Work-in-Progress: A Unified Framework for Throughput Analysis of Synchronous Data Flow Graphs under Memory Constraints

Xue-Yang Zhu
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
Beijing, China 100190
zxy@ios.ac.cn

ABSTRACT

Streaming applications are often modeled with Synchronous data flow graphs (SDFGs). A proper analysis of the models is helpful to predict the performance of a system. In this paper, we focus on the throughput analysis of memory-constrained SDFGs (MC SDFGs), which needs to choose a memory abstraction that decides when the space of consumed data is released and when the required space is claimed. Different memory abstractions may lead to different achievable throughputs. The existing techniques, however, consider only a certain abstraction. If a model is implemented according to other abstractions, the analysis result may not truly evaluate the performance of the system. In this paper, we present a unified framework for throughput analysis of MC SDFGs for difference abstractions, aiming to provide evaluations matching up to the corresponding implementations.

ACM Reference format:
Xue-Yang Zhu. 2017. Work-in-Progress: A Unified Framework for Throughput Analysis of Synchronous Data Flow Graphs under Memory Constraints. In *Proceedings of CODES/ISSS '17 Companion, Seoul, Republic of Korea, October 15–20, 2017,* 2 pages.
DOI: 10.1145/3125502.3125535

1 INTRODUCTION

An important class of applications in embedded systems are streaming applications, which usually run under limited resources and are required to achieve a high throughput. *Synchronous data flow graphs* (SDFGs) [3] are widely used to model streaming applications. Each node (also called *actor*) in an SDFG represents a computation and each edge models a FIFO channel which carries streams of data. A *token* is an atomic data object. A *sample rate* is the number of tokens produced or consumed by an actor. The sample rates of actors in an SDFG may differ.

In the semantics of SDFGs, an actor consumes tokens from its input edges at the start of its firing and produces tokens on its output edges at the end of its firing. When memory constraints are concerned, an analysis need to choose an abstraction about when to release the buffer space of the consumed tokens and when to claim the space for the produced tokens. Many studies conducted to investigate storage aspects for SDFGs [4, 5] choose a conservative abstraction. They assume that for a firing of an actor, the buffer space needed by the produced tokens is claimed at the start of the firing and the buffer space of consumed tokens is released at the end of the firing. An implementation scheduled according to another scheme may achieve a higher throughput than the analysis result according to the conservative abstraction, however. The reason is that different memory abstractions may lead to different achievable throughputs of a system.

To match up to the implementation, we present a unified framework for throughput analysis of SDFGs to deal with different memory abstractions. To the best of our knowledge, this is the first work that can provide throughput evaluations under different memory abstractions in a unified framework.

2 PRELIMINARIES

An SDFG is a finite directed graph $G = (V, E)$. V is the set of actors, modeling the computations of a system. Actor v is weighted with its computation time $t(v)$. E is the set of directed edges, modeling interconnections between computations. The source actor and sink actor of edge e are denoted by $src(e)$ and $snk(e)$, respectively. Edge e is weighted with three properties, $d(e)$, $p(e)$ and $c(e)$. Property $d(e)$ is the number of initial tokens on e, $p(e)$ is the number of tokens produced onto e by each firing of $src(e)$, and $c(e)$ is the number of tokens consumed from e by each firing of $snk(e)$. The set of incoming edges to actor v is denoted by $InE(v)$, and the set of outgoing edges from v by $OutE(v)$.

SDFG G is *sample rate consistent* [3] if and only if there exists a positive integer vector $q(V)$ satisfying *balance equations*, $q(src(e)) \times p(e) = q(snk(e)) \times c(e)$ for all $e \in E$. The smallest such q is called the *repetition vector*. We use q to represent the repetition vector directly. An *iteration* of an SDFG is a firing sequence in which each actor v occurs exactly $q(v)$ times. The average computation time per iteration is called the *iteration period* (IP). The IP is the reciprocal of the *throughput*. We will use IP and throughput alternatively in the remained of the paper.

Definition 2.1. A memory constraint of SDFG $G = (V, E)$ is a vector $MC(E)$, in which $MC(e) \geq d(e)$. When $MC(e) > 0$, it defines the buffer bound of edge $e \in E$.

We use two variables $x \in \{0, 1\}$ and $y \in \{0, 1\}$ to model different abstractions. Variable x is used for the *release abstraction*. The buffer space is released at the start of a firing

CODES/ISSS '17 Companion, Seoul, Republic of Korea
© 2017 ACM. 978-1-4503-5185-0/17/10...$15.00
DOI: 10.1145/3125502.3125535

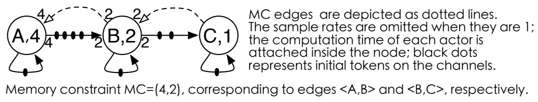
A,4
B,2
C,1
MC edges are depicted as dotted lines. The sample rates are omitted when they are 1; the computation time of each actor is attached inside the node; black dots represents initial tokens on the channels.
Memory constraint MC=(4,2), corresponding to edges <A,B> and <B,C>, respectively.

Figure 1: An MC SDFG.

when $x = 1$, at the end when $x = 0$. Variable y is used for the *claim abstraction*. The buffer space is claimed at the start of a firing when $y = 1$, at the end when $y = 0$. A *memory abstraction* is a combination of release abstraction and claim abstraction, denoted by the value of xy. For example, above-mentioned conservative abstraction is memory abstraction 01, representing that $x = 0$ and $y = 1$.

Let $IP_{xy}(G, MC)$ be the minimal achievable IP of SDFG $G = (V, E)$ under memory constraint $MC(E)$ based on the memory abstraction xy. Then $\frac{1}{IP_{xy}(G,MC)}$ is the maximal achievable throughput. The problems addressed in this paper are: given an SDFG G and a memory constraint MC, how to compute $IP_{xy}(G, MC)$ for $x \in \{0, 1\}$ and $y \in \{0, 1\}$.

3 THROUGHPUT ANALYSIS OF MEMORY CONSTRAINED SDFGS

The bound on buffer of an edge $\langle u, v\rangle$ of an SDFG can be modeled by adding edge $\langle v, u\rangle$ with tokens to model available storage space [4].

Definition 3.1. A memory-constrained SDFG (MC SDFG) of $G = (V, E)$ under $MC(E)$ is an SDFG $G_{MC} = (V, E \cup E_{MC})$, in which $E_{MC} = \{\langle v, u\rangle | \langle u, v\rangle \in E \wedge MC(\langle u, v\rangle) > 0\}$. For all $e' = \langle v, u\rangle \in E_{MC}$, there is an edge $e = \langle u, v\rangle \in E$, such that $p(e') = c(e)$, $c(e') = p(e)$ and $d(e') = MC(e) - d(e)$.

Edge $e' \in E_{MC}$ is called *MC edge* of corresponding $e \in E$. Tokens on e' model the available space. Denote $InE_{MC}(v)$ as the set of MC edges of $OutE(v)$ and $OutE_{MC}(v)$ as the set of MC edges of $InE(v)$. An MC SDFG with $MC = (4, 2)$ is shown in Fig. 1. Since there are already 4 and 2 tokens on edges $\langle A, B\rangle$ and $\langle B, C\rangle$, respectively, no tokens, which model buffer space, available on their corresponding MC edges.

Scheduling an SDFG under memory constraint is equivalent to scheduling the corresponding MC SDFG. An STE analysis method is used in our solutions. Instead of using a clock in the operational semantics of SDFGs as traditional STE analysis methods [4, 5], we use time stamps on tokens to model the time progress. Each token is tagged with a *time stamp* to indicate the time when it is produced. All initial tokens have a time stamp 0. There are different firing rules for different abstractions. Below we illustrate the basic ideas of proposal method via an example shown in Fig. 2, in which $fire_{xy}(v)$ indicates a firing of actor v based on abstraction xy. The firing starts at $v.st$ and ends at $v.et$. The memory constraint $MC = (4, 2)$. There are no constraints on the self-loop edges.

At state s, there are sufficient tokens on the incoming edges of actor B. That is, $d(\langle A, B\rangle) \geq c(\langle A, B\rangle)$ and $d(\langle C, B\rangle) \geq c(\langle C, B\rangle)$. Therefore actor B is ready for a firing. The time stamps of tokens produced on edges of $OutE(B)$ are not affected by the abstractions. They are always $B.et$, although the values of $B.et$ may differ with respect to claim abstraction y. The time stamps of tokens on $OutE_{MC}(B)$ are decided by the abstraction used.

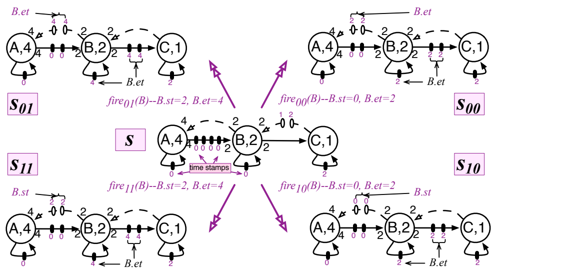
B.et
B.st
time stamps
s_{01}
s_{00}
s
s_{11}
s_{10}
$fire_{01}(B)$--B.st=2, B.et=4
$fire_{00}(B)$--B.st=0, B.et=2
$fire_{11}(B)$--B.st=2, B.et=4
$fire_{10}(B)$--B.st=0, B.et=2
A,4
B,2
C,1

Figure 2: The effects of firings under different abstractions.

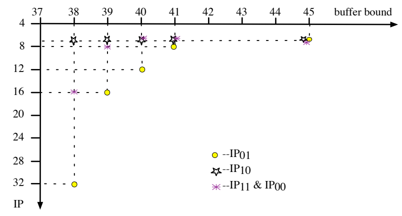

Figure 3: $IP_{xy}(modem, MC)$**s under abstractions** xy**.**

When the release time of consumed tokens is set to the end of the firing, i.e. $x = 0$, these time stamps are tagged with $B.et$, e.g. states s_{01} and s_{00}; otherwise, they are tagged with $B.st$, e.g. states s_{11} and s_{10}. Abstraction y affects the start time of a firing. When the required space is claimed at the start of the firing, i.e. $y = 1$, $B.st$ is set to be the largest time stamp of the required tokens for the firing, e.g. $fire_{01}$ and $fire_{11}$. Otherwise, the largest time stamp of the required tokens for the firing can be seen as the end time of the firing, i.e. $B.et$, then $B.st = B.et - t(B)$, e.g. $fire_{00}$ and $fire_{10}$.

Similar to the methods in [4, 5], the $IP_{xy}(G, MC)$ can be computed via the STE according to firing rules $fire_{xy}$. The IP_{xy}s of SDFG of a modem [1] are shown in Fig. 3. The IPs are computed under different MCs, which are storage distributions of the Pareto points obtained by the method in [4], which is under the abstraction 01.

We also consider the effects of the order of concurrent occurs of start and end of firings on the analysis results and the extension of the proposed technique to CSDF graphs [2], but omit them here because of space limitations.

4 ACKNOWLEDGMENTS

This work is supported in part by the National Natural Science Foundation of China (Nos. 61572478, 61472406 and 61472474).

REFERENCES

[1] SS Bhattacharyya, PK Murthy, and EA Lee. 1996. *Software synthesis from dataflow graphs*. Vol. 360. Springer.

[2] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. 1996. Cyclo-static dataflow. *IEEE Trans. on signal processing* 44, 2 (1996), 397–408.

[3] EA Lee and DG Messerschmitt. 1987. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. on Comput* 36, 1 (1987), 24–35.

[4] S. Stuijk, M. Geilen, and T. Basten. 2008. Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs. *IEEE Trans. on Computers* 57, 10 (2008), 1331–1345.

[5] XY Zhu, M Geilen, T Basten, and S Stuijk. 2014. Memory-Constrained Static Rate-Optimal Scheduling of Synchronous Dataflow Graphs via Retiming. In *Proc. of 17th Design, Automation and Test in Europe (DATE)*. 1–6.