Buffer allocation for real-time streaming on a multi-processor without back-pressure

Citation for published version (APA):

Document status and date:
Published: 01/01/2014

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Buffer Allocation for Real-Time Streaming on a Multi-Processor without Back-Pressure

Hrishikesh Salunkhe, Orlando Moreira and Kees van Berkel

ISSN 0926-4515

All rights reserved
editors: prof.dr. P.M.E. De Bra
prof.dr.ir. J.J. van Wijk

Reports are available at:
http://library.tue.nl/catalog/TUEPublication.csp?Language=dut&Type=ComputerScienceReports&Sort=Author&level=1 and
http://library.tue.nl/catalog/TUEPublication.csp?Language=dut&Type=ComputerScienceReports&Sort=Year&Level=1

Computer Science Reports 14-05
Eindhoven, September 2014
Buffer Allocation for Real-Time Streaming on a Multi-Processor without Back-Pressure

Hrishikesh Salunkhe*, Orlando Moreira†, Kees van Berkel*†
* Eindhoven University of Technology, The Netherlands; † Ericsson B.V. Eindhoven, The Netherlands
{h.l.salunkhe, c.h.v.berkel}@tue.nl; {orlando.moreira, kees.vanberkel}@ericsson.com

Abstract

The goal of buffer allocation for real-time streaming applications, modeled as dataflow graphs, is to minimize total memory consumption while reserving sufficient space for each production without overwriting any live tokens and guaranteeing the satisfaction of real-time constraints. We present a buffer allocation solution for dataflow graphs scheduled on a system without back-pressure.

Our contributions are 1) We extend the available dataflow techniques by applying best-case analysis. 2) We introduce dominator based relative life-time analysis. For our benchmark set, it exhibits up to 12% savings on memory consumption compared to traditional absolute life-time analysis. 3) We investigate the effect of variation in execution times on the buffer sizes for systems without back-pressure. It turns out that reducing the variation in execution times reduces the buffer sizes. 4) We compare the buffer allocation techniques for systems with and without back-pressure. For our benchmark set, we show that the system with back-pressure reduces the total memory consumption by as much as 30% compared to the system without back-pressure. Our benchmark set includes wireless communications and multimedia applications.

I. INTRODUCTION

Real-time streaming applications, such as multimedia streaming and wireless transceivers, are becoming increasingly complex. They have strict end-to-end latency and throughput requirements, and run continuously, processing virtually infinite input sequences in a pipelined manner. Since their performance must meet rigorous standards, they are often mapped onto Heterogeneous Multi-Processor (HMP) platforms, and both simulation and formal analytical techniques are used to verify the timing behavior.

Dataflow is a well-known temporal analysis and programming model which is well suited to model concurrent real-time streaming applications [11], [7], [21] In dataflow, an application is modeled as a directed graph, where nodes (actors) represent processing elements and edges represent data dependencies. In static variants of dataflow, bounds on actor execution times and the number of data items (tokens) consumed/produced on input/output edges for each execution (firing) of an actor are known at compile time, and for these variants, there exist techniques to verify real-time requirements such as deadlock-freedom and execution in bounded memory.

In dataflow, data is communicated through queues with First-In-First-Out (FIFO) behavior. The computation of the minimum amount of memory needed by FIFO queues (edges) of an application modeled as a dataflow graph such that it meets its real-time requirements, is called buffer sizing [13], [10], [23]. Often, execution platforms are equipped with a back-pressure mechanism. In such a platform, for an actor to be able to fire, not only a sufficient amount of tokens is required at each of its input edges, but also a sufficient amount of space is required to produce tokens on each of its output edges. However, on a platform without back-pressure, an actor can fire as soon as it has a sufficient amount of tokens on each of its input edges without checking space for output tokens. This can result in corruption if the producer fires before the token stored at that location is processed by the consumer. Since a back-pressured system assumes that the producer must wait for buffer availability, the existing buffer sizing techniques cannot be applied to platforms without back-pressure.

The architectures described in [14], [22], [8] do not support back-pressure. Generally, such platforms are equipped with several programmable processors and hardware accelerators. These accelerators are often weakly programmable, their communication interface allows them to read/write from/to buffers in a memory, but do not handle buffer management: they are simply given a memory address from/onto which to read/write, and will do so unconditionally.

One may wonder why such platforms are designed, since back-pressured systems offer safer execution from a functional perspective, at a cost, which in general-purpose platforms may sound negligible. However, and specially in the context of embedded real-time systems, back-pressure mechanisms are not without disadvantages: they consume chip area, complicate the design of the interface between accelerators and interconnect, and cause inter-processor communication and synchronization overheads [24], [22]. Also, when dealing with sampled signals (such as external sources), back-pressure cannot be applied, since the producer cannot be stopped. Moreover, the cyclic dependencies introduced by the back-pressure mechanism make timing analysis difficult for traditional real-time analysis techniques [6], and although dataflow analysis does not suffer from such limitations, most HMPs are not designed specifically with dataflow analysis in mind.

Therefore, these platforms require buffer allocation techniques which can handle non-back-pressured execution of dataflow.

1 In this paper, we propose buffer allocation techniques for dataflow graphs running in a self-timed manner (i.e. actors being...
activated by data availability) on an HMP without back-pressure. Please note that this paper is not trying to argue a choice for non-back-pressured systems. In fact, as we shall see, our results show that a non-back-pressured HMP will require larger buffers than a similar HMP with back-pressure, for the same throughput. But non-back-pressured HMPs do exist, and thus an efficient automated solution for computing buffer sizes for such platforms is necessary. Another contribution of this paper is a first comparison between buffer sizes for HMPs with and without back-pressure (Section VI), in cases where the temporal overhead of the back-pressure mechanism is negligible. We use Ericsson’s HMP modem platform [2] to show that for a given an HMP with back-pressure and another with similar architecture but without back-pressure, the HMP with back-pressure will always require smaller total memory expenditure on buffer allocation for the same throughput.

The rest of the paper is organized as follows. The need for a new buffer sizing technique is motivated in Section II. Our application model and hardware architecture are described in Section III. We then formalize the buffer allocation problem for systems without back-pressure in Section IV. In Section V, we study under which conditions finite buffer sizes exist for non-back-pressured systems. Buffer sizes for HMPs with and without back-pressure are compared in Section VI. We propose algorithms to approach the problem by introducing best-case temporal analysis of dataflow in Section VII. We introduce a novel dominator based relative life-time analysis for buffer sizing which is more accurate than traditional life-time analysis, and could have applications to other memory allocation problems, such as register allocation. To benchmark our technique, we use an LTE receiver [16] scheduled on our modem platform. Moreover, we also use a WLAN receiver [11], an MP3 Decoder [20] and an H263 Encoder [20] applications scheduled on an HMP platform (Section VIII). We discuss related work in Section IX and the paper concludes with Section X.

II. MOTIVATION

Single-Rate Dataflow (SRDF) is a static dataflow variant where an actor consumes and produces one token per firing. Figure 1a shows an example SRDF graph. A and B are connected by edge AB. The two other edges AA and BB, referred to as self-edges, hold an initial data token each, visualized by a black dot. Since there is a token already available on the input edge of A, it starts its first firing by consuming the token. A self-edge with one initial token forces its actor to fire once at a time i.e. there is no self-concurrency.

We assume that graph execution is strictly self-timed: actors fire immediately when a token is available on each of their inputs. Furthermore, we take the conservative assumption that a token resides in memory from the start of the actor firing that produces it to the end of the actor firing that consumes it.

![Graph without back-pressure](a) Graph without back-pressure ![Graph with back-pressure](b) Graph with back-pressure

Fig. 1: Back-pressure modeling

In SRDF semantics an edge has infinite capacity. However, in a realistic implementation, a buffer must have a finite size. This is typically imposed by Back-Pressure (BP). In hardware, back-pressure is realized using mechanisms such as blocking read/write and semaphore [22]. A back-pressured implementation of FIFO can easily be modeled in SRDF by introducing a back-edge as depicted in Figure 1b. For every finite back-pressured FIFO buffer, an edge is added from its consumer to its producer with a number of initial tokens equivalent to the capacity of the buffer. As shown in Figure 1b, the FIFO buffer between A and B is dimensioned to hold at most two tokens. In terms of SRDF execution, we see that after A fires the first two times, consuming the two initial tokens in BA and producing two tokens on AB, its third firing can only happen once B has finished its first firing and thus producing a token on BA which represents a release of space in the back-pressured implementation. Thus back-edge BA regulates the firings of A, and guarantees that there can only be at most two tokens in edge AB at any point in time.

We plot a Gantt chart for Self-Timed Schedule (STS) of this graph assuming Worst-Case Execution Time (WCET) of 2 units for both actors, as shown in Figure 2a. Each position in the FIFO is represented by a rectangle. The token stored in each FIFO position is indicated by the name of its producer and its firing count. The buffer size per edge is given by the number of rectangles per FIFO. Actor firings are also annotated by their firing counts. Note that, in this case, for the same execution times the execution without back-pressure would yield the same schedule, and the same buffer occupations.
When applying back-pressure, as shown in Figure 2b, if the second firing of A executes faster than its WCET, the third firing must wait for a token on BA which is produced by the first firing of B later in time. The buffer size thus obtained is valid for any STS with Varying Execution Times (VET) including the STS with WCET [10]. Any self-timed execution will have at least as many firings of each actor per time interval as in the STS with WCETs. This considerably simplifies traditional dataflow analysis, as local WCETs correspond to worst-case temporal behavior of the graph as a whole. However, if the same temporal variation occurs without back-pressure, the result is as shown in Figure 2c. A starts its third firing, denoted as \((A, 3)\), immediately after it finishes the second firing, requiring either a larger buffer space, or overwriting a previously produced token prematurely, thus leading to incorrect functional behavior.

Another fundamental limitation is that, even for fixed execution times, a graph without back-pressure is not guaranteed to execute in bounded buffer space. If the execution times of A and B are constant, but equal to 1 and 2 respectively, then the self-timed, non-back-pressured execution of the graph will require a buffer of ever increasing size. Thus, before designing algorithms to allocate bounded buffers, one must define under which conditions do such bounded buffers exist.

### III. Preliminaries

#### A. Application model and properties

We model real-time streaming applications using Time-Bounded SRDF (Tb-SRDF), which is an SRDF graph extended with Best-Case Execution Time (BCET) and WCET per actor. A Tb-SRDF graph \(G\) is denoted by \((V, E, d, \hat{\tau}, \bar{\tau})\), where \(V\) is the set of vertices of \(G\), \(E\) is the set of edges of \(G\). \(d\) is a valuation \(d : E \rightarrow \mathbb{N}_0\), such that \(d(i,j)\) is the number of initial tokens in edge \((i,j)\) where \(i,j \in V\). Valuations \(\tau, \hat{\tau}, \bar{\tau} : V \rightarrow \mathbb{N}_0\) give BCET and WCET of an actor respectively. The timing function is a valuation \(\tau : V \times \mathbb{N}_0 \rightarrow \mathbb{N}_0\), such that \(\tau(i,k)\) is the execution time for the \((k+1)\)th firing of \(i\), and \(\tau(i,k)\) for all \(i \in V, k \in \mathbb{N}_0\) is bounded from both sides by \(\hat{\tau}(i)\) and \(\bar{\tau}(i)\) respectively: \(\hat{\tau}(i) \leq \tau(i,k) \leq \bar{\tau}(i)\). We write \(\hat{\tau} \preceq \tau \preceq \bar{\tau}\) if \(\forall i \in V, \tau(i) \leq \bar{\tau}(i)\).

A Self-Timed Schedule (STS) of a Tb-SRDF graph is a schedule where each actor firing starts immediately if there are enough tokens on all of its input edges. In an STS with \(\tau : \hat{\tau} \preceq \tau \preceq \bar{\tau}\), the start times of actors occur as soon as all precedence constraints, as shown in Equation 1, are met. Valuation \(s : V \times \mathbb{N}_0 \rightarrow \mathbb{N}_0\) is such that \(s(i,k)\) gives the start time of the \((k+1)\)th firing of \(i\).

\[
s(i,k) = \max_{(x,i) \in E} \begin{cases} s(x,k-d(x,i)) + \tau(x,k-d(x,i)), & k \geq d(x,i) \\ 0, & k < d(x,i) \end{cases}
\]  

\(G\) produces a unique STS for given \(\tau\) where the start time of every actor \(i\) in the \((k+1)\)th firing is given by Equation 2.
\[ s(\tau, i, k) = \max_{(x, i) \in E} \begin{cases} s(\tau, x, k - d(x, i)) + \tau(x), & k \geq d(x, i) \\ 0, & k < d(x, i) \end{cases} \] (2)

An STS for a given \( \tau \) for a strongly-connected (i.e. there exists a directed path from any actor to any other actor) graph \( G \) has a property [1] that after a finite number of iterations \( K(\tau, G) \) (referred to as Transient Phase), it reaches a periodic regime and is characterized by the following Equation:

\[ s(\tau, i, k + N(G)) = s(\tau, i, k) + N(G) \mu(\tau, G), k > K(\tau, G) \] (3)

This periodic regime has a period of \( N(G) \mu(\tau, G) \) units. \( N(G) \) is the cyclicity of the graph [1] and is determined by the sum of delays on the graph’s critical cycle. \( \mu(\tau, G) \) is the Maximum Cycle Mean (MCM) and is defined as

\[ \mu(\tau, G) = \max_{c \in C(G)} \sum_{c} \tau(i) \] (4)

where \( C(G) \) is the set of all the cycles in \( G \). The inverse of \( \mu(\tau, G) \) gives the maximum attainable throughput of graph \( G \) for \( \tau \). The transient phase \( K(\tau, G) \) can be computed by simulating the execution of \( G \) and detecting the first periodic iteration. Another known method for calculating \( K(\tau, G) \) is presented in [1]. For fixed \( \tau \), we will use \( \mu(G) \) and \( K(G) \) notations instead of \( \mu(\tau, G) \) and \( K(\tau, G) \) respectively.

STSs given by \( \hat{\tau} \) and \( \hat{\tau} \) are termed as Best-Case STS (BC-STS) and Worst-Case STS (WC-STS) respectively. Start times for BC-STS and WC-STS can be obtained by replacing \( \tau \) with \( \hat{\tau} \) and \( \hat{\tau} \) in Equation 2 respectively. For a given graph \( G \), for the sake of brevity, we denote \( s(\hat{\tau}, i, k), s(\hat{\tau}, i, k) \) and \( \hat{s}(\tau, i, k) \) by \( s(i, k), \hat{s}(i, k) \) and \( \hat{s}(\tau, i, k) \) respectively. The finish time of the \((k+1)^{th}\) firing of \( i \) is defined as \( f(i, k) = s(i, k) + \tau(i, k) \). We assume FIFO ordering of actor firings i.e. for every edge, the tokens are consumed in the same order as they are produced. This forces actors to fire in the same order as graph iteration i.e. the first firing of an actor will occur in the first graph iteration, the second firing occurs in the second graph iteration and so on.

B. Cyclo-periodic source

![Cyclo-periodic source](image)

Fig. 3: Cyclo-periodic source

Real-time streaming applications such as LTE have different processing modes for different input symbols (refer Appendix A and A-A). Moreover, the symbol arrival period is constant across different processing modes. Each processing mode has a static repeating pattern. We model the temporal behavior of such a source, termed as Cyclo-Periodic Source (CPS), as a subset of the graph actors having constant execution time as shown in Figure 3. These source actors \( \{S_1, S_2, ..., S_n\} \) are connected one to another through edges with zero delay, except the edge which connects the last source \( S_n \) to the first source \( S_1 \) with exactly one delay, forming a cycle with a total delay sum of 1, which we refer to as Source Cycle. Such a graph must have \( \mu(\hat{\tau}, G) \) and \( \mu(\hat{\tau}, G) \) equal to the cycle mean of the source cycle. This condition enables the graph to keep up with the rate of the source, which is required for a realistic application. Furthermore, we require that there is a directed path to every actor through delay-less edges from a source actor present in the source cycle. This is typical in real-life applications, where each task must depend on input data received at run-time.

C. Hardware architecture

The hardware architecture of Ericsson’s modem platform is shown in Figure 4. All the processing elements are connected through a ring-bus. A processing element can be an Embedded Vector Processor (EVP) [2], general purpose processor such as ARM or a programmable accelerator. When a processing element receives a control packet configured for itself, it executes consuming the packet. It fetches its input data from the shared memory at the start of execution, writes output data and creates a new control packet which is then passed to the next PE and so on.
∀
must be respected:

consumption denoted as

max (returns false:

Overlap

z:

occupies a location in memory which is characterized by its starting memory address,

SA

µ

of the example SRDF graph. From the example graph and its component graph, we know that

SCC

governed by Equation 3 is not guaranteed. Suppose that

back-pressure. When back-pressure is not modeled in a dataflow graph, strong connectedness and thereby the periodic behavior

graph

G

Acyclic Graph (DAG) [4]. Since each node in

There is an edge

(v

i

, v

j

) = max ( ∀e∈E ∃k∈N0 SA(e, k) + z(e)). When computing buffer allocation, any overlapping constraint must be respected: ∀e,f∈E ∃k,l∈N0 (Overlap(e, k, f, l) = True) ⇔ (SA(e, k) + z(e) ≤ SA(f, l)).

V. BUFFER BOUNDEDNESS

This section introduces the properties under which bounded buffers exist for a self-timed execution of a system without back-pressure. When back-pressure is not modeled in a dataflow graph, strong connectedness and thereby the periodic behavior governed by Equation 3 is not guaranteed. Suppose that SCC

G

is a set of Strongly Connected Components (SCCs) [4] of a graph G, then the component graph G

SCC

is (V

SCC

, E

SCC

) where the vertex set V

SCC

contains v

i

for each C

i

in SCC

G

. There is an edge (v

i

, v

j

) ∈ E

SCC

if G contains an edge (x, y) for some x ∈ C

i

and y ∈ C

j

[4]. As a result, G

SCC

is a Directed Acyclic Graph (DAG) [4]. Since each node in G

SCC

is an SRDF graph, MCM (µ) of each such node can be computed.

Consider an SRDF graph shown in Figure 5a, every actor is annotated with its WCET. Figure 5b shows a component graph of the example SRDF graph. From the example graph and its component graph, we know that µ(V

S

) = 8, µ(V

X

) = µ(V

Y

) = 6. Consider an edge (V

S

, V

X

); if µ(V

S

) > µ(V

X

) then eventually V

S

will force V

X

to run as slow as V

S

i.e. V

S

will induce µ(V

S

) on V

X

. Similarly, for a component graph, each v

j

∈ V

SCC

will eventually be forced to adapt to the µ of the slowest of its predecessor nodes including itself. Equation 5 gives the induced µ i.e. µ

SCC

for every v

j

∈ V

SCC

of a component graph G

SCC

.

\[
\mu^{idc}(v_j) = \max(\mu(v_j), \max_{(v_i,v_j) \in E_{SCC}} \mu^{idc}(v_i))
\]

(5)

It is shown in [18] that an STS of a non-strongly connected dataflow graph, after its transient phase, settles onto a periodic regime provided that actors have constant execution times across iterations and such behavior is characterized as follows:
where \( v_j \in V_{SCC} \) and \( i \in C_j \). For every \( v_i \), \( \mu^{idc}(v_i) \) is calculated using Equation 5. Equation 6, however, does not guarantee buffer boundedness. If \( v_j \) has larger \( \mu \) than its predecessors then it will induce its own \( \mu \) and thereby its own periodicity \( N(G), \mu^{idc}(v_j) \). However, this case will lead to accumulation of an ever increasing number of tokens on all input edges of \( v_j \). Therefore, buffer boundedness is guaranteed if \( \forall_{(v_i,v_j) \in E_{SCC}}, \mu^{idc}(v_i) \geq \mu^{idc}(v_j) \). Source node \( v_s \) of \( G_{SCC} \) induces its own \( \mu \) i.e. \( \mu^{idc}(v_s) = \mu(v_s) \). \( \geq \) is a transitive relation and hence transitive closure of \( \geq \) on \( G_{SCC} \) with \( \forall_{v_i \in V_{SCC}} \mu^{idc}(v_i) \) gives one of the important conditions for the bounded buffer: \( \mu^{idc}(v_s) \geq \forall_{v_i \in V_{SCC}} \mu^{idc}(v_i) \). Since \( \forall_{v_i \in V_{SCC}} \mu^{idc}(v_i) \geq \mu(v_i) \) then it follows that \( \mu(v_s) \geq \forall_{v_i \in V_{SCC}} \mu(v_i) \).

Our analysis considers a graph \( G \) having a cyclo-periodic source \( v_{SRC} \), as described in Section III-B, consisting of actors with constant execution times and having strictly higher \( \mu \) than any other \( v_i \in V_{SCC} \) for all \( \tau : \bar{\tau} \leq \tau \leq \hat{\tau} \). Moreover, \( v_{SRC} \) dominates all other \( v_i \) in \( G_{SCC} \). We say that a node \( d \) dominates \( i \), if every possible execution path from source node to \( i \) includes \( d \) [12]. For such a graph, \( \mu(v_{SRC}) \geq \forall_{v_i \in V_{SCC}} \mu(v_i) \) condition holds, hence \( \mu(G) = \mu(\bar{\tau}, G) = \mu(\hat{\tau}, G) = \mu(v_{SRC}) \). Since, \( \mu(v_{SRC}) \) is strictly higher than any other \( \mu(v_i) \) and the critical cycle of \( v_{SRC} \) has a single delay, \( N(G) \) is 1. For such graph \( G \), any STS of \( G \), after its transient phase, will settle onto a periodic regime provided that actors have constant execution times across iterations. It is characterized as follows:

\[
s(i, k + 1) = s(i, k) + \mu(G), k > K(G)
\]  

(7)

![Fig. 6: Example SRDF graph](image)

![Fig. 7: Bounds on actor start times](image)

For any Tb-SRDF graph \( G \), the start time of every actor in any STS of \( G \) is bounded from below and above by its BC-STS and its WC-STS respectively. Figure 7 shows BC-STS, WC-STS and STS with varying execution times for the example graph shown in Figure 6, where \( \bar{\tau}(A) = \bar{\tau}(B) = 1 \). As we can see, start time of an actor in an iteration is always bounded from below and above by the start time of the same actor in the same iteration belonging to its BC-STS and its WC-STS respectively. Lower (blue line) and upper (red line) bounds on the start times of \( B \) are shown in Figure 7. This is proven in [11] using the monotonicity principle (Equation 8): if the execution time of an actor increases from \( \tau_1 \) to \( \tau_2 \), it can only cause subsequent firings of actors to happen later or at the same time than in STS with \( \tau_1 \).

\[
s(\tau_1, i, k) \leq s(\tau_2, i, k)
\]  

(8)
Theorem 1: (Bounds on actor start times) Given a Tb-SRDF graph $G = (V, E, d, \tau, \hat{\tau})$ with $\tau \leq \tau \leq \hat{\tau}$. Then for any $i \in V$ and $k \geq 0$, it holds that $s(\hat{\tau}, i, k) \leq s(\tau, i, k) \leq s(\tau, i, k)$ i.e. $s(i, k) \leq s(i, k) \leq \hat{s}(i, k)$.

Proof: From Equation 8, we can prove for any $\tau$ that $\hat{s}(i, k) \leq s(i, k)$ and $s(i, k) \leq \hat{s}(i, k)$. Hence, $s(i, k) \leq s(i, k) \leq \hat{s}(i, k)$.

Let’s assume an STS with $\tau$. Recall from Section III-A that we assume FIFO ordering. Therefore, for an edge $(i, j)$ with $d(i, j)$ initial tokens, in the $k^{th}$ iteration, all the tokens produced by $i$ belonging to iterations $p < k$ have been consumed. Therefore the total number of tokens on an edge $(i, j)$ in the $k^{th}$ iteration is the total number of firings of $i$ occurring in the interval $[s(i, k), f(j, k + d(i, j))]$, given as:

$$b(\tau, i, j, k) = \sum_{k' = 0}^{\infty} \left\{ \begin{array}{ll}
1, & s(i, k) \leq s(i, k') \leq f(j, k + d(i, j)) \\
0, & \text{otherwise}
\end{array} \right. \quad (9)$$

For simplicity, we assume token size for all edges to be 1 unit i.e. $\forall (i, j) \in E z(i, j) = 1$. The total buffer size for an edge $(i, j)$ will be the maximum buffer size needed across all iterations, and is given by Equation 10.

$$b(\tau, i, j) = \max_{k \in \mathbb{N}_0} b(\tau, i, j, k) \quad (10)$$

Definition 1: $B(\tau)$ is the total buffer size needed for an STS of a graph with $\tau$ and it is given by

$$B(\tau) = \sum_{(i, j) \in E} b(\tau, i, j) \quad (11)$$

Definition 2: $B(\hat{\tau})$ is the total buffer size needed for the WC-STS of a graph and it is given by

$$B(\hat{\tau}) = \sum_{(i, j) \in E} b(\hat{\tau}, i, j) \quad (12)$$

The total buffer size needed for an edge $(i, j)$ for non-back-pressured execution will be the maximum of the buffer sizes needed for an edge $(i, j)$ among all the STSs with $\tau: \tau \leq \tau \leq \hat{\tau}$.

$$\hat{b}(i, j) = \max_{\tau \leq \tau \leq \hat{\tau}} b(\tau, i, j) \quad (13)$$

Definition 3: $\hat{B}$ is the total buffer size needed for non-back-pressed execution of a graph and it is given by

$$\hat{B} = \sum_{(i, j) \in E} \hat{b}(i, j) \quad (14)$$

Equation 14 provides buffer sizes for non-back-pressed execution where buffers per FIFO edge are independent of each other. This buffer size is optimal in the sense that for correct execution of any STS with $\tau \leq \tau \leq \hat{\tau}$, Equation 14 gives the minimum amount of memory required.

The bound on the buffer sizes needed for non-back-pressed execution is obtained when the interval $[s(i, k), f(j, k + d(i, j)))]$ is maximum. The maximum interval is obtained using the lower bound on $s(i, k)$ and the upper bound on $f(j, k + d(i, j))$. Moreover, the maximum buffer size is equivalent to the maximum number of firings of $i$ during the interval $[s(i, k), f(j, k + d(i, j))]$, $\forall k \in \mathbb{N}_0$. Theorem 1 infers that the maximum number of firings of $i$ occur in the BC-STS. Hence, the bound on the buffer size for an edge $(i, j)$ is given by:

$$\hat{b}(i, j) \leq \max_{k \in \mathbb{N}_0} \sum_{k' = 0}^{\infty} \left\{ \begin{array}{ll}
1, & \hat{s}(i, k) \leq \hat{s}(i, k') \leq \hat{f}(j, k + d(i, j)) \\
0, & \text{otherwise}
\end{array} \right. \quad (15)$$

For $G$ and $k > \max(K(\hat{\tau}, G), K(\hat{\tau}, G))$, the BC and the WC-STS are periodic; hence $i$ will have finite number of firings in the interval under the BC-STS. Therefore buffer sizes for any STS of $G$ are bounded. Equation 16 provides the bound on the buffer sizes for a Tb-SRDF graph $G$ scheduled on a non-back-pressed platform.

$$\hat{B} \leq \sum_{(i, j) \in E} \max_{k \in \mathbb{N}_0} \sum_{k' = 0}^{\infty} \left\{ \begin{array}{ll}
1, & \hat{s}(i, k) \leq \hat{s}(i, k') \leq \hat{f}(j, k + d(i, j)) \\
0, & \text{otherwise}
\end{array} \right. \quad (16)$$
VI. BACK-PRESSURE VS. NO BACK-PRESSURE

Dataflow graphs modeling back-pressure are equipped with back edges to regulate token productions and thereby to reduce buffer sizes by delaying actor firings if required. This implies that when back-pressure is supported, buffer sizes can be reduced compared to the buffer sizes needed when back-pressure is not supported. The WC-STS of a graph is always rate-optimal, i.e. the WC-STS always realizes the maximal possible throughput [11]. We assume back-pressured systems that are rate-optimal and have minimum buffer sizes among such rate-optimal schedules. Lemma 1 states that the buffer sizes for non-back-pressured the WC-STS always realizes the maximal possible throughput [11]. We assume back-pressured systems that are rate-optimal.

Lemma 1: Given a Tb-SRDF graph \( G = (V, E, d, \tau, \tilde{\tau}) \) with \( \bar{B} \) and \( B(\tilde{\tau}) \). Then \( \bar{B} \geq B(\tilde{\tau}) \)

Proof: Recall Equation 13, as can be seen, \( b(i,j) \) is as low as \( b(\tilde{\tau}, i,j) \) or higher. Hence, \( b(i,j) \geq b(\tilde{\tau}, i,j) \). Thereby, it is proven that \( \bar{B} \geq B(\tilde{\tau}) \).

Definition 4: \( \bar{B} \) is the minimum amount of buffer size needed for the rate-optimal execution of a system with back-pressure.

![Fig. 8: Capacity constrained model](image)

It can easily be proved that \( B(\tilde{\tau}) \geq \bar{B} \). It is already proven in [10] that the buffer sizes obtained for the WC-STS will be valid for the rate-optimal execution of a back-pressed system. We rephrase the proof provided in [10] retaining relevant details. The capacity-constrained model \( G_{ccm} \) of an SRDF graph \( G \) with the total buffer size \( B(\tau) \) is itself an SRDF graph that is obtained from \( G \) by adding, for every edge \((i,j)\) a reverse edge \((j,i)\) with \( b(i,j) - d(i,j) \) initial tokens. Given an edge \((i,j)\), a reverse edge \((j,i)\) in \( G_{ccm} \) captures precisely the remaining buffer space for edge \((i,j)\). The start of a firing of \( i \) claims space by consuming a token from edge \((j,i)\); the end of a firing of actor \( j \) releases space in the buffer of \((i,j)\) by producing a token on \((j,i)\). This is in line with the conservative token production/consumption assumptions. Figure 8 shows a \( G_{ccm} \) derived from the SRDF graph shown in Figure 6, with \( b(\tau,A,B) = 1 \) and \( b(\tau,S,A) = 2 \) tokens respectively.

Lemma 2: Given a Tb-SRDF graph \( G = (V, E, d, \tilde{\tau}, \tilde{\tau}) \) with \( B(\tilde{\tau}) \) and \( G_{ccm} \) with buffer sizes \( B(\tilde{\tau}) \), then \( G_{ccm} \) has a rate-optimal schedule.

Proof: As a result of construction of \( G_{ccm} \) that uses buffer space \( B(\tilde{\tau}) \), is also a schedule of \( G_{ccm} \). The WC-STS of \( G \) is always rate-optimal [10]. Therefore, \( G_{ccm} \) has a rate-optimal schedule.

With varying execution times in back-pressured systems, firings will be delayed because of the unavailability of tokens in some back edges. In back-pressured systems, buffer sizes can be reduced by dimensioning them such that if possible, certain actor firings are delayed keeping the schedule rate-optimal. Figure 9 describes the scenario where buffer sizes can be reduced in case of back-pressured systems.

Figure 9a and 9c are SRDF graphs without and with back-pressure. Each actor is annotated with its name and WCET. Note that each actor has a self-edge with one initial token which is not shown in the graphs. Figure 9b shows the Gantt chart of the WC-STS of the non-back-pressed graph shown in Figure 9a. From the Gantt chart, we can see that every edge needs to store some back edges. In back-pressured systems, buffer sizes can be reduced by dimensioning them such that if possible, certain actor firings are delayed keeping the schedule rate-optimal. The example shown in Figure 9 shows the WC-STS of the back-pressured graph shown in Figure 9c. We can see that the initial token present on edge \( BC \) forces all the firings of \( C \) to happen later compared to the WC-STS of the non-back-pressed graph. This reduces the buffer size needed by edge \( CB \) from 2 to 1 tokens and still back-pressured schedule realizes the maximal throughput. The example shown in Figure 9 and the example shown in Section II demonstrate that the buffer sizes obtained by employing back-pressure can be smaller than or equal to the buffer sizes obtained for non-back-pressed system i.e. \( B(\tilde{\tau}) \geq \bar{B} \).

Theorem 2: Given a Tb-SRDF graph \( G = (V, E, d, \tilde{\tau}, \tilde{\tau}) \) with \( \bar{B} \) and \( B \). Then \( \bar{B} \geq B \)

Proof: From Lemma 1, it is proven that \( \bar{B} \geq B(\tilde{\tau}) \). Moreover, along with Lemma 2, the example shown in Figure 9 shows that \( B(\tilde{\tau}) \geq \bar{B} \). Hence, \( \bar{B} \geq B \).

We can infer several observations from the relationship we obtained between the buffer sizes for back-pressed and non-back-pressed systems. Back-pressure systems allow smaller buffer sizes compared to non-back-pressed systems. Back-pressure minimizes the life-times of tokens by delaying actors firings and retaining the maximal throughput which cannot be achieved with self-timed execution (no back-pressure) where actors fire as soon as they are enabled.
According to Equation 14, the buffer size for each edge is the maximum of the buffer sizes calculated for that edge for all possible STSs generated using every possible combination of $\tilde{\tau} \leq \tau \leq \hat{\tau}$. This makes the algorithm based on exhaustive solution space exploration combinatorially exponential. Therefore we compute sufficient buffer sizes using the BC-STS and the WC-STS bounds. Buffer sizes are computed using simulation. Simulation, in this context, means that we imitate the execution of a graph with a self-timed execution and thus obtain the start time of actors. It is evident from Equation 15 that we need to carry out simulations for the BC-STS and the WC-STS separately. It is sufficient to carry out the simulation until the end of the second periodic iteration, since, all the tokens produced in the first periodic iteration will have been consumed in the second periodic iteration. Moreover, for each edge $(i, j)$, for the token consumed by $j$ in the second iteration, all the productions (i.e. firings of $i$) overlapping its life-time will be recorded.

### A. Sharing policy

Figure 10 shows a simplified SRDF graph of the processing of a typical radio application such as WLAN [11]. At the start of a radio frame, the source sends a synchronization signal. After synchronization, a header is sent which is then used to process payloads in the current frame. For the next $k$ firings of the source, $k$ payloads are sent. The token sizes for all the edges from $Src$ to their respective processing actors are the same. As we can see, producers of these edges i.e. $Src_1$, $Src_2$,.. $Src_n$ always fire in the same static order and consumers i.e. $Sync$, $Header$, ..$Payload_k$ also fire in the same static order. Note that the activities of the producers and consumers of these edges may be different. This implies that the productions and consumptions of tokens, on these edges, are statically ordered i.e. tokens from these edges are consumed in the same order as they are produced, which mimics FIFO behavior. Hence, we can implement these edges using a single FIFO by implementing circular buffers. We call this sharing policy as FIFO ordered buffer sharing.

This type of behavior also arises from graph transformations such as graph unrolling. Graph unrolling is a graph transformation that alters a graph so that, for a positive integer $f$, $f$ consecutive iterations are (visible simultaneously) turned as a cycle in the unrolled graph. For example, a graph shown in Figure 14 is 3 times unrolled version of the graph shown in Figure 11. More details on graph unrolling are presented in Section VII-B2. Transformations of Synchronous Dataflow (SDF) or Cyclo-Static Dataflow (CSDF) graph to an SRDF graph are special case of graph unrolling transformations. SDF is a generalization of SRDF.
which allows actors to consume and produce fixed amounts of tokens per firing. In CSDF, a generalization of SDF, actors have predefined cyclically repeating firing rules i.e. for each firing the number of produced and consumed tokens is known and follow a cyclic pattern. Our FIFO ordered buffer sharing policy is more generalized than previously proposed techniques for SDF and CSDF buffer sizing; e.g. in SDF or CSDF, edge \((\text{Src}_1, \text{Sync})\) and \((\text{Src}_2, \text{Header})\), in Figure 10, cannot be modeled as a single FIFO since their consumers have different activities.

Let’s assume a set of circular buffers \(CB\). Each \(C_i \in CB\) is shared by a set of FIFO edges \(E_i \subseteq E\) based on life-time analysis: if the life-time of two or more edges do not overlap then they share the same circular buffer and its size is given by the FIFO edge \((i,j) \in E_i\) which has the maximum buffer size. Therefore, buffer size for a Tb-SRDF graph is the sum of buffer sizes of all circular buffers as shown in Equation 17.

\[
\tilde{B} = \sum_{C_i \in CB} \max_{(i,j) \in E_i} \tilde{b}(i,j)
\]

**B. Life-time Analysis**

Life-Time analysis (LTA) is not new to buffer sizing and has been employed previously [13]. For any token on any edge \((i,j)\), its life-time starts when it is produced by \(i\) and ends when it is consumed by \(j\). Therefore token production and consumption times are given by the start time of its producer and finish time of its consumer respectively. For every edge \((i,j)\), the maximum number of overlapping token life-times give the buffer size for edge \((i,j)\).

We will use a Tb-SRDF graph example shown in Figure 11. Let’s assume \(\tilde{\tau} = 2\) and \(\hat{\tau} = 3\) for all the actors except \(A\), which has \(\tilde{\tau} = 1\) and \(\hat{\tau} = 5\).

1) **Absolute Life-time Analysis:** In case of varying execution times, maximum possible life-time of a token is given by its earliest production time and its latest consumption time. The earliest production time of a token is given by the start time of its producer from the BC-STS and its latest consumption time is given by the finish time of its consumer from the WC-STS. Figure 12 shows the life-times computed using the BC-STS and WC-STS of the Tb-SRDF example graph from Figure 11. For the sake of simplicity, we consider only the first iteration of the graph. Figure 12 shows the conventional way of computing life-times which we term as absolute life-times. However, absolute LTA has not been employed for buffer allocation for non-back-pressured systems before this work. As shown in Figure 12, life-times of all the tokens overlap with each other and hence sharing among tokens of any edges is not possible.

2) **Relative Life-time analysis:** Let’s consider the topology of the graph shown in Figure 11, all the edges to \(C, D, E\) and \(F\) go through \(B\) which means execution of \(C, D, E\) and \(F\) is always dependent on \(B\). Any variation or shift in start or execution time of \(B\) will shift the start times of \(C, D, E\) and \(F\). Thus, if we compute best-case production time (source actor start time) and worst-case consumption time (sink actor finish time) of \(C, D, E\) and \(F\) with respect to \(B\) instead of \(A\) then we can obtain more accurate life-times.

Figure 13 shows the life-times computed using the BC-STS and WC-STS with respect to \(B\) i.e. start times of \(C, D, E\) and \(F\) are computed from their relative common source \(B\). We term these life-times as relative life-times. From Figure 13, it can easily be inferred that the life-time of \(BC\) does not conflict with \(EF\), allowing sharing between \(BC\) and \(EF\) which was not
achievable using absolute LTA. This infers that the relative LTA is more accurate than the absolute LTA and can lead to more sharing possibilities.

The determination of the relative common source for an actor pair is carried out using dominators based analysis [12]. We say that a node $d$ dominates $i$ (shown as $d \text{ dom } i$), if every possible execution path from source node to $i$ includes $d$. The common dominator is a relation, $\text{CoDom} : V \times V \rightarrow V$, $\text{CoDom}(a,b) = c \leftrightarrow c \text{ dom } a \land c \text{ dom } b$. The Closest Common Dominator (CCD) is a relation $\text{CCD} : V \times V \rightarrow V$, $\text{CCD}(a,b) = c \leftrightarrow c = \text{CoDom}(a,b) \land (\neg \exists d : d = \text{CoDom}(a,b) \land c \text{ dom } d)$. The (closest) relative common source and the CCD for an actor pair are equivalent.

For a token pair, if the consumer actor of one token finishes before the producer actor of the other token fires then their life-times do not conflict. Relative LTA can let us calculate these timings more accurately. We obtain the CCD of the pair formed by the consumer actor of one token and the producer actor of another token. We then compute relative production and consumption times with respect to their CCD and perform relative LTA. To find the CCD of the firings of actors belonging to different iterations, inter-iteration dependencies have to be considered. These dependencies are exposed by graph unrolling. When we unroll a graph by $u$ times ($u \geq 1$), we create $u$ copies of each actor, having same properties as their parent actor, in the unrolled graph $G_u(V_u,E_u)$. The executional behavior of the original and its unrolled graph are similar i.e. the $k^{th}$ ($k \in \mathbb{N}_0$) firing of the $i^{th}$ ($0 \leq i < u$) copy of an actor $a$ in the unrolled graph is equivalent to the $(k.u + i)^{th}$ iteration of that actor in original graph. Therefore simulating a graph $G$ till $n$ iterations and executing unrolled graph $G_n$ ($n$ times unrolled graph) are equivalent in terms of actor start times.

Figure 14 shows a ($u = 3$) unrolled graph $G_3$ of the example graph shown in Figure 11. For our buffer calculation, we carry out simulation till the end of the second periodic iteration. That means we can find out the maximum number of iterations, let’s say $n$, running in parallel with the second periodic iteration. Then we need to unroll the graph for $(K(G) + n)$ to cover all the firings of all the actors occurring till the end of the second periodic iteration. We can now compute the CCD of actor pairs belonging to different iterations using the unrolled graph and perform relative LTA.

When a graph, having one or more edges with one or more initial tokens, is unrolled then these edges run across iterations, which may eliminate advantages of relative LTA. For example, let’s consider $B_1$ and $B_2$ from Figure 14. Because of the inter-iteration edge $(B_0,B_1)$, their CCD is $A_0$ which is also the graph (absolute) source. Therefore, their absolute start (finish) times are identical to their relative start (finish) times. Let’s assume that $B_0$ always finishes its firing before $A_1$ under any STS, then the start time of $B_1$ always depends on the finish time of $A_1$ and not on $B_0$. As a result, the inter-iteration edge $(B_0,B_1)$ can be ignored for CCD computation. So if we ignore $(B_0,B_1)$ then the CCD of $B_1$ and $B_2$ will be $A_1$ instead of $A_0$, which gives us tighter relative start (finish) times for $B_1$ and $B_2$. This test is performed for each inter-iteration edge in an unrolled graph to improve our CCD based relative life-time analysis.
C. Buffer Interference Graph

We employ a Buffer Interference Graph (BIG) to model conflicts in life-times of different tokens. The BIG is a graph $G(V_B, E_B)$ where $V_B$ is a set of tokens and $E_B$ is a set of edges. If there is an edge between token $A$ and $B$ then their life-times overlap which means they cannot reuse or share the same memory. Generation of BIG using absolute LTA ($BIG_A$) is trivial. Figure 15a shows the $BIG_A$ (for the tokens belonging to the first iteration) of the graph shown in Figure 11.

![Fig. 15: Buffer Interference Graph](image)

To generate BIG using relative LTA ($BIG_R$), we employ CCD based relative LTA. Since relative life-times are tighter, $BIG_R$ will have at most the same number of interferences as $BIG_A$. Hence, $BIG_R$ is generated using $BIG_A$ by pruning edges using relative LTA. We take every edge from $BIG_A$ and recompute the life-times of its source and sink tokens using their CCD. If the recomputed life-times do not overlap then we remove that edge else we keep it, refining the interference relations further. The graph obtained after this step is called $BIG_R$. Figure 15b shows the $BIG_R$ of the graph shown in Figure 11. For example, let’s consider $(B, C)$ and $(E, F)$ from the graph shown in Figure 11. As we know, absolute LTA infers conflict between the tokens produced on these edges (reflects in $BIG_A$). Relative best-case start time $\tilde{s}_{rlv}(E, k)$ and relative worst-case finish time $\hat{f}_{rlv}(C, k)$ with respect to their CCD i.e. $B$ are given by Equation 18 and 19 respectively. Since, $\tilde{s}_{rlv}(E, k) \geq \hat{f}_{rlv}(C, k)$, the relative life-times do not conflict and hence the interference between $(B, C)$ and $(E, F)$ shall be removed from $BIG_A$; and the same is reflected in the $BIG_R$ as shown in Figure 15b.

\[
\tilde{s}_{rlv}(E, k) = \tilde{s}(E, k) - \tilde{s}(B, k) \\
\hat{f}_{rlv}(C, k) = \hat{f}(C, k) - \hat{s}(B, k)
\]

D. Cycle optimization

Both absolute and relative life-time analysis for varying execution times can be pessimistic when it comes to computing buffer sizes for the edges involved in a cycle. Let’s consider the example shown in Figure 16 with $\tau = 1$ and $\hat{\tau} = 5$ units for all actors. In this case, LTA will infer that the life-time of the tokens belonging to edges $(i, j)$, $(j, k)$ and $(k, l)$ conflict and the same will be reflected in the generated BIG. However, since there is only one token present in the cycle, only one of the actors $i, j, k, l$ can be executing at any given time. On the other hand, at the time of firing, input and output edges of the same actor will conflict because we assume that a token resides in memory from the start of the firing that produces it to the end of the actor firing that consumes it. As a result, buffer interference edges $((i, j), (k, l))$ and $((j, k), (l, i))$, present in the BIG of the given example, must be removed. In this way, for any Tb-SRDF graph, for every cycle, interference relations in its BIG are refined. Note that the cycle optimization involves finding all possible paths between a given pair of actors, which makes the algorithm exponentially complex. Therefore, we perform the cycle optimization on the critical cycles of the graph. Consequently, this may yield smaller reduction in the buffer sizes.

![Fig. 16: Cycle optimization](image)
E. Algorithmic flow

Figure 17 shows the algorithm flow for our buffer calculation. A Tb-SRDF application model is given as input. If absolute LTA is chosen, then absolute LTA is performed to generate $BIG_A$. If relative LTA is chosen, then first absolute LTA is performed and then $BIG_A$ is refined with the help of relative LTA and cycle optimization to generate $BIG_R$. Depending on the chosen LTA, $BIG_A$/$BIG_R$ is used to allocate buffers to memory based on the FIFO ordered buffer sharing policy. Analysis algorithms comprising LTA and the dominator analysis have a time complexity of $O(n^3)$ for a graph having $n$ actors [12]. Moreover, the length of the transient phase of an input graph during simulation impacts the running time of the algorithm.

![Algorithmic flow](image)

Fig. 17: Algorithmic flow

VIII. RESULTS

We use an LTE receiver, a WLAN receiver, an MP3 decoder and an H263 video encoder [20] applications to benchmark our technique. The LTE receiver, WLAN receiver and MP3 decoder models are described in Appendix A, B and C respectively. Note that we use WLAN receiver assuming a packet size of 100 payloads (Appendix B).

A. Buffer sizes for systems without back-pressure

We compute buffer sizes for platforms without back-pressure as shown in Table I. We compare the buffer sizes obtained by absolute LTA with the buffer sizes obtained by relative LTA with FIFO ordered buffer sharing and cycle optimization.

<table>
<thead>
<tr>
<th>Application</th>
<th>Buffer Sizes (KBytes)</th>
<th>Savings (%)</th>
<th>Running-Time (#sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Absolute</td>
<td>Relative</td>
<td></td>
</tr>
<tr>
<td>LTE Receiver</td>
<td>529</td>
<td>465</td>
<td>12</td>
</tr>
<tr>
<td>WLAN Receiver</td>
<td>3.62</td>
<td>3.54</td>
<td>2</td>
</tr>
<tr>
<td>H263 Encoder</td>
<td>344.2</td>
<td>306.1</td>
<td>11</td>
</tr>
<tr>
<td>MP3 Decoder</td>
<td>61.2</td>
<td>61</td>
<td>1</td>
</tr>
</tbody>
</table>

TABLE I: Buffer sizes: Absolute Vs. Relative

As expected, the relative LTA always provide smaller or equal buffer sizes compared to absolute LTA. These savings can increase if a graph has a tree-like topology. This is because the accuracy of life-times for a set of actors depends on the closeness of their CCD to the set of actors i.e. closer the CCD to the set, better the accuracy. For a tree-shaped structure, for a parent actor $i$, the dominator analysis will yield $i$ as the CCD for its child actors which is the closest possible CCD for the child actors.

B. Effect of variation in execution times on the buffer sizes

This experiment studies the effect of variation in actor execution times on the buffer sizes for non-back-pressured systems. We use absolute LTA with FIFO ordered buffer sharing for this experiment. For a given Tb-SRDF graph, to compute the buffer sizes that guarantee correct execution for any possible variation in execution time of any actor with respect to its WCETs, we assume the maximum variation in actor execution times i.e. we set BCETs of all the actors (except the source) to zero.
We define Execution Variance as the difference between WCET and BCET of an actor. Maximum execution variance occurs when BCETs of actors are set to zero, referred to as 100% variance. Buffer sizes are computed for every step, where at each step the execution variance is reduced by 10% by increasing BCETs. The last step, where the variance becomes zero (0%), implies non-back-pressured execution with fixed (worst-case) execution times. For each application, we plot these buffer sizes on Y-axis with respect to the execution variance on X-axis as shown in Figure 18. 0% means there is no variation in execution times, which gives the buffer sizes $B(\hat{\tau})$, i.e. buffer sizes for the WC-STS. Buffer sizes obtained for any execution variance from 100% till 10% can be treated as $\tilde{B}$ for corresponding updated BCETs and WCETs. Results show that reducing the variation in execution times reduce the buffer sizes which confirms our result: $\tilde{B} \geq B(\hat{\tau})$. This can be inferred from the fact that reducing the variation in execution times reduce the life-times i.e. the distance between the best-case production time and the worst-case consumption time of a token, which in turn reduce the number of overlapping token life-times.

C. Back-pressure Vs. No back-pressure

We already know from Section VI that the buffer sizes $B(\hat{\tau})$ obtained for the WC-STS are valid for the rate-optimal execution of back-pressured system. Hence, for this experiment, we use $B(\hat{\tau})$ as the buffer sizes for back-pressured systems. Note that these back-pressured buffer sizes are not optimal since we do not alter self-timed schedules to reduce the buffer sizes like shown in [20]. Therefore, an algorithm that computes optimal buffer sizes for back-pressured systems may produces smaller buffer sizes than $B(\hat{\tau})$. Table II shows the buffer sizes required by each application for back-pressured ($B(\hat{\tau})$) and non-back-pressured implementation. An LTE receiver is scheduled on Ericsson’s modem platform [2]. As we can see in Table II, the buffer sizes required by the systems with back-pressure are significantly smaller compared to the systems without back-pressure for the same throughput.

<table>
<thead>
<tr>
<th>Application</th>
<th>Buffer Sizes (KBytes)</th>
<th>Savings (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No back-pressure</td>
<td>Back-pressure ($B(\hat{\tau})$)</td>
<td></td>
</tr>
<tr>
<td>LTE Receiver</td>
<td>465</td>
<td>343</td>
</tr>
<tr>
<td>WLAN Receiver</td>
<td>3.54</td>
<td>2.48</td>
</tr>
<tr>
<td>H263 Encoder</td>
<td>306.1</td>
<td>268.3</td>
</tr>
<tr>
<td>MP3 Decoder</td>
<td>61</td>
<td>43.8</td>
</tr>
</tbody>
</table>

TABLE II: Buffer sizes: Back-pressure ($B(\hat{\tau})$) Vs. No back-pressure

IX. RELATED WORK

Buffer sizing and buffer allocation constitute a topic that is thoroughly studied in the context of static dataflow [10], [20], [13]. Various approaches have been proposed for buffer sizing for the dataflow graphs scheduled on a platform with back-pressure [20], [23]. In [20], authors have provided optimal buffer sizing algorithms for back-pressured systems by altering self-timed schedules. Unlike any other techniques, our technique compute buffer sizes for a dataflow graph running on a platform without back-pressure. To our knowledge, buffer sizing for dataflow graphs executing without back-pressure has not been performed previously.
Our technique uses dominators to perform relative life-time analysis. Closest Common Dominator (CCD) is a novel concept used in our analysis to build relative life-times, which, to our knowledge, have not been carried out previously. In [15], relative life-time analysis is performed for reducing the jitter in fixed priority scheduling analysis, however, the application model is restricted to tree-like structures. It cannot handle a more general graph topology unlike our technique. Furthermore, our dominator based technique systematically determines a relative source for arbitrary graph structures which is not possible with the technique provided in [15].

Several other temporal analysis tools such as SymTA/S [5] and RTC [3] support an execution model without back-pressure and can carry out buffer sizing. However, they cannot perform FIFO ordered buffer sharing (recall Section VII-A) which can lead to large buffer sizes. Moreover, SymTA/S can handle specific cases of cyclic dependencies whereas our technique can handle general cyclic dependencies efficiently.

X. Conclusions

In this paper, we presented a buffer allocation technique that can handle systems without back-pressure. We have extended the available dataflow analysis techniques with best-case temporal analysis. Our buffer allocation technique is the first dataflow based technique that can be applied to many practical systems without back-pressure. Moreover, for our benchmark set, the relative life-time analysis proposed in the work shows up to 12% gain in memory consumption compared to traditional life-time analysis. The paper also presented, in Section VI, a first comparison between buffer sizing for systems with and without back-pressure. We show that for a given system with and without back-pressure, the system with back-pressure will always require smaller or equal total memory expenditure on buffer allocation.

As a future work, we are extending the technique for the dataflow variant called Mode-Controlled Dataflow, which can handle dynamic inter-task dependencies.

REFERENCES


APPENDIX A

LTE Receiver

In this Section, we describe an LTE receiver and the construction of the SRDF graph of an LTE Receiver used in the experiments. In an LTE receiver, data is transmitted in terms of Radio Frames (10 msec in duration) [17]. They are divided into 10 sub-frames, each sub-frame being 1 msec long as shown in Figure 19. Each sub-frame is further divided into two slots, each of 0.5 msec. A slot may contain 6 or 7 OFDM symbols, depending on the normal or extended cyclic prefix used.
The transmitted downlink signal consists of $N_{BW}$ sub-carriers in a resource grid as shown in Figure 20. The value of $N_{BW}$ depends on the system bandwidth (ranging from 1.25 to 20 MHz) and the sub-carrier bandwidth (15 kHz). A Physical Resource Block (PRB) is defined as consisting of 12 consecutive sub-carriers for one slot duration. Each location present in the resource grid is called a Resource Element, which is the basic unit of physical resources. LTE employs special Reference Signals (RS) in each resource block to facilitate channel estimation and timing synchronization. As shown in Figure 20, all the RSs are used for multiple antennas case and only highlighted RSs are used for single antenna case.

Different (control and data) types of information are mapped to different physical channels (a subset of the resource elements in the resource grid) in a sub-frame. We consider three physical channels: the Physical Control Format Indicator Channel (PCFICH), the Physical Downlink Control Channel (PDCCH) and the Physical Downlink Shared Channel (PDSCH). The PCFICH ($C_1$ channel) is a control channel that carries a Control Format Indicator (CFI) message which contains information about the structure and size of PDCCH. A PDCCH ($C_2$ channel) carries a Downlink Control Information (DCI) message which includes resource assignments and other control information for one or more User Equipments (UE). The PDSCH (Data channel) is the main shared (among all UEs) data channel which carries all the user data. Henceforth, wherever necessary, we will refer both $C_1$ and $C_2$ channels as $Cntrl$ channel.
A. LTE Receiver model

Figure 21 shows the LTE Receiver Tb-SRDF model used for our experiments. It shows processing for a single sub-frame having 14 OFDM symbols. Src0, Src2, ... Src14 actors model arrival of (14) input symbols for a single sub-frame. OFDM symbols arrive periodically with the period of 71 μsec. Each input OFDM symbol is demodulated by respective OFDM Demodulator (DMOD). Channel estimates are computed by ChEst. Multiple Input Multiple Output i.e. MIMO computes the combined response in case of multiple antennas. OFDM Demapper (DMAP) demaps symbols to softbits with the help of combined response from MIMO. MIMO and DMAP for Ctrl and Data channels exhibit distinct activities and hence, we model MIMO and DMAP for both the channels separately; MIMO_Ctrl and DMAP_Ctrl process Ctrl channel symbols, whereas MIMO_Data and DMAP_Data process Data channel symbols. DMAP_Ctrl and DMAP_Data forward the softbits to respective decoders i.e. CDEC and DDEC. PCPH decodes the C1 channel and feeds the information to CDEC, which facilitates decoding of the C2 channel. DCID (DCI Done) extracts the location and other Data channel related information from the Ctrl channel. MAC is a higher layer interface (sink actor).

Every function such as DMOD, DMAP, CDEC and DDEC have dedicated processing element as described in Figure 4. For instance, DMOD must execute for the \((i + 1)th\) symbol after the \((i)th\) symbol. For a sub-frame, DMOD function is executed in the following order: DMOD1, DMOD2, ... DMOD14. This execution order is specified by having a static-order among the DMOD actors. Any static-order imposed to a group of SRDF actors executing on the same processing element is represented by adding edges with no tokens between them. From the last to the first actor in the static-order an edge is also added, with a single initial token; when the static-order finishes execution for a given iteration, it re-starts it from the first actor in the static-order for the next iteration.

In LTE Receiver, variation in the mapping of C1, C2 and Data channels to the symbols in a sub-frame give rise to different sub-frame formats. The 1st symbol always has C1 mapped on it; C2 may occupy the remaining part of the 1st symbol and also can occupy the 2nd and even the 3rd symbol. Depending on the C2 mapping, Data channel occupies the remaining symbols in a sub-frame from the 2nd or 3rd or 4th to the 14th symbol. Recall that we refer both the C1 and C2 channels as Ctrl channel. For our modeling, we assume the worst-case for both the channels i.e. we assume 3 Ctrl symbols and 13 Data symbols in a sub-frame. This means, we assume that the 2nd and 3rd symbol in a sub-frame contain both the Ctrl and Data channel symbols. Consequently, as we can see in Figure 21, Ctrl channel decoder i.e. CDEC consumes three symbols from three respective DMAP_Ctrl actors. Also, Data channel decoder i.e. DDEC consumes 13 symbols from respective DMAP_Data actors to decode the sub-frame.

Reference Signals are only transmitted for few resource elements in the LTE resource grid (Figure 20). ChEst only consumes the symbols having the Reference Signals (RS), whose positions in any sub-frame are fixed. The LTE model shown in Figure 21 models this cyclo-static behavior of ChEst accurately. ChEst consumes symbols only for the 1st, 2nd, 5th, 8th, 9th and 12th symbol positions in every sub-frame as shown in Figure 21. Consequently, to obtain the channel estimates for every resource element, we need to carry out interpolation along frequency direction and then along time direction. This forces the ChEst to run 6 tokens ahead of decoding stage to decode the current symbol. Moreover, fast time filtering in ChEst allows decoding the control (Ctrl) channels faster than the data (Data) channels. Consequently, the ChEst stages for Ctrl and Data channels run 4 and 6 symbols ahead of their respective decoding stages respectively. Consequently, for the Ctrl channel, the 1st (2nd and 3rd) control symbols will be demapped in the 5th (6th and 7th) symbols. This is shown in Figure 21: DMAP_Ctrl0, DMAP_Ctrl5 and DMAP_Ctrl7 produce softbits of the 1st, 2nd and 3rd symbols respectively; which is modeled by edges (DMOD1, DMAP_Ctrl5), (DMOD2, DMAP_Ctrl6) and (DMOD3, DMAP_Ctrl7). These demapped control symbols are then consumed by CDEC. Similarly, for Data channel, the 1st data symbol i.e. the 2nd OFDM symbol in a sub-frame is demapped by DMAP_Data8; the 2nd data symbol i.e. the 3rd OFDM symbol in a sub-frame is demapped by DMAP_Data9 and so on.

DMAP for Ctrl and Data are statically ordered i.e. DMAP_Ctrl5 is always fired before DMAP_Data5. However, our model, for a sub-frame where it has 2 Ctrl symbols and 12 Data symbols, will still fire DMAP_Ctrl7 (i.e. DMAP for the 3rd Ctrl symbol) which does not exist for the given sub-frame format. This will delay DMAP_Data7 firing, which in practice will not be delayed. We model this scenario by assigning BCET of DMAP_Ctrl7 as 0. In the best-case of DMAP_Ctrl7, which is equivalent to having 2 Ctrl scenarios in a sub-frame, firing of DMAP_Data7 is not delayed. In this way, we model different sub-frame formats in a single graph conservatively.
Fig. 21: LTE Receiver Tb-SRDF graph
APPENDIX B
WLAN RECEIVER

Figure 22 depicts the Tb-SRDF model of a WLAN 802.11a receiver. We assume a cyclo-periodic source (Section III-B). Nodes with names starting by `Src` model the source (inputs from an external RF unit). The nodes `LatencyHeader` and `LatencyPayload` and their adjacent edges are used to convert latency into throughput constraints [9]. The WLAN packet synchronization step is represented in Multi-Rate Dataflow (MRDF) syntax: 5 `CFEnSync` nodes process the output of 5 `Src` nodes in a chain of sources and synchronization nodes. `HDemode` and `HDecode` actors infer header demodulation and decoding. `PDemode_1/2/3` and `PDecode_1/2/3` actors infer demodulation and decoding of a single payload. A single packet of WLAN receiver can have from 1 to 255 payloads. This variation is depicted by a production/consumption rates (MRDF syntax) of $n$ whose value ranges between 1 and 255. For given $n$, the WLAN MRDF graph is converted into an equivalent SRDF graph. For more details on WLAN, please refer [9].

Unlike in [9], we assume that each actor is mapped on a dedicated processing element. Each edge in Figure 22 is annotated with its token size in bytes. Table III shows the best-case and the worst-case execution times for actors in nanoseconds for a hardware platform described in [9].

![WLAN Receiver Tb-SRDF graph](Fig. 22: WLAN Receiver Tb-SRDF graph)
In this Section, we introduce a Tb-SRDF model of the decoder side of an MPEG-1 Layer 3 (MP3) standard i.e. MP3 decoder. An MP3 decoder reverses the operations performed by the MP3 audio encoder in order to reconstruct the audio stream. A Tb-SRDF model of an MP3 decoder is shown in Figure 23. For more details about the model, please refer [19]. Unlike the model shown in [19], the source of the model is assumed to be a cyclo-periodic source III-B. We assume that each actor in the model is scheduled on a separate processing element such as ARM. The best-case and worst-case execution times for each actor (in terms of processor cycles) are shown in Table IV for a hardware platform similar to ARM7 architecture. Token sizes for Huffman and Req. self-edges are 1024 and 64 bytes respectively. Token sizes for all the other edges except the self-edges is 576 bytes. For rest of the self-edges, the token sizes are 0 bytes.

![MP3 Decoder Tb-SRDF Graph](image)

Fig. 23: MP3 Decoder Tb-SRDF graph

<table>
<thead>
<tr>
<th>Actors</th>
<th>BCET</th>
<th>WCET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>2400</td>
<td>2400</td>
</tr>
<tr>
<td>Src2</td>
<td>800</td>
<td>500</td>
</tr>
<tr>
<td>Src3</td>
<td>2400</td>
<td>2400</td>
</tr>
<tr>
<td>Src4</td>
<td>1600</td>
<td>1600</td>
</tr>
<tr>
<td>Src5</td>
<td>2400</td>
<td>2400</td>
</tr>
<tr>
<td>Src_Header</td>
<td>4000</td>
<td>4000</td>
</tr>
<tr>
<td>Src_Payload</td>
<td>4000</td>
<td>4000</td>
</tr>
<tr>
<td>SIFS</td>
<td>1600</td>
<td>1600</td>
</tr>
<tr>
<td>Detect</td>
<td>1000</td>
<td>5000</td>
</tr>
<tr>
<td>CFEnSync</td>
<td>1000</td>
<td>5000</td>
</tr>
<tr>
<td>FFEnMst</td>
<td>1000</td>
<td>5000</td>
</tr>
<tr>
<td>HDemode</td>
<td>1000</td>
<td>5000</td>
</tr>
<tr>
<td>HDecode</td>
<td>1000</td>
<td>5000</td>
</tr>
<tr>
<td>PDmode_1</td>
<td>460</td>
<td>1500</td>
</tr>
<tr>
<td>PDmode_2</td>
<td>460</td>
<td>1500</td>
</tr>
<tr>
<td>PDmode_3</td>
<td>460</td>
<td>1500</td>
</tr>
<tr>
<td>PD_decode_1</td>
<td>460</td>
<td>1500</td>
</tr>
<tr>
<td>PD_decode_2</td>
<td>460</td>
<td>1500</td>
</tr>
<tr>
<td>PD_decode_3</td>
<td>460</td>
<td>1500</td>
</tr>
<tr>
<td>MacCRC</td>
<td>1000</td>
<td>5000</td>
</tr>
<tr>
<td>MacAnalyse</td>
<td>400</td>
<td>2000</td>
</tr>
<tr>
<td>BuildHeader</td>
<td>1000</td>
<td>5000</td>
</tr>
<tr>
<td>CodeHeader</td>
<td>1000</td>
<td>5000</td>
</tr>
<tr>
<td>ModHeader</td>
<td>1000</td>
<td>5000</td>
</tr>
<tr>
<td>AckCode</td>
<td>1000</td>
<td>5000</td>
</tr>
<tr>
<td>AckMod</td>
<td>1000</td>
<td>5000</td>
</tr>
<tr>
<td>SendPreamble</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SendHeader</td>
<td>1600</td>
<td>8000</td>
</tr>
<tr>
<td>SendPayload</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LatencyHeader</td>
<td>20000</td>
<td>20000</td>
</tr>
<tr>
<td>LatencyPayload</td>
<td>20000</td>
<td>20000</td>
</tr>
</tbody>
</table>

TABLE III: WLAN receiver: actor best- and worst-case execution times (#nanoseconds)

<table>
<thead>
<tr>
<th>Actors</th>
<th>BCET</th>
<th>WCET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Huffman</td>
<td>151977</td>
<td>151977</td>
</tr>
<tr>
<td>Req.</td>
<td>3635</td>
<td>72695</td>
</tr>
<tr>
<td>Reorder</td>
<td>1734</td>
<td>34684</td>
</tr>
<tr>
<td>Stereo</td>
<td>2680</td>
<td>53602</td>
</tr>
<tr>
<td>Antialias</td>
<td>93250</td>
<td>1865001</td>
</tr>
<tr>
<td>Hybrid Synth._L</td>
<td>271</td>
<td>74114</td>
</tr>
<tr>
<td>Freq. Inv._L</td>
<td>246</td>
<td>4912</td>
</tr>
<tr>
<td>Subh. Inv._L</td>
<td>93250</td>
<td>1865001</td>
</tr>
</tbody>
</table>

TABLE IV: MP3 decoder: actor best- and worst-case execution times for ARM7 (#cycles)
If you want to receive reports, send an email to: wsinsan@tue.nl (we cannot guarantee the availability of the requested reports).

**In this series appeared (from 2012):**

<table>
<thead>
<tr>
<th>Date</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/01</td>
<td>S. Cranen</td>
<td>Model checking the FlexRay startup phase</td>
</tr>
<tr>
<td>12/02</td>
<td>U. Khadim and P.J.L. Cuijpers</td>
<td>Appendix C / G of the paper: Repairing Time-Determinism in the Process Algebra for Hybrid Systems ACP</td>
</tr>
<tr>
<td>12/03</td>
<td>M.M.H.P. van den Heuvel, P.J.L. Cuijpers, J.J. Lukkien and N.W. Fisher</td>
<td>Revised budget allocations for fixed-priority-scheduled periodic resources</td>
</tr>
<tr>
<td>12/04</td>
<td>Ammar Osaiweran, Tom Fransen, Jan Friso Groote and Bart van Rijnsoever</td>
<td>Experience Report on Designing and Developing Control Components using Formal Methods</td>
</tr>
<tr>
<td>12/05</td>
<td>Sjoerd Cranen, Jeroen J.A. Keiren and Tim A.C. Willemse</td>
<td>A cure for stuttering parity games</td>
</tr>
<tr>
<td>12/06</td>
<td>A.P. van der Meer</td>
<td>CIF MSOS type system</td>
</tr>
<tr>
<td>12/07</td>
<td>Dirk Fahland and Robert Prüfer</td>
<td>Data and Abstraction for Scenario-Based Modeling with Petri Nets</td>
</tr>
<tr>
<td>12/08</td>
<td>Luc Engelen and Anton Wijs</td>
<td>Checking Property Preservation of Refining Transformations for Model-Driven Development</td>
</tr>
<tr>
<td>12/09</td>
<td>M.M.H.P. van den Heuvel, M. Behnam, R.J. Bril, J.J. Lukkien and T. Nolte</td>
<td>Opaque analysis for resource-sharing components in hierarchical real-time systems - extended version –</td>
</tr>
<tr>
<td>12/10</td>
<td>Milosh Stoljki, Pieter J. L. Cuijpers and Johan J. Lukkien</td>
<td>Efficient reprogramming of sensor networks using incremental updates and data compression</td>
</tr>
<tr>
<td>12/11</td>
<td>John Businge, Alexander Serebrenik and Mark van den Brand</td>
<td>Survival of Eclipse Third-party Plug-ins</td>
</tr>
<tr>
<td>12/12</td>
<td>Jeroen J.A. Keiren and Martijn D. Klabbers</td>
<td>Modelling and verifying IEEE Std 11073-20601 session setup using mCRL2</td>
</tr>
<tr>
<td>12/13</td>
<td>Ammar Osaiweran, Jan Friso Groote, Mathijs Schuts, Jozef Hooman and Bart van Rijnsoever</td>
<td>Evaluating the Effect of Formal Techniques in Industry</td>
</tr>
<tr>
<td>12/14</td>
<td>Ammar Osaiweran, Mathijs Schuts, and Jozef Hooman</td>
<td>Incorporating Formal Techniques into Industrial Practice</td>
</tr>
<tr>
<td>13/01</td>
<td>S. Cranen, M.W. Gazda, J.W. Wesselink and T.A.C. Willemse</td>
<td>Abstraction in Parameterised Boolean Equation Systems</td>
</tr>
<tr>
<td>13/02</td>
<td>Neda Noroozi, Mohammad Reza Mousavi and Tim A.C. Willemse</td>
<td>Decomposability in Formal Conformance Testing</td>
</tr>
<tr>
<td>13/03</td>
<td>D. Bera, K.M. van Hee and N. Sidorova</td>
<td>Discrete Timed Petri nets</td>
</tr>
<tr>
<td>13/04</td>
<td>A. Kota Gopalakrishna, T. Ozelebi, A. Liotta and J.J. Lukkien</td>
<td>Relevance as a Metric for Evaluating Machine Learning Algorithms</td>
</tr>
<tr>
<td>13/05</td>
<td>T. Ozelebi, A. Weffers-Albu and J.J. Lukkien</td>
<td>Proceedings of the 2012 Workshop on Ambient Intelligence Infrastructures (WAmI)</td>
</tr>
<tr>
<td>13/06</td>
<td>Lotfi ben Othmane, Pelin Angin, Harold Weffers and Bharat Bhargava</td>
<td>Extending the Agile Development Process to Develop Acceptably Secure Software</td>
</tr>
<tr>
<td>13/08</td>
<td>Mark van den Brand and Jan Friso Groote</td>
<td>Software Engineering: Redundancy is Key</td>
</tr>
<tr>
<td>13/09</td>
<td>P.J.L. Cuijpers</td>
<td>Prefix Orders as a General Model of Dynamics</td>
</tr>
<tr>
<td>Time</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>14/01</td>
<td>Jan Friso Groote, Remco van der Hofstad and Matthias Raffelsieper</td>
<td>On the Random Structure of Behavioural Transition Systems</td>
</tr>
<tr>
<td>14/02</td>
<td>Maurice H. ter Beek and Erik P. de Vink</td>
<td>Using mCRL2 for the analysis of software product lines</td>
</tr>
<tr>
<td>14/03</td>
<td>Frank Peeters, Ion Barosan, Tao Yue and Alexander Serebrenik</td>
<td>A Modeling Environment Supporting the Co-evolution of User Requirements and Design</td>
</tr>
<tr>
<td>14/04</td>
<td>Jan Friso Groote and Hans Zantema</td>
<td>A probabilistic analysis of the Game of the Goose</td>
</tr>
<tr>
<td>14/05</td>
<td>Hrishikhesh Salunkhe, Orlando Moreira and Kees van Berkel</td>
<td>Buffer Allocation for Real-Time Streaming on a Multi-Processor without Back-Pressure</td>
</tr>
</tbody>
</table>