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, Swenden

ASP–DAC'09 January 18–22, 2009

Outline

Outline

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 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

- Overview
- Introduction
- Improving MC Scalability
- Performance
- Conclusions

- 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

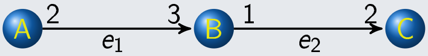
Overview

- 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 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

Outline

Introduction to SDF

a SDF example



Balance Equations:

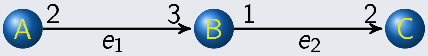
- for each edge e: $r_{src} \times p(e) = r_{snk} \times c(e)$

For this example:

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

a SDF example



Balance Equations:

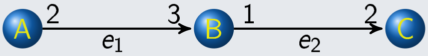
- for each edge e: $r_{src} \times p(e) = r_{snk} \times c(e)$

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:

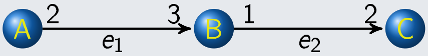
- for each edge e: $r_{src} \times p(e) = r_{snk} \times c(e)$

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:

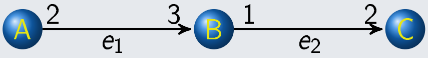
- for each edge e: $r_{src} \times p(e) = r_{snk} \times c(e)$

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:

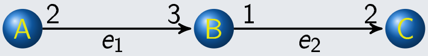
- for each edge e: $r_{src} \times p(e) = r_{snk} \times c(e)$

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

a SDF example



Balance Equations:

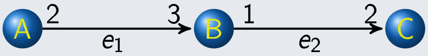
- for each edge e: $r_{src} \times p(e) = r_{snk} \times c(e)$

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:

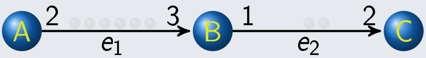
- for each edge e: $r_{src} \times p(e) = r_{snk} \times c(e)$

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

SDF Scheduling and Edge Buffer Sizes

s1: A A A B B C A A A B B C ··· ···



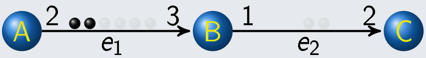
Max Req.= 6 Max Req.=2

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: A A A B B C A A A B B C ··· ···



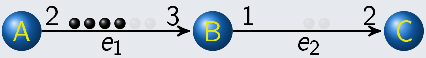
Max Req.= 6   Max Req.=2

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: A A A B B C A A A B B C ··· ···



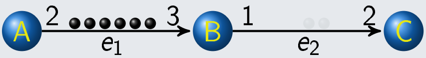
Max Req.= 6   Max Req.=2

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 ··· ···



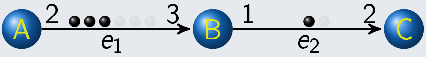
Max Req.= 6   Max Req.=2

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: AAAB BC AAABBC ··· ···



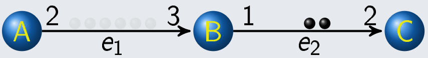
Max Req.= 6   Max Req.=2

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: A A A B B C A A A B B C ··· ···



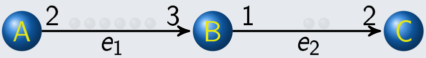
Max Req.= 6 Max Req.=2

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 ··· ···



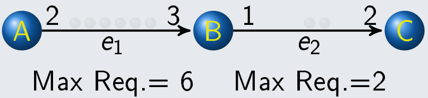
Max Req.= 6 Max Req.=2

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

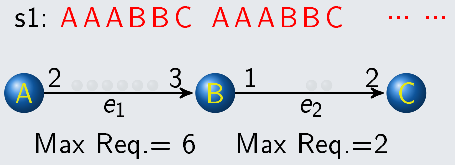
s1: AAABBC AAABBC ··· ···



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

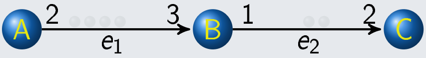


Total Required Buffer Size: $6+2=8$

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 ...

s2: AABABC AABABC ... ...

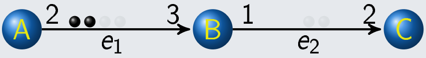


Max Req.= 4   Max Req.=2

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 ... ...

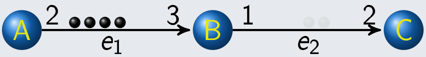


Max Req.= 4 Max Req.=2

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 ··· ···

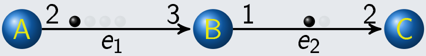


Max Req.= 4 Max Req.=2

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 ··· ···

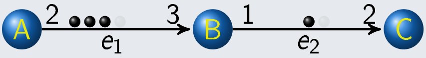


Max Req.= 4 Max Req.=2

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 ··· ···

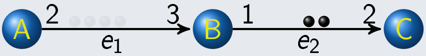


Max Req.= 4 Max Req.=2

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 ... ...

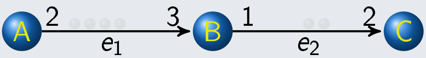


Max Req.= 4 Max Req.=2

Total Required Buffer Size: 4+2 =6

SDF Scheduling and Edge Buffer Sizes ...

s2: AABABC AABABC ··· ···

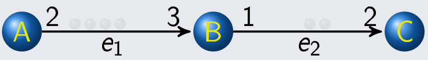


Max Req.= 4 Max Req.=2

Total Required Buffer Size: 4 + 2 =6

SDF Scheduling and Edge Buffer Sizes ...

s2: AABABC AABABC ... ...

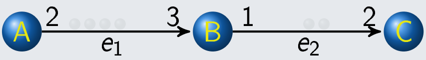


Max Req.= 4 Max Req.=2

Total Required Buffer Size: 4+2 =6

SDF Scheduling and Edge Buffer Sizes ...

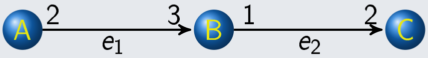
s2: AABABC AABABC ··· ···



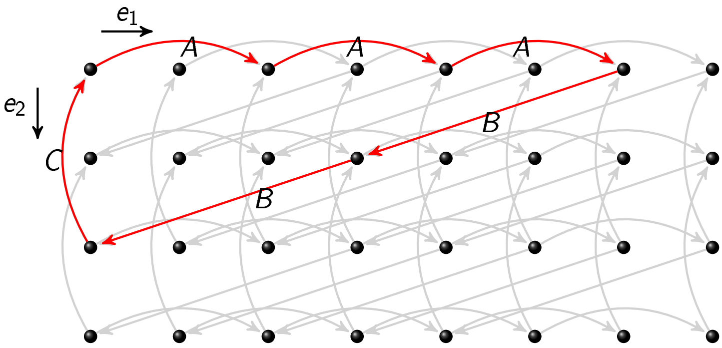
Max Req.= 4 Max Req.=2

Total Required Buffer Size: $4 + 2 = 6$

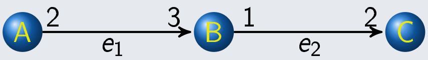
State Space Representation



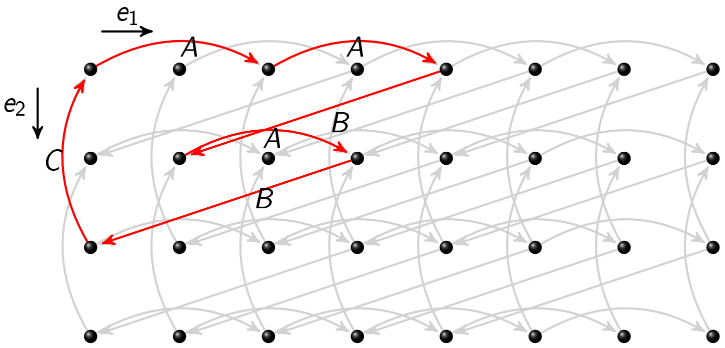
s_1: A A A B B C



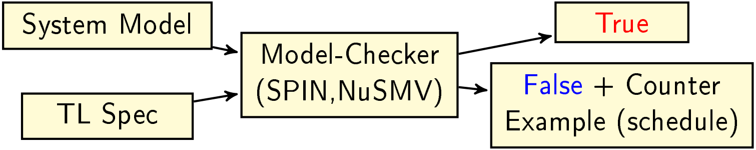
State Space Representation ...



s_2: A A B A B C

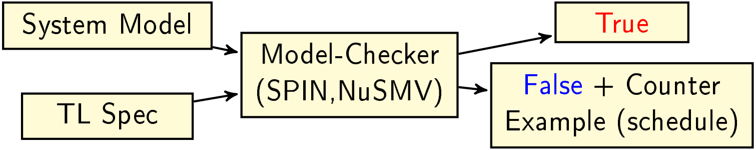


Using MC to Find Minimal Buffer Size



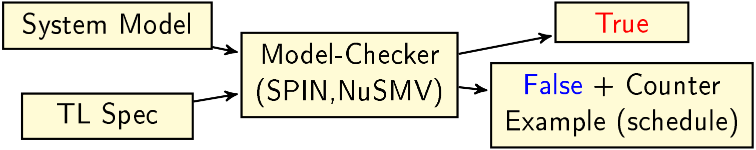
- Verification Claim: Linear Temporal Logic (LTL) formula (for SPIN):
  - $<> BufReq \geq BOUND$
  - "All possible schedules will eventually lead to a state where the total buffer size requirement is larger than or equal to a user-specified bound."
- If proven False, then a feasible schedule has been found with buffer size requirement $BufReq < BOUND$.
- Set $BOUND = BufReq$ and run MC. $BOUND$ is reduced iteratively until the LTL formula is proven True.

Using MC to Find Minimal Buffer Size



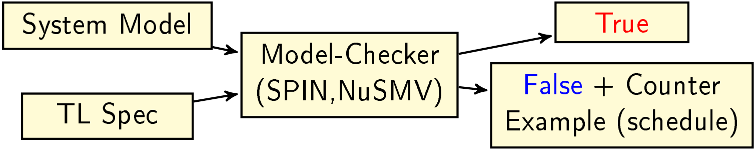
- Verification Claim: Linear Temporal Logic (LTL) formula (for SPIN):
  - $<> BufReq \geq BOUND$
  - "All possible schedules will eventually lead to a state where the total buffer size requirement is larger than or equal to a user-specified bound."
- If proven False, then a feasible schedule has been found with buffer size requirement $BufReq < BOUND$.
- Set $BOUND = BufReq$ and run MC. $BOUND$ is reduced iteratively until the LTL formula is proven True.

Using MC to Find Minimal Buffer Size



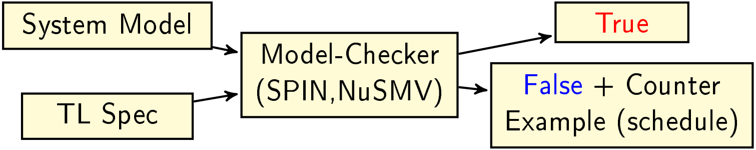
- Verification Claim: Linear Temporal Logic (LTL) formula (for SPIN):
  - $<> BufReq \geq BOUND$
  - "All possible schedules will eventually lead to a state where the total buffer size requirement is larger than or equal to a user-specified bound."
- If proven False, then a feasible schedule has been found with buffer size requirement $BufReq < BOUND$.
- Set $BOUND = BufReq$ and run MC. $BOUND$ 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):
  - $<> BufReq \geq BOUND$
  - "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 $BufReq < BOUND$.
- Set $BOUND = BufReq$ and run MC. $BOUND$ is reduced iteratively until the LTL formula is proven True.

Using MC to Find Minimal Buffer Size



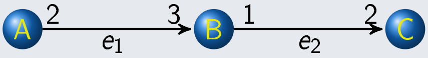
- Verification Claim: Linear Temporal Logic (LTL) formula (for SPIN):
  - $<> BufReq \geq BOUND$
  - "All possible schedules will eventually lead to a state where the total buffer size requirement is larger than or equal to a user-specified bound."
- If proven False, then a feasible schedule has been found with buffer size requirement $BufReq < BOUND$.
- Set $BOUND = BufReq$ and run MC. $BOUND$ is reduced iteratively until the LTL formula is proven True.

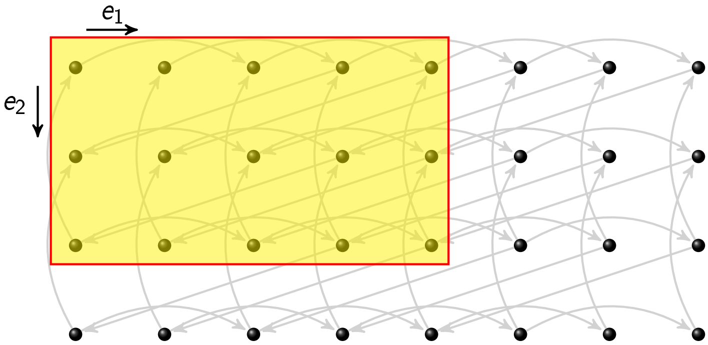
Outline

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





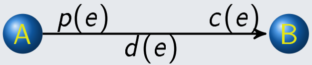
Firing Count Restriction ...

- 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

Edward A. Lee, David G. Messerschmitt: Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing. IEEE Trans. Computers 36(1): 24-35 (1987)

Tighter Upper Bounds – Technique 1

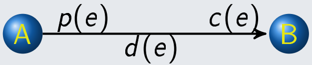


A Naive Upper Bound (UB):

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

"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]:

$$LB = \begin{cases} d & d > p + c - g \\ p + c - g + d \bmod g & \text{otherwise} \end{cases}$$

$$g = gcd(p, c)$$

S.S. Bhattacharyya, P.K. Murthy and E.A. Lee, Software Synthesis from Dataflow Graphs, Kluewer Academic Publishers, 1996

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]:

$$LB = \begin{cases} d & d > p+c-g \\ p+c-g+d \bmod g & \text{otherwise} \end{cases}$$

$$g = gcd(p, c)$$

S.S. Bhattacharyya, P.K. Murthy and E.A. Lee, Software Synthesis from Dataflow Graphs, Kluewer Academic Publishers, 1996

Tighter Upper Bounds – Technique 1 ...

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

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

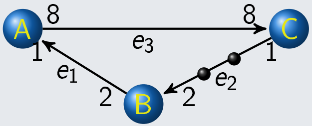
- A heuristic algorithm [Bh'96] can be used to obtain a feasible schedule s.
  - Optimal for acyclic, delayless SDF graphs, but not for general SDF graphs.
- Edge buffer lower bound (LB) can be obtained [Bh'96]:

$$LB = \begin{cases} d & d > p + c - g \\ p + c - g + d \bmod g & \text{otherwise} \end{cases}$$

$$g = gcd(p, c)$$

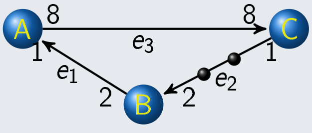
S.S. Bhattacharyya, P.K. Murthy and E.A. Lee, Software Synthesis from Dataflow Graphs, Kluewer Academic Publishers, 1996

Tighter Upper Bounds – Technique 2: Heavy Edges



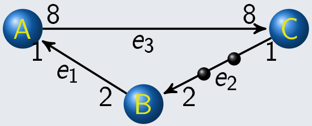
- s_1: C C A A B
  - $R(s_1) = 2(e_1) + 2(e_2) + 16(e_3) = 20$
  - s_1 is unadvisable, since e_3 is a "heavy edge", and we should avoid accumulating tokens on it
- s_2: C A C A B
  - $R(s_2) = 2(e_1) + 2(e_2) + 8(e_3) = 12$

Tighter Upper Bounds – Technique 2: Heavy Edges



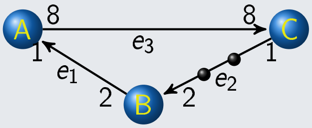
- s_1: C C A A B
  - $R(s_1) = 2(e_1) + 2(e_2) + 16(e_3) = 20$
  - s_1 is unadvisable, since e_3 is a "heavy edge", and we should avoid accumulating tokens on it
- s_2: C A C A B
  - $R(s_2) = 2(e_1) + 2(e_2) + 8(e_3) = 12$

Tighter Upper Bounds – Technique 2: Heavy Edges



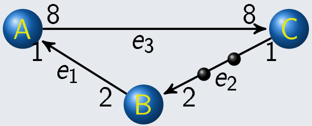
- s_1: C C A A B
  - $R(s_1) = 2(e_1) + 2(e_2) + 16(e_3) = 20$
  - s_1 is unadvisable, since e_3 is a "heavy edge", and we should avoid accumulating tokens on it
- s_2: C A C A B
  - $R(s_2) = 2(e_1) + 2(e_2) + 8(e_3) = 12$

Tighter Upper Bounds – Technique 2: Heavy Edges



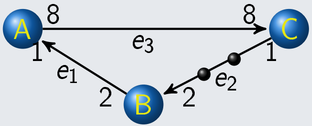
- s_1: C C A A B
  - $R(s_1) = 2(e_1) + 2(e_2) + 16(e_3) = 20$
  - s_1 is unadvisable, since e_3 is a "heavy edge", and we should avoid accumulating tokens on it
- s_2: C A C A B
  - $R(s_2) = 2(e_1) + 2(e_2) + 8(e_3) = 12$

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



- 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 ...

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

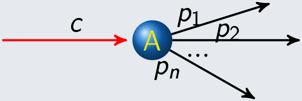
Backward Heavy Edge (BHE) —— $p > c_1 + c_2 + \ldots + c_n$

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) —– $c > p_1 + p_2 + \ldots + p_n$



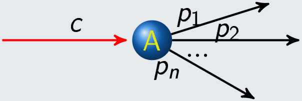
Backward Heavy Edge (BHE) —– $p > c_1 + c_2 + \ldots + c_n$

Regular Heavy Edge (RHE)

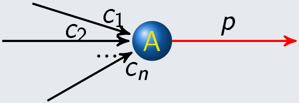
A FHE where $p(e)$ and $d(e)$ are integer multiples of c(e), or
A BHE where $c(e)$ and $d(e)$ are integer multiples of p(e)

Tighter Upper Bounds – Technique 2: Heavy Edges ...

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



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

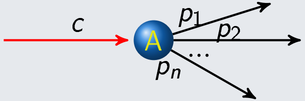


Regular Heavy Edge (RHE)

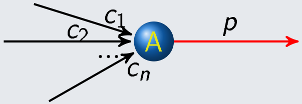
A FHE where $p(e)$ and $d(e)$ are integer multiples of c(e), or
A BHE where $c(e)$ and $d(e)$ are integer multiples of p(e)

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)

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 ....

Upper Bounds for Heavy Edges:

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

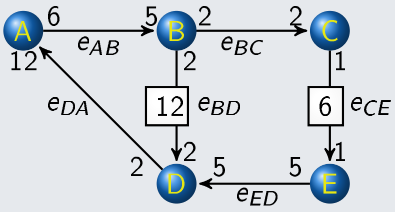
$$\max(p(e_f) + c(e_f), d(e_f)) + c(e_f)$$

- 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 .....

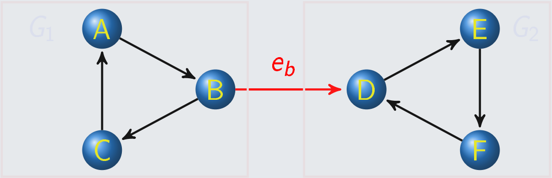


	e_{AB}	e_{BC}	e_{CE}	e_{BD}	e_{ED}	e_{DA}
Naive UB	30	12	6	12	30	60
Improved UB	16	2	6	12	5	32

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.

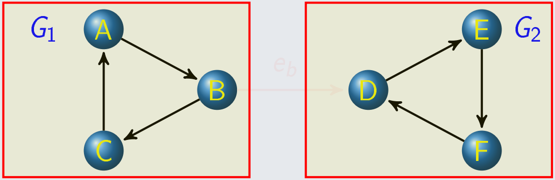


- Given known optimal schedules s_1 and s_2 for subgraphs G_1 and G_2 , we can get an optimal schedule s of G by
  - firing each node by following the known optimal schedules s_1 and s_2, and
  - firing the sink of the bridge e_b as soon as possible
- $R_{opt}(G) = R_{opt}(G_1) + R_{opt}(G_2) + LB(e_b)$

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.

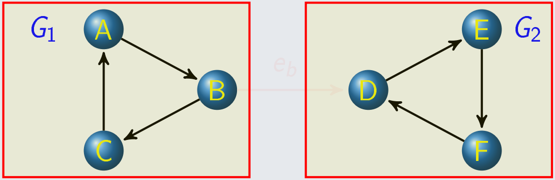


- Given known optimal schedules s_1 and s_2 for subgraphs G_1 and G_2 , we can get an optimal schedule s of G by
  - firing each node by following the known optimal schedules s_1 and s_2, and
  - firing the sink of the bridge e_b as soon as possible
- $R_{opt}(G) = R_{opt}(G_1) + R_{opt}(G_2) + LB(e_b)$

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.

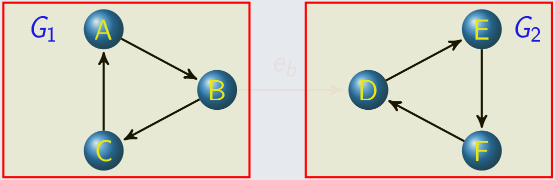


- Given known optimal schedules s_1 and s_2 for subgraphs G_1 and G_2 , we can get an optimal schedule s of G by
  - firing each node by following the known optimal schedules s_1 and s_2, and
  - firing the sink of the bridge e_b as soon as possible
- $R_{opt}(G) = R_{opt}(G_1) + R_{opt}(G_2) + LB(e_b)$

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.

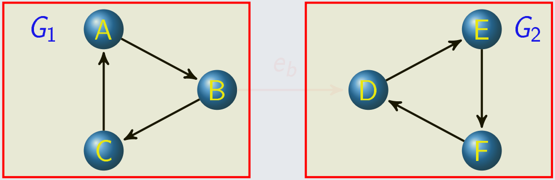


- Given known optimal schedules s_1 and s_2 for subgraphs G_1 and G_2 , we can get an optimal schedule s of G by
  - firing each node by following the known optimal schedules s_1 and s_2, and
  - firing the sink of the bridge e_b as soon as possible
- $R_{opt}(G) = R_{opt}(G_1) + R_{opt}(G_2) + LB(e_b)$

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

Outline

Performance

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

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

M. Geilen, S. Stuijk and T. Basten. SDF^3: SDF for free. ACSD 2006.

M. Geilen, T. Basten and S. Stuijk: Minimizing buffer requirements of synchronous dataflow graphs with model checking. DAC 2005.

Performance

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

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

M. Geilen, S. Stuijk and T. Basten. SDF^3: SDF for free. ACSD 2006.

M. Geilen, T. Basten and S. Stuijk: Minimizing buffer requirements of synchronous dataflow graphs with model checking. DAC 2005.

Performance

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

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

M. Geilen, S. Stuijk and T. Basten. SDF^3: SDF for free. ACSD 2006.

M. Geilen, T. Basten and S. Stuijk: Minimizing buffer requirements of synchronous dataflow graphs with model checking. DAC 2005.

Performance

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

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

M. Geilen, S. Stuijk and T. Basten. SDF^3: SDF for free. ACSD 2006.

M. Geilen, T. Basten and S. Stuijk: Minimizing buffer requirements of synchronous dataflow graphs with model checking. DAC 2005.

Outline

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