
LOOPED SCHEDULES FOR DATAFLOW DESCRIPTIONS OF MULTIRATE SIGNAL PROCESSING ALGORITHMS

Shuvra S. Bhattacharyya  
Edward A. Lee  

Department of Electrical Engineering and Computer Sciences  
University of California  
Berkeley, California 94720  

December 7, 1993

ABSTRACT

The synchronous dataflow (SDF) programming paradigm has been used extensively in design environments for multirate signal processing applications. In this paradigm, the repetition of computations is specified by the relative rates at which the computations consume and produce data. This implicit specification of iteration allows a compiler to easily explore alternative nested loop structures for the target code with respect to their effects on code size, buffering requirements and throughput. In this paper, we develop important relationships between the SDF description of an algorithm and the range of looping structures offered by this description, and we discuss how to improve code efficiency by applying these relationships.

1 Introduction

Synchronous dataflow (SDF) is a restricted form of the dataflow model of computation [5]. In the dataflow model, a program is represented as a directed graph. The nodes of the graph, also called actors, represent computations and the arcs represent data paths between computations. In SDF [15], each node consumes a fixed number of data items, called tokens or samples, per invocation and produces a fixed number of output samples per invocation. Figure 1 shows an SDF graph that has three actors A, B, and C. Each arc is annotated with the number of samples

This research was sponsored by ARPA and the United States Air Force.
produced by its source actor and the number of samples consumed by its sink actor. The “D” on
the arc between B and C represents a unit *delay*, which can be viewed as an initial sample that is
queued on the arc. SDF and related models have been studied extensively in the context of syn-
thesizing assembly code for signal processing applications, for example [7, 8, 9, 10, 17, 18, 19,
20].

In SDF, *iteration* is defined as the repetition induced when the number of samples pro-
duced on an arc (per invocation of the source actor) does not match the number of samples con-
sumed (per sink invocation) [12]. For example, in figure 1, actor B must be invoked two times for
every invocation of A. Multirate applications often involve a large amount of iteration and thus
subroutine calls must be used extensively, code must be replicated, or loops must be organized in
the target program. The use of subroutine calls to implement repetition may reduce throughput
significantly, however, particularly for graphs involving small granularity. On the other hand, we
have found that code duplication can quickly exhaust on-chip program memory [11]. As an alter-
native, we examine the problem of arranging loops in the target code.

In [11], How demonstrated that by clustering connected subgraphs that operate at the same
repetition-rate, and scheduling these consolidated subsystems each as a single unit, we can often
synthesize loops effectively. This technique was extended in [3] to cluster across repetition-rate
changes and to take into account the minimization of buffering requirements. Although these
techniques proved effective over a large range of applications, they do not always yield the most
compact schedule for an SDF graph [2].

In this paper we define a simple optimality criterion for the synthesis of compact loop-
structures from an SDF graph. The criterion is based on the *looped schedule* notation introduced
in [3], in which loops in a schedule are represented by parenthesized terms of the form (*n M₁ M₂
... Mₖ*), where *n* is a positive integer, and each *Mᵢ* represents an SDF actor or another (nested)

![Fig. 1. A simple SDF graph. Each arc is annotated with the number of samples produced by its source and the number of samples consumed by its sink. The “D” designates a unit delay.](image-url)
loop. For the graph in figure 1, for example, the looped schedule A(2 BC) specifies the firing sequence ABCBC. Using this notation, we can define an optimally-compact looped schedule as one that contains only one lexical appearance of each actor in the SDF graph. We call such an “optimal” looped schedule a **single appearance** schedule. More precisely, a single appearance schedule is a looped schedule in which no actor appears more than once. For example the looped schedule CA(2 B)C for figure 1 is not a single appearance schedule since C appears twice. Thus, either C must be implemented with a subroutine, or we must insert two versions of C’s code block into the synthesized code. In the schedule A(2 CB) however, no actor appears more than once, so it is a single appearance schedule; thus it allows in-line code generation without a code-size penalty.

Our observations suggest that we can construct single appearance schedules for most practical SDF graphs [2]. In this paper, we formally develop transformations that can be applied to single appearance schedules to improve the efficiency of the target code. We also determine necessary and sufficient conditions for an SDF graph to have a single appearance schedule. These conditions were developed independently, in a different form, by Ritz et al. [20], although their application of the condition is quite different to ours. Ritz et al. discuss single appearance schedules in the context of **minimum activation** schedules, which minimize the number of “context-switches” between actors. For example, in the looped schedule A(2 CB) for figure 1, the invocations of B and C are interleaved, and thus a separate activation is required for each invocation — 5 total activations are required. On the other hand, the schedule A(2 B)(2 C) requires only three activations, one for each actor. In the objectives of [20], the latter schedule is preferable, because in that code-generation framework, there is a large overhead involved with each activation. With effective register allocation and instruction scheduling, such overhead can often be avoided, however, as [18] demonstrates. Thus, we prefer the former schedule, which has less looping overhead and requires less memory for buffering.

Our focus has been on creating a general framework for developing scheduling algorithms that provably generate single appearance schedules when possible, and that incorporate other scheduling objectives, such as the minimization of buffering requirements, in a manner that is guaranteed not to interfere with code compaction goals. The framework modularizes different
parts of the scheduling process, and the compiler developer has freedom to experiment with the component modules, while the framework guarantees that the interaction of the modules does not impede code size minimization goals. We have applied conditions for the existence of a single appearance schedule to define our scheduling framework. Due to space limitations, we do not elaborate further on this scheduling framework in this paper; instead, we refer the reader to [2].

We begin with a review of the SDF model of computation and the terminology associated with looped schedules for SDF graphs. SDF principles were introduced [13] in terms of connected graphs. However, for developing scheduling algorithms it is useful to consider non-connected graphs as well, so in section 3 we extend SDF principles to non-connected SDF graphs. In sections 4 and 5, we discuss a schedule transformation called factoring, which can produce large reductions in the amount of memory required for buffering. Finally, in section 6, we develop conditions for the existence of a single appearance schedule, and we discuss the application of these conditions to synthesizing single appearance schedules whenever they exist. The sections form a linear dependence chain — each section depends on the previous ones. For reference, a summary of terminology and notation can be found in the glossary at the end of the paper.

2 Background

2.1 Synchronous Dataflow

An SDF program is normally translated into a loop, where each iteration of the loop executes one cycle of a periodic schedule for the graph. In this section we summarize important properties of periodic schedules.

For an SDF graph G, we denote the set of nodes in G by N(G) and the set of arcs in G by A(G). For an SDF arc α, we let source(α) and sink(α) denote the nodes at the source and sink of α; we let p(α) denote the number of samples produced by source(α), c(α) denote the number of samples consumed by sink(α), and we denote the delay on α by delay(α). We define a subgraph of G to be that SDF graph formed by any Z ⊆ N(G) together with the set of arcs {α ∈ A(G) |
source(α), sink(α) ∈ Z}. We denote the subgraph associated with the subset of nodes Z by subgraph(Z, G); if G is understood, we may simply write subgraph(Z).

We can think of each arc in G as having a FIFO queue that buffers the tokens that pass through the arc. Each FIFO contains an initial number of samples equal to the delay on the associated arc. Firing a node in G corresponds to removing c(α) tokens from the head of the FIFO for each input arc α, and appending p(β) tokens to the FIFO for each output arc β. After a sequence of 0 or more firings, we say that a node is fireable if there are enough tokens on each input FIFO to fire the node. An admissible schedule for G is a sequence S = S₁ S₂ ... Sₙ of nodes in G such that each Sᵢ is fireable immediately after S₁, S₂, ..., Sᵢ₋₁ have fired in succession.

If some Sᵢ is not fireable immediately after its antecedents have fired, then there is least one arc α such that (1) sink(α) = Sᵢ, and (2) the FIFO associated with sink(α) contains less than c(α) tokens just prior to the i-th firing in S. For each such α, we say that S terminates on α at firing Sᵢ. Clearly then, S is admissible if and only if it does not terminate on any arc α.

We say that a schedule S is a periodic schedule if it invokes each node at least once and produces no net change in the number of tokens on a FIFO — for each arc α, (the number of times source(α) is fired in S) × p(α) = (the number of times sink(α) is fired in S) × c(α). A valid schedule is a schedule that is both periodic and admissible. For a given schedule, we denote the i-th firing, or invocation, of actor N by Nᵢ, and we call i the invocation number of Nᵢ.

In [14], it is shown that for each connected SDF graph G, there is a unique minimum number of times that each node needs to be invoked in a periodic schedule. We specify these minimum numbers of firings by a vector of positive integers q₇, which is indexed by the nodes in G, and we denote the component of q₇ corresponding to a node N by q₇(N). Corresponding to each periodic schedule S, there is a positive integer J(S) called the blocking factor of S, such that S invokes each N ∈ N(G) exactly J(S)q₇(N) times. We call q₇ the repetitions vector of G. If G is understood from context, we may refer to q₇ simply as q. From the definition of a repetitions vector, we obtain the following three facts:
**Fact 1:** The components of a repetitions vector are collectively coprime; that is, they have no common divisor that exceeds unity.

**Fact 2:** The balance equation $q(source(\alpha)) \times p(\alpha) = q(sink(\alpha)) \times c(\alpha)$ is satisfied for each arc $\alpha$ in $G$. Also, any positive-integer vector that satisfies the balance equations is a positive-integer multiple of the repetitions vector.

**Fact 3:** Suppose that $G$ is a connected SDF graph and $S$ is an admissible schedule for $G$. If there is a positive integer $J_0$ such that $S$ invokes each $N \in N(G)$ exactly $J_0q(N)$ times, then $S$ is a valid schedule.

Thus a positive-integer vector indexed by the nodes in a connected SDF graph is the repetitions vector for that SDF graph iff it satisfies the balance equations and its components are coprime.

Given an SDF graph $G$, we say that $G$ is **strongly connected** if for any pair of distinct nodes $A$, $B$ in $G$, there is a directed path from $A$ to $B$ and a directed path from $B$ to $A$. We say that a strongly connected SDF graph is **nontrivial** if it contains more than one node. Also, we say that a subset $Z$ of nodes in $G$ is strongly connected if $subgraph(Z, G)$ is strongly connected. Finally, a **strongly connected component** of $G$ is a strongly connected subset of $N(G)$ such that no strongly connected subset of $N(G)$ properly contains $Z$.

Although there is no theoretical impediment to infinite SDF graphs, we currently do not have any practical use for them, so in this paper, we deal only with SDF graphs that have a finite number of nodes and arcs. Also, unless otherwise stated, we deal only with SDF graphs for which a valid schedule exists.

### 2.2 Looped Schedule Terminology

**Definition 1:** A **schedule loop** is a parenthesized term of the form $(n \ T_1 \ T_2 \ldots \ T_m)$, where $n$ is a positive integer and each $T_i$ represents an SDF node or another schedule loop. $(n \ T_1 \ T_2 \ldots \ T_m)$ represents the successive repetition $n$ times of the firing sequence $T_1 \ T_2 \ldots \ T_m$. If $L = (n \ T_1 \ T_2 \ldots \ T_m)$ is a schedule loop, we say that $n$ is the **iteration count** of $L$, each $T_i$ is an **iterand** of $L$, and $T_1 \ T_2 \ldots$
T_m constitutes the body of L. A **looped schedule** is a sequence V_1 V_2 ... V_k, where each V_i is either an actor or a schedule loop. Since a looped schedule is usually executed repeatedly, we refer to each V_i as an *iterand* of the associated looped schedule.

When referring to a looped schedule, we often omit the “looped” qualification if it is understood from context; similarly, we may refer to a schedule loop simply as a “loop”. Given a looped schedule S, we refer to any contiguous sequence of actor appearances and schedule loops in S as a *subschedule* of S. More formally, a subschedule of S is any sequence of successive iterands of S or any sequence of successive iterands of a schedule loop contained in S. For example, the schedules (3 AB)C and (2 B(3 AB)C)A are both subschedules of A(2 B(3 AB)C)A(2 B), whereas (3 AB)CA is not. By this definition, every schedule loop in S is a subschedule of S. If the same firing sequence appears in more than one place in a schedule, we distinguish each instance as a separate subschedule. For example, in (3 A(2 BC)D(2 BC)), “(2 BC)” appears twice, and these correspond to two distinct subschedules. In this case, the content of a subschedule is not sufficient to specify it — we must also specify the lexical position, as in “the second appearance of (2 BC”).

Given a looped schedule S and an actor N that appears in S, we define \( \text{inv}(N, S) \) to be the number of times that S invokes N. Similarly, if \( S_0 \) is a subschedule, we define \( \text{inv}(S_0, S) \) to be the number of times that S invokes \( S_0 \). For example, if \( S = A(2 (3 BA)C)BA(2 B) \), then \( \text{inv}(B, S) = 9 \), \( \text{inv}(3 BA), S) = 2 \), and \( \text{inv}(\text{first appearance of BA, S}) = 6 \). Also, we refer to the schedule that a looped schedule S represents as the firing sequence generated by S. For example, the firing sequence generated by \( A(2 (3 BA)C)BA(2 B) \) is ABABABACBABABACBB. When there is no ambiguity, we occasionally do not distinguish between a looped schedule and the firing sequence that it generates.

Finally, given an SDF graph G, an arc \( \alpha \) in G, a looped schedule S for G, and a nonnegative integer \( i \), we define \( P(\alpha, i, S) \) to denote the number of firings of \( \text{source}(\alpha) \) that precede the \( i \)th invocation of \( \text{sink}(\alpha) \) in S. For example, consider the SDF graph in figure 1 and let \( \alpha \) denote the arc from B to C. Then \( P(\alpha, 2, A(2 BC)) = 2 \), the number of firings of B that precede invocation \( C_2 \) in the firing sequence ABCBC.
3 Non-connected SDF Graphs

The fundamentals of SDF were introduced in terms of connected SDF graphs [13, 15]. In this section, we extend some basic principles of SDF to non-connected SDF graphs. We begin with two definitions.

**Definition 2:** Suppose that G is an SDF graph, M is any subset of nodes in G, and \( M_s \subseteq M \). We say that \( M_s \) is a *maximal connected subset of M* if \( \text{subgraph}(M_s, G) \) is connected, and no subset of M that properly contains \( M_s \) induces a connected subgraph in G. Every subset of nodes in an SDF graph has a unique partition into one or more maximal connected subsets. For example in figure 2, the subset of nodes \{A, B, C, E, G, H\} has three maximal connected subsets: \{A, H\}, \{B, E, C\} and \{G\}. If \( M_s \) is a maximal connected subset of \( N(G) \), then we say that \( \text{subgraph}(M_s, G) \) is a *maximal connected subgraph* of G. We denote the set of maximal connected subgraphs in G by \( \text{max}_{-}\text{connected}(G) \). Thus, for figure 3, \( \text{max}_{-}\text{connected}(G) = \{\text{subgraph}\{A, B\}, \text{subgraph}\{C, D\}\} \).

![Diagram](image)

**Fig. 1.** An example used to illustrate maximally connected subsets. The direction and the SDF parameters for each arc are not shown because they are not relevant to connectedness.

**Definition 3:** Suppose that S is a looped schedule for an SDF graph and \( N_s \subseteq N(G) \). If we remove from S all actors that are not in \( N_s \), and remove all empty loops — schedule loops that contain no actors in their bodies — that result, we obtain another looped schedule, which we call the *restriction* of S to \( N_s \), and which we denote by \( \text{restriction}(S, N_s) \). For example, \( \text{restriction}((2(2\ B)(5\ A)), \{A, C\}) = (2(5\ A)) \), and \( \text{restriction}((5\ C), \{A, B\}) \) is the *null schedule*. If \( G_s \) is a subgraph of G, then we define \( \text{restriction}(S, G_s) = \text{restriction}(S, N(G_s)) \).
The restriction of an admissible schedule $S$ to a subset of actors $N_a$ fully specifies the sequence of token populations occurring on each arc in the corresponding subsystem. More precisely, for any actor $A \in N_a$, any positive integer $i$ such that $1 \leq i \leq inv(A, S)$, and any input arc $\alpha$ of $A$ contained in $\text{subgraph}(N_a)$, the number of tokens queued on $\alpha$ just prior to the $i$th execution of $A$ in $S$ equals the number of tokens queued on $\alpha$ just prior to the $i$th invocation of $A$ in an execution of $\text{restriction}(S, N_a)$. Thus, we have the following fact.

**Fact 4:** If $S$ is a valid schedule for an SDF graph $G$ and $G_a$ is a subgraph of $G$, then $\text{restriction}(S, G_a)$ is a valid schedule for $G_a$.

The concept of blocking factor does not apply directly to SDF graphs that are not connected. For example, in figure 3 the minimal vector of repetitions for a periodic schedule is given

$$\rho(A, B, C, D) = (1, 1, 1, 1).$$

The schedule $A(2)B(2)D$ is a valid schedule for this example, but this schedule corresponds to a blocking factor of 1 for $\text{subgraph}(\{A, B\})$ and a blocking factor of 2 for $\text{subgraph}(\{C, D\})$ — there is no single scalar blocking factor associated with $A(2)B(2)D$.

Now suppose that $S$ is a valid schedule for an arbitrary SDF graph $G$. By fact 4, for each $C \in \text{max_connected}(G)$, we have that $\text{restriction}(S, C)$ is a valid schedule for $C$. Thus, associated with $S$, there is a vector of positive integers $J_S$, indexed by the maximal connected subgraphs of $G$, such that $\forall C \in \text{max_connected}(G), \forall N \in N(C), inv(N, S) = J_S(C)q_C(N)$. We call $J_S$ the blocking vector of $S$. For example, if $S = A(2)B(2)D$ for figure 3, then $J_S(\text{subgraph}(\{A, B\})) = 1$, and $J_S(\text{subgraph}(\{C, D\})) = 2$. On the other hand, if $G$ is connected, then $J_S$ has only one component, which is the blocking factor of $S$, $J(S)$.

It is often convenient to view parts of an SDF graph as subsystems that are invoked as single units. The invocation of a subsystem corresponds to invoking a minimal valid schedule for the associated subgraph. If this subgraph is connected, its repetitions vector gives the minimum num-
ber of invocations required for a periodic schedule. However, if the subgraph is not connected, then the minimum number of invocations involved in a periodic schedule is not necessarily obtained by concatenating the repetitions vectors of the maximal connected subcomponents.

For example, consider the subsystem \( \text{subgraph}\{A, B, D, E\} \) in the SDF graph of figure 4(a). It is easily verified that \( q(A, B, C, D, E) = (2, 2, 1, 4, 4) \). Thus, for a periodic schedule, the actors in \( \text{subgraph}\{D, E\} \) must execute twice as frequently as those in \( \text{subgraph}\{A, B\} \). We see that the minimal repetition rates for \( \text{subgraph}\{A, B, D, E\} \) as a subgraph of the original graph are given by \( \rho(A, B, D, E) = (1, 1, 2, 2) \), which can be obtained dividing each corresponding entry in \( q \) by \( \gcd(q(A), q(B), q(D), q(E)) = \gcd(2, 2, 4, 4) = 2 \). On the other hand, concatenating the repetitions vectors of \( \text{subgraph}\{A, B\} \) and \( \text{subgraph}\{D, E\} \) yields the repetition rates \( \rho'(A, B, D, E) = (1, 1, 1, 1) \). However, repeatedly invoking the subsystem with these relative rates can never lead to a periodic schedule for the enclosing SDF graph. We have motivated the following definition.

**Definition 4:** Let \( G \) be a connected SDF graph, suppose that \( Z \) is a subset of \( N(G) \), and let \( R = \text{subgraph}(Z) \). We define \( q_G(Z) = \gcd(\{q_G(N) | N \in Z\}) \), and we define \( q_{R/G} \) to be the vector of positive integers indexed by the members of \( Z \) that is defined by \( q_{R/G}(N) = q_G(N) / q_G(Z), \forall N \in Z \). \( q_G(Z) \) can be viewed as the number of times a minimal periodic schedule for \( G \) invokes the subgraph \( R \), and we refer to \( q_{R/G} \) as the repetitions vector of \( R \) as a subgraph of \( G \). For example, in

![Graph A](imageA.png)

![Graph B](imageB.png)

**Fig. 4.** An example of clustering a subgraph in an SDF graph.

1. \( \gcd \) denotes the greatest common divisor.
figure 4, if $R = \text{subgraph}\{A, B, D, E\}$, then $q_G(N(R)) = 2$, and $q_{R/G} = q_{R/G}(A, B, D, E) = (1, 1, 2, 2)$.

**Fact 5:** If $G$ is a connected SDF graph and $R$ is a connected subgraph of $G$, then $q_{R/G} = q_R$. Thus for a connected subgraph $R$, $\forall N \in N(R)$, $q_G(N) = q_G(N(R))q_R(N)$.

*Proof.* Let $S$ be any valid schedule for $G$ of unit blocking factor, and let $S' = \text{restriction}(S, R)$. Then from fact 4 and fact 2, for all $N \in N(R)$, we have $q_G(N) = J(S')q_R(N)$. But from fact 1, we know that the components of $q_R$ are coprime. It follows that $J(S') = \gcd\{q_G(N') | N' \in N(R)\} = q_G(N(R))$. Thus, $\forall N \in N(R)$, $q_R(N) = q_G(N) / q_G(N(R)) = q_{R/G}(N)$. QED.

For example, in figure 4(a), let $R = \text{subgraph}\{A, B\}$. We have $q_G(A, B, C, D, E) = (2, 2, 1, 4, 4)$, $q_R(A, B) = (1, 1)$, and from definition 4, $q_G(N(R)) = \gcd(2, 2) = 2$, and $q_{R/G} = (2, 2) / 2 = (1, 1)$. As fact 4 assures us, $q_R = q_{R/G}$.

Finally, we formalize the concept of clustering a subgraph of a connected SDF graph $G$, which as we discussed above, is used to organize hierarchy for scheduling purposes. This process is illustrated in figure 4. Here $\text{subgraph}\{A, B, D, E\}$ of figure 4(a) is clustered into the hierarchical node $\Omega$, and the resulting SDF graph is shown in figure 4(b). Each input arc $\alpha$ to a clustered subgraph $R$ is replaced by an arc $\alpha'$ having $p(\alpha') = p(\alpha)$, and $c(\alpha') = c(\alpha) \times q_{R/G}(\text{sink}(\alpha))$, the number of samples consumed from $\alpha$ in one invocation of $R$ as a subgraph of $G$. Similarly we replace each output arc $\beta$ with $\beta'$ such that $c(\beta') = c(\beta)$, and $p(\beta') = p(\beta) \times q_{R/G}(\text{source}(\beta))$. We will use the following property of clustered subgraphs.

**Fact 6:** Suppose $G$ is an SDF graph, $R$ is a subgraph of $G$, $G'$ is the SDF graph that results from clustering $R$ into the hierarchical node $\Omega$, $S'$ is a valid schedule for $G'$, and $S_R$ is a valid schedule for $R$ such that $\forall N \in N(R)$, $\text{inv}(N, S_R) = q_{R/G}(N)$. Let $S^*$ denote the schedule that results from replacing each appearance of $\Omega$ in $S'$ with $S_R$. Then $S^*$ is a valid schedule for $G$.

As a simple example, consider figure 4 again. Now, $(2 \Omega)C$ is a valid schedule for the SDF graph in figure 4(b), and $S = AB(2 \text{ DE})$ is a valid schedule for $R = \text{subgraph}\{A, B, D, E\}$ such that $\text{inv}(N, S) = q_{R/G}(N) \forall N$. Thus, fact 6 guarantees that $(2 \ AB(2 \text{ DE}))C$ is a valid schedule for figure 4(a).
Proof of fact 6. Given a schedule $\sigma$ and an SDF arc $\alpha$, we define

$$\Delta(\alpha, \sigma) = \text{inv}(\text{source}(\alpha), \sigma) \times p(\alpha) - \text{inv}(\text{sink}(\alpha), \sigma) \times c(\alpha).$$

Then $\sigma$ is a periodic schedule iff it invokes each actor and $\Delta(\alpha, \sigma) = 0 \ \forall \ \alpha$.

We can decompose $S'$ into $s_1 \Omega s_2 \Omega \ldots \Omega s_k$, where each $s_j$ denotes the sequence of firings between the $(j-1)$th and $j$th invocations of $\Omega$. Then $S^* = s_1 S_R s_2 S_R \ldots S_R s_k$.

First, suppose that $\theta$ is an arc in $G$ such that $\text{source}(\theta), \text{sink}(\theta) \not\in N(R)$. Then $S_R$ contains no occurrences of $\text{source}(\theta)$ nor $\text{sink}(\theta)$, so $P(\theta, i, S') = P(\theta, i, S')$ for any invocation number $i$ of $\text{sink}(\theta)$. Thus, since $S'$ is admissible, $S^*$ does not terminate on $\theta$. Also, $\Delta(\theta, S^*) = \Delta(\theta, s_1 s_2 \ldots s_k) = \Delta(\theta, S') = 0$, since $S'$ is periodic.

If $\text{source}(\theta), \text{sink}(\theta) \in N(R)$, then none of the $s_j$’s contain any occurrences of $\text{source}(\theta)$ or $\text{sink}(\theta)$. Thus for any $i$, $P(\theta, i, S^*) = P(\theta, i, S'^*)$ and $\Delta(\theta, S^*) = \Delta(\theta, S'^*)$, where $S'^* = S_R S_R \ldots S_R$ denotes $S^*$ with all of the $s_j$’s removed. Since $S'^*$ consists of successive invocations of a valid schedule, it follows that $S^*$ does not terminate on $\theta$, and $\Delta(\theta, S^*) = 0$.

Now suppose that $\text{source}(\theta) \not\in N(R)$, and $\text{sink}(\theta) \not\in N(R)$. Then corresponding to $\theta$, there is an arc $\theta'$ in $G'$, such that $\text{source}(\theta') = \Omega \ \text{sink}(\theta') = \text{sink}(\theta), p(\theta') = q_{RG}(\text{source}(\theta))p(\theta)$, and $c(\theta') = c(\theta)$. Now each invocation of $S_R$ produces $\text{inv}(\text{source}(\theta), S_R)p(\theta) = q_{RG}(\text{source}(\theta))p(\theta) = p(\theta')$ samples onto $\theta$. Since $c(\theta') = c(\theta)$ and $S'$ is a valid schedule, it follows that $\Delta(\theta, S') = 0$ and $S^*$ does not terminate on $\theta$.

Similarly, if $\text{source}(\theta) \not\in N(R)$, and $\text{sink}(\theta) \in N(R)$, we see that each invocation of $S_R$ consumes the same number of samples from $\theta$ as $\Omega$ consumes from the corresponding arc in $G'$, and thus $\Delta(\theta, S') = 0$ and $S^*$ does not terminate on $\theta$.

We conclude that $S^*$ does not terminate on any arc in $G$, and thus $S^*$ is admissible. Furthermore $\Delta(\alpha, S^*) = 0$ for all arcs $\alpha$ in $G$, and since $S'$ and $S_R$ are both periodic schedules, it easily verified that $S^*$ invokes each actor in $G$ at least once, so we conclude that $S^*$ is a periodic schedule. \textit{QED.}

We conclude this section with a fact that relates the repetitions vector of an SDF graph obtained by clustering a subgraph to the repetitions vector of the original graph.
Factoring Schedule Loops

Fact 7: If $G$ is a connected SDF graph, $Z \subseteq N(G)$, and $G'$ is the SDF graph obtained from $G$ by clustering $\text{subgraph}(Z)$ into the node $\Omega$, then $\mathbf{q}_{G'}(\Omega) = \mathbf{q}_G(Z)$, and $\forall N \notin Z$, $\mathbf{q}_{G'}(N) = \mathbf{q}_G(N)$.

Proof. Let $\mathbf{q}'$ denote the vector that we claim is the repetitions vector for $G'$, and recall from facts 1 and 2 that $\mathbf{q}' = \mathbf{q}_G'$ iff $\mathbf{q}'$ satisfies the balance equations for $G'$ and the components of $\mathbf{q}'$ are coprime. It can easily be verified that $\mathbf{q}'$ satisfies the balance equations for $G'$. Furthermore, from fact 1, no positive integer greater than one can divide all members of $\{\mathbf{q}_G(N) \mid N \notin Z\} \cup \{\gcd(\{\mathbf{q}_G(N) \mid N \in Z\})\}$. Since $\mathbf{q}_G(Z) = \gcd(\{\mathbf{q}_G(N) \mid N \in Z\}$, it follows that the components of $\mathbf{q}'$ are collectively coprime. QED.

4 Factoring Schedule Loops

In this section, we show that in a single appearance schedule, we can “factor” common terms from the iteration counts of inner loops into the iteration count of the enclosing loop. An important practical advantage of factoring is that it may significantly reduce the amount of memory required for buffering.

For example, consider the SDF graph in figure 5. One valid single appearance schedule for this graph is (100 A)(100 B)(10 C) D. With this schedule, prior to each invocation of C, 100 tokens are queued on each of C’s input arcs, and a maximum of 10 tokens are queued on D’s input arc. Thus 210 words of memory are required to implement the buffering for this schedule.

Now observe that this schedule induces the same firing sequence as (1 (100 A)(100 B)(10 C)) D. The result developed in this section allows us to factor the common divisor of 10 in the iteration counts of the three inner loops into the iteration count of the outer loop. This yields the

![Figure 5](image_url)

Fig. 5. An SDF graph used to illustrate the factoring of loops. For this graph, $\mathbf{q}(A, B, C, D) = (100, 100, 10, 1)$. 

13 of 29
new single appearance schedule \((10 \ (10 \ A)(10 \ B)C)D\), for which at most ten tokens simultaneously reside on each arc. Thus this factoring application has reduced the buffering requirement by a factor of 7.

There is, however a trade-off involved in factoring. For example, the schedule \((100 \ A)(100 \ B)(10 \ C)D\) requires 3 \textit{loop initiations} per schedule period, while the factored schedule \((10 \ (10 \ A)(10 \ B)C)D\) requires 21. Thus the runtime cost of starting loops — usually, initializing the loop indices — has increased by the same factor by which the buffering cost has decreased. However the loop-startup overhead is normally much smaller than the penalty that is paid when the memory requirement exceeds the on-chip limits. Unfortunately, we cannot in general perform the reverse of the factoring transformation — i.e. moving a factor of the outer loop’s iteration count into the inner loops. This reverse transformation would be desirable in situations where minimizing buffering requirements is not critical.

In this section, we prove the validity of factoring for an arbitrary “factorable” loop in a single appearance schedule.

\textbf{Definition 5:} Given a schedule \(S_0\), we denote the set of actors that appear in \(S_0\) by \(\text{actors}(S_0)\). For example, \(\text{actors}((2 \ (2 \ B)(5 \ A))) = \{A, B\}\) and \(\text{actors}((3 \ X(2 \ Y(3 \ Z)X))) = \{X, Y, Z\}\).

\textbf{Lemma 1:} Suppose that \(S\) is a single appearance schedule (that is not necessarily a valid schedule) for the SDF graph \(G\), and \(S_0\) is a subschedule of \(S\) such that \(S_0\) is a valid schedule for \(\text{subgraph}(\text{actors}(S_0), G)\). Then \(S\) does not terminate on any arc \(\theta\) for which \(\text{source} (\theta), \text{sink}(\theta) \in \text{actors}(S_0)\).

For example, suppose that \(S\) is the schedule \(D(2 \ A(2 \ BC))E\) for the SDF graph in figure 6, and \(S_0\) is the subschedule \((2 \ A(2 \ BC))\). Lemma 1 guarantees that \(S\) does not terminate on any arc that is contained in \(\text{subgraph}(\{ABC\})\): No matter what the values of the delays \(\{d_i\}\) are, \(S\) does not terminate on the arc from \(A\) to \(B\), nor the arc from \(A\) to \(C\).

\textit{Proof of lemma 1:} Since \(S\) is a single appearance schedule, \(\text{source}(\theta)\) and \(\text{sink}(\theta)\) are invoked only through invocations of \(S_0\). Since \(S_0\) is admissible, the number of samples on \(\theta\) prior to each invocation of \(\text{sink}(\theta)\) is at least \(c(\theta)\). Thus \(S\) does not terminate on \(\theta\). \textit{QED.}
**Lemma 2:** Suppose that $G$ is an SDF graph, $S$ is an admissable looped schedule for $G$, and $S_0$ is a subschedule of $S$. Suppose also that $S'_0$ is any looped schedule such that $\text{actors}(S'_0) = \text{actors}(S_0)$, and $\text{inv}(N, S_0) = \text{inv}(N, S'_0)$ $\forall N \in \text{actors}(S_0)$. Let $S'$ denote the schedule obtained by replacing $S_0$ with $S'_0$ in $S$. Then $S'$ does not terminate on any arc $\theta$ that is not contained in $\text{subgraph}(\text{actors}(S_0), G)$; equivalently, $(\text{source}(\theta) \notin \text{actors}(S_0)$ or $\text{sink}(\theta) \notin \text{actors}(S_0)) \Rightarrow S'$ does not terminate on $\theta$.

Again consider the example in figure 6 and suppose that $D(2 \ A(2 \ BC))E$ is an admissable schedule for this SDF graph. Then lemma 2 (with $S_0 = A(2 \ BC)$, and $S'_0 = BCABC$) tells us that $D(2 \ BCABC)E$ does not terminate on any of the four arcs that lie outside of $\text{subgraph}(\{A, B, C\})$.

Before moving to the proof, we emphasize that lemma 2 applies to general looped schedules, not just single appearance schedules.

**Proof of lemma 2:** Let $\theta$ be any arc that is not contained in $\text{subgraph}(\text{actors}(S_0), G)$. Let $i$ be any invocation number of $\text{sink}(\theta)$; that is, $1 \leq i \leq \text{inv}(\text{sink}(\theta), S')$. The sequence of invocations fired in one period of $S$ can be decomposed into $(s_1 \ b_1 \ s_2 \ ... \ b_i \ s_{n+1})$, where $b_j$ denotes the sequence of firings associated with the $j$th invocation of subschedule $S_0$, and $s_i$ is the sequence of firings between the $(j-1)$th and $j$th invocations of $S_0$. Since $S'$ is derived by rearranging the firings in $S_0$, we can express it similarly as $(s_1 \ b'_1 \ s_2 \ b'_2 \ ... \ b'_i \ s_{n+1})$, where $b'_j$ corresponds to the $j$th invocation of $S'_0$ in $S'$.

If neither $\text{source}(\theta)$ nor $\text{sink}(\theta)$ is contained in $\text{actors}(S_0)$, then none of the $b_j$’s nor any of the $b'_j$’s contain any occurrences of $\text{sink}(\theta)$ or $\text{source}(\theta)$. Thus $P(\theta, i, S) = P(\theta, i, s_1 \ s_2 \ ... \ s_{n+1}) = P(\theta, i, S')$.

![Fig. 6. An example used to illustrate the application of lemmas 1 and 2. Each $d_i$ represents the number of delays on the corresponding arc. Here $q(A, B, C, D, E) = (2, 4, 4, 1, 1)$](image-url)
Factoring Schedule Loops

Now suppose \( source(\theta) \in actors(S_0) \) and \( sink(\theta) \notin actors(S_0) \). Let \( k \) denote the number of invocations of \( S_0 \) that precede \( sink(\theta)_i \) in \( S \). Then, since \( inv(sink(\theta), b_j) = inv(sink(\theta), b'_j) = 0 \ \forall \ j \), we have that \( k \) invocations of \( S_0'_i \) precede \( sink(\theta)_i \) in \( S' \). It follows that \( P(\theta, i, S) = P(\theta, i, S, s_1 s_2 \ldots s_{n+1} + k \times inv(source(\theta), S_0), P(\theta, i, S, s_1 s_2 \ldots s_{n+1}) \) or \( P(\theta, i, S) = P(\theta, i, S, s_1 s_2 \ldots s_{n+1} + k \times inv(source(\theta), S_0) \). But, by assumption, \( inv(source(\theta), S_0) = inv(source(\theta), S_0') \), so \( P(\theta, i, S) = P(\theta, i, S') \).

Finally, suppose \( source(\theta) \notin actors(S_0) \) and \( sink(\theta) \in actors(S_0) \). There are two sub-cases to consider here: (1) In \( S \), \( sink(\theta)_i \) occurs in one of the \( s'_j \)'s, say \( s_k \). Since \( inv(sink(\theta), S_0) = inv(sink(\theta), S_0') \), it follows that in \( S' \), \( sink(\theta)_i \) occurs in \( s_k \) as well. Since \( source(\theta) \notin actors(S_0) \), we have \( P(\theta, i, S) = P(\theta, i, S, s_1 s_2 \ldots s_k) = P(\theta, i, S, s_1 s_2 \ldots s_k) \). (2) In \( S \), \( sink(\theta)_i \) occurs in one of the \( b'_j \)'s, say \( b_m \). Then \( inv(sink(\theta), S_0) = inv(sink(\theta), S_0') \) implies that in \( S' \), \( sink(\theta)_i \) occurs in \( b'_m \). Since \( source(\theta) \notin actors(S_0) \), \( P(\theta, i, S) = P(\theta, i, S) \).

Thus, for arbitrary \( i \), \( P(\theta, i, S) = P(\theta, i, S') \) From the admissability of \( S \), it follows that \( S' \) does not terminate on \( \theta \). \( QED \).

The following theorem establishes the validity of our factoring transformation.

**Theorem 1**: Suppose that \( S \) is a valid single appearance schedule for \( G \) and suppose that \( L = (m (n_1 S_1) (n_2 S_2) \ldots (n_k S_k)) \) is a schedule loop within \( S \) of any nesting depth. Suppose also that \( \gamma \) is any positive integer that divides \( n_1, n_2, \ldots, n_k \), and let \( L' \) denote the loop \( (\gamma m (\gamma^4 n_1 S_1) (\gamma^4 n_2 S_2) \ldots (\gamma^4 n_k S_k)) \). Then replacing \( L \) with \( L' \) in \( S \) results in a valid schedule for \( G \).

**Proof**: We will use the following definition in our proof of this theorem.

**Definition 6**: Given a schedule loop \( L \) in \( S \) and an arc \( \theta \) in \( G \), we define \( consumed(\theta, L) \) to be the number of samples consumed from \( \theta \) by \( sink(\theta) \) during one invocation of \( L \). Similarly, we define \( produced(\theta, L) \) to be the number of samples produced onto \( \theta \) during one invocation of \( L \). Clearly, if the number of samples on \( \theta \) is at least \( consumed(\theta, L) \) just prior to a particular invocation of \( L \), then \( S \) will not terminate on \( \theta \) during that invocation of \( L \).

Let \( S' \) denote the schedule that results from replacing \( L \) with \( L' \) in \( S \). By construction of \( S' \), we have that for all actors \( A \) in \( G \), \( inv(A, S') = inv(A, S) \). Since \( S \) is valid, and hence periodic, it
follows that \( S' \) is a periodic schedule. It remains to be shown that \( S' \) is admissible. We will demonstrate this by induction on \( k \).

First, observe that for \( k = 1 \), \( L \) and \( L' \) generate the same firing sequence, and thus \( S \) and \( S' \) generate the same firing sequence. We conclude that \( S' \) is admissible for \( k = 1 \).

Now consider the case \( k = 2 \). Then \( L = (m (n_1 S_1) (n_2 S_2)) \) and \( L' = (\gamma n (\gamma^1 n_1 S_1) (\gamma^1 n_2 S_2)) \). By construction, \( J(S') = J(S) \) and \( S' \) is also a single appearance schedule. Now let \( \theta \) be an arc in \( G \). If \( source(\theta) \in actors(S_1) \) and \( sink(\theta) \in actors(S_2) \) then

\[
produced(\theta, (\gamma^1 n_1 S_1)) = J(S)q_G(source(\theta))p(\theta) / \text{inv}((\gamma^1 n_1 S_1), S')
\]

\[
= J(S)q_G(source(\theta))p(\theta) / (\gamma m \times \text{inv}(L', S'))
\]

\[
= J(S)q_G(sink(\theta))c(\theta) / (\gamma n \times \text{inv}(L', S')) \quad \text{(by fact 2)}
\]

\[
= consumed(\theta, (\gamma^1 n_2 S_2)).
\]

Similarly, if \( source(\theta) \in actors(S_2) \) and \( sink(\theta) \in actors(S_1) \), \( produced(\theta, (\gamma^1 n_2 S_2)) = consumed(\theta, (\gamma^1 n_1 S_1)) \). Summarizing, we have

\[
source(\theta) \in actors(S_1), sink(\theta) \in actors(S_2) \Rightarrow produced(\theta, (\gamma^1 n_1 S_1)) = consumed(\theta, (\gamma^1 n_2 S_2));
\]

and

\[
source(\theta) \in actors(S_2), sink(\theta) \in actors(S_1) \Rightarrow produced(\theta, (\gamma^1 n_2 S_2)) = consumed(\theta, (\gamma^1 n_1 S_1)). \quad \text{(EQ 1)}
\]

Now we will show that \( S \) does not terminate on \( \theta \) for an arbitrary arc \( \theta \) in \( G \).

Case 1: \( source(\theta) \in actors(S_1), sink(\theta) \in actors(S_2) \). From EQ 1, we know that prior to each invocation of \( (\gamma^1 n_2 S_2) \), at least \( consumed(\theta, (\gamma^1 n_2 S_2)) \) samples reside on \( \theta \). Thus \( S' \) never terminates on \( \theta \) during an invocation of \( (\gamma^1 n_2 S_2) \). Furthermore, since \( S' \) is a single appearance schedule, \( sink(\theta) \) is fired only through invocations of \( (\gamma^1 n_2 S_2) \), and it follows that \( S' \) does not terminate on \( \theta \).

Case 2: \( source(\theta) \in actors(S_2) \) and \( sink(\theta) \in actors(S_1) \). Since \( S \) is an admissible schedule, \( delay(\theta) \geq consumed(\theta, (n_1 S_1)) \), otherwise \( S \) would terminate on \( \theta \) during the first invocation of \( (n_1 S_1) \). Since \( \gamma \geq 1 \), it follows that \( delay(\theta) \geq consumed(\theta, (\gamma^1 n_1 S_1)) \), so \( S' \) does not terminate on \( \theta \) during the first invocation of \( (\gamma^1 n_1 S_1) \). From EQ 1, we know that prior to each subsequent invocation of \( (\gamma^1 n_1 S_1) \), at least \( consumed(\theta, (\gamma^1 n_1 S_1)) \) samples reside on \( \theta \), so \( S' \) does not terminate on \( \theta \) for invocations \( 2, 3, 4, \ldots \) of \( (\gamma^1 n_1 S_1) \). We conclude that \( S' \) does not terminate on \( \theta \).
Factoring Schedule Loops

Case 3: \(\text{source}(\theta), \text{sink}(\theta) \in \text{actors}(S_1)\). Since \(S\) is a valid single appearance schedule, \(S_1\) must be a valid schedule for \(\text{subgraph(actors}(S_1))\). Applying lemma 1 with \(S_0 = S_1\), we see that \(S'\) does not terminate on \(\theta\).

Case 4: \(\text{source}(\theta), \text{sink}(\theta) \in \text{actors}(S_2)\). From lemma 1 with \(S_0 = S_2\), \(S'\) does not terminate on \(\theta\).

Case 5: \(\text{source}(\theta) \notin (\text{actors}(S_1) \cup \text{actors}(S_2))\), or \(\text{sink}(\theta) \notin (\text{actors}(S_1) \cup \text{actors}(S_2))\). Applying lemma 2 with \(S_0 = L\) and \(S_0' = L'\), we see that \(S'\) does not terminate on \(\theta\).

From our conclusions in cases 1-5, \(S'\) does not terminate on any arc in \(\theta\), and it follows that \(S'\) is an admissible schedule. Thus theorem 1 holds for \(k = 2\).

Now suppose that theorem 1 holds whenever \(k \leq k'\), for some \(k' \geq 2\). We will show that this implies the validity of theorem 1 for \(k \leq k' + 1\). For \(k = k' + 1\), \(L = (m (n_1 S_1) (n_2 S_2) \ldots (n_{k+1} S_{k+1})\) and \(L' = (\gamma m (\gamma^1 n_1 S_1) (\gamma^1 n_2 S_2) \ldots (\gamma^1 n_{k+1} S_{k+1}))\). Let \(S_a\) denote the schedule that results from replacing \(L\) with the loop \(L_a\equiv (m (1 (n_1 S_1) (n_2 S_2) \ldots (n_k S_k)) (n_{k+1} S_{k+1})))\). Since \(L_a\) and \(L\) induce the same firing sequence, \(S_a\) induces the same firing sequence as \(S\). Now theorem 1 for \(k = k'\) guarantees that replacing \((1 n_1 S_1) (n_2 S_2) \ldots (n_{k'} S_{k'})\) with \((\gamma (\gamma^1 n_1 S_1) (\gamma^1 n_2 S_2) \ldots (\gamma^1 n_{k'} S_{k'}))\) in \(S_a\) results in a valid schedule \(S_b\).

Observe that \(S_b\) is the schedule \(S\) with \(L\) replaced by \(L_b\equiv (m (\gamma (\gamma^1 n_1 S_1) (\gamma^1 n_2 S_2) \ldots (\gamma^1 n_{k'} S_{k'})) (n_{k+1} S_{k+1}))\). Theorem 1 for \(k = 2\) guarantees that replacing \(L_b\) with \(L_c\equiv (\gamma m (1 (\gamma^1 n_1 S_1) (\gamma^1 n_2 S_2) \ldots (\gamma^1 n_{k'} S_{k'})) (\gamma^1 n_{k+1} S_{k+1}))\) yields another valid schedule \(S_c\). Now \(L_c\) yields the same firing sequence as \(L' = (\gamma m (\gamma^1 n_1 S_1) (\gamma^1 n_2 S_2) \ldots (\gamma^1 n_{k+1} S_{k+1})))\), so replacing \(L_c\) with \(L'\) in \(S_c\) yields an admissible schedule \(S_d\). But, by our construction, \(S_d = S'\), so \(S'\) is a valid schedule for \(G\).

We have shown that theorem 1 holds for \(k = 1\) and \(k = 2\), and we have shown that if the result holds for \(k \leq k'\), then it holds for \(k \leq (k' + 1)\). We conclude that theorem 1 holds for all \(k\).

QED.

Note that factoring is not necessarily a legitimate transformation in admissible schedules that are not single appearance schedules. As a counter-example, consider the SDF graph in figure 7. One can easily verify that the looped schedule A(2 B)(2 CCBB)CC is a valid schedule for this SDF graph. If we factor the two adjacent schedule loops, which have a common iteration count, we obtain the looped schedule A(2 BCCBB)CC. The firing sequence generated by this latter
looped schedule, ABCCCBBCCBBCC, terminates on the input arc of C at invocation C_2. Thus, factoring has converted a valid schedule into a schedule that is not valid.

5 Reduced Single Appearance Schedules

We begin this section with a definition.

**Definition 7:** Suppose that \( \Lambda \) is either a schedule loop or a looped schedule. We say that \( \Lambda \) is **non-coprime** if all iterands of \( \Lambda \) are schedule loops, and there exists an integer \( j > 1 \) that divides all of the iteration counts of the iterands of \( \Lambda \). If \( \Lambda \) is not non-coprime, we say that \( \Lambda \) is **coprime**.

For example, the schedule loops (3 (4 A)(2 B)) and (10 (7 C)) are both non-coprime, while the loops (5 (3 A)(7 B)) and (70 C) are coprime. Similarly, the looped schedules (4 AB) and (6 AB)(3 C) are both non-coprime, while the schedules A(7 B)(7 C) and (2 A)(3 B) are coprime. From our discussion in the previous section, we know that non-coprime schedule loops may result in much higher buffering requirements than their factored counterparts.

**Definition 8:** Given a single appearance schedule \( S \), we say that \( S \) is **fully reduced** if \( S \) is coprime, and every schedule loop contained in \( S \) coprime.

In this section, we show that we can always convert a valid single appearance schedule that is not fully reduced into a valid fully reduced schedule. Thus, we can always avoid the overhead associated with using non-coprime schedule loops over their corresponding factored forms. To prove this, we use another useful fact: that any fully reduced schedule has blocking factor 1. This implies that any schedule that has blocking factor greater than one is not fully reduced. Thus,
if we decide to generate a schedule that has nonunity blocking factor, then we risk introducing higher buffering requirements.

**Theorem 2**: Suppose that $S$ is a valid single appearance schedule for a connected SDF graph $G$. If $S$ is fully reduced then $S$ has blocking factor 1.

*Proof*: First, suppose that not all iterands of $S$ are schedule loops. Then some actor $N$ appears as an iterand. Since $N$ is not enclosed by a loop in $S$, and since $S$ is a single appearance schedule, $\text{inv}(N, S) = 1$, and thus $J(S) = 1$.

Now suppose that all iterands of $S$ are schedule loops, and suppose that $j$ is an arbitrary integer that is greater than that one. Then since $S$ is fully reduced, $j$ does not divide at least one of the iteration counts associated with the iterands of $S$. Define $i_0 = 1$ and let $L_1$ denote one of the iterands of $S$ whose iteration count $i_1$ is not divisible by $j = j / \gcd(j, i_0)$. Again, since $S$ is fully reduced, if all iterands of $L_1$ are schedule loops then there exists an iterand $L_2$ of $L_1$ such that $j / \gcd(j, i_1i_2)$ does not divide the iteration count $i_2$ of $L_2$. Similarly, if all iterands of $L_2$ are schedule loops, there exists an iterand $L_3$ of $L_2$ whose iteration count $i_3$ is not divisible by $j / \gcd(j, i_1i_2i_3)$.

Continuing in this manner, we generate a sequence $L_1, L_2, L_3, \ldots$ such that the iteration count $i_k$ of each $L_k$ is not divisible by $j / \gcd(j, i_1i_2 \ldots i_{k-1})$. Since $G$ is of finite size, we cannot continue this process indefinitely — for some $m \geq 1$, not all iterands of $L_m$ are schedule loops. Thus, there exists an actor $N$ that is an iterand of $L_m$. Since $S$ is a single appearance schedule,

\[
\text{inv}(N, S) = \text{inv}(L_1, S)\text{inv}(L_2, L_1)\text{inv}(L_3, L_2) \ldots \text{inv}(L_m, L_{m-1})\text{inv}(N, L_m) = i_1i_2 \ldots i_m.
\]

By our selection of the $L_k$’s, $j / \gcd(j, i_1i_2 \ldots i_{m-1})$ does not divide $i_m$, and thus $j$ does not divide $\text{inv}(N, S)$.

We have shown that given any integer $j > 1$, $\exists N \in N(G)$ such that $\text{inv}(N, S)$ is not divisible by $j$. It follows that $S$ has blocking factor 1. $QED$.

**Theorem 3**: If an SDF graph $G$ has a valid single appearance schedule, then $G$ has a valid fully reduced schedule.
Proof. We prove theorem 3 by construction. This construction process can easily be automated to yield an efficient algorithm for synthesizing a valid fully reduced schedule from an arbitrary valid single appearance schedule.

Given a looped schedule $\Psi$, we define $\text{non-coprime}(\Psi)$ to be the set of schedule loops in $\Psi$ that are non-coprime. Now suppose that $S$ is a valid single appearance schedule for $G$, and let $\lambda_1 = (m (n_1 \Psi_1) (n_2 \Psi_2) \ldots (n_k \Psi_k))$ be any innermost member of $\text{non-coprime}(S)$ — i.e. $\lambda_1$ is non-coprime, but every loop nested within $\lambda_1$ is coprime. From theorem 1, replacing $\lambda_1$ with $\lambda_1' = (\gamma m (\gamma^{-1} n_1 \Psi_1) (\gamma^{-1} n_2 \Psi_2) \ldots (\gamma^{-1} n_k \Psi_k))$, where $\gamma = \gcd(n_1, n_2, \ldots, n_k)$, yields another valid single appearance schedule $S_1$. Furthermore, $\lambda_1'$ is coprime, and since every loop nested within $\lambda_1$ is coprime, every loop nested within $\lambda_1'$ is coprime as well. Now let $\lambda_2$ be any innermost member of $\text{non-coprime}(S_1)$, and observe that $\lambda_2$ cannot equal $\lambda_1'$. Theorem 1 guarantees a replacement $\lambda_2'$ for $\lambda_2$ that leads to another valid single appearance schedule $S_2$. If we continue this process, it is clear that no replacement loop $\lambda_k'$ ever replaces one of the previous replacement loops $\lambda_1' \lambda_2' \ldots \lambda_{k-1}'$, since these loops, and the loops nested within these loops, are already coprime. Also, no replacement changes the total number of loops in the schedule. It follows that we can continue this process only a finite number of times — eventually, we will arrive at an $S_n$ such that $\text{non-coprime}(S_n)$ is empty.

Now if $S_n$ is a coprime looped schedule, we are done. Otherwise, $S_n$ is of the form $(p_1 Z_1) (p_2 Z_2) \ldots (p_m Z_m)$, where $\gamma = \gcd(p_1, p_2, \ldots, p_m) > 1$. Applying theorem 1 to the schedule $(1 S_n) = (1 (p_1 Z_1) (p_2 Z_2) \ldots (p_m Z_m))$, we have that $(\gamma^{-1} p_1 Z_1) ((\gamma^{-1} p_2 Z_2) \ldots ((\gamma^{-1} p_m Z_m))$ is a valid schedule for $G$. From the definition of a valid schedule, it follows that $S_n' = ((\gamma^{-1} p_1 Z_1) ((\gamma^{-1} p_2 Z_2) \ldots ((\gamma^{-1} p_m Z_m)$ is also a valid schedule, and by our construction of $S_n$ and $S_n'$, $S_n'$ is coprime, and all schedule loops in $S_n'$ are coprime. Thus $S_n'$ is a valid fully reduced schedule for $G$. $QED.$

6 Constructing Single Appearance Schedules

Since valid single appearance schedules implement the full repetition inherent in an SDF graph without requiring subroutines or code duplication, we examine the topological conditions required for such a schedule to exist. First suppose that $G$ is an acyclic SDF graph containing $n$
nodes. Then we can take some root node \( r_1 \) of \( G \) and fire all \( q_G(r_1) \) invocations of \( r_1 \) in succession. After all invocations of \( r_1 \) have fired, we can remove \( r_1 \) from \( G \), pick a root node \( r_2 \) of the new acyclic graph, and schedule its \( q_G(r_2) \) repetitions in succession. Clearly, we can repeat this process until no nodes are left to obtain the single appearance schedule \( (q_G(r_1) \ r_1) \ (q_G(r_2) \ r_2) \ldots (q_G(r_n) \ r_n) \) for \( G \). Thus we see that any acyclic SDF graph has a valid single appearance schedule.

Also, observe that if \( G \) is an arbitrary SDF graph, then we can cluster the subgraphs associated with each nontrivial strongly connected component of \( G \). Clustering a strongly connected component into a single node never results in deadlock since there can be no directed loop containing the clustered node. Since clustering all strongly connected components yields an acyclic graph, it follows from fact 4 and fact 6 that \( G \) has a valid single appearance schedule if and only if each strongly connected component has a valid single appearance schedule.

Observe that we must, in general, analyze a strongly connected component \( R \) as a separate entity, since \( G \) may have a valid single appearance schedule even if there is a node \( N \) in \( R \) for which we cannot fire all \( q_G(N) \) invocations in succession. The key is that \( q_R \) may be less than \( q_G \), so we may be able to generate a single appearance subschedule for \( R \) (e.g., we may be able to schedule \( N \) \( q_R(N) \) times in succession). Since we can schedule \( G \) so that \( R \)'s subschedule appears only once, this will translate to a single appearance schedule for \( G \). For example, in figure 8(a), it can be verified that \( q(A, B, C) = (10, 4, 5) \), but we cannot fire so many invocations of \( A, B \), or \( C \) in succession. However, consider the strongly connected component \( R^* \) consisting of nodes \( A \) and \( B \). Then we obtain \( q_{R^*}(A) = 5 \) and \( q_{R^*}(B) = 2 \), and we immediately see that \( q_{R^*}(B) \) invocations of \( B \) can be scheduled in succession to obtain a subschedule for \( R^* \). The SDF graph that results

![Diagram](https://example.com/diagram.png)

**Fig. 8.** An example of how clustering strongly connected components can improve looping.
from clustering \{A, B\} is shown in figure 8(b). This leads to the valid single appearance schedule
(2 (2 B)(5 A))(5 C).

**Theorem 4:** Suppose that G is a connected SDF graph and suppose that G has a valid single
appearance schedule of some arbitrary blocking factor. Then G has valid single appearance sche-
dules for all blocking factors.

*Proof*: Clearly, any valid schedule \(S\) of unity blocking factor can be converted into a valid sche-
dule of arbitrary blocking factor \(j\) simply by encapsulating \(S\) inside a loop of \(j\) iterations. Thus, it
suffices to show that G has a valid single appearance schedule of unity blocking factor. Now, the-
orem 3 guarantees that G has a valid fully reduced single appearance schedule, and theorem 2
tells us that this schedule has blocking factor 1. *QED.*

**Corollary 1:** Suppose that G is an arbitrary SDF graph that has a valid single appearance sche-
dule. Then G has a valid single appearance schedule for all blocking vectors.

*Proof.* Suppose \(S\) is a valid single appearance schedule for G, let \(R_1, R_2, \ldots, R_k\) denote the maxi-
mal connected subgraphs of G, let \(\mathbf{J}'(R_1, R_2, \ldots, R_k) = (z_1, z_2, \ldots, z_k)\) be an arbitrary blocking vec-
tor for G, and for \(1 \leq i \leq k\), let \(S_i\) denote the restriction of \(S\) to \(R_i\). Then from fact 4 each \(S_i\) is a valid
single appearance schedule for the corresponding \(R_i\). From theorem 4, for \(1 \leq i \leq k\), there exists a
valid single appearance schedule \(S_i'\) of blocking factor \(z_i\) for \(R_i\). Since the \(R_i'\)’s are mutually disjoint
and non-adjacent, it follows that \(S_1' \ S_2' \ldots \ S_k'\) is a valid single appearance schedule of blocking
vector \(\mathbf{J}'\) for G. *QED.*

Our condition for the existence of a valid single appearance schedule involves a form of
precedence independence that we call *subindependence.*

---

1. An alternative proof has been suggested by Sebastian Ritz of the Aachen University of Technology. This proof is based on the
observations that (1) constructing blocking factor 1 schedules for acyclic graphs is easy — we simply use the process described at
the beginning of this section, and (2) if a strongly connected SDF graph G has a single appearance schedule then it has a subin-
dependent subset of nodes (see definition 9), which allows us to decompose G into smaller collections of strongly connected compo-
nents. By hierarchically scheduling the input graph based on observations (1) and (2), we can always construct a blocking factor 1
schedule.
**Definition 9:** Suppose that $G$ is a connected SDF graph. If $Z_1$ and $Z_2$ are disjoint subsets of $N(G)$ we say that “$Z_1$ is subindependent of $Z_2$ in $G$” if for every arc $\alpha$ in $G$ such that $source(\alpha) \in Z_2$ and $sink(\alpha) \in Z_1$, we have $delay(\alpha) \geq q_G(sink(\alpha))c(\alpha)$.

Thus, $Z_1$ is subindependent of $Z_2$ in $G$ if given a minimal (unit blocking factor) valid schedule for $G$, data produced by $Z_2$ is never consumed by $Z_1$ in the same schedule period of $G$ in which it is produced. Thus, at the beginning of each schedule period for $G$, all of the data required by $Z_1$ from $Z_2$ for that schedule period is available at the inputs of $Z_1$. For example, consider the SDF graph in figure 9. Here $q(A, B, C, D) = (2, 1, 2, 2)$, and we see that $\{A, D\}$ is subindependent of $\{B, C\}$ and trivially, $\{B, C, A\}$ is subindependent of $\{D\}$.

We are now ready to establish a recursive condition for the existence of a valid single appearance schedule. Recall that an arbitrary SDF graph has a valid single appearance schedule iff each strongly connected component has a single appearance schedule. Theorem 5 gives necessary and sufficient conditions for a strongly connected SDF graph to have a valid single appearance schedule.

**Theorem 5:** Suppose that $G$ is a nontrivial strongly connected SDF graph. Then $G$ has a valid single appearance schedule if and only if there exists a nonempty proper subset $N_s$ of $N(G)$ such that

1. $N_s$ is subindependent of $(N(G) - N_s)$ in $G$; and
2. $subgraph(N_s, G)$ and $subgraph((N(G) - N_s), G)$ both have valid single appearance schedules.

**Proof:** $\Leftarrow$ Let $S$ and $T$ denote valid single appearance schedules for $Y \equiv subgraph(N_s, G)$ and $Z = subgraph((N(G) - N_s), G)$ respectively; let $y_1, y_2, \ldots, y_k$ denote the maximal connected

![Diagram](image-url)

**Fig. 9.** An example used to illustrate subindependence.
subsets of \( N(Y) \); and let \( z_1, z_2, \ldots, z_l \) denote the maximal connected subsets of \( N(Z) \). From corollary 1, we can assume without loss of generality that for \( 1 \leq i \leq k \), \( J_G(subgraph(y_i)) = q_G(y_i) \), and that for \( 1 \leq i \leq l \), \( J_T(subgraph(z_i)) = q_G(z_i) \). From fact 5, it follows that \( S \) invokes each \( N \in N_s \) \( q_G(N) \) times, and \( T \) invokes each \( N \in (N(G) - N_s) \) \( q_G(N) \) times, and since \( N_s \) is subindependent, it follows that \( (S \circ T) \) is a valid single appearance schedule (of blocking factor 1) for \( G \).

\( \Rightarrow \) Suppose that \( S \) is a valid single appearance schedule for \( G \). From theorem 4, we can assume without loss of generality that \( S \) has blocking factor 1. Then \( S \) can be expressed as \( S_a S_b \), where \( S_a \) and \( S_b \) are nonempty single appearance subchedules of \( S \) that are not encompassed by a loop (if we could represent \( S \) as a single loop \( (n (\ldots) (\ldots)(\ldots)) \) then \( \gcd(\{q_G(N) \mid N \in N(G)\}) \geq n \), so \( S \) is not of unity blocking factor — a contradiction). Since \( S_a S_b \) is a valid single appearance schedule for \( G \), every actor \( N \in actors(S_a) \) is fired \( q_G(N) \) times before any actor outside of \( actors(S_a) \) is invoked. It follows that \( actors(S_a) \) is subindependent of \( actors(S_b) \). Also by fact 4, \( S_a \) is a valid single appearance schedule for \( subgraph(actors(S_a)) \) and \( S_b \) is a valid single appearance schedule for \( subgraph(actors(S_b)) \). QED.

Theorem 5 shows that a strongly connected SDF graph \( G \) has a valid single appearance schedule only if we can find a subindependent partition of the nodes — a partition into two subsets \( Z_1 \) and \( Z_2 \) such that \( Z_1 \) is subindependent of \( Z_2 \). If we can find such \( Z_1 \) and \( Z_2 \), then we can construct a valid single appearance schedule for \( G \) by constructing a valid single appearance schedule for all invocations associated with \( Z_1 \) and then concatenating a valid single appearance schedule for all invocations associated with \( Z_2 \). By repeatedly applying this decomposition, we can construct valid single appearance schedules whenever they exist [2].

The partitioning process on which this decomposition method is based can be performed efficiently. Given a nontrivial strongly connected SDF graph \( G \), we first remove all arcs from \( G \) whose delay is not less than the total number of samples consumed from the arc in a schedule period. If the resulting graph \( G' \) is still strongly connected then no subindependent partition exists. Otherwise, any root strongly connected component of \( G' \) is subindependent. This method of determining a subindependent partition is illustrated in figure 10.
Fig. 10. An example of constructing a subindependent partition. For the strongly connected SDF graph on the left, \( q(A, B, C, D) = (1, 10, 2, 20) \). Thus the delay on the arc directed from D to B (25) exceeds the total number of samples consumed by B in a schedule period (20). We remove this arc to obtain the new graph on the right. Since this graph is not strongly connected, a subindependent partition exists: the root strongly connected component \( \{A, B\} \) is subindependent of the rest of the graph \( \{C, D\} \).

7 Conclusion

We have formally discussed the organization and manipulation of loops in uniprocessor schedules for synchronous dataflow programs. We have introduced two main techniques: (1) constructing single appearance schedules, which permit the efficiency of inlined code without requiring any code duplication across multiple invocations of the same functional block; and (2) factoring loops in a single appearance schedule to reduce the amount of memory required for buffering. Based on our technique for constructing single appearance schedules, we have implemented a scheduling framework for synthesizing optimally compact programs for a large class of applications. This framework defines a way to incorporate other scheduling objectives in a manner that does not conflict with the full code compaction potential offered by subindependent partitions. For example the incorporation of techniques to make buffering more efficient is discussed in [2].

The trade-offs involved in compiling an SDF program are complex. These tradeoffs include the effects of parallelization; code compactness; the amount of memory required for buffering; the amount of data transfers that occur only through machine registers; the number of subroutine calls and their associated overhead; the amount of context-switch overhead, as [20] addresses; and the total loop overhead (initiation and indexing). We have only begun to explore
these tradeoffs — the existing techniques focus on a small subset of the full range of considerations. A more global objective of the formal techniques for working with looped schedules that this paper presents is to facilitate the exploration of tradeoffs involved in compiling SDF programs. This is demonstrated to some extent by our scheduling framework [2]; there is much more room for work in this area.

Acknowledgment

The authors are grateful to Sebastian Ritz of the Aachen University of Technology, and to the anonymous reviewers for their helpful comments.

Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>actors(S)</td>
<td>The set of actors that appear in the looped schedule S.</td>
</tr>
<tr>
<td>admissible schedule</td>
<td>A schedule that does not deadlock.</td>
</tr>
<tr>
<td>c(α)</td>
<td>The number of samples consumed from SDF arc α by one invocation of sink(α).</td>
</tr>
<tr>
<td>delay(α)</td>
<td>The number of delays on SDF arc α.</td>
</tr>
<tr>
<td>gcd</td>
<td>Greatest common divisor.</td>
</tr>
<tr>
<td>inv(X, S)</td>
<td>The number of times that the looped schedule S invokes actor or subschedule X.</td>
</tr>
<tr>
<td>iterand</td>
<td>Given a schedule loop ( n \Psi_1 \Psi_2 \ldots \Psi_k ), we refer to each ( \Psi_i ) as an iterand of the schedule loop. Given a looped schedule ( S = V_1V_2 \ldots V_k ), where each ( V_i ) is either an actor or a schedule loop, we refer to each ( V_i ) as an iterand of ( S ).</td>
</tr>
<tr>
<td>iteration count</td>
<td>Given a schedule loop ( n \Psi_1 \Psi_2 \ldots \Psi_k ), we refer to ( n ) as the iteration count.</td>
</tr>
<tr>
<td>( J(S) )</td>
<td>Given a periodic schedule ( S ) for a connected SDF graph, ( J(S) ) denotes the blocking factor of ( S ). Every periodic schedule invokes each actor ( N ) some multiple of ( q_G(N) ) times. This multiple is the blocking factor.</td>
</tr>
<tr>
<td>( J_S )</td>
<td>Given a periodic schedule ( S ) for an SDF graph ( G ), ( J_S ) denotes the blocking vector of ( S ). ( J_S ) is indexed by the maximal connected subgraphs of ( G ). For each maximal connected subgraph ( C ), ( J_S(C) = J(restriction(S, C)) ).</td>
</tr>
<tr>
<td>looped schedule</td>
<td>A schedule that has zero or more parenthesized terms of the form ( n \Psi_1 \Psi_2 \ldots \Psi_k ), where ( n ) is a nonnegative integer, and each ( \Psi_i ) represents either an SDF node or another parenthesized term. ( (n \Psi_1 \Psi_2 \ldots \Psi_k) ) represents the successive repetition ( n ) times of the firing sequence ( \Psi_1 \Psi_2 \ldots \Psi_k ).</td>
</tr>
</tbody>
</table>
max_connected(G)  The set of maximal connected subgraphs of the SDF graph G.

N(G)  The set of nodes in the SDF graph G.

P(α, i, S)  The number of invocations of the source actor of SDF arc α that precede the i-th invocation of sink(α) in schedule S.

p(α)  The number of samples produced onto SDF arc α by one invocation of source(α).

periodic schedule  A schedule that invokes each actor at least once and produces no net change in the number of samples buffered on any arc.

qG  The repetitions vector qG of the connected SDF graph G is a vector that is indexed by the nodes in G. qG gives the minimum number of times that each node must be invoked in a periodic schedule for G.

restriction  Given a looped schedule S and a set of actors Z, the restriction of S to Z, denoted restriction(S, Z), is the looped schedule obtained by removing from S all actors that are not contained in Z, and removing all empty schedule loops that result. Also, given an SDF subgraph G, restriction(S, G) = restriction(S, N(G)).

capture schedule  A looped schedule in which no actor appears more than once.

sink(α)  The actor at the sink of SDF arc α.

source(α)  The actor at the source of SDF arc α.

subgraph  A subgraph of an SDF graph G is the graph formed by any subset Z of nodes in G together with all arcs α in G for which source(α), sink(α) ∈ Z. We denote the subgraph corresponding to the subset of nodes Z by subgraph(Z, G), or simply by subgraph(Z) if G is understood from context.

subschedules  A subschedule of a looped schedule S is a sequence of successive iterands of S or a sequence of successive iterands of a schedule loop contained in S.

termination of a schedule  If S is not an admissible schedule then some invocation f in S is not fireable immediately after all of its antecedents in the schedule have fired. Thus f does not have sufficient data on at least one of its input arcs. If α is one such input arc, we say that S terminates on α at f.

valid schedule  A schedule that is both periodic and admissible.

References


References


