McGill University
School of Computer Science
ACAPS Laboratory

Advanced Compilers, Architectures
and Parallel Systems

Minimizing Buffer Requirements under Rate-Optimal Schedule in Regular Dataflow Networks

R. Govindarajan
Guang R. Gao
Palash Desai †

ACAPS Technical Memo 69
February 25, 1994

govind@cs.mcgill.ca, gao@cs.mcgill.ca, palashd@macs.ee.mcgill.ca

†Dept. of Electrical Enginering, McGill University, 3480 University Street, Montréal, Québec

Contents

List of Figures

Abstract

Large-grain synchronous dataflow graphs or multi-rate graphs have the distinct feature that the nodes of the dataflow graph fire at different rates. Such multi-rate large-grain dataflow graphs have been widely regarded as a powerful programming model for DSP applications. In this paper we propose a method to minimize buffer storage requirement in constructing rate-optimal compile-time (MBRO) schedules for multi-rate dataflow graphs. We demonstrate that the constraints to minimize buffer storage while executing at the optimal computation rate (i.e. the maximum possible computation rate without storage constraints) can be formulated as a unified linear programming problem in our framework. A novel feature of our method is that it tries to minimize the memory requirement while *simultaneously* maximizing the computation rate.

We have constructed an experimental testbed which implements our MBRO scheduling algorithm as well as (i) the widely used periodic admissible parallel schedules (also known as block schedules) proposed by Lee and Messerschmitt [16], (ii) the optimal scheduling buffer allocation (OSBA) algorithm of Ning and Gao [19], and (iii) the multi-rate software pipelining (MRSP) algorithm [9]. Schedules generated for a number of random dataflow graphs and for a set of DSP application programs using the different scheduling methods are compared. The experimental results have demonstrated a significant improvement in buffer requirements for the MBRO schedules compared to the schedules generated by the other three methods. Compared to block schedules, MBRO schedules perform better in terms of both computation rate and buffer requirements. The average observed improvement in buffer storage is 26% with respect to the MRSP algorithm, 17% with respect to block schedules and 5% with respect to the OSBA formulation. At best, an MBRO schedule can perform with only 50% of the buffer requirement for an MRSP schedule or for a block schedule and without compromising the computation rate.

1 Introduction

Large-grain synchronous dataflow graphs (originally proposed by Lee et al. [16] and subsequently used by [23, 25]) have been widely accepted as a powerful programming model for DSP applications. Here large grain means that each task in the applications generates (or consumes) multiple, sometimes in tens or even in hundreds, samples per invocation. Each of these tasks represent operations such as an FIR filter, FFT, or bandshifting. Synchronous means that the amount of input consumed by each task, and the amount of output generated, are known a priori and are fixed. Large-grain regular dataflow networks are important as they can represent a large class of DSP computations, such as speech coding, auto-correlation, power spectrum, voice band modulation, and a variety of filters [16, 23].

A large-grained dataflow graph consists of tasks (also called nodes or actors) and arcs (or channels) representing communication between tasks. The communication can be viewed as a sequence of *tokens* passing through an arc. The temporal sequences of values passed along the communication channels can be viewed as *streams*. A Stream is an abstraction of temporal sequence of values. Therefore, in this paper, we call these graphs Regular Stream Flow Graphs (RSFGs). (These graphs are called synchronous dataflow graphs by Lee and Messerschmitt [16].)

In this paper, we investigate the construction of efficient compile-time (static) schedules as opposed to run-time (dynamic) schedules for DSP programs expressed using RSFGs. Compared to the scheduling of conventional (also known as homogeneous) dataflow graphs [4], a distinct feature of the scheduling problem for RSFGs is that the production and consumption rates of tokens on an arc in an RSFG may be different. As a consequence the actors of the RSFG may be executed at different rates — hence the name *multi-rate* — in order not to accumulate tokens on any of the arcs.

We are interested in the class of RSFGs which may contain feedback loops representing the precedence constraints between successive iterations. In constructing schedules, operations of successive iterations may overlap, without violating the precedence constraints. Such overlapped or *software-pipelined* schedules are desirable as they maximize the *computation rate*, which is measured in number of iterations completed per unit time.

A major challenge facing researchers in constructing schedules for RSFGs is taking tradeoff decisions between computation rate and the storage requirements. In other words,

> Given an RSFG, how does one find a schedule which minimizes buffer space requirement while still executing the program graph at an optimal computation rate?

By optimal computation rate, we mean the maximum possible computation rate without any storage constraints. Henceforth, we refer to the above problem as Minimizing Buffer requirements under Rate-Optimal schedules (MBRO) problem.

Our earlier work on scheduling RSFGs [9] discusses a method to construct rate-optimal schedules, called Multi-Rate Software Pipelined (MRSP) schedules. In this work, we extend this framework to address the MBRO problem. We demonstrate that the constraints to minimize the buffer requirements under rate-optimal schedules can be formulated as a unified linear programming problem. The objective of this linear programming problem is to minimize the total buffer requirements. A novel feature of our formulation is that it tries to *simultaneously* minimize the buffer requirements while maximizing the computation rate.

The linear programming problem can be solved using existing linear programming techniques such as the *simplex* method. Efficient software to solve complex linear programs in a reasonably short computation time are available these days. Our formulation leads to the construction of a schedule with reduced buffer requirements (though not necessarily with the optimal buffer requirements) without compromising the computation rate. Compared to existing scheduling methods which optimize exclusively for either computation rate [2, 21] or buffer storage [3], our scheduling framework maximizes the computation rate and tries to minimize the buffer requirements *simultaneously.*

In order to evaluate the performance of our scheduling algorithm and to compare it with other scheduling methods, we have constructed an experimental scheduling testbed. In the testbed, in addition to our MBRO scheduling method, we have implemented three other scheduling algorithms, namely (i) the periodic admissible parallel schedules, also known as block schedules, proposed by Lee and Messerschmitt [16], (ii) the optimal scheduling buffer allocation (OSBA) algorithm of Ning and Gao [19], and (iii) our own multi-rate software pipelining (MRSP) algorithm [9]. As in [27], to obtain general trends over a broad range of test inputs, we propose to evaluate our scheduling method on randomly generated RSFGs. We will supplement these results with those obtained from a set of sample DSP algorithms namely, (1) phase-locked loop, (2) voice band modulation, (3) power spectrum, (4) auto-correlation, (5) periodogram, (6) comb filters, (7) satellite receiver, and (8) spectrum analyzer. The buffer requirements of the schedules generated by the various scheduling algorithms are compared. The experimental results have demonstrated a significant improvement in buffer requirements for the MBRO schedules compared to schedules generated by the other three methods. Compared to block schedules, MBRO schedules perform, on the average, 35% better in terms of computation rate and 16% better in terms of buffer requirements. The average observed improvement in buffer storage is 26% with respect to the MRSP algorithm, and 5% with respect to the OSBA formulation. For some test graphs, the MBRO schedule can execute at the maximum computation rate using only 50% of the total buffer required for a block schedule or for an MRSP schedule. Lastly our experiments establish that the existing scheduling methods, such as the ones reported in [19], for homogeneous graphs when extended to RSFGs do not lead to better memory-efficient schedules in a majority of cases (in more than 80% cases).

The rest of this paper is organized as follows. The following section motivates our work with the help of an example. Section 3 deals with the formulation of the Minimum Buffer Rate-Optimal (MBRO) schedule problem. In Section 4, solutions for the MBRO formulation are discussed using the motivating example. In Section 5, we report experimental results, comparing MBRO formulation with other scheduling methods. A discussion of related work is included in Section 6. Concluding remarks and plans for future work are presented in Section 7.

2 Motivating Example

In this section we informally introduce the concept of software pipelining for the RSFG model. Subsequently, we will motivate our work on minimizing buffer requirements in rate-optimal schedules for multi-rate RSFGs using a simple example.

2.1 Software Pipelining of Regular Stream Flow Graphs (RSFG): A Motivating Example

Regular stream flow graphs have been proposed to express DSP computations in a graphical representation. A brief introduction of RSFG can be found in the appendix, and for more information the readers are referred to [7].

Consider the RSFG shown in Fig. 1. The firing rate of the actors of the RSFG so as not accumulate tokens on any arc is given by [2,1,3,2]. Let the initial number of tokens on the arcs (d, c), (b, a), (b, d), and (d, b) be six, three, two, and one respectively. It may be observed that firing the actors in the above rate restores the state of graph (informally, the number of tokens on each arc) to the initial state. Thus firing the actors according to the multi-rate causes neither

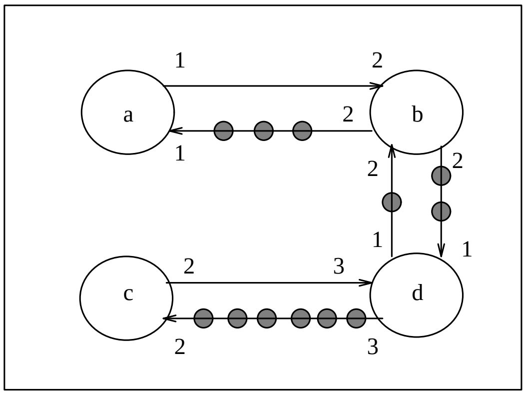

Figure 1: An Example RSFG

	Time Step												
	0	1	2	3	4	5	6	7	8	9	10	11	12
iteration = 0	c,a	c,_a	c,_a	d,a	d,_a	_a	b						
iteration = 1							c,a	c,_a	c,_a	d,a	d,_a	_a	b
iteration = 2													c,a

Table 1: **Schedule for Motivating Example**

token accumulation nor token depletion on any arc in the RSFG. Intuitively, this is referred to as *consistency* [15] or *well-behavedness* [6].

Let the execution time of actor a be 3 time units while that of other actors be unity. In this paper we do not address issues relating to (processor) resource constraints. Further, throughout this paper, we do not allow two instances of an actor to fire concurrently. A schedule for the RSFG is shown in Table 1 where the actors (belonging to one or many iterations) in a single column fire concurrently. The symbol _a is used in Table 1 to indicate that the execution of actor a is continuing from the previous time step. The schedule of Table 1 is repetitive starting from time step 3, with a repetitive pattern from time step 3 to time step 8. A new iteration is initiated after every 6 time units, or equivalently the computation rate is $\frac{1}{6}$. Observe further that in the repetitive pattern, the firings of the actors in various iterations overlap, and the schedule is called *software pipelined*. More details on software pipelining for RSFGs can be found in [9].

The maximum computation rate of an RSFG can be derived by inspecting the equivalent *homogeneous dataflow graph* (of the original REFG) where each actor will consume (output) only one token per arc at each firing.

A method to construct the equivalent homogeneous graph is given in [27, 10]. Fig. 2 depicts the equivalent homogeneous graph for the given RSFG. The equivalent homogeneous graph consists of multiple instances of each actor (as dictated by its firing rate) and arcs indicating data dependencies between the various instances of the actors. Multiple instances of a node are represented by the actor name followed by the instance number, as in $a0$, $a1$, $c0$, $c1$, and $c2$. In the homogeneous graph, there is a self-dependency between two successive instances of actor represented by dotted arcs. This is because any two instances of an actor are not allowed to execute concurrently under our model.

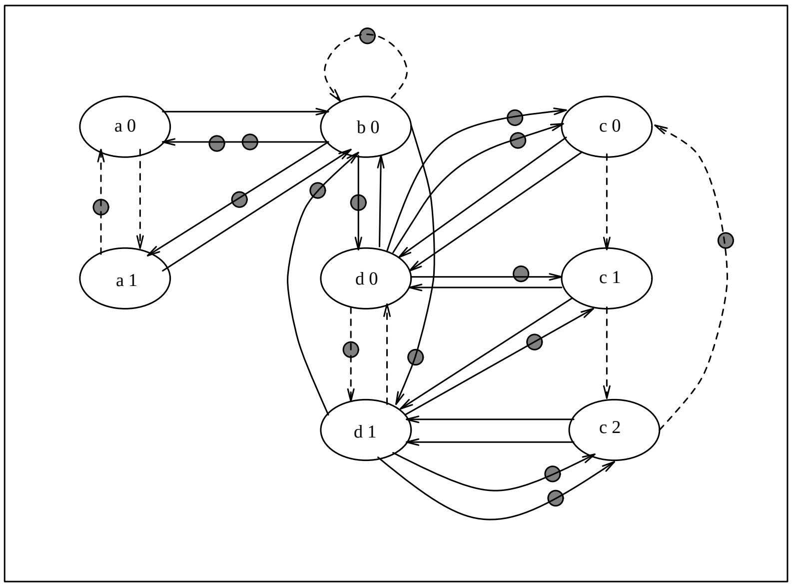

Figure 2: Equivalent Homogeneous Graph for Example 2.1

The following theorem due to Reiter characterizes the maximum rate attainable in such a graph. The proof can be found in [24].

Theorem 2.1 (Reiter [24]) *Consider a homogeneous graph $G = (V,E,m,d)$ where V is the set of nodes, E is the set of arcs, m represents the initial marking, and d the delay function of the nodes in the graph. The maximal computation rate of every node in the above graph is given by $\min_{\forall C} \frac{M(C)}{D(C)}$ where $M(C) = \sum_{(i,j) \in C} m_{ij}$, $D(C) = \sum_{i \in C} d_i$, and C is any cycle in the graph.*

The maximum computation rate $\frac{1}{T}$ is determined by the cycle, called the critical cycle, which has the smallest above ratio. It can be seen that the optimal rate for the graph shown in Fig. 2 is $\frac{1}{6}$, corresponding to the critical cycle, $a0 \rightarrow a1 \rightarrow a0$. Observe that this optimal computation rate equals the computation rate of the schedule given in Table. 1 — hence the schedule is rate-optimal.

2.2 The Storage Requirements of Rate-Optimal Schedules

Next we compute the buffer requirement for each arc of the RSFG using an operation model[1]. In this method we compute the buffer requirements for an arc by firing the nodes of the RSFG

[1]A more sophisticated method for determining the buffer requirements for homogeneous graphs is presented in [19], and for RSFGs in [10]. However both methods are based on the homogeneous dataflow graphs and hence require the RSFG to be transformed into an equivalent homogeneous graph in the first place. Since the operational model deals directly with the RSFG, we prefer to use it in this paper.

Time Step	Actors Fired†	Buffer size on Arc (c,d) before	Buffer size on Arc (c,d) after	Buffer size on Arc (d,c) before	Buffer size on Arc (d,c) after
6	c	0	2	6	4
7	c	2	4	4	2
8	c	4	6	2	0
9	d	6	3	0	2
10	d	3	0	2	4
11	-	0	0	4	6

† Only actors c and d are considered here.

Table 2: **Buffer Requirements on Arcs** (c, d) **and** (d, c)

according to the schedule and by determining the maximum tokens present on the arc at any time during the repetitive pattern. In evaluating the buffer requirements for a schedule, we do not include the requirements for the prologue (steps 0 to 2 in our example). This is because the the prologue is executed only once, and even if the prologue requires more buffers (than the repetitive pattern) separate buffer allocation can be done for the prologue and the repetitive part of the schedule.

In computing the buffer requirements for the arcs of the RSFG, we make the following assumption without any loss of generality. Input tokens remain on the input arc until the activation (firing) is completed and output tokens are produced (all at once) at the end of the firing. This assumption is not crucial to our formulation and similar formulations can be derived for other assumptions on the time tokens are consumed (or produced). Table 2 shows the number of buffer locations used for arcs (c, d) and (d, c) after each time step in the repetitive pattern. Also, without loss of generality, we will consider the repetitive pattern starting from time step 6.

Clearly, under this schedule, the maximum buffer size required for each of the arcs (c, d) and (d, c) is 6. A buffer size of 2 is required for the arcs (a, b) and (b, d), and a buffer size of 3 is required for (b, a) and (d, b). Thus the total buffer requirement for the schedule of Table 1 is 22.

It is important to note that there may be more than one rate-optimal schedule for a given RSFG, and they may have different buffer requirements. For instance, for the RSFG of Fig. 1, an alternative schedule with the optimal computation rate is given Table 3. Once again, the schedule has a repetitive pattern starting from time step 3 with a period 6.

The buffer requirements for the arcs (c, d) and (d, c) at each time step, starting from $t = 6$ is shown in Table 4.

In this schedule, the maximum buffer size required for arcs (c, d) and (d, c) are 5 and 4 respectively. Hence, the total buffer requirement for all the arcs in the RSFG is 18 using the schedule given in Table 3. Thus by choosing the schedule of Table 3 we reduce the buffer requirement without compromising the computation rate.

	Time Step												
	0	1	2	3	4	5	6	7	8	9	10	11	12
iteration = 0	c,a	_a	c,_a	a	d,c,_a	_a	b	d					
iteration = 1							c,a	_a	c,_a	a	d,c,_a	_a	b
iteration = 2													c,a

Table 3: **An Alternative Schedule for the Motivating Example**

Time Step	Actors Fired†	Buffer size on Arc (c, d) before	after	Buffer size on Arc (d, c) before	after
6	c	3	5	3	1
7	d	5	2	1	4
8	c	2	4	4	2
9	–	4	4	2	2
10	d,c	4	3	2	3
11	–	3	3	3	1

† Only actors c and d are considered here.

Table 4: **Buffer Requirements for the Alternative Schedule**

The motivating example elucidates the MBRO problem introduced in Section 1. Given an RSFG there could be several rate-optimal schedules. In [9], we presented a method to construct a rate-optimal schedule which may not necessarily use minimum buffers. The problem is how to chose, among the rate-optimal schedules, the one that has lesser memory requirement. In this paper, we answer this question by deriving linear constraints for buffer requirements and integrating them with our MRSP formulation. The obtained linear programming formulation can be solved using existing linear programming techniques. In the following section we formulate this problem in the linear programming framework.

3 Minimizing Buffer Requirements in Rate-Optimal Schedules (MBRO): Problem Formulation

In [9], the problem of obtaining rate-optimal schedules for RSFGs (without buffer constraints) has been formulated as a linear programming problem, called the Multi-Rate Software Pipelining (MRSP) problem. In section 3.1 we briefly review the MRSP formulation as well as introduce the necessary terminology for the rest of this section. Readers who are familiar with the MRSP formulation can skip this subsection. In section 3.2, we show that the buffer storage requirements can also be represented as linear constraints, and the MRSP formulation can be extended to present a unified linear program formulation of the MBRO problem, capturing both the precedence and storage constraints.

3.1 Rate-Optimal Schedules

A schedule σ is a function which maps the ith instance of an actor u in the RSFG to a time instance t if the $(i+1)$-th firing of u commences at time t under the schedule σ. That is, $\sigma(i,u)$ represents the time t at which the i-th instance of actor u begins its execution. The instance number i is counted from 0 rather than from 1. Therefore the i-th instance of an actor actually corresponds to the $(i+1)$-th firing of the actor.

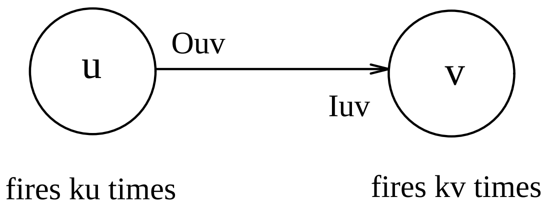

Figure 3: An Arc in the RSFG

Consider an arc (u,v) in an RSFG (refer to Fig. 3). Let u produce O_{uv} tokens in each firing and let I_{uv} be the number of tokens consumed by v in each firing. Further, let $b_{uv}(t)$ denote the number of tokens present in the arc connecting u to v at time t. We will use the special notation m_{uv} to represent the initial number of tokens on the arc (u,v).

Definition 3.1 *A schedule σ is said to be* **admissible** *if for every arc (u,v) in the graph, and for all $i \geq 0$,*

$$\sigma(i,v) = t \Rightarrow b_{uv}(t) \geq I_{uv}.$$

That is, a schedule is *admissible* if it does not violate the dependency constraints imposed by the arcs of the RSFG. The following necessary and sufficient condition for the class of admissible schedules for RSFGs has been established in [9].

Theorem 3.1 *A schedule σ is admissible if and only if for each arc (u,v) in the graph,*

$$\sigma(i,v) \geq \sigma\left(\left\lceil \frac{(i+1)*I_{uv} - m_{uv} - O_{uv}}{O_{uv}} \right\rceil, u\right) + d_u. \tag{1}$$

An *iteration* of an RSFG is formally defined as:

Definition 3.2 *If $[k_{v_1}, k_{v_2}, \ldots, k_{v_n}]$ represents the (fundamental) firing rates of the actors of the RSFG, then an iteration of the graph consists of k_{v_i} firings of actor v_i for every $v_i \in \{v_1, v_2, \ldots, v_n\}$.*

Here the term 'fundamental firing rates' refers to the fact that the rates (i.e. the $k'_{v_i}s$) do not have a common divisor. In other words, the greatest common divisor of the k_{v_i}'s is 1.

Next, we formalize the notion of r-periodic schedules for RSFGs. Intuitively, in an r-periodic schedule, the firing of the k-th instance of an actor in every r iterations takes place exactly after T time units.

Definition 3.3 *A schedule is called* **r-periodic** *with period T if for any node v_i, for $0 \le j < r$, and for any k in the semi-open interval $[0, k_{v_i})$ the k-th instance in iterations $j, j+r, j+2r, \ldots$, take place, respectively, at time $t_{v_i,k},\ T + t_{v_i,k},\ 2T + t_{v_i,k},\ \ldots$*

Since r iterations of the RSFG are executed in T time steps, the computation rate of an r-periodic schedule is $\frac{r}{T}$.

The notation (j, k) is used to represent the k-th instance in the j-th iteration. Similarly, $\sigma(j, k, v)$ denotes the time at which (j, k)-th firing of v commences. Finally, we use r-periodic schedules of the form $\left\lfloor \frac{j*T+A[k,v]}{r} \right\rfloor$ for the (j, k)-th firing of actor v. That is,

$$\sigma(j, k, v) = \left\lfloor \frac{j * T + A[k, v]}{r} \right\rfloor, \tag{2}$$

where T is the period of an r-periodic schedule for the given RSFG, and $\forall k \in [0, k_v)$ and $\forall v, \quad A[k, v] \ge 0$ are constants that uniquely determine a schedule for the given T and r values. Throughout this paper we will assume that the execution time of each actor and time at which an actor is scheduled for execution are always integer values. However, since we use the integer scheduling form with the 'floor' function (see Equation (2)), $A[k, v]$'s can take real values.

Using Equation (2) in the necessary and sufficient condition (Equation (1)), and after a few algebraic manipulations, we obtain the following constraints on $A[k, v]$'s (refer to [9] for the details).

$\forall (u, v)$ in the RSFG and $\forall k \in [0, k_v)$,

$$\begin{aligned} A[k, v] - A\left[\left(\left\lceil \frac{(k+1) * I_{uv} - m_{uv} - O_{uv}}{O_{uv}} \right\rceil \ \ mod \ \ k_u\right), u\right] \ \ \ge \\ T * \left\lfloor \frac{1}{k_u} \left\lceil \frac{(k+1) * I_{uv} - m_{uv} - O_{uv}}{O_{uv}} \right\rceil \right\rfloor + r * d_u \end{aligned} \tag{3}$$

Our assumption that two instances of an actor are not allowed to fire concurrently can be enforced by including a self arc to each actor in the RSFG. The self dependency arcs lead to the constraints:

$\forall v$ in the RSFG, and $\forall k \in [0, k_v)$,

$$A[k, v] - A[((k-1) \ \ mod \ \ k_v), v] \ge T * \left\lfloor \frac{(k-1)}{k_v} \right\rfloor + r * d_v \tag{4}$$

Now to obtain rate-optimal schedules, first we need to find the optimal computation rate. That is, to maximize $\frac{r}{T}$ subject to the above constraints (Equations (3) and 4)). An efficient solution for this problem can be obtained by relating it to the Minimum Cost-to-Time Ratio Cycle problem in network flow [14, 13]. Once the optimal $\frac{r}{T}$ is found, the value of various $A[k, v]$'s can be obtained by solving the linear equation with the optimal value of $\frac{r}{T}$. This can be reduced to the shortest path problem for which polynomial time solutions exist [14].

In the following subsection we extend the above formulation to MBRO schedules.

3.2 MBRO Schedules

In this section, we develop a linear programming problem formulation for the MBRO problem.

We first show that the buffer requirement on each arc (u, v) in a RSFG can be represented as a linear constraint. Consider a time instant t such that node u is between its i-th and $(i+1)$-th firing for some i, and node v is between its i'-th and $(i'+1)$-th firing for some i'. Formally,

$$\begin{array}{llll} \sigma(i,u)+d_u & \leq t & < \sigma(i+1,u)+d_u & \text{and} \\ \sigma(i',v)+d_v & \leq t & < \sigma(i'+1,v)+d_v, & \text{for some } i,\ i' \end{array} \tag{5}$$

Since the i-th instance (i.e. the $(i+1)$-th firing) of u has finished, actor u would have produced $(i+1)*O_{uv}$ tokens on the arc (u, v). Further, as the i'-th instance of v has finished its execution, it must have consumed $(i'+1)*I_{uv}$ tokens from the arc (u, v). If there were m_{uv} tokens initially on the arc (u, v), then at time t, there would be $(i+1)*O_{uv}+m_{uv}-(i'+1)*I_{uv}$ tokens. Thus, the buffer size for the arc (u, v) at time t should be

$$b_{uv}(t) \geq (i+1)*O_{uv}+m_{uv}-(i'+1)*I_{uv} \tag{6}$$

We can represent the i-th instance of actor u as $\left(\lfloor \frac{i}{k_u} \rfloor, i \bmod k_u\right)$ in our notation. Similarly,

$$\begin{aligned} i+1 &= \left(\left\lfloor \frac{i+1}{k_u} \right\rfloor, (i+1) \bmod k_u\right) \\ i' &= \left(\left\lfloor \frac{i'}{k_v} \right\rfloor, i' \bmod k_v\right) \quad \text{and} \\ i'+1 &= \left(\left\lfloor \frac{i'+1}{k_v} \right\rfloor, (i'+1) \bmod k_v\right) \end{aligned} \tag{7}$$

where 'mod' stands for the modulo operator.

Let $\frac{r}{T}$ be the optimal computation rate. Using the the scheduling form of Equation (2) in Equation (5), we obtain

$$\left\lfloor \frac{\lfloor \frac{i}{k_u} \rfloor * T + A[(i \bmod k_u), u]}{r} \right\rfloor + d_u \leq t < \left\lfloor \frac{\lfloor \frac{i+1}{k_u} \rfloor * T + A[(i+1) \bmod k_u, u]}{r} \right\rfloor + d_u \tag{8}$$

$$\left\lfloor \frac{\lfloor \frac{i'}{k_v} \rfloor * T + A[(i' \bmod k_v), v]}{r} \right\rfloor + d_v \leq t < \left\lfloor \frac{\lfloor \frac{i'+1}{k_v} \rfloor * T + A[(i'+1) \bmod k_v, v]}{r} \right\rfloor + d_v \tag{9}$$

Our estimate for the buffer size needs to be conservative. Thus, while approximating $b_{uv}(t)$, higher values for i and lower values i' should be used to obtain a conservative bound for the buffer size. Therefore we will use the lower bound of t in Equation (8) and the upper bound in Equation (9). That is,

$$\left\lfloor \frac{\lfloor \frac{i}{k_u} \rfloor * T + A[(i \bmod k_u), u]}{r} \right\rfloor + d_u \leq t \tag{10}$$

$$t < \left\lfloor \frac{\lfloor \frac{i'+1}{k_v} \rfloor * T + A[(i'+1), v]}{r} \right\rfloor + d_v \tag{11}$$

Equation (10) can be rewritten as

$$\frac{\lfloor \frac{i}{k_u} \rfloor * T + A[(i \bmod k_u), u]}{r} + d_u < t + 1$$

Let $i \bmod k_u = k$, where $k \in [0, k_u)$, in the above equation. We can rewrite it as

$$\left\lfloor \frac{i}{k_u} \right\rfloor < \frac{(t+1-d_u) * r - A[k, u]}{T} \tag{12}$$

Similarly, substituting $k' = (i'+1) \bmod k_v$ in Equation (11), we get

$$\frac{\lfloor \frac{i'+1}{k_v} \rfloor * T + A[k', v]}{r} + d_v > t$$

That is,

$$\left\lfloor \frac{i'+1}{k_v} \right\rfloor > \frac{(t-d_v) * r - A[k', v]}{T} \tag{13}$$

Notice that i and $i'+1$ can be written as

$$i = \left\lfloor \frac{i}{k_u} \right\rfloor * k_u + k \quad \text{and} \quad i'+1 = \left\lfloor \frac{i'+1}{k_v} \right\rfloor * k_v + k'$$

Substituting the values of $\lfloor \frac{i}{k_u} \rfloor$ and $\lfloor \frac{i'+1}{k_v} \rfloor$ from Equations (12) and (13),

$$i < \frac{(t+1-d_u) * r - A[k, u]}{T} * k_u + k \tag{14}$$

$$i'+1 > \frac{(t-d_v) * r - A[k', v]}{T} * k_v + k' \tag{15}$$

Now substituting the upper bound for i (Equation (14)) and the lower bound for $i'+1$ (Equation (15)) in Equation (6), we get a conservative estimate for the buffer size required for arc (u, v).

$$\begin{aligned} b_{uv}(t) \quad \geq \quad & \left(\frac{(t+1-d_u) * r - A[k, u]}{T} * k_u + k + 1\right) * O_{uv} \\ & - \left(\frac{(t-d_v) * r - A[k', v]}{T} * k_v + k'\right) * I_{uv} + m_{uv} \end{aligned} \tag{16}$$

By the property of multi-rate graphs, $k_u * O_{uv} = k_v * I_{uv}$. Using this in Equation (16) and after simple algebraic manipulations, we get

$$\begin{aligned} & T * b_{uv}(t) + k_u * O_{uv} * (A[k, u] - A[k', v]) \geq \\ & \quad (k+1) * T * O_{uv} - k' * T * I_{uv} + r * k_u * O_{uv} * (d_v - d_u + 1) + m_{uv} * T \end{aligned}$$

The right hand side of the above equation depends on the instance numbers k and k', and on constants k_u, k_v, O_{uv}, and I_{uv}. We will represent this as a function $\mathcal{C}(k, k')$. Note that the equation is linear in $b_{uv}(t)$, $A[k, u]$, and $A[k', v]$. Further the above equation is independent of t and hence

$$T * b_{uv} + (k_u * O_{uv}) * (A[k, u] - A[k', v]) \geq \mathcal{C}(k, k') \tag{17}$$

Notice that the buffer estimate is independent of the iteration numbers of the actors u and v, just like the dependence constraints (Equation (3)) are independent of the iteration numbers. The buffer requirement depends on k and k' which represent some instance of actors u and v respectively. More specifically, k is the most recent instance of actor u that has finished its firing. Likewise k' is the next instance of actor v that is to finish its firing. The difference in the interpretation of k and k' is due to the fact that $k = i \bmod k_u$ while $k' = (i' + 1) \bmod k_v$. For the buffer estimate to be valid, the above constraint must be satisfied for all values of $k \in [0, k_u)$ and for all values of $k' \in [0, k_v)$.

The MBRO problem can be formulated as follows. Our objective is to minimize the total buffer requirements for all arcs while maintaining rate-optimality. The value of $A[k, v]$'s must satisfy dependency constraints (3) and (4). To ensure rate-optimality, use the optimal computation rate $\frac{r}{T}$, obtained from the MRSP formulation, in the constraints. The complete MBRO formulation is shown in Fig. 4.

[MBRO Problem] Minimize $\sum_{(u,v)} b_{uv}$ subject to the conditions

(I) $\forall (u, v)$, $\forall k \in [0, k_u)$, and $\forall k' \in [0, k_v)$,

$$T * b_{uv} + k_u * O_{uv} * (A[k, u] - A[k', v] \geq \mathcal{C}(k, k')$$

(II) $\forall (u, v)$, and $\forall k \in [0, k_v)$,

$$A[k, v] - A\left[\left(\left\lceil \frac{(k+1) * I_{uv} - m_{uv} - O_{uv}}{O_{uv}} \right\rceil \quad mod \quad k_u \right), u\right] \quad \geq$$
$$T * \left\lfloor \frac{1}{k_u} \left\lceil \frac{(k+1) * I_{uv} - m_{uv} - O_{uv}}{O_{uv}} \right\rceil \right\rfloor + r * d_u$$

(III) $\forall v$, and $\forall k \in [0, k_v)$

$$A[k, v] - A[((k-1) \quad mod \quad k_v), v] \geq T * \left\lfloor \frac{(k-1)}{k_v} \right\rfloor + r * d_v$$

Figure 4: MBRO Formulation

In the following subsection we will present a solution method for the MBRO formulation. Also we will obtain the MBRO formulation for our motivating example and derive its solution.

4 Solution to the MBRO Formulation and Discussions

The linear programming formulation can be solved using standard linear programming methods, such as the simplex method. Once the value of various $A[k, v]$'s are obtained, the buffer size for each arc needs to be calculated. Recall that in obtaining the MBRO formulation, we have used a conservative approach. Hence if we use Equation (17) to compute the buffer requirements, the estimates could be more than what is required by the schedule. However, once the schedule is finalized buffer requirements on each arc can be computed using an operational method. In this method, the actors are fired according to the generated schedule. In particular, the algorithm considers the prologue and the repetitive pattern of the schedule and fires the actors accordingly. The algorithm also keeps track of the maximum number of tokens present on every arc during the execution of the repetitive pattern. These numbers give the buffer requirements for the individual arcs of the RSFG. From these values the total buffer requirements for all the arcs in the RSFG is computed.

In the following subsection, we derive the MBRO formulation for the motivating example. A few remarks on the MBRO formulation are presented in Section 4.2.

4.1 MBRO Formulation for Motivating Example

The RSFG used in our motivating example (refer to Fig. 1) has 4 actors a, b, c, and d. The fundamental firing rate of the actors is given by $[2, 1, 3, 2]$. The optimal computation rate for the graph is $\frac{1}{6}$. That is $r = 1$ and $T = 6$.

First we will obtain the dependency constraints (Constraint II in the MBRO formulation) for each arc of the RSFG. For each arc (u, v), using different values for $k \in [0, k_v)$, we obtain the following equations.

$$A[0,b] - A[1,a] \geq 3; \qquad \text{for } (a,b), \text{ and for } k \in [0,1)$$

$$A[0,b] - A[0,d] >= 1; \qquad \text{for } (d,b), \text{ and for } k \in [0,1)$$

$$\left.\begin{array}{r} A[0,a] - A[0,b] \geq -11; \\ A[1,a] - A[0,b] \geq -5; \end{array}\right\} \quad \text{for } (b,a) \text{ and for } k \in [0,2)$$

$$\left.\begin{array}{r} A[0,d] - A[0,b] \geq -5; \\ A[1,d] - A[0,b] \geq -5; \end{array}\right\} \quad \text{for } (b,d) \text{ and for } k \in [0,2)$$

$$\left.\begin{array}{r} A[0,d] - A[1,c] \geq 1; \\ A[1,d] - A[2,c] \geq 1; \end{array}\right\} \quad \text{for } (c,d) \text{ and for } k \in [0,2)$$

$$\left.\begin{array}{r} A[0,c] - A[0,d] \geq -5; \\ A[1,c] - A[1,d] \geq -5; \\ A[2,c] - A[1,d] \geq -5; \end{array}\right\} \quad \text{for } (d,c) \text{ and } k \in [0,3)$$

Constraint III in the MBRO formulation is the self-dependency constraint which enforces that no two instances of an actor fire concurrently. This constraint leads to the following inequalities for the actors a, c, and d of the RSFG. Since actor b has only one instance (in an iteration), no constraint need to be enforced on $A[0,b]$.

$$
\begin{aligned}
A[0,a]-A[1,a] &\geq -3; & A[1,a]-A[0,a] &\geq 3;\\
A[0,c]-A[2,c] &\geq -5; & A[1,c]-A[0,c] &\geq 1;\\
A[2,c]-A[1,c] &\geq 1; & A[0,d]-A[1,d] &\geq -5;\\
A[1,d]-A[0,d] &\geq 1;
\end{aligned}
$$

Lastly, the constraints on buffer requirements (Constraint I) for the arcs of the RSFG are:

$$
\left.\begin{aligned}
6*b_{ab}+2*A[0,a]-2*A[0,b] &\geq 2;\\
6*b_{ab}+2*A[1,a]-2*A[0,b] &\geq 8;
\end{aligned}\right\} \quad \text{for } (a,b),\ \ k\in[0,2),\ \text{and } k'\in[0,1)
$$

$$
\left.\begin{aligned}
6*b_{ba}+2*A[0,b]-2*A[0,a] &\geq 34;\\
6*b_{ba}+2*A[0,b]-2*A[1,a] &\geq 28;
\end{aligned}\right\} \quad \text{for } (b,a),\ \ k\in[0,1),\ \text{and } k'\in[0,2)
$$

$$
\left.\begin{aligned}
6*b_{bd}+2*A[0,b]-2*A[0,d] &\geq 24;\\
6*b_{bd}+2*A[0,b]-2*A[1,d] &\geq 18;
\end{aligned}\right\} \quad \text{for } (b,d),\ \ k\in[0,1),\ \text{and } k'\in[0,2)
$$

$$
\left.\begin{aligned}
6*b_{db}+2*A[0,d]-2*A[0,b] &\geq 12;\\
6*b_{db}+2*A[1,d]-2*A[0,b] &\geq 18;
\end{aligned}\right\} \quad \text{for } (d,b),\ \ k\in[0,2),\ \text{and } k'\in[0,1)
$$

$$
\left.\begin{aligned}
6*b_{cd}+6*A[0,c]-6*A[0,d] &\geq 12;\\
6*b_{cd}+6*A[0,c]-6*A[1,d] &\geq -6;\\
6*b_{cd}+6*A[1,c]-6*A[0,d] &\geq 24;\\
6*b_{cd}+6*A[1,c]-6*A[1,d] &\geq 6;\\
6*b_{cd}+6*A[2,c]-6*A[0,d] &\geq 36;\\
6*b_{cd}+6*A[2,c]-6*A[1,d] &\geq 18;
\end{aligned}\right\} \quad \text{for } (c,d),\ \ k\in[0,3),\ \text{and } k'\in[0,2)
$$

$$
\left.\begin{aligned}
6*b_{dc}+6*A[0,d]-6*A[0,c] &\geq 54;\\
6*b_{dc}+6*A[0,d]-6*A[1,c] &\geq 42;\\
6*b_{dc}+6*A[0,d]-6*A[2,c] &\geq 30;\\
6*b_{dc}+6*A[1,d]-6*A[0,c] &\geq 72;\\
6*b_{dc}+6*A[1,d]-6*A[1,c] &\geq 60;\\
6*b_{dc}+6*A[1,d]-6*A[2,c] &\geq 48;
\end{aligned}\right\} \quad \text{for } (d,c),\ \ k\in[0,2),\ \text{and } k'\in[0,3)
$$

The objective of the MBRO formulation is

$$\text{minimize} \quad (b_{ab}+b_{ba}+b_{bd}+b_{db}+b_{cd}+b_{dc})$$

subject to the above sets of constraints. Solving the above linear programming problem, we get

$$
\begin{aligned}
A[0,a]&=0; & A[1,a]&=3; & A[0,b]&=6; & A[0,c]&=0;\\
A[1,c]&=2; & A[2,c]&=4; & A[0,d]&=4; & A[1,d]&=7;
\end{aligned}
$$

Using the operational method we get the buffer estimates for each arc in RSFG as

$$
\begin{aligned}
b_{(a,b)}&=2; & b_{(b,a)}&=3; & b_{(b,d)}&=3;\\
b_{(d,b)}&=2; & b_{(c,d)}&=5; & b_{(b,d)}&=4;
\end{aligned}
$$

4.2 Discussions

Recall that in obtaining the MBRO formulation, we have used a conservative approach and made the following two approximations. First, we considered all combinations of (k, k') for Equation (17). However, not all possible combinations of k and k' may arise in a schedule. For example, for arc (c, d) the following combinations do not arise at any time step in the repetitive pattern of the schedule shown in Table 3.

$$k = 2, \quad k' = 0 \quad \text{and} \quad k = 1, \quad k' = 1$$

All other (k, k') combinations for the arc (c, d) are possible at the time steps indicated below.

$$\begin{array}{llll} k = 0, \ k' = 0 & \text{after time step} \ \ 7; \qquad & k = 1, \ k' = 0 & \text{after time step} \ \ 8; \\ k = 0, \ k' = 1 & \text{after time step} \ \ 6; \qquad & k = 2, \ k' = 1 & \text{after time step} \ \ 10; \end{array}$$

Notice that our formulation estimates the buffer requirements even before the schedule is finalized and uses this estimate to arrive at a schedule. Since the schedule is not finalized at the time of formulation, it may not be possible, in general, to detect all combinations of (k, k') that may not arise. As a second approximation, we used larger values for i and smaller values for i' in Equation (6) to estimate the buffer size. Lastly, in the MBRO formulation the b_{uv} variables must take integer values, since they represent buffer sizes. Making the the b_{uv} variables as integer variables makes the MBRO formulation an integer program formulation whose complexity is NP-Complete. However since the buffer estimates are conservative, the solution obtained from the MBRO formulation (with or without the constraint that the b_{uv} variables are integers) may not always be optimal. Thus to reduce the complexity, we solve the MBRO formulation as a linear programming problem rather than an integer program. Our experimental results demonstrate that the MBRO formulation, in a large majority of cases, produces schedules that are buffer efficient compared to the MRSP formulation. That is, even though the MBRO schedules are not buffer *optimal*, they use significantly less buffer storage compared to the MRSP schedules. In Section 5.2 we compare the schedules obtained from MBRO formulation and the MRSP formulation and show that except in 2 cases (out of nearly 200 experiments), the MBRO formulation always produces buffer efficient schedules. Also, our experiments establish that solving the MBRO problem with the integer constraints leads to only a very small improvement (less than 0.5%) in buffer requirements.

5 Experimental Results

In this section we describe the implementation of our testbed and evaluate the proposed scheduling method. A comparison of the MBRO formulation with other existing methods is presented in Section 5.2.

5.1 The Scheduling Testbed

The experimental testbed is implemented using C programming language on a Unix platform. Due to space restriction, only a brief and informal description of the experimental testbed is presented here; some additional details on the testbed are provided in Appendix B. The testbed consists of four main modules, namely the random graph generator, the module to generate and solve (for optimal $\frac{r}{T}$) the linear programming formulation for rate-optimal schedules, the module to generate and solve the linear programming problem with buffer minimization constraints, and the module to compute the buffer requirements for each arc in the RSFG.

Due to the lack of availability of standard multi-rate benchmark programs, and to extract general trends over a broad range of test inputs, we propose to evaluate our scheduling method on randomly generated well-behaved RSFGs. A similar approach was followed in other studies, e.g. in [27, 17]. We supplement this study with results obtained from a set of DSP algorithms. The DSP algorithms considered in our experiments are; (1) phase-locked loop, (2) voice band modulation, (3) power spectrum, (4) auto-correlation, (5) periodogram, (6) comb filters, (7) satellite receiver, and (8) spectrum analyzer. The random graph generator generates well-behaved multi-rate graphs in an incremental fashion. The parameters of the graph generator are chosen in such way that the generated graphs reasonably model DSP application programs. The subsequent module generates the linear constraints (corresponding to Equations (3) and (4)) of the MRSP formulation. The optimal computation rate $\frac{r}{T}$ is computed efficiently by relating the problem to the Minimum Cost-to-Time Ratio Cycle (MCTRC) problem. Using this $\frac{r}{T}$ value, the constraints for the MBRO problem are generated. The linear programming problem is solved using a public domain software package called *lp_solve*, developed in the Eindhoven University of Technology, The Netherlands.

In the testbed, we have also implemented an algorithm to compute the maximal buffer requirements for the arcs of the RSFG. The algorithm is based on an operational model in which the actors are fired according to the generated schedule. In particular, the algorithm considers the prologue and the repetitive pattern of the schedule and fires the actors accordingly. The algorithm also keeps track of the maximum number of tokens present on every arc during the execution of the repetitive pattern. These numbers give the buffer requirements on individual arcs of the RSFG. From these values the total buffer requirements for the RSFG is computed.

5.2 Experimental Results and Comparisons

We compare the schedules generated by the MBRO formulation with those generated using (i) the MRSP formulation [9], (ii) the block scheduling method [16], and (iii) the linear programming formulation based on homogeneous graphs [19].

5.2.1 Comparison with MRSP Schedules

In the first group of experiments, we evaluate how well the MBRO formulation minimizes the buffer requirements in comparison with the MRSP schedules [9]. The performance of the MBRO schedules is compared with the MRSP schedules for more than 200 randomly generated RSFGs, and the results for 10 candidate cases are shown in Table 5. For each case, we report the number of nodes and the number of arcs in the RSFG, the optimal computation period $\frac{T}{r}$, the buffer requirements in the MRSP and in the MBRO schedules, execution times (in CPU seconds) to solve the respective formulations on a Sun-SPARC workstation. We also report the

No. of Nodes	No. of Arcs	MRSP Schedules			MBRO Schedules			% Buffer Improvement
		Period	Buffer Required	Execution Time (CPU Sec.)	Period	Buffer Required	Execution Time (CPU Sec.)	
6	6	15	36	0.083	15	35	0.200	2.78
16	15	25	265	0.217	25	219	0.683	17.36
20	19	240	295	0.550	240	179	3.833	39.32
25	24	243	653	1.033	243	350	8.566	46.40
28	27	392	493	0.550	392	240	3.817	51.31
28	47	945	1646	2.100	945	1334	70.997	18.95
35	45	875	2278	2.433	875	1206	91.280	47.05
39	43	150	876	0.933	150	835	11.066	4.68
43	53	315	1586	2.050	315	1223	103.179	22.88
55	93	252	3895	3.400	252	3016	112.762	22.56

Table 5: **MRSP vs. MBRO Schedules on Random RSFGs**

Application Program	MRSP Schedules			MBRO Schedules			Buffer Improvement
	Period	Buffer Required	Execution Time (CPU Sec.)	Period	Buffer Required	Execution Time (CPU Sec.)	
Phase Locked Loop	5	23	0.033	5	19	0.133	21.05
Voice Band	16	23	0.083	16	23	0.133	0.00
Power Spectrum	128	180	0.350	128	153	4.000	15.00
Auto Correlation	256	364	2.700	256	229	120.812	37.08
Periodogram	256	245	2.100	256	197	104.462	19.59
Comb Filter	2	33	0.117	2	33	0.133	0.00
Satellite Receiver	151	625	3.333	151	427	295.172	31.68
Spectrum Analyzer	84	53	0.083	84	47	0.217	11.32

Table 6: **MRSP vs. MBRO Schedules on DSP Applications**

percentage improvement achieved by the MBRO schedules with respect to the MRSP schedules computed as:

$$\%\ \text{improvement} \quad = \frac{\text{buffer for MRSP} - \text{ buffer for MBRO}}{\text{buffer for MRSP}} \tag{18}$$

Table 6 summarizes the comparisons for the considered DSP applications. We observe an average improvement of 26% in buffer requirements for the MBRO schedules. In certain cases, e.g. for the random RSFG with 28 nodes and 27 arcs, the improvement is as high as 51% and for the auto-correlation program it is 37%. The execution time of the MBRO schedule is comparable to that of the MRSO schedules for smaller RSFGs. But for larger RSFGs even though the execution time for the MBRO formulation is significantly larger, it is still in the acceptable range of compilation time for application specific programs. Out of the 200 experiments conducted, in all but 2 cases, the MBRO schedules perform better than the MRSP schedules in terms of buffer requirements.

No. of Nodes	No. of Arcs	Block Schedules			MBRO Schedules			% Buffer Improvement	% Improvement in Period
		Period	Buffer Required	Execution Time (CPU Sec.)	Period	Buffer Required	Execution Time (CPU Sec.)		
6	6	17	36	0.017	15	35	0.200	2.78	11.76
10	13	39	385	0.000	27	347	1.350	9.86	30.76
20	19	299	295	0.017	240	179	3.833	39.32	19.17
28	27	424	493	0.017	392	240	3.817	51.31	7.54
34	38	185	1135	0.033	139	797	13.716	29.78	24.86
37	46	385	1062	0.050	384	942	55.748	11.30	0.25
39	43	229	876	0.017	150	835	11.066	4.68	34.49
41	67	511	1686	0.050	240	1562	54.464	7.35	53.03
55	93	327	3637	0.050	252	3016	112.762	17.07	22.93
56	82	442	1693	0.067	243	1863	51.031	-8.36	45.02

Table 7: **Block Schedules vs. MBRO Schedules on Random RSFGs**

5.2.2 Comparison with Block Schedules

In the second group of experiments, we compare our scheduling method with the well-known block scheduling method of Lee and Messerschmitt [16]. The block scheduling method is based on an operational approach which uses the *earliest firing rule*[8]. The block schedules are also known as Periodic Admissible Parallel Schedules (PAPS). In all experiments, we assume the availability of infinite processor resources for the block schedules since the MBRO schedules are also based on the same assumption. The block scheduling method is applied to an unrolling RSFG, unrolled a suitable number of times. In our experiments we unroll the RSFG by r times, where $\frac{r}{T}$ is the optimal computation rate in the irreducible form. A similar unrolling effect takes place when generating MBRO schedules, even though the unrolling is never performed explicitly in either of the scheduling methods.

The results of both MBRO and the block scheduling for 10 randomly RSFGs are presented in Table 7. In this table, we report the computation period, buffer requirements, and execution time of the two scheduling methods. The columns on percentage improvement shows the improvement in computation period and buffer requirements calculated as in Equation (18). From Tables 7, we observe that the MBRO schedules perform better than the block schedules in terms of both computation period and buffer requirements. In a block schedule, as the name indicates, the execution of nodes from different iterations are not overlapped. In fact, the execution of all nodes in one iteration must be completed before the execution of any node in the next iteration. However, it is interesting to note that the lower computation rate of the block schedules do not bring any buffer storage improvement. The average improvement is 35% in computation period and 16% in buffer requirements.

In Table 8 we compare the block scheduling method with the MBRO formulation on the considered DSP application programs. For the DSP applications (refer to Table 8), the MBRO schedules perform better than the block schedules only in terms of computation rate. The buffer requirements for the block schedule are less than that for the corresponding MBRO schedules However the computation period of the block schedule is significantly larger. In an MBRO schedule, since the operations from different iterations overlap, there is a good possibility that

Application Program	Block Schedules			MBRO Schedules			Buffer Improvement	% Improvement in Period
	Period	Buffer Required	Execution Time (CPU Sec.)	Period	Buffer Required	Execution Time (CPU Sec.)		
Phase Locked Loop	15	5	0.001	5	19	0.133	-42.42	66.67
Voice Band	21	10	0.001	16	23	0.133	-36.11	23.80
Power Spectrum	168	180	0.017	128	153	4.000	15.00	23.80
Auto Correlation	559	156	0.083	256	229	120.812	-24.17	54.20
Periodogram	532	113	0.050	256	197	104.462	-29.89	51.87
Comb Filter	12	7	0.001	2	33	0.133	-44.06	83.33
Satellite Receiver	175	673	0.067	151	427	296.321	36.55	13.71
Spectrum Analyzer	96	44	0.017	84	47	0.217	-6.82	12.50

Table 8: **Block Schedules vs. MBRO Schedules on DSP Applications**

the MBRO schedule require more memory than the non-overlapped block schedules. Lastly, the compile time to construct the block schedules are significantly lower than that for MBRO schedules.

5.2.3 Comparison with Homogeneous Scheduling Methods

A linear programming formulation (called the Optimal Schedule Buffer Allocation (OSBA) formulation) to obtain rate-optimal buffer-optimal schedule for homogeneous graphs has been proposed in [19]. Unlike the MBRO formulation, the one proposed in [19] uses tight bounds for buffer estimates and therefore always generates schedules that have optimal buffer requirements. Hence it may be interesting to evaluate MBRO schedules against the schedules generated by the OSBA formulation. The OSBA formulation works on the equivalent homogeneous graph (e.g. the graph shown in Fig. 2), rather on the original RSFG itself (refer to Fig. 1). Once the optimal schedule is obtained from the OSBA formulation, buffer requirements for the arcs of the RSFG can be determined by using the operational method on the original RSFG. The reason for working with the RSFG, rather than with the homogeneous graph, in determining the buffer requirement is as follows. In the RSFG, buffer space allocated for an arc is shared by all instances of the producing and consuming nodes. However, in the homogeneous graph, independent buffers need to be allocated for each homogeneous arc. Since the former allows the sharing of buffers between various instances of the producer and consumer nodes, it results in lesser buffer requirements.

Tables 9 and 10 summarize the comparison between MBRO schedules and OSBA schedules. The computation rates of an OSBA schedule is equal to that of an MBRO schedule. However, surprisingly, in 75% of the test graphs used in our experiments, the MBRO schedules require lesser storage than the buffer-optimal schedules generated by the OSBA formulation. This could be due to the following reason. The OSBA schedules are in fact buffer-optimal when

No. of Nodes	No. of Arcs	OSBA Schedules			MBRO Schedules			% Buffer Improvement
		Period	Buffer Required	Execution Time (CPU Sec.)	Period	Buffer Required	Execution Time (CPU Sec.)	
10	13	27	373	2.483	27	347	1.350	6.96
16	15	25	223	1.367	25	219	0.683	1.79
20	19	240	187	5.316	240	179	3.833	4.27
25	24	243	423	9.783	243	350	8.566	17.25
28	27	392	279	4.983	392	240	3.817	13.97
35	45	875	1113	47.698	875	1206	91.280	-7.15
37	46	384	1029	44.048	384	942	55.748	8.45
39	43	150	775	14.899	150	835	11.066	-6.70
41	67	240	1646	51.531	240	1562	54.464	5.10
56	82	243	1884	46.215	243	1863	51.031	1.11

Table 9: **OSBA Schedules vs. MBRO Schedules on Random RSFGs**

Application Program	OSBA Schedules			MBRO Schedules			Buffer Improvement
	Period	Buffer Required	Execution Time (CPU Sec.)	Period	Buffer Required	Execution Time (CPU Sec.)	
Phase Locked Loop	5	19	0.133	5	19	0.133	0.00
Voice Band	16	23	0.150	16	23	0.133	0.00
Power Spectrum	128	173	2.000	128	153	4.000	11.56
Auto Correlation	256	247	12.416	256	229	120.812	7.28
Periodogram	256	197	8.166	256	197	104.462	0.00
Comb Filter	2	33	0.117	2	33	0.133	0.00
Satellite Receiver	151	434	34.515	151	427	296.321	1.61
Spectrum Analyzer	84	44	0.183	84	47	0.217	-6.82

Table 10: **OSBA Schedules vs. MBRO Schedules on DSP Applications**

no buffer-sharing is considered. However, without buffer-sharing the total buffer requirements for an RSFG could increase by 50% to 100%. This is the reason why we used the RSFGs in the first place to compute the buffer requirements. Buffer sharing brings in a significant improvement; however, with buffer sharing, the OSBA schedules will no longer be buffer optimal. Since the MBRO formulation uses the RSFGs directly, it captures the notion of buffer sharing. As a consequence the objective function of the MBRO formulation is more effective in minimizing the buffer requirements.

5.2.4 Summary

We summarize the observations made from our experiments as follows

- For the examples we used, the MBRO schedules perform significantly better than the MRSP schedules, in terms of buffer requirements. The improvement is 26% on the average and upto 51% in certain cases.

- Compared to Lee's block scheduling method, MBRO schedules perform 35% better in terms of the computation period for both the random RSFGs and for the considered DSP applications. However a 16% improvement in buffer requirements is observed in the MBRO schedules only for the random RSFGs.

- The MBRO formulation obtains rate-optimal buffer minimized schedules directly from the RSFG. In 75% of our experiments the MBRO schedules require lesser storage than buffer-optimal OSBA schedule [19] obtained from the equivalent homogeneous graph.

- Lee's block scheduling method requires the least amount of compile time (time to construct a schedule). The overall compile time for the MBRO and OSBA schedules do not differ significantly for the test cases we used, while that for the MRSP schedules are comparable to the block schedules.

- In a total of 200 experiments, only in 1% of the cases (i.e. 2 cases) MBRO formulation produced schedules which are inferior, in terms of buffer requirements, to either MRSP schedules or block schedules. However, in all cases, the computation period of the MBRO schedules is significantly better than that of the block schedules.

6 Related Work

Our earlier work on rate-optimal schedules [10] deals only with (computation) rate-optimality; it does not consider minimizing buffer requirement. This paper extends the above work to choose, among the various rate-optimal schedules, the schedule that has lesser buffer space requirement.

In [16], a static scheduling method for synchronous dataflow graphs has been presented by Lee, *et al.* Lee's method, which is based on the operational model, constructs non-overlapped schedules and hence may not always lead to rate-optimality. Further this scheduling does not consider reducing buffer requirements. However, the block scheduling method can handle processor resource constraints which are not considered by either MRSP or MBRO schedules. Further, several other scheduling methods, some of them based on the block scheduling method, with different objectives have been proposed by Lee's research group. For example, methods to construct compact loop schedules, called single-appearance schedules have been proposed in [1]. Sih and Lee have studied the scheduling of acyclic synchronous dataflow graphs taking into consideration the interprocessor communication delays, another issue not addressed by MRSP or MBRO formulation. They have proposed several solution methods for scheduling the acyclic graphs [27]. Compile-time scheduling of dataflow program graphs with dynamic constructs was reported in [11].

In [3], algorithms to construct either space-optimal or maximally-concurrent schedules have been discussed. Their method is based on the operational model and does not overlap operations belonging to successive iterations. Parhi and Messerschmitt [21] have studied the static rate-optimal scheduling of iterative dataflow flow programs via optimal unfolding. However, their method has focused *only* on homogeneous dataflow graphs and does not try to optimize buffer space requirements. Recently, Lucke and Parhi [18] have proposed a novel resource-constrained scheduling method for homogeneous graphs. Their approach is based on integer programming and can optimally unfold, retime, and pipeline the graph to achieve rate-optimal schedules with minimum processor requirement. The method presented by Chao and Sha [2] to obtain the optimal computation rate for a homogeneous dataflow graph is somewhat similar to Parhi's method [21], but is capable of obtaining either a pipelined or a non-pipelined schedule.

Printz [23] discusses a method to map large-grain synchronous signal processing programs onto multiprocessor machines using a style which combines serial, systolic, and data-parallel implementations. The scheduling and mapping obtained using this method are not optimal, though it is guaranteed that schedule is within a certain factor from the optimality. A complete DSP design environment, called McDAS was developed by Hoang [12]. McDAS consists of a scheduler which is based on the *iterative clustering* approach. A method to construct compact looped schedules for a variant of synchronous dataflow graphs, called scalable synchronous dataflow graphs, has been proposed by Ritz, et. al. [26]. Related work applied in the areas of high-level synthesis and code generation can be found in [25, 22]. A wealth of literature has been reported in the area of software pipelining for general-purpose computations. The reader is referred to [20] for an excellent review of these work.

7 Conclusion

In this paper we have studied the problem of constructing rate-optimal schedules which also require minimum buffer requirements. The problem of rate-optimal scheduling has been formulated, called the MBRO formulation, as a linear programming problem. A unique feature of our method is that it does scheduling and buffer minimization *simultaneously*. The construction process also gives the optimal unfolding factor for rate-optimal schedules. Further, in the proposed schedules the execution of nodes from successive iterations do overlap, exploiting software-pipelining.

The proposed scheduling method is implemented and a scheduling testbed is constructed to evaluate the proposed method on a set of DSP applications and on a set of randomly generated RSFGs. In addition to the MBRO formulation, the rate-optimal multi-rate scheduling method (MRSP) [9], the Optimal Scheduling and Buffer Allocation (OSBA) method [19], and the Periodic Admissible Parallel Schedule method (PAPS) [16] have also been implemented. In a large majority of cases, the MBRO schedules perform significantly better than the MRSP and the PAPS schedules in terms of buffer requirements. Also, the MBRO schedules perform considerably better than the PAPS schedules in terms of computation rate. The total execution time for the MBRO method is comparable to MRSP or OSBA scheduling methods.

As future work, we plan to investigate methods to further reduce the memory requirements of MBRO schedules. Currently, the MBRO scheduling method captures buffer sharing between a pair of producer and consumer nodes. However, it does not capture the sharing of buffers among different pairs of producer and consumer nodes. Ning and Gao have a proposed a method (as a second phase following their OSBA scheduling method) to perform buffer sharing among all arcs in a homogeneous graph using a register coloring method [19]. We plan to investigate the coloring method to further minimize the buffer requirements of the MBRO schedules. Also we plan to implement Cubric and Panangaden's space-optimal scheduling method in our testbed and compare the generated schedules with the MBRO schedules.

8 Acknowledgments

We wish to thank Qi Ning, Erik R. Altman, Guoning Liao, and Eric Masson for the fruitful discussions we had with them. The authors are thankful to Prof. Edward A. Lee and Shuvra Bhattacharyya for their suggestions in the comparative study of various scheduling methods. Further some of the DSP applications reported in this paper are taken from Ptolemey : An Environment for Heterogeneous System Simulation and Prototyping, developed in the University of California, Berkeley by Prof. Lee and his research group. The authors thank V. Živojnović and S. Ritz for their contribution in modeling the spectrum analyzer and the satellite receiver applications. The authors acknowledge Natural Science and Engineering Research Council (NSERC) and MICRONET, Network of Centers of Excellence for supporting this work. Finally the support received from the members of ACAPS (Advanced Computer Architecture and Program Structures Group) at McGill University is acknowledged.

References

[1] S.S. Bhattacharyya and E.A. Lee. Scheduling synchronous dataflow graphs for efficient looping. *Journal of VLSI Signal Processing*, 1993. to appear.

[2] L-F. Chao and E.H-M. Sha. Rate-optimal static scheduling for DSP dataflow programs. In *Proc. of the 1993 Great Lakes Symposium on VLSI*, pages 80–84, March 1993.

[3] M. Cubric and P. Panangaden. Minimal memory schedules for data flow networks. In E.Best, editor, *Proc. of the 4th Intl. Conf. on Concurrency Theory*, Aug. 1993. Lecture Notes in Computer Science 715.

[4] Jack B. Dennis. First version of a data-flow procedure language. In *Proc. of the Colloque sur la Programmation*, number 19 in Lec. Notes in Comp. Sci., pages 362–376. Springer-Verlag, 1975.

[5] P. Desai. An implementation of a multi-rate scheduling testbed. ACAPS Technical Note, School of Computer Science, McGill University, Montreal, Que., 1993.

[6] G.R. Gao, R. Govindarajan, and P. Panangaden. Construction rules for well-behaved stream programs. ACAPS Technical Memo 26, School of Computer Science, McGill University, Montreal, Que., April 1992.

[7] G.R. Gao, R. Govindarajan, and P. Panangaden. Well-behaved dataflow programs for DSP computation. In *Proc. of the 1992 Intl. Conf. on Acoustics, Speech, and Signal Processing*, pages 561–564, San Francisco, CA, March 1992.

[8] Guang R. Gao, Yue-Bong Wong, and Qi Ning. A Petri-Net model for fine-grain loop scheduling. ACAPS Tech. Memo 18, Sch. of Comp. Sci., McGill U., Montréal, Qué., Jan. 1991.

[9] R. Govindarajan and G.R. Gao. A novel framework for multi-rate schedule in DSP applications. In *Proc. of the 1993 Intl. Conf. on Application Specific Array Processors*, pages 77–88, Venice, Italy, October 1993.

[10] R. Govindarajan and Guang R. Gao. Rate-optimal schedule for multi-rate DSP computations. ACAPS Tech. Memo 61, Sch. of Comp. Sci., McGill U., Montréal, Qué., Aug. 1993.

[11] S. Ha and E.A. Lee. Compile-time scheduling and assignment of data-flow program graphs with data-dependent iteration. *IEEE Transactions on Computers*, C40(11):1225–1238, Nov. 1991.

[12] P.D. Hoang. Compiling real-time digital signal processing applications onto multiprocessor systems. Memorandum UCB/ERL M92/68 (Ph.D. Dissertation), College of Engineering, University of California, Berkeley, 1992.

[13] R.M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[14] Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Saunders College Publishing, Ft Worth, TX, 1976.

[15] E.A. Lee. Static scheduling of dataflow programs for DSP. In L. Bic and J.-L. Gaudiot, editors, *Dataflow Computation*, chapter 19, pages 501–526. Prentice-Hall, Engelwood Cliffs, NJ, 1991.

[16] E.A. Lee and D.A. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, C36(1):24–35, January 1987.

[17] G. Liao, E.R. Altman, V.K. Agarwal, and G.R. Gao. A comparative study of DSP multiprocessor list scheduling heuristics. In *Proc. of the Intl. Hawaii System Science Conf.*, Hawaii, 1994.

[18] L.E. Lucke and K.K. Parhi. Generalized ILP scheduling and allocation for high-level DSP synthesis. In *Proc. of the 1993 IEEE Custom Integrated Circuits Conf.*, 1993.

[19] Q. Ning and G. R. Gao. A novel framework of register allocation for software pipelining. In *Conf. Rec. of the Twentieth Ann. ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pages 29–42, Charleston, South Carolina, Jan. 10–13, 1993. ACM SIGACT and SIGPLAN.

[20] Qi Ning. *Register Allocation for Optimal Loop Scheduling*. PhD thesis, School of Computer Science, McGill University, 3480 University Street, Montreal, H3A 2A7, Canada, 1993.

[21] K.K. Parhi and D.G. Messerschmitt. Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding. *IEEE Transactions on Computers*, pages 178–195, Feb. 1991.

[22] D.B. Powell, E.A. Lee, and W.C. Newmann. Direct synthesis of optimized DSP assembly code from signal flow block diagrams. In *Proc. of the 1992 Conf. on Acoustics, Speech, and Signal Processing*, San Francisco, March 1992.

[23] H. Printz. Automatic mapping of large signal processing systems to a parallel machine. Memorandum CMU-CS-91-101 (Ph.D. Dissertation), Computer Science Department, Carnegie-Mellon University, 1991.

[24] Raymond Reiter. Scheduling parallel computations. *J. of the ACM*, 15(4):590–599, 1968.

[25] S. Ritz, M. Pankert, and H. Meyr. High level software synthesis for signal processing systems. In *Proc. of the 1992 Conf. on Application Specific Array Processors*, San Francisco, August 1992.

[26] S. Ritz, M. Pankert, V. Živojnović, and H. Meyr. Optimum vectorization of scalable synchronous dataflow graphs. In *Proc. of the 1993 Intl. Conf. on Application Specific Array Processors*, pages 285–296, Venice, Italy, October 1993.

[27] G.C. Sih. Multiprocessor scheduling to account for interprocessor communication. Memorandum, UCB/ERL M91/29 (Ph.D. Dissertation), EECS Department, University of California, Berkeley, 1991. to appear.

A Regular Stream Flow Graphs

In this section, we present an informal description of RSFGs. For more details the reader is referred to [6].

In an RSFG the nodes, also called *regular actors*, are connected by means of arcs or channels. Associated with each node is a set of input channels and a set of output channels. The nodes communicate by passing a sequence of values, called tokens. A sequence of tokens is also referred to as a *stream.* A node consumes a fixed number of tokens (greater than or equal to 1) on each input channel and produces a fixed number of tokens on every output channel. A node represents an action performed on input tokens to produce certain output tokens on the output arcs.

A potential problem in RSFGs is that since the rate at which the tokens are produced in a channel may be different from the rate at which they are consumed, there could be potential token accumulation. An RSFG may have a multiple firing rate of the actors such that after firing the actors with this rate there is neither token accumulation nor token depletion on any arc. Informally, if such a multiple firing rate exists, then we say the graph is *well-behaved.* In this paper we are interested only in well-behaved RSFGs. If a schedule of an RSFG does not obey the multiple firing rate, at least one of the arcs in the RSFG will require unbounded buffer size.

The multiple firing rate of an RSFG can be obtained by solving a set of flow balance equations, relating the number of firing of the actors of a graph and the number of tokens produced (or consumed) on each channel. As an example, for the RSFG shown in Fig. 1 the firing rates are 2, 1, 3, and 2 for the actors a, b, c, and d respectively.

B The Multi-Rate Scheduling Testbed

In this section we present some details of the scheduling testbed used in our experiments. Due to space restriction, only a brief and informal description of the experimental testbed is presented here. The reader is referred to [5] for further details on the implementation. The testbed is implemented using C programming language on a Unix platform. It consists of four main modules, namely the random graph generator, the module to generate and solve the MRSP formulation, the module to generate and solve the MBRO formulation, and the module to compute the buffer requirements for each arc in the RSFG.

The random graph generator generates multi-rate graphs in an incremental fashion. At each incremental step, the graph generated thus far is ensured to be well-behaved. The following attributes of the generated RSFG are exponentially distributed random values. (i) the number of arcs per node, (ii) the number of tokens consumed/produced by a node on an input/output arc, and (iii) the execution time of each node, The mean value for the above parameters can be set independently. The total number of nodes in the RSFG can also be specified. Once a graph of a given number of nodes has been generated, the graph is checked for connectivity and deadlocks. Informally, connectivity ensures that the randomly generated graph is a single graph and not a collection of a number of disjoint graphs. The deadlock algorithm introduces sufficient delays (or tokens) on (feedback) arcs such that the graph is not deadlocked due to directed cycles in the graph. After connectivity check and deadlock detection, the user is allowed to interactively add additional delays on the arcs of the RSFG.

The subsequent module generates the linear constraints (corresponding to Equations (3) and (4)) for the random RSFG. As explained in [10], the linear constraints can be efficiently solved by relating them to the Minimum Cost-to-Time Ratio Cycle (MCTRC) problem. The MCTRC problem itself can be solved using a binary search for optimal $\frac{T}{r}$ and using a shortest path algorithm [14]. A rate-optimal schedule (not necessarily with minimal buffer requirement) for the RSFG can be generated by solving the linear inequalities with the optimal $\frac{T}{r}$ value. To solve the linear equations, we use a freeware software package called *lp_solve*, developed in the Eindhoven University of Technology, The Netherlands. Once the optimal $\frac{T}{r}$ is calculated the inequalities corresponding to the MBRO problem can be generated. The linear programming problem to minimize $\sum_{(u,v)} b_{uv}$, subject to the buffer constraints (Constraint I of the MBRO formulation), and the dependency constraints (Constraints II and III of the MBRO formulation) is solved using *lp_solve*.

In the testbed, we have also implemented an algorithm to compute the maximal buffer requirement on each arc of the RSFG. The algorithm is based on an operational model in which the actors are fired according to the generated schedule. In particular, the algorithm considers the prologue and the repetitive pattern of the schedule and fires the actors accordingly. The algorithm also keeps track of the maximum number of tokens present on every arc during the execution of the repetitive pattern. These numbers give the buffer requirement on individual arcs of the RSFG. From these values the total buffer requirements for all the arcs in the RSFG is computed.

In addition to the MRSP and MBRO scheduling methods, we have also implemented the block scheduling method of Lee and Messerschmitt [16] and the homogeneous scheduling method based on the OSBA formulation [19].