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

Nan Guan$^1$, Zonghua Gu$^2$, Wang Yi$^3$, Ge Yu$^1$

$^1$Northeastern University, China
$^2$Hong Kong University of Science and Technology, China
$^3$Uppsala University, Sweden

ASP–DAC’09 January 18–22, 2009
Outline

1 Overview

2 Introduction

3 Improving MC Scalability
   - Firing Count Restriction
   - Tighter Edge Buffer Size Upper Bounds
     - Technique 1
     - Technique 2
   - Graph Decomposition

4 Performance

5 Conclusions
Overview

- Synchronous dataflow (SDF)
  - Also called Statically-Schedulable Dataflow (SSDF)
  - Widely used in multimedia, signal processing, etc.
  - Each actor invocation consumes and produces a constant number of data tokens.

- Buffer Size minimization
  - Memory is a scare resource in embedded systems
  - NP-complete

- Model-checking (MC)
  - pro: obtain provably-optimal solution
  - con: state space explosion limits scalability

- Contribution: improve MC scalability by exploiting SDF-specific properties
Overview

■ Synchronous dataflow (SDF)
  • Also called Statically-Schedulable Dataflow (SSDF)
  • Widely used in multimedia, signal processing, etc.
  • Each actor invocation consumes and produces a constant number of data tokens.

■ Buffer Size minimization
  • Memory is a scarce resource in embedded systems
  • NP-complete

■ Model-checking (MC)
  • pro: obtain provably-optimal solution
  • con: state space explosion limits scalability

■ Contribution: improve MC scalability by exploiting SDF-specific properties
Overview

- Synchronous dataflow (SDF)
  - Also called Statically-Schedulable Dataflow (SSDF)
  - Widely used in multimedia, signal processing, etc.
  - Each actor invocation consumes and produces a constant number of data tokens.

- Buffer Size minimization
  - Memory is a scarce resource in embedded systems
  - NP-complete

- Model-checking (MC)
  - pro: obtain provably-optimal solution
  - con: state space explosion limits scalability

Contribution: improve MC scalability by exploiting SDF-specific properties
Overview

- Synchronous dataflow (SDF)
  - Also called Statically-Schedulable Dataflow (SSDF)
  - Widely used in multimedia, signal processing, etc.
  - Each actor invocation consumes and produces a constant number of data tokens.

- Buffer Size minimization
  - Memory is a scarce resource in embedded systems
  - NP-complete

- Model-checking (MC)
  - pro: obtain provably-optimal solution
  - con: state space explosion limits scalability

- Contribution: improve MC scalability by exploiting SDF-specific properties
Outline

1 Overview

2 Introduction

3 Improving MC Scalability
   • Firing Count Restriction
   • Tighter Edge Buffer Size Upper Bounds
     • Technique 1
     • Technique 2
   • Graph Decomposition

4 Performance

5 Conclusions
Introduction to SDF

Balance Equations:
- for each edge $e$: $r_{src} \times p(e) = r_{snk} \times c(e)$

For this example:
- $r_A \times 1 = r_C \times 3$, $r_A \times 2 = r_B \times 3$, $r_B \times 1 = r_C \times 2$
- solution (repetition vector): $r_A = 3$, $r_B = 2$, $r_C = 1$
- any legal schedule must contain 3 firings of A, 2 firings of B and 1 firing of C
- possible schedules: AAABBC, AABABC
Introduction to SDF

a SDF example

Balance Equations:

- for each edge $e$: $r_{\text{src}} \times p(e) = r_{\text{snk}} \times c(e)$

For this example:

- $r_A \times 1 = r_C \times 3$, $r_A \times 2 = r_B \times 3$, $r_B \times 1 = r_C \times 2$
- solution (repetition vector): $r_A = 3$, $r_B = 2$, $r_C = 1$
- any legal schedule must contain 3 firings of $A$, 2 firings of $B$ and 1 firing of $C$
- possible schedules: AAABBC, AABABC
Introduction to SDF

a SDF example

Balance Equations:
- for each edge \( e \): \( r_{\text{src}} \times p(e) = r_{\text{snk}} \times c(e) \)

For this example:
- \( r_A \times 1 = r_C \times 3 \), \( r_A \times 2 = r_B \times 3 \), \( r_B \times 1 = r_C \times 2 \)
- solution (repetition vector): \( r_A = 3, r_B = 2, r_C = 1 \)
- any legal schedule must contain 3 firings of A, 2 firings of B and 1 firing of C
- possible schedules: AAABBC, AABABC
Introduction to SDF

Balance Equations:

- for each edge $e$: $r_{src} \times p(e) = r_{snk} \times c(e)$

For this example:

- $r_A \times 1 = r_C \times 3$, $r_A \times 2 = r_B \times 3$, $r_B \times 1 = r_C \times 2$

- solution (repetition vector): $r_A = 3, r_B = 2, r_C = 1$

- any legal schedule must contain 3 firings of A, 2 firings of B and 1 firing of C

- possible schedules: AAABBC, AABABC
Introduction to SDF

Balance Equations:
- for each edge $e$: $r_{src} \times p(e) = r_{snk} \times c(e)$

For this example:
- $r_A \times 1 = r_C \times 3$, $r_A \times 2 = r_B \times 3$, $r_B \times 1 = r_C \times 2$
- solution (repetition vector): $r_A = 3$, $r_B = 2$, $r_C = 1$
- any legal schedule must contain 3 firings of A, 2 firings of B and 1 firing of C
- possible schedules: AAABBC, AABABC
Balance Equations:

- for each edge $e$: \[ r_{\text{src}} \times p(e) = r_{\text{snk}} \times c(e) \]

For this example:

- $r_A \times 1 = r_C \times 3$, \quad $r_A \times 2 = r_B \times 3$, \quad $r_B \times 1 = r_C \times 2$
- solution (repetition vector): $r_A = 3$, $r_B = 2$, $r_C = 1$
- any legal schedule must contain 3 firings of A, 2 firings of B and 1 firing of C

possible schedules: AAABBC, AABABC
Introduction to SDF

Balance Equations:

- for each edge $e$: $r_{src} \times p(e) = r_{snk} \times c(e)$

For this example:

- $r_A \times 1 = r_C \times 3$, $r_A \times 2 = r_B \times 3$, $r_B \times 1 = r_C \times 2$
- solution (repetition vector): $r_A = 3, r_B = 2, r_C = 1$
- any legal schedule must contain 3 firings of A, 2 firings of B and 1 firing of C
- possible schedules: AAABBC, AABABC
Total Required Buffer Size: $6 + 2 = 8$

We assume that each edge has its dedicated buffer space in this paper, instead of a global shared buffer space for all edges.
SDF Scheduling and Edge Buffer Sizes

We assume that each edge has its dedicated buffer space in this paper, instead of a global shared buffer space for all edges.

Total Required Buffer Size: $6 + 2 = 8$
SDF Scheduling and Edge Buffer Sizes

s1: AA ABBC AAABBC ... ...

Max Req. = 6  Max Req. = 2

Total Required Buffer Size: 6 + 2 = 8

We assume that each edge has its dedicated buffer space in this paper, instead of a global shared buffer space for all edges.
SDF Scheduling and Edge Buffer Sizes

\[ s1: \text{AAA} \quad \text{BBC} \quad \text{AAABBBC} \ldots \ldots \]

Max Req. = 6  Max Req. = 2

Total Required Buffer Size: 6 + 2 = 8

We assume that each edge has its dedicated buffer space in this paper, instead of a global shared buffer space for all edges.
SDF Scheduling and Edge Buffer Sizes

s1: AAAB

Max Req. = 6

Max Req. = 2

Total Required Buffer Size: $6 + 2 = 8$

We assume that each edge has its dedicated buffer space in this paper, instead of a global shared buffer space for all edges.
SDF Scheduling and Edge Buffer Sizes

Total Required Buffer Size: $6 + 2 = 8$

We assume that each edge has its dedicated buffer space in this paper, instead of a global shared buffer space for all edges.
SDF Scheduling and Edge Buffer Sizes

s1: AAABBC  AAABBC  ...

Max Req. = 6  Max Req. = 2

Total Required Buffer Size: 6 + 2 = 8

We assume that each edge has its dedicated buffer space in this paper, instead of a global shared buffer space for all edges.
SDF Scheduling and Edge Buffer Sizes

s1: AAABBC AAABBC  

Max Req. = 6  Max Req. = 2

Total Required Buffer Size: 6 + 2 = 8

We assume that each edge has its dedicated buffer space in this paper, instead of a global shared buffer space for all edges.
SDF Scheduling and Edge Buffer Sizes

s1: AAABBC AAABBC ... ...

Max Req. = 6
Max Req. = 2

Total Required Buffer Size: 6 + 2 = 8

We assume that each edge has its dedicated buffer space in this paper, instead of a global shared buffer space for all edges.
SDF Scheduling and Edge Buffer Sizes ...

s2: A A B A B C A A B A B C ...

Max Req. = 4  Max Req. = 2

Total Required Buffer Size: 4 + 2 = 6
SDF Scheduling and Edge Buffer Sizes ...

s2: A A B A B C A A B A B C ...

Max Req. = 4    Max Req. = 2

Total Required Buffer Size: 4 + 2 = 6
SDF Scheduling and Edge Buffer Sizes ...

s2: A A B A B C A A B A B C ... ...

\[ \begin{align*}
\text{Max Req.} &= 4 \\
\text{Max Req.} &= 2 \\
\text{Total Required Buffer Size}: 4 + 2 &= 6
\end{align*} \]
SDF Scheduling and Edge Buffer Sizes ...

s2: AAB ABC AABABC ...

Max Req. = 4  Max Req. = 2

Total Required Buffer Size: 4 + 2 = 6
SDF Scheduling and Edge Buffer Sizes ...

s2: AABA

Max Req. = 4

Total Required Buffer Size: 4 + 2 = 6
SDF Scheduling and Edge Buffer Sizes ...

s2: A A B A B C A A B A B C ... ...

A \rightarrow B \rightarrow C

\text{Max Req.} = 4 \quad \text{Max Req.} = 2

Total Required Buffer Size: 4 + 2 = 6
SDF Scheduling and Edge Buffer Sizes ...
SDF Scheduling and Edge Buffer Sizes ...

s2: AABABC AABABC ... ...  

Max Req. = 4  Max Req. = 2

Total Required Buffer Size: 4 + 2 = 6
SDF Scheduling and Edge Buffer Sizes...

s2: A A B A B C A A B A B C

A \[ e_1 \] 3 \[ e_2 \] B 1 \[ e_2 \] C

Max Req. = 4

Total Required Buffer Size: 4 + 2 = 6
State Space Representation

\[ s_1: \text{A A A B B C} \]
State Space Representation ...
Using MC to Find Minimal Buffer Size

Verification Claim: Linear Temporal Logic (LTL) formula (for SPIN):
- \( <> \text{BufReq} \geq \text{BOUND} \)
- "All possible schedules will eventually lead to a state where the total buffer size requirement is larger than or equal to a user-specified bound."

If proven False, then a feasible schedule has been found with buffer size requirement \( \text{BufReq} < \text{BOUND} \).
Set \( \text{BOUND} = \text{BufReq} \) and run MC. \( \text{BOUND} \) is reduced iteratively until the LTL formula is proven True.
Using MC to Find Minimal Buffer Size

- Verification Claim: Linear Temporal Logic (LTL) formula (for SPIN):
  - $<> \text{BufReq} \geq \text{BOUND}$
  - "All possible schedules will eventually lead to a state where the total buffer size requirement is larger than or equal to a user-specified bound."

- If proven False, then a feasible schedule has been found with buffer size requirement $\text{BufReq} < \text{BOUND}$.

- Set $\text{BOUND} = \text{BufReq}$ and run MC. $\text{BOUND}$ is reduced iteratively until the LTL formula is proven True.
Verification Claim: Linear Temporal Logic (LTL) formula (for SPIN):

- \(<> \text{BufReq} \geq \text{BOUND} \)
- "All possible schedules will eventually lead to a state where the total buffer size requirement is larger than or equal to a user-specified bound."

If proven False, then a feasible schedule has been found with buffer size requirement \( \text{BufReq} < \text{BOUND} \).

Set \( \text{BOUND} = \text{BufReq} \) and run MC. \( \text{BOUND} \) is reduced iteratively until the LTL formula is proven True.
Using MC to Find Minimal Buffer Size

- System Model
- TL Spec
- Model-Checker (SPIN, NuSMV)
- True
- False + Counter Example (schedule)

Verification Claim: Linear Temporal Logic (LTL) formula (for SPIN):
- \(<> \text{BufReq} \geq BOUND\)
- "All possible schedules will eventually lead to a state where the total buffer size requirement is larger than or equal to a user-specified bound."

If proven False, then a feasible schedule has been found with buffer size requirement \(\text{BufReq} < BOUND\).

Set \(BOUND = \text{BufReq}\) and run MC. \(BOUND\) is reduced iteratively until the LTL formula is proven True.
Using MC to Find Minimal Buffer Size

- **System Model**
- **TL Spec**
- **Model-Checker (SPIN, NuSMV)**
- **True**
- **False + Counter Example (schedule)**

- **Verification Claim**: Linear Temporal Logic (LTL) formula (for SPIN):
  - $<> \text{BufReq} \geq \text{BOUND}$
  - "All possible schedules will eventually lead to a state where the total buffer size requirement is larger than or equal to a user-specified bound."

- If proven **False**, then a feasible schedule has been found with buffer size requirement $\text{BufReq} < \text{BOUND}$.

- Set $\text{BOUND} = \text{BufReq}$ and run MC. $\text{BOUND}$ is reduced iteratively until the LTL formula is proven **True**.
Outline

1 Overview

2 Introduction

3 Improving MC Scalability
   - Firing Count Restriction
   - Tighter Edge Buffer Size Upper Bounds
     - Technique 1
     - Technique 2
   - Graph Decomposition

4 Performance

5 Conclusions
Rationale Behind the Techniques

- **Firing Count Restriction**
  - helps reduce system state space

- **Tighter Edge Buffer Size Upper Bounds (UB)**
  - helps reduce system state space
  - also helps reduce the number of model-checker invocations in the iterative procedure to obtain the minimum buffer size requirement

- **Graph Decomposition**
  - use divide-and-conquer to decompose a large problem into multiple smaller sub-problems for certain SDF graphs with a special topology
Firing Count Restriction

Overview

Introduction

Improving MC Scalability

Firing Count Restriction

Tighter Edge Buffer Size Upper Bounds Technique 1 Technique 2 Graph Decomposition

Performance

Conclusions
Firing Count Restriction ...

- If a SDF graph has a schedule with bounded memory requirement, it must have a periodic schedule where each actor firing count is equal to its firing count in the repetition vector [Lee’87].

- To help reduce MC state space, we restrict each actor’s firing count to not exceed its entry in the repetition vector.

Tighter Upper Bounds – Technique 1

A Naive Upper Bound (UB):

\[ UB(e) = p(e) \times r_{src}(e) + d(e) \]

“This upper bound is too loose!”
Tighter Upper Bounds – Technique 1

A Naive Upper Bound (UB):

\[ UB(e) = p(e) \times r_{src}(e) + d(e) \]

“This upper bound is too loose!”
Tighter Upper Bounds – Technique 1 ...

- Given a known feasible schedule $s$ with total buffer requirement $R(s)$:
  \[ UB(e_i) \leq R(s) - \sum_{e_j \neq e_i} LB(e_j) \]

- A heuristic algorithm [Bh’96] can be used to obtain a feasible schedule $s$.
  - Optimal for acyclic, delayless SDF graphs, but not for general SDF graphs.

- Edge buffer lower bound ($LB$) can be obtained [Bh’96]:
  \[ LB = \begin{cases} 
    d & d > p + c - g \\
    p + c - g + d \mod g & \text{otherwise}
  \end{cases} \]
  \[ g = \gcd(p, c) \]

Tighter Upper Bounds – Technique 1 ...

- Given a known feasible schedule \( s \) with total buffer requirement \( R(s) \):

\[
UB(e_i) \leq R(s) - \sum_{e_j \neq e_i} LB(e_j)
\]

- A heuristic algorithm [Bh’96] can be used to obtain a feasible schedule \( s \).
  - Optimal for acyclic, delayless SDF graphs, but not for general SDF graphs.

- Edge buffer lower bound (\( LB \)) can be obtained [Bh’96]:

\[
LB = \begin{cases} 
  d & \text{if } d > p + c - g \\
  p + c - g + d \mod g & \text{otherwise}
\end{cases}
\]

\[
g = \gcd(p, c)
\]

Tighter Upper Bounds – Technique 1 ...

- Given a known feasible schedule \( s \) with total buffer requirement \( R(s) \):
  \[
  UB(e_i) \leq R(s) - \sum_{e_j \neq e_i} LB(e_j)
  \]

- A heuristic algorithm [Bh’96] can be used to obtain a feasible schedule \( s \).
  - Optimal for acyclic, delayless SDF graphs, but not for general SDF graphs.

- Edge buffer lower bound (\( LB \)) can be obtained [Bh’96]:
  \[
  LB = \begin{cases} 
  d & d > p + c - g \\
  p + c - g + d \mod g & \text{otherwise}
  \end{cases}
  \\[g = gcd(p, c)\]

Tighter Upper Bounds – Technique 2: Heavy Edges

- \( s_1: C C A A B \)
  - \( R(s_1) = 2(e_1) + 2(e_2) + 16(e_3) = 20 \)
  - \( s_1 \) is unadvisable, since \( e_3 \) is a "heavy edge", and we should avoid accumulating tokens on it

- \( s_2: C A C A B \)
  - \( R(s_2) = 2(e_1) + 2(e_2) + 8(e_3) = 12 \)
Tighter Upper Bounds – Technique 2: Heavy Edges

- **s_1**: C C A A B
  - \( R(s_1) = 2(e_1) + 2(e_2) + 16(e_3) = 20 \)
  - \( s_1 \) is unadvisable, since \( e_3 \) is a "heavy edge", and we should avoid accumulating tokens on it

- **s_2**: C A C A B
  - \( R(s_2) = 2(e_1) + 2(e_2) + 8(e_3) = 12 \)
Tighter Upper Bounds – Technique 2: Heavy Edges

- $s_1$: C C A A B
  - $R(s_1) = 2(e_1) + 2(e_2) + 16(e_3) = 20$
  - $s_1$ is unadvisable, since $e_3$ is a "heavy edge", and we should avoid accumulating tokens on it

- $s_2$: C A C A B
  - $R(s_2) = 2(e_1) + 2(e_2) + 8(e_3) = 12$
Tighter Upper Bounds – Technique 2: Heavy Edges

- $s_1$: C C A A B
  - $R(s_1) = 2(e_1) + 2(e_2) + 16(e_3) = 20$
  - $s_1$ is unadvisable, since $e_3$ is a "heavy edge", and we should avoid accumulating tokens on it

- $s_2$: C A C A B
  - $R(s_2) = 2(e_1) + 2(e_2) + 8(e_3) = 12$
Tighter Upper Bounds – Technique 2: Heavy Edges

- \( s_1 \): C C A A B
  - \( R(s_1) = 2(e_1) + 2(e_2) + 16(e_3) = 20 \)
  - \( s_1 \) is unadvisable, since \( e_3 \) is a "heavy edge", and we should avoid accumulating tokens on it

- \( s_2 \): C A C A B
  - \( R(s_2) = 2(e_1) + 2(e_2) + 8(e_3) = 12 \)
Tighter Upper Bounds – Technique 2: Heavy Edges

- **s₁:** C C A A B
  - \( R(s₁) = 2(e₁) + 2(e₂) + 16(e₃) = 20 \)
  - \( s₁ \) is unadvisable, since \( e₃ \) is a "heavy edge", and we should avoid accumulating tokens on it

- **s₂:** C A C A B
  - \( R(s₂) = 2(e₁) + 2(e₂) + 8(e₃) = 12 \)
Tight Upper Bounds – Technique 2: Heavy Edges ...

- Forward Heavy Edge (FHE): $c > p_1 + p_2 + \ldots + p_n$

- Backward Heavy Edge (BHE): $p > c_1 + c_2 + \ldots + c_n$

- Regular Heavy Edge (RHE): A FHE where $p(e)$ and $d(e)$ are integer multiples of $c(e)$, or A BHE where $c(e)$ and $d(e)$ are integer multiples of $p(e)$
Tighter Upper Bounds – Technique 2: Heavy Edges …

Forward Heavy Edge (FHE) \[ c > p_1 + p_2 + \ldots + p_n \]

Backward Heavy Edge (BHE) \[ p > c_1 + c_2 + \ldots + c_n \]

Regular Heavy Edge (RHE)
A FHE where \( p(e) \) and \( d(e) \) are integer multiples of \( c(e) \), or
A BHE where \( c(e) \) and \( d(e) \) are integer multiples of \( p(e) \)
Tighter Upper Bounds – Technique 2: Heavy Edges ...

**Forward Heavy Edge (FHE)**

\[ c > p_1 + p_2 + \ldots + p_n \]

**Backward Heavy Edge (BHE)**

\[ p > c_1 + c_2 + \ldots + c_n \]

**Regular Heavy Edge (RHE)**

A FHE where \( p(e) \) and \( d(e) \) are integer multiples of \( c(e) \), or

A BHE where \( c(e) \) and \( d(e) \) are integer multiples of \( p(e) \).
Tighter Upper Bounds – Technique 2: Heavy Edges ...

Forward Heavy Edge (FHE) — $c > p_1 + p_2 + \ldots + p_n$

Backward Heavy Edge (BHE) — $p > c_1 + c_2 + \ldots + c_n$

Regular Heavy Edge (RHE)
A FHE where $p(e)$ and $d(e)$ are integer multiples of $c(e)$, or
A BHE where $c(e)$ and $d(e)$ are integer multiples of $p(e)$
Upper Bounds for Heavy Edges:

- If $e_f$ is an FHE, we can set UB of $e_f$ as
  \[ \max(p(e_f) + c(e_f), d(e_f)) + c(e_f) \]

- If $e_b$ is an BHE, we can set UB of $e_b$ as
  \[ \max(p(e_b) + c(e_b), d(e_b)) + p(e_b) \]

- If $e_r$ is an RHE, we can set the upper bound of $e_r$ as $LB(e_r)$
Tighter Upper Bounds – Technique 2: Heavy Edges ....

Upper Bounds for Heavy Edges:

- If $e_f$ is an FHE, we can set UB of $e_f$ as
  \[
  \max(p(e_f) + c(e_f), d(e_f)) + c(e_f)
  \]

- If $e_b$ is an BHE, we can set UB of $e_b$ as
  \[
  \max(p(e_b) + c(e_b), d(e_b)) + p(e_b)
  \]

- If $e_r$ is an RHE, we can set the upper bound of $e_r$ as
  \[
  LB(e_r)
  \]
Tighter Upper Bounds – Technique 2: Heavy Edges ....

Upper Bounds for Heavy Edges:

- If \(e_f\) is an FHE, we can set UB of \(e_f\) as
  \[
  \max(p(e_f) + c(e_f), d(e_f)) + c(e_f)
  \]

- If \(e_b\) is an BHE, we can set UB of \(e_b\) as
  \[
  \max(p(e_b) + c(e_b), d(e_b)) + p(e_b)
  \]

- If \(e_r\) is an RHE, we can set the upper bound of \(e_r\) as
  \[
  LB(e_r)
  \]
Tighter Upper Bounds – Technique 2: Heavy Edges ....

Upper Bounds for Heavy Edges:

- If $e_f$ is an FHE, we can set UB of $e_f$ as

  $$\max(p(e_f) + c(e_f), d(e_f)) + c(e_f)$$

- If $e_b$ is an BHE, we can set UB of $e_b$ as

  $$\max(p(e_b) + c(e_b), d(e_b)) + p(e_b)$$

- If $e_r$ is an RHE, we can set the upper bound of $e_r$ as $LB(e_r)$
**Tighter Upper Bounds – Technique 2: Heavy Edges .....**

![Graph Diagram](image)

<table>
<thead>
<tr>
<th></th>
<th>$e_{AB}$</th>
<th>$e_{BC}$</th>
<th>$e_{CE}$</th>
<th>$e_{BD}$</th>
<th>$e_{ED}$</th>
<th>$e_{DA}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive UB</td>
<td>30</td>
<td>12</td>
<td>6</td>
<td>12</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>Improved UB</td>
<td>16</td>
<td>2</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>32</td>
</tr>
</tbody>
</table>
Definition: Bridge

A bridge in graph theory is an edge s.t. if it is deleted, the graph will become two separate subgraphs.

Given known optimal schedules $s_1$ and $s_2$ for subgraphs $G_1$ and $G_2$, we can get an optimal schedule $s$ of $G$ by
- firing each node by following the known optimal schedules $s_1$ and $s_2$, and
- firing the sink of the bridge $e_b$ as soon as possible

$$R_{opt}(G) = R_{opt}(G_1) + R_{opt}(G_2) + LB(e_b)$$
Graph Decomposition

Definition: Bridge

A bridge in graph theory is an edge s.t. if it is deleted, the graph will become two separate subgraphs.

Given known optimal schedules $s_1$ and $s_2$ for subgraphs $G_1$ and $G_2$, we can get an optimal schedule $s$ of $G$ by

- firing each node by following the known optimal schedules $s_1$ and $s_2$, and
- firing the sink of the bridge $e_b$ as soon as possible.

$$R_{opt}(G) = R_{opt}(G_1) + R_{opt}(G_2) + LB(e_b)$$
Graph Decomposition

**Definition: Bridge**

A bridge in graph theory is an edge s.t. if it is deleted, the graph will become two separate subgraphs.

![Graph Decomposition Diagram](image)

- Given known optimal schedules $s_1$ and $s_2$ for subgraphs $G_1$ and $G_2$, we can get an optimal schedule $s$ of $G$ by
  - firing each node by following the known optimal schedules $s_1$ and $s_2$, and
  - firing the sink of the bridge $e_b$ as soon as possible

$$R_{opt}(G) = R_{opt}(G_1) + R_{opt}(G_2) + LB(e_b)$$
**Graph Decomposition**

**Definition: Bridge**

A *bridge* in graph theory is an edge s.t. if it is deleted, the graph will become two separate subgraphs.

---

**Example**

Given known optimal schedules $s_1$ and $s_2$ for subgraphs $G_1$ and $G_2$, we can get an optimal schedule $s$ of $G$ by:

- firing each node by following the known optimal schedules $s_1$ and $s_2$, and
- firing the sink of the bridge $e_b$ as soon as possible

\[ R_{opt}(G) = R_{opt}(G_1) + R_{opt}(G_2) + LB(e_b) \]
**Graph Decomposition**

**Definition: Bridge**

A bridge in graph theory is an edge s.t. if it is deleted, the graph will become two separate subgraphs.

Given known optimal schedules $s_1$ and $s_2$ for subgraphs $G_1$ and $G_2$, we can get an optimal schedule $s$ of $G$ by

- firing each node by following the known optimal schedules $s_1$ and $s_2$, and
- firing the sink of the bridge $e_b$ as soon as possible

$$R_{opt}(G) = R_{opt}(G_1) + R_{opt}(G_2) + LB(e_b)$$
Performance

- Use SDF$^3$ [Gelein’06], to generate random SDF graphs
- Compare the state space size with and without our optimizations

<table>
<thead>
<tr>
<th>Experiment</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Actors</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>Number of States with the original approach in [Gelein’05]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOUND-1 (MB)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7064</td>
<td>26394</td>
</tr>
<tr>
<td>BOUND (MB)</td>
<td>13</td>
<td>88</td>
<td>115</td>
<td>452</td>
<td>193</td>
<td>195</td>
<td>216</td>
<td>18341</td>
</tr>
<tr>
<td>Number of States with the optimized approach in this paper</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOUND-1 (s)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1244</td>
<td>11111</td>
</tr>
<tr>
<td>BOUND (s)</td>
<td>13</td>
<td>64</td>
<td>82</td>
<td>114</td>
<td>112</td>
<td>92</td>
<td>91</td>
<td>4120</td>
</tr>
</tbody>
</table>


Performance

- Use SDF³ [Gelein’06], to generate random SDF graphs
- Compare the state space size with and without our optimizations

<table>
<thead>
<tr>
<th>Experiment</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Actors</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>Number of States with the original approach in [Gelein’05]</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7064</td>
<td>26394</td>
</tr>
<tr>
<td>BOUND -1 (MB)</td>
<td>13</td>
<td>88</td>
<td>115</td>
<td>452</td>
<td>193</td>
<td>195</td>
<td>216</td>
<td>18341</td>
</tr>
<tr>
<td>BOUND (MB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of States with the optimized approach in this paper</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1244</td>
<td>11111</td>
</tr>
<tr>
<td>BOUND -1 (s)</td>
<td>13</td>
<td>64</td>
<td>82</td>
<td>114</td>
<td>112</td>
<td>92</td>
<td>91</td>
<td>4120</td>
</tr>
<tr>
<td>BOUND (s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Performance

- Use SDF$^3$ [Gelein’06], to generate random SDF graphs
- Compare the state space size with and without our optimizations

<table>
<thead>
<tr>
<th>Experiment</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Actors</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of States with the original approach in [Gelein’05]</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOUND$^1$ (MB)</td>
</tr>
<tr>
<td>BOUND (MB)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of States with the optimized approach in this paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOUND$^1$ (s)</td>
</tr>
<tr>
<td>BOUND (s)</td>
</tr>
</tbody>
</table>


Performance

- Use SDF$^3$ [Gelein’06], to generate random SDF graphs
- Compare the state space size with and without our optimizations

<table>
<thead>
<tr>
<th>Experiment</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Actors</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>Number of States with the original approach in [Gelein’05]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOUND$^{-1}$ (MB)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7064</td>
<td>26394</td>
</tr>
<tr>
<td>BOUND (MB)</td>
<td>13</td>
<td>88</td>
<td>115</td>
<td>452</td>
<td>193</td>
<td>195</td>
<td>216</td>
<td>18341</td>
</tr>
<tr>
<td>Number of States with the optimized approach in this paper</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOUND$^{-1}$ (s)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1244</td>
<td>11111</td>
</tr>
<tr>
<td>BOUND (s)</td>
<td>13</td>
<td>64</td>
<td>82</td>
<td>114</td>
<td>112</td>
<td>92</td>
<td>91</td>
<td>4120</td>
</tr>
</tbody>
</table>


Outline

1 Overview

2 Introduction

3 Improving MC Scalability
   - Firing Count Restriction
   - Tighter Edge Buffer Size Upper Bounds
     - Technique 1
     - Technique 2
   - Graph Decomposition

4 Performance

5 Conclusions
Conclusions

- Presented a set of techniques for improving MC efficiency
  - Actor firing count restriction
  - Tighter upper bounds for edge buffer size
  - Graph decomposition

- Performance evaluation shows their effectiveness in reducing state space