Improving Scalability of Model-Checking for Minimizing Buffer Requirements of Synchronous Dataflow Graphs

Abstract— Synchronous Dataflow (SDF) is a well-known model of computation for dataflow-oriented applications such as signal processing and multimedia. It is important to minimize the buffer size requirements of the source code generated from a given SDF model, since memory space is often a scarce resource for these applications due to cost or power consumption constraints. Some authors have proposed using model-checking for minimizing buffer size requirements. However, scalability is a key limiting factor in using model-checking due to state space explosion. In this paper, we present several techniques for improving scalability of model-checking by exploiting problem-specific properties of SDF models.

I. Introduction and Related Work

Synchronous Dataflow (SDF) is a widely-used model of computation for signal processing applications that allows for powerful static analysis and synthesis techniques. Since signal processing and multimedia applications are often implemented on resource constrained embedded systems, it is important to minimize their memory size. The total memory size requirement of an application consists of two parts: *code size* and *data buffer size*. Since effective techniques have been developed to minimize the code size for any feasible schedule with acceptable runtime overhead [1], [2], we do not consider the code size in this paper, but focus on finding the feasible schedule with minimum the data buffer size.

The data buffer minimization problem of SDF is known to be NP-complete [3]. Some authors have used model-checking to obtain the optimal solution [4], [2] by exploring the entire state space. Geilen *et al* [4] used the SPIN model-checker to find the minimum buffer size requirement of a given SDF graph. Gu *et al* [2] used the symbolic model-checker NuSMV [5], which uses Ordered Binary Decision Diagrams (OBDD) to encode the state space, to solve the buffer size minimization problem for SDF as well as several variants of it, including Cyclo-Static Dataflow and Multidimensional SDF. In addition, [2] presented techniques for synthesizing efficient *dynamic single-appearance* code, where each actor firing appears only once, while minimizing runtime overhead due to conditional branches. One key limitation of model-checking is its lack of scalability due to state space explosion. In this paper, we present several techniques for reducing the state space and improving scalability of model-checking when applied to the SDF buffer minimization problem. We focus on the NuSMV model-checker, but our techniques are at the application-level, and independent of the specific model-checker used.

This paper is structured as follows: we present the basic technique of model-checking for buffer size minimization in Section II; techniques for obtaining tighter variable upper bounds in Section III; techniques for graph decomposition in Section IV; performance evaluation in Section V; conclusions in Section VI.

II. Model-checking for Buffer Size Minimization of SDF Graphs

A SDF graph is a directed graph $G = (V, E)$, where V is the set of nodes representing actors and E is the set of edges. An edge $e \in E$ has a source actor $src(e)$ that produces $p(e)$ tokens on e at each invocation (referred to as an actor *firing* in the rest of the paper), and a sink actor $snk(e)$ that consumes $c(e)$ tokens on e at each firing. $d(e)$ denotes the initial number of tokens on e, also called the *initial delay*. The argument (e) in these definitions can be omitted when there is no ambiguity.

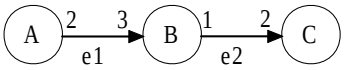

Fig. 1. SDF graph Example 1.

A *feasible schedule* of a SDF graph is a finite actor firing sequence, which when executed by the SDF graph, returns the edge buffer token states to the initial state. The feasible schedule can be repeatedly executed at run time without any deadlock or buffer overflow. For example, the balance equation of the SDF graph in Fig. 1 is:

$$\begin{cases} r_A * 2 = r_B * 3 \\ r_B * 1 = r_C * 2 \end{cases}$$

where r_A is the number of firings of actor `A`. Solving the balance equation yields the repetition vector $(r_A, r_B, r_C) = (3, 2, 1)$, which means that any minimum-length feasible schedule must consist of 6 actor firings in sequence, including 3 firings of actor `A`, 2 firings of actor `B` and 1 firing of actor `C`. Different feasible schedules may have different data buffer size requirements. In this paper, we assume that each edge has its own dedicated buffer space without any buffer sharing. The buffer size requirements of `e1` and `e2` for the feasible schedule `AAABBC` are 6 and 2, respectively, with total buffer size requirement of 8; the buffer size requirements of `e1` and `e2` for the feasible schedule `AABABC` are 4 and 2, respectively, with total buffer size requirement of 6. Our objective in this paper is to find the minimum total buffer size requirement for a SDF graph to have at least one feasible schedule. Any buffer size less than that will cause the SDF graph to run into a deadlock for any possible schedule.

In order to use model-checking to find the minimum-buffer size schedule of a SDF graph, we transform the SDF graph into a Finite State Machine (FSM) encoding its execution semantics, e.g., the FSM in Fig. 2 corresponds to the SDF graph in Fig. 1. Each FSM transition models an actor firing and its effects on the input and output buffers

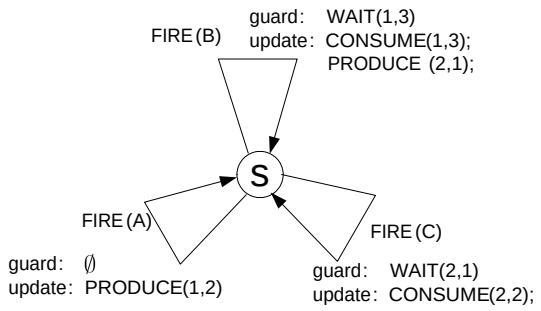

Fig. 2. FSM that encodes the execution semantics of the SDF graph in Figure 1.

of the actor. The macros `WAIT(c,n)`, `PRODUCE(c,n)` and `CONSUME(c,n)` in Fig. 2 are defined as follows:

```
#define WAIT(c,n) ch[c]>=n
#define PRODUCE(c,n) ch[c] = ch[c] + n; UPDATE(c)
#define CONSUME(c,n) ch[c] = ch[c] - n
#define UPDATE(c) if (ch[c]>sz[c]) then sz[c]=ch[c]
```

where `ch[c]` denotes the current number of tokens on edge `c`, and `sz[c]` denotes the buffer size requirement of edge `c`, i.e., the maximum number of tokens on edge `c` throughout the entire schedule. The guard `WAIT(c,n)` encodes the condition that an actor can be invoked only if there are enough tokens on its input edge(s). The actions `PRODUCE(c,n)` (`CONSUME(c,n)`) encodes the semantics that each actor firing produces (consumes) a certain number of tokens on its output (input) edges. Our objective is to find a feasible schedule with minimum `SUM=sz[c1]+sz[c2]+...`, i.e., the sum of all edge buffer size requirements. To find the schedule with minimum buffer size requirement, we formulate and check a series of CTL formulas. There are two different ways of formulating the CTL formulas:

1) Each CTL formula has the form of `SPEC AF SUM>=BOUND`, which means "All possible feasible schedules will eventually satisfy `SUM>=BOUND`". If the formula is proven true for a certain value of `BOUND`, but false for `BOUND+1`, then we can conclude that the total buffer size requirement is `BOUND`, and the model-checker returns a counter-example trace as the schedule with buffer size requirement `BOUND` when proving the property `SPEC AF SUM>=BOUND+1` false.
2) Check for absence of deadlocks under the invariant constraint that the total buffer size requirement does not exceed a given bound. Given the invariant `INVAR SUM<=BOUND`, if the property `SPEC AF EX 1` is proven true for a certain value of `BOUND`, but false for a certain value of `BOUND-1`, then we can conclude that the the total buffer size requirement is `BOUND`. The CTL formula `AF EX 1` means that "for all paths from the initial state, the system can always make a transition into the next state". If this is proven false, then there is a deadlock.

In both approaches, binary search can be used to narrow down the range and eventually find the correct `BOUND` value. Our experience shows that the 2nd approach exhibits much better scalability than the 1st approach, perhaps due to more efficient implementation of deadlock-checking algorithm than general CTL properties in NuSMV. However, the 2nd approach does not provide a feasible buffer size requirement along with a feasible schedule like in the first approach when `SPEC AF SUM>=BOUND` is proven false. Since our objective in this paper is to obtain the buffer size requirement only, this is not a major shortcoming.

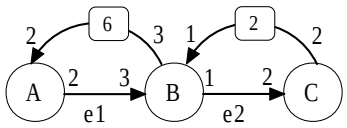

Fig. 3. The SDF graph Example 1 in Fig. 1 with back edges to encode buffer size upper bound on each edge.

The *Buffer Size Lower Bound* (BSLB) of a directed edge e connecting from actor `A` to actor `B` is the minimum buffer size requirement of edge e in any feasible schedule. It can be obtained from the following equation [3]:

$$BSLB(e) = \begin{cases} p + c - g + d \bmod g & \text{if } 0 \le d \le p + c - g \\ d & \text{if } d > p + c - g \end{cases} \quad (1)$$

where $g = gcd(p, c)$ (gcd stands for *greatest common divisor*); each of p, c, d has an argument (e) that is omitted for text formatting reasons.

A safe but pessimistic value for the *Buffer Size Upper Bound* (BSUB) of an edge can be obtained from the *repetition vector*, which is in turn obtained from solving the balance equation of the SDF graph. For example, the SDF graph in Fig. 1 has the repetition vector (3,2,1), hence $BSUB(e1) = 3 * 2 = 6, BSUB(e2) = 2 * 1 = 2$. However, BSUB values determined this way are typically very pessimistic and grossly overestimate the actual buffer size needed. It is desirable to obtain correct and tight BSUB value for each edge for a number of reasons:

- The starting point of the binary search for the correct total buffer size requirement (`BOUND`) is the maximum and minimum total buffer size requirements obtained from summing up the *Buffer Size Upper Bound* (BSUB) and *Buffer Size Lower Bound* (BSLB) values of all edges. Therefore, tight BSUB values helps reduce the number of iterations in the binary search by setting a good starting point.
- BSUB on each edge is reflected as additional back-edges to the original SDF graph to encode the buffer size constraints, as shown in Fig. 3. Therefore, tight BSUB values help reduce the system state space.
- NuSMV is a symbolic model-checker, which uses Ordered Binary Decision Diagrams (OBDD) to encode the state space. Integer variables are encoded in their binary representation, i.e., i bits are needed to encode any integer $0 \le n \le 2^i$. As a result, the state space is very sensitive to variable sizes, and it is desirable to set the maximum sizes of variables to be as small as possible. In our case, the relevant variables are those that encode the number of tokens on each edge. Therefore, tight BSUB values help

further reduce the system state space for the symbolic model-checker NuSMV.

- Incorrect BSUB values can cause the model-checker to arrive at an incorrect buffer size requirement. Gu *et al* [2] first uses BSUB values obtained from solving the balance equation in the NuSMV model. If the state space is too large for NuSMV to handle, then the BSUB values are set to the BSLB values obtained from Equation (1), which is not safe and may cause a deadlock. If a feasible schedule is found, then finish. If a deadlock occurs, then the BSUB values are gradually increased until no deadlock occurs.

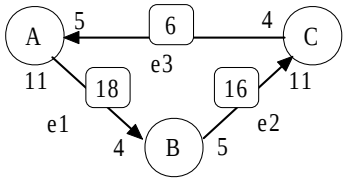

Fig. 4. SDF Graph Example 2.

We use the SDF graph in Fig. 4 as an example to show that the technique in [2] is not guaranteed to find the minimum buffer size requirement. The BSLB values of `e1`, `e2` and `e3` obtained from Equation (1) are $(18, 16, 8)$, respectively. Setting the BSUB values equal to the BSLB values in the NuSMV model results in a deadlock, but it is unclear as to BSUB of which edge should be increased. If we choose to always increase BSUB of edge $e1$, then we will obtain the minimum buffer size requirement of 56 when BSUB of $e1$ is increased to 32. However, if we adopt the safe but pessimistic BSUB values obtained from solving the balance equation $(62, 71, 26)$, then we can obtain the true minimum buffer size requirement of 49.

In the next section, we discuss techniques for setting correct yet tight BSUB values.

III. Tight Edge Buffer Size Upper Bounds

Theorem 1. *For a given SDF graph $G = (V, E)$, let s denote any feasible schedule s with buffer size requirement $R(s)$; let $R(s, e_i)$ denote the buffer requirement of edge e_i for schedule s; let s_{opt} denote an optimal schedule with minimum buffer size requirement $R(s_{opt})$. Then for each edge $e_i \in E$, we have:*

$$R(s_{opt}, e_i) \leq R(s) - \sum_{e_j \in E - e_i} BSLB(e_j) \tag{2}$$

where $BSLB(e_j)$ is the BSLB of edge e_j obtained from Equation (1).

Proof: Since values of $BSLB$ are valid lower bounds for any feasible schedule, we have:

$$R(s_{opt}, e_i) + \sum_{e_j \in E - e_i} BSLB(e_j) \leq R(s_{opt}) = \sum_{e_j \in E} R(s_{opt}, e_j) \tag{3}$$

We also know that $R(s_{opt}) \leq R(s)$, so Equation (2) follows. ■

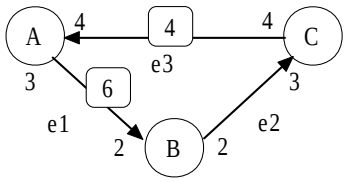

Fig. 5. SDF graph Example 3.

Theorem 2 allows us to set $BSUB(e_i)$ to be equal to $R(S) - \sum_{e_j \in E - e_i} BSLB(e_j)$ without losing optimality. We use the well-known heuristic algorithm in [6] for finding a feasible schedule s with close-to-minimum buffer size requirement $R(s)$. This algorithm is optimal for acyclic, delayless SDF graphs, but not for general SDF graphs with cycles and/or initial delays. It works by firing each actor in a demand-driven manner, and when multiple choices of actor firings exist, always choosing the one that results in the smallest increase in the number of tokens. Our experience shows that this approach gives us much tighter BSUB values than the BSUB values obtained from solving the balance equation. The maximum and minimum total buffer size requirements are set to be $R(s)$ and sum of the BSLB values from Equation 1, respectively. Subsequently, we use binary search to narrow down the range of the buffer size requirement while keeping all the BSUB values fixed in the NuSMV model.

Next, we present another technique for setting the BSUB of so-called *Heavy Edges*. At first we introduce the definition of *Forward Heavy Edge* and *Backward Heavy Edge*.

Definition 1 (Forward Heavy Edge)**.** *An edge e_f is a* Forward Heavy Edge (FHE) *if it is the only input edge to its sink actor $snk(e_f)$, and*

$$c(e_f) > \sum_{src(e_j) = snk(e_f)} p(e_j) \tag{4}$$

Definition 2 (Backward Heavy Edge)**.** *An edge e_b is a* Backward Heavy Edge (BHE) *if it is the only output edge from its source actor $src(e_b)$, and*

$$p(e_b) > \sum_{snk(e_j) = src(e_b)} c(e_j) \tag{5}$$

If the sink actor of a FHE is fired, then the number of produced tokens is smaller than the number of consumed tokens, hence we should try to avoid accumulating tokens on a FHE by transferring tokens on a FHE forward to its downstream edges. On the other hand, if the source actor of a BHE is fired, then the number of produced tokens is greater than the number of consumed tokens, hence we should try to avoid accumulating tokens on a BHE by transferring tokens on a BHE backward to its upstream edges. However, one should not simply fire the sink actor of a FHE as soon as possible (or the source actor of a BHE as late as possible), which may lead to higher buffer size requirement. For example, the edge `e3` is both a FHE and a BHE, if we simply fire actor `A` as soon as possible, and fire actor `C` as late as possible, we may get a feasible schedule `AABBBCC` with buffer size requirement of 20, while the optimal buffer requirement is 16.

We can get upper bounds for the heavy edges as shown in the following theorem:

Theorem 2. *In any optimal feasible schedule s_{opt} of a SDF graph, any FHE e_f must satisfy*

$$R(s_{opt}, e_f) < \max(p(e_f) + c(e_f), d(e_f)) + c(e_f) \quad (6)$$

and any BHE e_b must satisfy

$$R(s_{opt}, e_b) < \max(p(e_b) + c(e_b), d(e_b)) + p(e_b) \quad (7)$$

We will prove Theorem 2 by Lemma 1 and Lemma 2 shown as follows:

Lemma 1. *For a SDF graph G, if there exists a feasible schedule s, where an FHE e_f satisfying*

$$R(s, e_f) \geq \max(p(e_f) + c(e_f), d(e_f)) + c(e_f) \quad (8)$$

then there must exist a feasible schedule s' satisfying:

$$R(s', e_f) = R(s, e_f) - c(e_f) \quad \wedge \quad R(s', G) < R(s, G) \quad (9)$$

Similarly, we can prove the following lemma for BHE:

Lemma 2. *For a SDF graph G, if there exists a feasible schedule s, in which a BHE e has buffer requirement*

$$R(s, e_b) \geq \max(p(e_b) + c(e_b), d(e_b)) + p(e_b) \quad (10)$$

then there must exist a feasible schedule s' satisfying:

$$R(s', e_b) = R(s, e_b) - p(e_b) \quad \wedge \quad R(s', G) < R(s, G) \quad (11)$$

Theorem 2 can be proved by iteratively applying Lemma 1 and Lemma 2.

We can further bound the buffer requirement of a special kind of heavy edges.

Definition 3 (Regular Heavy Edge)**.** *An edge e_r is called a Regular Heavy Edge (RHE) if*

1) *e_r is an FHE, $p(e) = k_1 * c(e)$, $d(e) = k_2 * c(e)$, or*
2) *e_r is a BHE, $c(e) = k_1 * p(e)$, $d(e) = k_2 * p(e)$*

where k_1 is any positive integer and k_2 is any non-negative integer.

Theorem 3. *The buffer requirement of a RHE e_r must be $BSLB(e_r)$ in any optimal feasible schedule.*

The proof of Theorem 3 is similar to that of Theorem 2.

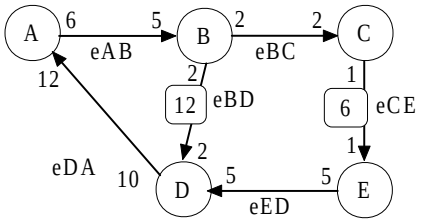

Fig. 6. SDF graph Example 4.

For SDF graph Example 4 in Fig. 6, the BSUB values obtained from the balance equation are:

```
BSUB(eAB)=30, BSUB(eBC)=12, BSUB(eCE)=6
BSUB(eBD)=12, BSUB(eED)=30, BSUB(eDA)=60
```

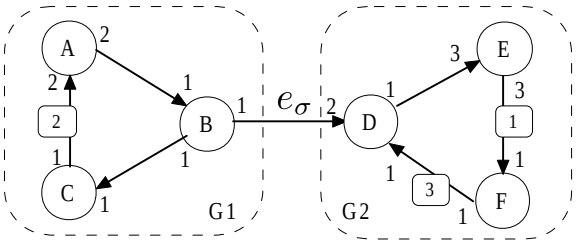

Fig. 7. Example to illustrate Lemma 3.

Since eAB is an FHE; eBC is a RHE; eED is a RHE; eDA is both an FHE and a BHE, we can obtain tighter upper bounds as follows:

```
BSUB(eAB)=16, BSUB(eBC)=2, BSUB(eCE)=6
BSUB(eBD)=12, BSUB(eED)=5, BSUB(eDA)=32
```

IV. Graph Decomposition

In this section, we will show that we can decompose the SDF graph into several subgraphs connected by bridges, analyze each subgraph separately, and then obtain the buffer size requirement of the whole SDF graph from that of each subgraph.

We first present some basic definitions. Suppose $s(G)$ is a feasible schedule of SDF graph $G = (V, E)$, and $G_i = (V_i, E_i)$ is a subgraph of G. The *projection* of $s(G)$ on G_i is the schedule sequence denoted by $s(G_i)$, in which the firings of actors that do not belong to G_i are removed, and the remaining firing sequence is kept with the same order as in $s(G)$. A *bridge* (also known as a *cut-edge* in graph theory) e is special edge s.t. if e is deleted, the SDF graph will become two separated subgraphs.

Next, we introduce a schedule composition algorithm called **CA**. For any SDF graph consisting of subgraphs connected with a bridge, (**CA**) can be used to compose the valid schedules of its two subgraphs to obtain a valid schedule of the overall SDF graph. Suppose the SDF graph $G = (V, E)$ consists of two subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ connected with a bridge e_b, as shown in Fig. 7.

CA consists of the following steps to obtain a schedule for G:

1) Fire each actor in s_1 in sequence repeatedly. If actor $src(e_\sigma)$ is fired, then goto (2), otherwise
2) If there are enough tokens on e_σ for $snk(e_\sigma)$ to fire, then fire $snk(e_\sigma)$ and goto (3); otherwise go to (1).
3) Fire each actor in $s2$ in sequence repeatedly. If there are not enough tokens on e_σ for $snk(e_\sigma)$ to fire, then goto (1); otherwise repeat (3).
4) Whenever $src(e_\sigma)$ or $snk(e_\sigma)$ is fired, check to see if the number of tokens on e_σ has returned to its initial value, all firings of $src(e_\sigma)$ in the current iteration of $s1$ have been finished, and all firings of $snk(e_\sigma)$ of the current iteration of $s2$ have been finished. If all three conditions are satisfied, then the remaining actor firings in the current iteration of $s1$ and $s2$ are performed, and the algorithm terminates afterwards.

As an example, in Fig. 7, s_1=ABCBC is a feasible schedule for G_1 and s_2=DDFDEFF is a feasible schedule for G_2. Now we apply **CA**:

1) Fire A and B in schedule s_1.
2) Number of tokens on e_σ is 1, which is not enough for D to fire. Continue to fire C and B in schedule s_1.
3) Number of tokens on e_σ is now 2. Fire D.
4) Number of tokens on e_σ has returned to its initial state of 0, but there are still two firings of D remaining in s_2. So we continue to fire the next actor in s_2, F.
5) Since the next actor firing in s_2, D, is not enabled, we return to schedule s_1.
6) The process continues until the final schedule of ABCB D CABCB DF CABCB DEFF C is obtained.

CA always tries to fire the sink actor of the bridge as soon as possible, in order to minimize the buffer requirement of the bridge. The benefit of **CA** is that, for any SDF graph consisting of two subgraphs connected by a bridge, we can get its optimal feasible schedule with minimum buffer size by composing the optimal feasible schedules of its two subgraphs with **CA**.

Lemma 3. *Suppose a SDF graph $G = (V, E)$ consists of two subgraphs G_1 and G_2 connected by a bridge e_σ. We have:*

$$R(s_{opt}, G) = R(s^1_{opt}, G_1) + R(s^2_{opt}, G_2) + BSLB(e_\sigma) \quad (12)$$

where s_{opt}, s^1_{opt} and s^2_{opt} is an optimal schedule of G, G_1 and G_2 respectively.

Proof: We can compose s^1_{opt} and s^2_{opt} with **CA** to obtain a valid schedule s for G. s^1_{opt} and s^2_{opt} is the projection of s on G_1 and G_2 respectively. We observe that if we schedule G by s, the buffer requirement on G_1, G_2 and e_σ is $R(s^1_{opt}, G_1)$, $R(s^2_{opt}, G_2)$ and $BSLB(e_\sigma)$ respectively. At the same time, $R(s^1_{opt}, G_1)$, $R(s^2_{opt}, G_2)$ and $BSLB(e_\sigma)$ is the lower bound of the buffer requirement of the corresponding part respectively, which means the buffer requirement of G can not be smaller than $R(s^1_{opt}, G_1) + R(s^2_{opt}, G_2) + BSLB(e_\sigma)$. So the minimal buffer requirement of G is $R(s^1_{opt}, G_1) + R(s^2_{opt}, G_2) + BSLB(e_\sigma)$. ■

For the example in Fig. 7, ABCBC is an optimal feasible schedule of G_1 and $BSLB(G_1) = 2 + 2 + 1 = 5$. DDFDEFF is an optimal feasible schedule of G_2 and $BSLB(G2) = 3 + 3 + 3 = 9$. $BSLB(e_{BD}) = 2$. So by Lemma 1, we have $BSLB(G) = 5 + 9 + 2 = 16$

We can generalize the conclusion of Lemma 3 to SDF graphs with more complex topology.

Theorem 4. *If $G = (V, E)$ is a SDF graph consisting of the set C of the minimal subgraphs connected by the set of bridges B.*

$$R(s_{opt}, G) = \sum_{G_i \in C} R(s^i_{opt}, G_i) + \sum_{e_i \in B} BSLB(e_i) \quad (13)$$

in which s_{opt} is an optimal schedule of G, and s^i_{opt} is an optimal schedule of G_i.

Theorem 4 can be easily proved by Lemma 3 recursively.

To use Theorem 4 to improve the efficiency of model-checking for the buffer requirement optimization of SDF G, we should find all subgraphs and bridges of the graph G, and compute the minimal buffer requirement of each subgraph separately, then compute the buffer requirement of S with Theorem 4. If a subgraph is cyclic, or it is acyclic with non-zero delay tokens, then we can use model-checking to derive its minimal buffer requirement and schedule. For acyclic and delayless subgraphs, we can use the algorithm in [7] to derive them analytically. Finding bridges of a graph can be done by the depth-first search with a complexity of $O(m+n)$ [8], where m is the number of actors and n is the number of edges.

V. Performance Evaluation

We used the software tool SDF^3 [9] to generate random SDF graphs. Since there exists fast and optimal algorithms [6] for deriving minimum buffer size requirements for acyclic and delayless SDF graphs, which makes it unnecessary to apply model-checking for these types of graphs, we ensured that any generated SDF graph contains cycles or initial delay tokens, or both. In our experiments, the minimum, maximum and average total input-output degree of each actor in the SDF graph are 1, 3 and 2, respectively. The experiments are run on a Linux PC with an Intel dual-core 2.83GHz 64-bit processor and 2GB of main memory. We use a utility program *Memtime* from the UPPAAL group to measure the running time and peak memory usage. Table I shows the performance evaluation results. We only record the running time and peak memory usage of the final step of the binary search process. If the optimal buffer size requirement is BOUND, then for the optimized approach, then we record the peak memory size and running time of two model-checking sessions given the invariant INVAR SUM<=BOUND, which should not lead to a deadlock, and the invariant INVAR SUM<=BOUND−1, which should lead to a deadlock; for the original approach in [2], we record the peak memory size and running time of two model-checking sessions for the CTL properties SUM>=BOUND, which should be proven true, and SUM>=BOUND+1, which should be proven false. For a SDF graph that is decomposed into several subgraphs using the approach in Section IV, we take the maximum model-checker peak memory size and running time among all its decomposed subgraphs as the memory size and running time for the overall SDF graph.

We set a timeout limit of 2 hours. If a model-checking session did not finish within 2 hours, then we denote it with "–" in the table. As shown in Table I, the optimization techniques presented in this paper can result in significant reduction in both the memory size and running time of model-checking.

VI. Conclusions

In this paper, we have presented several optimization techniques for reducing the state space and improving efficiency/scalability of model-checking for finding the minimum buffer size requirement of SDF graphs. Performance evaluation demonstrates the effectiveness of these techniques in reducing the peak memory size and running time of model-checking compared to the original approach without these optimizations.

TABLE I
PERFORMANCE EVALUATION.

No. Actors	4	6	8	10	12	14	16	18	20	22
Buff. Size.	32	36	56	64	188	168	222	430	324	471
Peak memory of NuSMV with the original approach in [2]										
BOUND (MB)	19.6	96.6	22.4	62.1	156.7	–	–	–	–	–
BOUND+1 (MB)	19.6	86.9	22.5	62.9	223.6	–	–	–	–	–
Running Time of NuSMV with the original approach in [2]										
BOUND (s)	0.6	11.2	1.2	15.2	327.3	–	–	–	–	–
BOUND+1 (s)	0.1	21.2	1.2	26.5	562.5	–	–	–	–	–
Peak memory of NuSMV with the optimized approach in this paper										
BOUND (MB)	2.5	26.1	13.6	25.8	36.6	17.2	21.3	12.6	16.7	56.8
BOUND-1 (MB)	2.5	2.5	2.5	2.5	26.7	2.5	2.5	2.5	2.5	22.4
Running Time of NuSMV with the optimized approach in this paper										
BOUND (s)	0.1	0.1	0.2	0.1	0.6	0.3	0.4	0.2	0.3	2.2
BOUND-1 (s)	0.1	0.1	0.1	0.1	0.3	0.1	0.1	0.1	0.1	0.4

REFERENCES

[1] H. Oh, N. Dutt, and S. Ha, "Memory optimal single appearance schedule with dynamic loop count for synchronous dataflow graphs," in *ASP-DAC*, F. Hirose, Ed. IEEE, 2006, pp. 497–502.

[2] Z. Gu, M. Yuan, N. Guan, M. Lv, X. He, Q. Deng, , and G. Yu, "Static scheduling and software synthesis for dataflow graphs with symbolic model-checking," in *Proc. IEEE Real-Time Systems Symposium (RTSS)*, 2007.

[3] P. K. Murthy and S. S. Bhattacharyya, *Memory Management for Synthesis of DSP Software*. CRC Press, 2006.

[4] M. Geilen, T. Basten, and S. Stuijk, "Minimising buffer requirements of synchronous dataflow graphs with model checking." in *DAC*, 2005, pp. 819–824.

[5] A. Cimatti and et al, "NuSMV 2: An OpenSource Tool for Symbolic Model Checking," in *International Conference on Computer-Aided Verification (CAV)*, 2002.

[6] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Software Synthesis from Dataflow Graphs*. Kluwer Academic Publishers, 1996.

[7] M. Cubric and P. Panangaden, "Minimal memory schedules for dataflow networks." in *CONCUR*, ser. Lecture Notes in Computer Science, E. Best, Ed., vol. 715. Springer, 1993, pp. 368–383.

[8] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM Journal of Computer*, 1972.

[9] S. Stuijk, M. Geilen, and T. Basten, "SDF^3: SDF For Free," in *Application of Concurrency to System Design, 6th International Conference, ACSD 2006, Proceedings*. IEEE Computer Society Press, Los Alamitos, CA, USA, June 2006, pp. 276–278. [Online]. Available: http://www.es.ele.tue.nl/sdf3

APPENDIX

Proof of Lemma 1.

Proof: We first consider the case of $p(e_f) + c(e_f) \geq d(e_f)$.

We visit each actor firing in s in order starting from its beginning, to find the step at which the buffer requirement becomes larger than $R(s, e_f) - c(e_f)$ for the first time. Suppose it is at the step j, and we know it must be a firing of node $src(e_f)$. Then we look for the the first firing of $snk(e_f)$ after step j, and suppose it is at step k. Note that there is no firing of $src(e_f)$ during step $j+1$ ... k, otherwise the buffer size of e_f will exceed $R(s, e_f)$.

We move the firing of $snk(e_f)$ at step k to j, and move every step between j and $k-1$ in s to its next step respectively. Now we will show that this moving action won't lead to negative token number on e_f:

Since $p(e_f) + c(e_f) \geq d(e_f)$, by Equation 8 we have $R(s, e_f) - c(e_f) > d(e_f)$, so the initial buffer size is smaller than $R(s, e_f) - c(e_f)$. Suppose the buffer size at step j (after the firing of $src(e)$ at step j) is x, then the buffer size at step $j-1$ is $x - p(e_f)$. After we move the firing of $snk(e_f)$ to step j, the buffer size at step j is $x - p(e_f) - c(e_f)$, and since $x > R(s, e_f) - c(e_f)$ and $R(s, e_f) > p(e_f) + 2 * c(e_f)$, we know $x - p(e_f) - c(e_f) > 0$, at step j the token number on e_f is positive. Since after the moving actions during step $j+1...k$, there is no firing of $snk(e)$, and after step k the token number on e_f is the same as s, it will not lead to a negative number of tokens on e_f.

After the moving actions, for the schedule sequence between step j and step k, the buffer requirement of e_f becomes smaller than $R(s, e_f) - c(e_f)$. At the same time, the buffer requirement of each edge e_i whose source node is $snk(e_f)$ is increased by at most $p(e_i)$.

Then from the step $k+1$ (there is no other firing of $src(e_f)$ before $k+1$, as mentioned above), we start to look for the next firing of $src(e_f)$ at which the buffer requirement exceeds $R(s, e_f) - c(e_f)$, and repeat the process above, until the end of schedule sequence is reached.

Now, we have transformed the schedule s to another feasible schedule s', in which the buffer size requirement of e_f is $R(s, e_f) - c(e_f)$; the buffer size requirement of any e_i whose source ode is the sink node of e_f is $R(s, e_i) + p(e_i)$; the buffer size requirement of all other edges are unchanged. Since e_f is a FHE, we have $c(e_f) > \sum_{e_j \in E'} p(e_j)$, so we know $R(s', G) < R(s, G)$.

Now, consider the case of $d(e_f) > p(e_f) + c(e_f)$. The only difference from the case discussed above is that, when $d(e_f) > p(e_f) + c(e_f)$, the condition to guarantee that the initial buffer size of e_f before is smaller than $R(s, e_f) - c(e_f)$, otherwise, after we change s to s' by the method introduced above, the buffer requirement of e_f may be reduced by a value smaller than $c(e_f)$, therefore it is not guaranteed that $R(s', G) < R(s, G)$. ■

Since the proof of Lemma 2 is very similar to that of Lemma 1, it is omitted due to space limitations.