous Data-Flow Graphs

There are a few processes where each of the operations needs to produce and consume multiple units of data. Data flow graphs cannot be used to represent these types of applications since each node in a DFG is capable of producing and consuming a single unit of data. In order to address this situation, Lee in [11] has introduced the concept of synchronous data flow graphs where each node is capable of producing or consuming multiple tokens of data on every iteration. The number of tokens produced or consumed by each node is predetermined.

2.1.1 Definition

Synchronous data flow graphs are an extension to data flow graphs where each edge in a SDFG is assigned two additional parameters to denote the number of data tokens produced and consumed at either end. It is represented as $G(V, E, d, t, p, c)$ where each of its parameters is explained below.

- V is the set of operations participating in the Iterative process denoted by nodes in the graph.
- E is the set of edges that connect the nodes in the graph and they determine the dependencies between the operations.

- $d(e)$ denotes the number of delays on the edge e , for any edge $d(e) \geq 0$
- $t(v)$ denotes the number of time steps required to compute the operation v .
- $p(e)$ denotes the number of data tokens produced by the source node of the edge e .
- $c(e)$ denotes the number of data tokens consumed by the sink node of the edge e .

For example, consider the SDFG in the Figure 1. In this figure, it can be observed that the set $\{A, B, C\}$ represent the set of vertices, the numbers above the nodes represent the computation times of each of the operations.

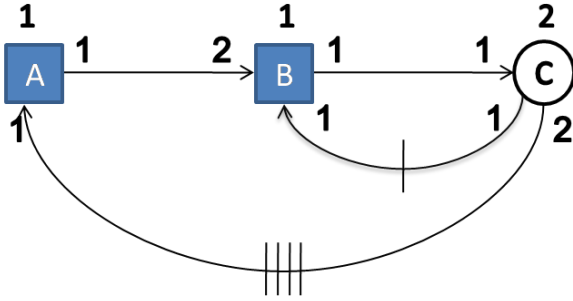


Fig. 1. Synchronous data flow graph example

The number of delays $d(e)$ on each edge e are denoted by the vertical bars cutting across the edges. On each edge e , we can find two numbers representing the number of data tokens produced and consumed by the edge. In the above graph, it can be seen that the node A produces a single data token but node B consumes 2 data tokens per iteration.

2.2 Topology matrix

A *topology matrix* is one that stores the information about every edge in the synchronous data flow graph. It is an $|E| \times |V|$ matrix where $|E|$ denotes the number of edges in the graph, and $|V|$ denotes the number of vertices in the graph. An entry in the (i, j) th position indicates the number of tokens produced or consumed by the vertex j on the edge i . A positive value indicates the number of data tokens produced by the vertex and a negative value indicates the number of data tokens consumed by the vertex. A value of zero indicates that the vertex has not used the edge to produce or consume data units. For example, the topology matrix for the SDFG shown in Figure 1 is shown below in Figure 2.

2.3 Basic Repetition Vector

The authors in [7] prove that a sequential schedule can be constructed for a SDFG if the rank of the topology matrix is one less than the number of nodes in the synchronous data flow graph. From this hypothesis, we can find a vector of q integers in the null space of the topology matrix which is called a repetition vector. This type of vector with the smallest

| | | |
|----|----|----|
| 1 | -2 | 0 |
| 0 | 1 | -1 |
| 0 | -1 | 1 |
| -1 | 0 | 2 |

Fig. 2. Topology matrix for the SDFG in Figure 1.

norm is called *basic repetition vector* (BRV). The number of elements in the basic repetition vector is equal to the number of vertices of the graph. Each element q_i in the basic repetition vector q indicates the number of copies of node v_i that are to be scheduled in each iteration.

The basic repetition vector for the topology matrix in Figure 2 is $[2 \ 1 \ 1]$. Hence we need to schedule two copies of node A, one copy of node B and one copy of node C in every iteration of the iterative process.

2.4 Equivalent Homogeneous Graph

In a SDFG, each operation is capable of producing and consuming multiple tokens of data. This characteristic makes it difficult to schedule the operations in a SDFG. To address this problem, equivalent homogeneous graphs were introduced in [7] which converts the SDFG into a graph where each node produces or consumes a single token of data. The topology matrix and the basic repetition vector play an important role in this conversion. The number of copies of each node that need to be scheduled is obtained from the BRV.

2.5 The Iteration Bound for SDFG

As defined earlier, the clock period of a data flow graph is the length of the longest zero delay sequence in the graph. This definition cannot be applied directly to the SDFG, since the above definition applies only to the single rate graphs, so we define the clock period of the SDFG as the clock period of its EHG.

We will be facing a similar situation while minimizing the clock period of the SDFG since an iteration of a synchronous data flow graph completes only when all the nodes of its EHG gets executed. The average computation time of an iteration is called the iteration period of the SDFG. If a SDFG contains a loop, then the iteration period is bounded from below by the iteration bound, which will be the loop with the maximum computation delay ratio among all the loops present in the EHG.

In [7], an equation to find the iteration bound for synchronous data flow graphs without converting them to EHG was developed. The equation is given as below.

$$I(G) = \max_{l \in G} \frac{\sum_{v \in l} t(v)}{\sum_{e: u \rightarrow v} \lfloor d(e) / \max(q_u, q_v) \rfloor} \quad (1)$$

In the above equation, $l \in G$ are the loops present in the graph G , $e: u \rightarrow v$ are the edges present in the loop l , q_u and q_v are the corresponding elements in the basic repetition vector for the vertices u and v .

For example, consider the SDFG in Figure 2. The EHG consists of three loops, (A1, B, C) and (A2, B, C) each with total computation time of 4 and delay count 2, and loop (B, C) has a computation time 2 and delay count 1. Here the first two loops have maximum time to delay ratio of 2, hence the Iteration period of this SDFG can be given as 2.

3 Implementation

In this chapter, we will discuss the mechanisms and methodologies used in implementing the proposed techniques. These techniques will be demonstrated with the help of algorithms.

3.1 Retiming

The retiming technique was initially introduced by Leiserson and Saxe in [1] for designing VLSI circuits. The basic idea of retiming is to adjust the delays in the circuit in order to minimize the clock period of the process. It is applied by pulling the delays from the incoming edges and pushing them onto the outgoing edges. The delays are more evenly distributed among the data flow graph so that it decreases the longest zero delayed path in the given circuit thereby reducing the clock period of the data flow graph. The iteration bound remains same even after the retiming the edges, since the total number of delays in a cycle remains constant.

Rotation scheduling is one resource - constrained retiming technique for producing an optimum clock period for the data flow graph. Even though a graph is retimed optimally, it is still a hard task to obtain the most optimum schedule. Rotation scheduling is a heuristic technique that combines both retiming and scheduling by performing cumulative retiming in a particular order. In this research, we demonstrate the application of this technique to achieve optimum schedules in a short amount of time.

The primary operation in the rotation scheduling technique is down rotation. When the down rotation operation of length N

is applied on a synchronous data flow graph, it extracts all the nodes that are scheduled in the first N time steps of the schedule. The retiming technique is then applied on these nodes. Below is the rotation scheduling algorithm that clearly explains the procedure.

The down rotation algorithm in Figure 3 is applied for a series of transformations cumulatively and the lengths of the schedules are recorded to check whether there is any improvement in the schedule length.

```

Input:  $G(V, E, t, d, p, c)$  is a SDFG,  $S_G$  is the schedule of the
SDFG  $G$ ,  $l$  is the number of time steps to rotate the
synchronous data flow graph.  $q$  is the basic rotation vector
BRV for the SDFG.
Output: The rotated SDFG based on the number of time
steps.
/*  $X$  = nodes scheduled within time step  $l$  */
 $X \leftarrow \{v \in V / S(v) - \min_{u \in V} \{S(u)\} < l\}$ 
/*  $I$  = incoming edges for the nodes in  $X$  */
 $I \leftarrow \{u \rightarrow v \in E \setminus u \in V \setminus X, v \in X\}$ 
/*  $O$  = outgoing edges from the node */
 $O \leftarrow \{u \rightarrow v \in E \setminus u \in X, v \in V \setminus X\}$ 
for all  $v \in X$  do
    extract( $S_G, v$ )
end for
for all incoming edges  $e \in I$ 
    /* Decrement the delay on edge  $e$  by  $q_x$  */
    /* where  $q_x$  is the repetition number for  $x$ . */
     $d(e) \leftarrow d(e) - q_x$ 
end for
for all outgoing edges  $e \in O$ 
    /* Increment the delays on the edge  $e$  by  $(q_x * p(x))$  */
     $d(e) \leftarrow d(e) + q_x * p(x)$ 
end for
/* Schedule the retimed vertices of the graph. */
Schedule( $S_G, G, X$ )

```

Fig. 3. Algorithm Down_Rotate (G, S_G, l, q)

3.2 Example to demonstrate down rotate operation

We now explain the above algorithm with the aid of an example to help familiarize the reader with the down rotate operation. Consider the synchronous data flow graph shown in Figure 1 above. The iterative process represented by the SDFG in Figure 1 consists of 3 operations A, B and C. The operations A and B are of similar kind represented by the square boxes, each of them taking 1 time unit for completion. The operation C is another kind of operation represented by circle which takes 2 time units for completion. The topology matrix for the above graph is shown in Figure 2 above. From the above topology matrix, the basic repetition vector can be computed as $q = [2 \ 1 \ 1]$. Hence we get the following values.

$q_A = 2, q_B = 1, q_C = 1$. So, we need to schedule 2 copies of A and 1 copy each for B and C in every iteration.

TABLE 4. INITIAL SCHEDULE FOR THE SDFG IN FIGURE 2

| Time Step | U+ | U* |
|-----------|-----|-----|
| 1 | A1 | NOP |
| 2 | A2 | NOP |
| 3 | B | NOP |
| 4 | NOP | C |
| 5 | NOP | C |

Now let us rotate the graph with the time step 1. In order to find all the operations that are scheduled under the time step 1, we need to find the initial schedule for the graph. The process has a dependency $A \rightarrow B \rightarrow C$. The initial schedule for the graph with one functional unit for each operation is given in the below Table 4. As we observe, the operation scheduled within the time step 1 is A. The set of incoming edges and outgoing edges for these operations are $I_A = \{CA\}$, $I_C = \{BC\}$, $O_A = \{AB\}$, $O_C = \{CB, CA\}$ where I_x and C_x represent the set of incoming and outgoing edges for the vertex x.

The next step is to redistribute the delays along the above edges. As per the algorithm we decrement the delays on the edge CA by q_A and increment the delays on the edge AB by $q_A * p(A)$. The new SDFG we get after single down rotation is shown in the Figure 5 below. The dependency we have in the new graph is $B \rightarrow C$. The schedule for the SDFG in Figure 5 is shown in Table 6 below.

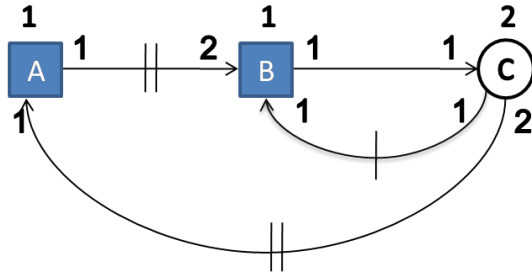


Fig. 5. Rotated SDFG

The above steps describe the basic idea behind the rotation scheduling operation. The series of applied down rotations can be termed as a *rotation phase*. The number of down rotations that need to be applied is η . This value of η is an experimentally determined constant. Since the length of the schedule gets reduced after each down rotation, the number of time steps for down rotation needs to be adjusted after every

TABLE 6. INITIAL SCHEDULE FOR THE SDFG AFTER SINGLE ROTATION

| Time Step | U+ | U* |
|-----------|----|-----|
| 1 | B | NOP |
| 2 | A1 | C |
| 3 | A2 | C |

step. The algorithm described in [4] is shown below.

Input: $G(V, E, t, d, p, c)$ is a SDFG, S_G is the schedule of the SDFG G , l is the number of time steps to rotate the Synchronous Data Flow Graph. q is the Basic Rotation Vector BRV for the SDFG.

Output: The rotated SDFG G , after a series of down rotations have been to the initial SDFG, The optimal Schedule S_{opt} .

// Initializing the S_{opt} with the initial Schedule.

$S_{opt} = S_G$

for $i = 1$ to ζ

 while $l \geq \text{length}(S_G)$ do

$l \leftarrow l/2$

 end while

 down_rotate(G, S_G, l, q)

 if $\text{length}(S_G) < \text{length}(S_{opt})$

$S_{opt} \leftarrow S_G$

 end if

end for

Fig. 7. Algorithm Rotation_Phase ($G, S_G, l, \zeta, S_{opt}, q$)

3.3 Rotation Heuristics

The application of every rotation phase to the initial schedule produces an optimal schedule for the SDFG. There are several rotation heuristics that can be applied to an SDFG to improve the schedule length of the graph [2]. These heuristics apply a series of rotation phases to the given schedule, and any improvements in the schedule length are recorded. The range of rotation phases to be tried is an experimentally determined constant κ . For our purpose we will use the heuristic RH1 described in [2, 4] and shown in Figure 8 below.

In this rotation heuristic, it can be observed that each time after the rotation phase is applied, the Schedule S_G is re-initialized to the original Schedule S_{ini} , i.e; the rotation phases

Input: G is a SDFG, ζ is the number of down rotations that need to be applied in each rotation phase. \hat{e} is the range of Rotation phases to be tried.

Output: The optimal solution found, is stored in S_{opt} .

$S_{ini} \leftarrow \text{Initial Schedule(SDFG)}$

$L \leftarrow \text{Length}(S_{ini})$

$S_{opt} \leftarrow S_{ini}$

for $i = 1$ to \hat{e} . L do

$S_G \leftarrow S_{ini}$

Rotation_Phase ($G, S_G, i, \zeta, S_{opt}$).

end for

Fig. 8. Algorithm Heuristic RH1 (G, ζ, \hat{e})

are applied independently to the original schedule and any improvements are recorded as S_{opt} . By applying the rotation phases cumulatively, we can avoid duplicate calculations that occur during the RH1 heuristic.

3.4 Limitations of the Heuristic Model

The rotation heuristics discussed above have a few limitations when implemented on the synchronous data flow graphs which are described in [4]. The first one is the time required to implement the above algorithms, which depends on the values η and κ . The algorithm generally produces good results for higher values of η and κ , which increases the running time of the algorithm.

The values η and κ are experimentally determined constants. Finding the correct values for the above constants is based on the trial and error method. The other issue with this kind of approach is that there is no attempt made to stop the method early if the optimal value is reached before the completion of the heuristic.

Another drawback we have from this heuristic is that we might encounter the same schedule after several iterations. It does not provide any mechanism to avoid re-computing the same schedule that can re-occur in the process of the finding the Optimum schedule for the given SDFG. Finally, the theoretical iteration bound which we find from the SDFG is hardware independent and does not take into consideration factors like the number of different functional units that are required for scheduling the graph and the number of copies available for each of the functional unit.

4 Conclusion

Significant amount of research has been done in implementing the technique of rotation scheduling on the data flow graphs. Through this research, we demonstrated the application of rotation scheduling on the synchronous data

flow graphs without converting them into equivalent homogeneous graphs.

We made an attempt to optimize the clock period of the graph with the help of *down_rotate* algorithm which moves the operations scheduled at the beginning to the end of the schedule to utilize the idle time of the hardware. This algorithm uses the concept of basic repetition vector, which stores the number of copies of each node that need to be scheduled on each iteration.

We discussed the Rotation_Phase algorithm which cumulatively applies the *down_rotate* operation on the graph for a constant number of times. This constant is determined experimentally.

We discussed the heuristic RH1 which makes use of the above two algorithms and produces an optimal clock period for the synchronous data flow graph. We then explained the limitations that this heuristic can potentially have.

Throughout this research, we assume that the exact computation times of the tasks involved are known to the scheduler. It is always a challenge to produce a good schedule without knowing the exact computation times of the tasks that need to be scheduled. The article [11] has done significant amount of research in scheduling these kinds of tasks. It used data flow graphs to represent these operations where each node of the graph represents a task with probabilistic computation time. It proposed two new algorithms called *probabilistic retiming* and *probabilistic rotation scheduling*. These algorithms achieved good results when applied on the above data flow graphs.

Our research work which currently applies rotation scheduling on synchronous data flow graphs can be extended to develop a rotation scheduling algorithm that generates optimal schedule for the synchronous data flow graphs with probabilistic computation times.

5 Acknowledgements

The authors are grateful to the University of Akron for their support of this work while they were all affiliated with the University.

6 References

- [1] C.E Leiserson and J.B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:5-35, 1991.
- [2] L.F. Chao, E.H.M. Sha. Scheduling data flow graphs via retiming and unfolding. *IEEE transactions on computers*, 8:1259-1267, 1997.
- [3] Timothy W. O'Neil, Edwin H.M Sha. On retiming synchronous data flow graphs. *Proceedings of the ISCA*

14th international conference on parallel and distributed computing systems, August 2001, Richardson, Texas, USA, pp. 103-108.

- [4] Michael E. Richter, *Variations in rotation scheduling*. Master's Thesis, Department of Computer Science, The University of Akron. August 2007.
- [5] Amirhossein Ghamarian, Marc Geilen, Twan Basten, Bart Theelen, MohammadReza Mousavi, Sander Stuijk. Liveness and boundedness of synchronous data flow graphs. *Proceedings of the formal methods in computer aided design*, pages 68-75, 2006.
- [6] Q.Zhuge, B. Xiao, Edwin.H.M Sha. Code size reduction technique and implementation for software pipelined DSP applications. *ACM transactions on embedded computing systems*, 2:590-613, 2003.
- [7] Samer F. Khasawneh, M.E.Richter, T.W. O'Neil. Static scheduling for synchronous data flow graphs. *Proceedings of ISCA 22nd international conference on computers and their applications*, pages 38-43, 2007.
- [8] A.H. Ghamarian, S.Stuijk, T. Basten, M.C.W. Geilen, B.D. Theelen. Latency minimization for synchronous data flow graphs. *Proceedings of the 10th euromicro conference on digital system design architectures, methods and tools*, pages 189-196, 2007.
- [9] S.Rosner, M.Scholles, D.Forchel. Near optimal scheduling of synchronous data flow graphs by exact calculation of Interprocess communication costs. *Proceedings of the international conference and exhibition on high-performance computing and networking*, pages 987-988, 1997.
- [10] S.Tongsima, E.H.M Sha, C. Chantrapornchai, D.R. Surma, Nelson L. Passos. Probabilistic Loop Scheduling for Applications with Uncertain Execution Time. *IEEE Transactions on Computers*. Volume 49, Issue 1, pages 65-80, 2000.
- [11] E.A. Lee, D.G. Messerschmitt. Static scheduling of synchronous data-flow programs for digital signal processing. *IEEE Transactions on Computers*, 36:24-35, 1987.