Performance Analysis of Reconfigurations in Adaptive Real-Time Streaming Applications

JUN ZHU, INGO SANDER, and AXEL JANTSCH,
Royal Institute of Technology, Stockholm, Sweden

We propose a performance analysis framework for adaptive real-time synchronous data flow streaming applications on runtime reconfigurable FPGAs. As the main contribution, we present a constraint based approach to capture both streaming application execution semantics and the varying design concerns during reconfigurations. With our event models constructed as cumulative functions on data streams, we exploit a novel compile-time analysis framework based on iterative timing phases. Finally, we implement our framework on a public domain constraint solver, and illustrate its capabilities in the analysis of design trade-offs due to reconfigurations with experiments.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-Based Systems**]: *Real-time and embedded systems*; C.4 [**Performance of Systems**]: *Modeling techniques*

General Terms: Design, Experimentation, Performance

Additional Key Words and Phrases: Performance analysis, reconfiguration, streaming applications, synchronous data flow, runtime reconfigurable FPGAs

ACM Reference Format:
Zhu, J., Sander, I., and Jantsch, A. 2012. Performance analysis of reconfigurations in adaptive real-time streaming applications. ACM Trans. Embed. Comput. Syst. 11S, 1, Article 12 (June 2012), 20 pages.
DOI = 10.1145/2180887.2180888 http://doi.acm.org/10.1145/2180887.2180888

1. INTRODUCTION

Partially runtime reconfigurable (RTR) FPGAs, such as Xilinx Virtex-4 [Xilinx Ltd], are becoming very popular infrastructures in today's embedded systems Chaudhuri et al. [2008]. They allow part of the hardware tasks to be reprogramed dynamically on the fly while the remaining part continues its operation without being affected. For a number of signal processing and multimedia streaming applications, this reconfigurable property enhances their capability to vary functionalities at runtime in a dynamic environment with varying demands, which significantly reduces the design cost while leveraging the ubiquity of embedded systems. Hence, RTR FPGAs can be used to built cost-effective hardware platform for streaming applications [Kirischian et al. 2008], and deliver high flexibility, besides breakthrough performance.

However, reconfigurability adds another dimension of complexity to the design process, while the system performance needs to be satisfied even during

This research has been partially supported by the ANDRES project within FP6, the sixth framework program of the European Commission.
J. Zhu is currently affiliated with University College Dublin, Ireland.
Authors' addresses: J. Zhu, School of Computer Science and Informatics, University College Dublin, Ireland; email: jun.zhu@ucd.ie; I. Sander and A. Jantsch, Department of Electronic, Computer, and Software Systems, School for Information and Communication Technology, Royal Institute of Technology (KTH), Stockholm, Sweden; email: {ingo, axel}@kth.se.

DOI 10.1145/2180887.2180888 http://doi.acm.org/10.1145/2180887.2180888

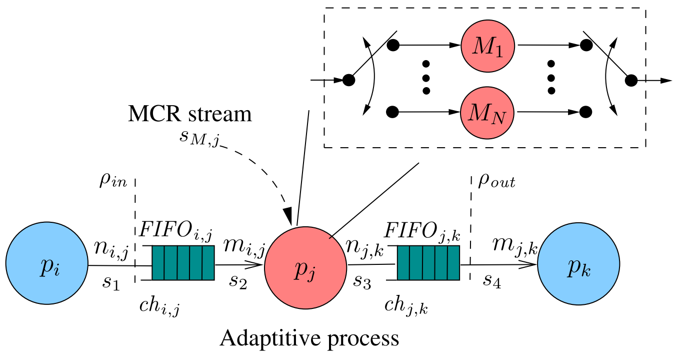


Fig. 1. An example adaptive streaming application model.

reconfigurations. The scheduling of regular SDF applications have been known to be NP-complete [Govindarajan et al. 2002; Stuijk et al. 2006]. Compared with traditional nonadaptive systems, the reconfigurations in adaptive systems add new challenges in performance analysis and lead to even more complexity.

An adaptive streaming application in the synchronous data flow (SDF) model [Lee and Messerschmitt 1987] is illustrated in Figure 1, which we use as a tutorial example in this article. Nodes denote the computation processes, which are connected with each other via communication channels denoted as edges. For instance, the communication channel $ch_{i,j}$ connects process p_i with p_j. FIFOs associated with each communication channel denote the storage buffers, which decouple the input and output data streams for each channel. For instance, $FIFO_{i,j}$ decouples the input data stream s_1 from the output data stream s_2 of $ch_{i,j}$. Processes read tokens from the input-side FIFOs, operate (compute) on the data, and emit the resulting data tokens to the output-side FIFOs. In SDF models, the amount of input (output) tokens is fixed in each firing of a process, and is called input (output) *rate*. For instance, the output rate $n_{i,j}$ from process p_i to channel $ch_{i,j}$ and input rate $m_{j,k}$ of process p_k from channel $ch_{j,k}$ are both static. While a process is computing, the data tokens remain on the input-side FIFO(s) until the computation is completed [Stuijk et al. 2006]. At the end of each execution, the output results are available in the output-side FIFO(s).

Besides the regular source and sink processes p_i and p_k, there is an adaptive process p_j, which has N different working modes from M_1 to M_N, as illustrated in the dashed box. The mode change is initiated by a (MCR). stream, that is, $s_{M,j}$ for p_j. Each mode transition circumstance introduces a temporal overhead, during which p_j does not work in either the old or new mode and is thus stalled.

While the stream source p_i provides an input throughput ρ_{in} (to the input of communication channel $ch_{i,j}$ cutting by the dashed line), an average output throughput ρ_{out} needs to be guaranteed by the application even during reconfigurations. Caused by the reconfiguration stalls of adaptive process p_j, it is critical that the backlog tokens in the FIFO(s) between p_j and the consumer process(es), the so called *playout buffer(s)*[1], are sufficient to sustain the application throughput. Therefore, to exploit the full potential of RTR adaptive systems, special resource requirement and scheduling techniques are needed when taking the behaviors and properties of reconfigurations into consideration. In this article, we address performance analysis, without losing throughput guarantees and design efficiency, for real-time adaptive streaming applications. To the extent of our knowledge, it is still an open topic.

[1]In the example application in Figure 1, the playout buffer is $FIFO_{j,k}$.

The rest of this article is structured as follows. Section 2 discusses the related work. Section 3 introduces the reconfiguration basics and the RTR FPGA architecture model used throughout this article. Section 4 and Section 5 discuss the simulation based approach on our synchronous model and the reconfiguration analysis framework for adaptive systems respectively. Then, we present our constraint based analysis approach in Section 6. Section 7 shows the experimental results. Finally, Section 8 concludes this article.

2. RELATED WORK

Models of computations (MoCs) have been widely used as the formal base in streaming applications design [Geilen and Basten 2004; Lee and Sangiovanni-Vincentelli 1998]. For a class of specifications with fixed consumption and production tokens in computation, Lee and Messerschmitt [1987] have introduced synchronous data flow (SDF) models, which facilitate the functionality validation, scheduling, and buffer analysis at compile time. To provide timing guarantees, Govindarajan et al. [2002] and Stuijk et al. [2006] exploit a timed-SDF model[2] to address the scheduling of SDF applications for buffer requirement minimization with performance guarantees (NP-complete problem). Preserving the static producing and consuming token properties, the SDF models can be transformed into synchronous models [Lublinerman and Tripakis 2008; Zhu et al. 2008a], in which timing-related properties (e.g., throughput and energy) can be captured. The synchronous MoC has been very successful in the context of industrial safety-critical and reactive real-time systems [Benveniste et al. 2003]. We adopt the synchronous model in a recent work [Sander and Jantsch 2008] for the modelling of adaptive systems with prespecified reconfiguration scenarios. However, buffer dimensioning and design cost efficiency of real-time embedded systems with runtime reconfigurability have not yet been addressed by all the previous work above.

Formal analysis at design time has been widely used in performance analysis of heterogeneous embedded systems, such as timing properties validation, scheduling policies optimization, and buffer dimensioning. It has been a promising option to overcome the limitations of simulation based methods in incomplete corner case coverage, and can thus preserve conservative system properties. In SymTA/S [Richter et al. 2002], a compositional way for scheduling analysis has been presented based on standard periodic/sporadic event models. Network calculus [Boudec and Thiran 2001] and real-time calculus (RTC) [Chakraborty et al. 2003] are both a collection of methods in deterministic queuing theories. Both methods formalize the incoming workloads and processing capabilities as cumulative functions over time, and suit performance analysis in network domain and real-time embedded system domain respectively. In the subsequent extensions of RTC, Chakraborty et al. [2005] and Phan et al. [2008] present a mode based RTC to handle the execution dependence between processing resources and buffers caused by their state information (i.e., the fill-levels of buffers and their effect on processing resources). However, none of them aims at buffer dimensioning and design cost (area) analysis for *adaptive systems* as we do.

Schedulability analysis and reconfiguration methods for multimode (adaptive) real-time systems has been studied in [Real and Crespo 2004; Shin et al. 2000], where each mode consists of a different set of tasks. They develop mode change protocols in mode transition stages, and exploit analysis techniques to ensure that no deadlines are violated during the transition periods. In the cyclo-SDF MoC [Bilsen et al. 1996], the number of tokens produced and consumed by a single task changes periodically. Since this cyclically changing behavior is known at compile time, static schedules can be

[2]To analyze timing related properties, the timed-SDF model is a timed extension to the regular untimed SDF model in Lee and Messerschmitt [1987].

constructed when the necessary and sufficient condition for scheduling holds [Bilsen et al. 1996]. Accordingly, the buffer requirement can be analyzed for specified throughput requirement [Moreira et al. 2009], similar as in SDF models [Stuijk et al. 2006]. Furthermore, Wiggers et al. [2008] propose an algorithm to compute sufficient buffer capacities for task graphs with data-dependent execution conditions (consumption token rate), but they do not consider the runtime overhead between the transition of different execution conditions. On the other hand, we address more practical adaptive systems on RTR FPGAs with reconfiguration overhead, according to the runtime (unpredictable at compile-time) configuration request.

Our constraint formulation is close in spirit to the work on nonadaptive SDF applications in Govindarajan et al. [2002], in which the authors establish their linear constraints based on data dependency and process start execution time. However, they relax the integer constraints on buffer sizes (the integer formulation is NP-complete) in implementation to utilize the efficiency of linear programming. It is argued to dimension the conservative *minimal* buffer requirement. This article is based on our previous work [Zhu et al. 2008b]. While the scheduling problem of adaptive streaming applications is formulated in integer linear programming (ILP) techniques, the simulation based on the synchronous model of computation ensures application throughput guarantees [Zhu et al. 2008b]. In this article, we propose a refined integral approach based on iterative timing phases for reconfiguration analysis, instead of the hybrid approach of simulation and analysis in Zhu et al. [2008b]. Inspired by the existing successful optimization techniques in constraint programming domain to solve NP-complete problems, we exploit a public domain constraint solver: *Gecode* [Gecode 2009] for solutions finding.[3]

3. RECONFIGURATION PRELIMINARIES

In this section, without losing generality, we use the adaptive process p_j in Figure 1 to illustrate the reconfiguration preliminaries used in this article.

3.1 Definitions and Assumptions

First, we introduce the reconfiguration definitions and the main assumptions used in our work.

For the adaptive process p_j, a mode change is triggered by the (MCR), this is, the stream $s_{M,j}$ in Figure 1, which might either come from an external controller or be retrieved from the input data streams. During the mode transition stage, the old configuration is deleted and released for the loading of new ones. The reconfiguration transition from an old mode M_1 to a new mode M_2 takes nonzero time $t_{R,j}^{M_1,M_2}$. $t_{R,j}^{M_1,M_2}$ depends on the circuit size of different reconfiguration modes and is usually nonignorable.

For convenience, to avoid the use of $t_{R,j}^{M_x,M_y}$, we will simply write $t_{R,j}$ to implicitly mean $t_{R,j}^{M_x,M_y}$ when the mode switching is known from context. Similarly, $t_{C,j}$ is used to implicitly mean the computation time $t_{C,j}^{M_x}$ or $t_{C,j}^{M_y}$ of process p_j in the respective working modes.

An MCR may occur during the execution of the system in a particular mode, but never during the transition stages. In the worst case, two succeeding configurations

[3]While our previous work [Zhu et al. 2008b] utilizes a Python script to glue the ILP analysis in *lp_solve* [lp_solve 2009] and the simulation in synchronous model of computation, the framework in this article is implemented using a single *Gecode* [Gecode 2009] solver.

have the minimal interval $t_{interR,i}$ to avoid violating the application throughput requirement caused by too-close consecutive reconfiguration stalls.

The input data stream arrives at a peak or average[4] throughput ρ_{in}. Meanwhile, a required average output throughput ρ_{out} is applied to the sink process p_k, to denote the stable application throughput requirement during the lifetime of adaptive systems even in reconfiguration transition stage.

3.2 Consistency

In this article, we only consider SDF applications, which can run infinitely with bounded buffer and are said to be consistent [Lee and Messerschmitt 1987]. Given the communication channel $ch_{i,j}$ with the input data token $n_{i,j}$ (from p_i) and output data token $m_{i,j}$ (to p_j), for consistent SDF models, process p_i and p_j can run in a repetitive pattern with nontrivial (nonzero) firing rates r_i and r_j. While r_i and r_j are called *repetition firing rate* of process p_i and p_j, they are the minimum integer solutions of a set of balance equations for all the communication channels as follows.

$$r_i \cdot n_{i,j} = r_j \cdot m_{i,j}. \tag{1}$$

The consistency of SDF models is known to be a necessary condition to allow them to be executed within bounded memory without deadlock [Ghamarian et al. 2006]. To execute adaptive SDF streaming applications within bounded memory without deadlock, the model consistency needs to be preserved. Besides the balance equations as defined in Equation (1) for communication channels between nonadaptive processes, furthermore, for each communication channel $ch_{i,j}$ from nonadaptive process p_i to adaptive process p_j, the following equation holds.

$$r_i \cdot n_{i,j} = \sum_{x=1}^{N} r_j^{M_x} \cdot m_{i,j}^{M_x}, \tag{2}$$

in which $r_j^{M_x}$ and $n_{i,j}^{M_x}$ are the firing rate and consumption data tokens for process p_j in each working mode M_x. While reconfiguration protocols determine the correctness of Equation (2) for general adaptive SDF applications, to our best knowledge, such protocols for SDF applications (similar as in [Real and Crespo 2004] for task graphs) are still lacking. Since to develop such protocols is out of the scope of this article, we assume all configurations of the same adaptive process have the same input/output token rate which meets Equation (1); that is, the reconfiguration scenarios addressed in our work always comply with the application consistency requirement, and we focus on the performance analysis of such kind of adaptive systems. However, our approach should be able to handle more general SDF applications, once the reconfiguration protocols are available and can be captured as extra constraints in our analysis framework.

3.3 Runtime Reconfigurable FPGAs

Our architecture template is a partially reconfigurable FPGA, as illustrated in Figure 2. For discussion, we divide the configuration-related area (excluding the nonreconfigurable area) into two parts.

(1) The *configuration memory* is used to store the bit stream of all configurations for different working modes in a compressed (with a ratio k_C) format. It can either be a local memory, or an external memory (Flash, DDRAM, SRAM, etc.).
(2) The *reconfigurable area* is the space for configurations that are only needed for a limited amount of time at runtime. It can be used to store several configurations

[4]Both situations will be covered in our analysis approach; see Section 6.4.

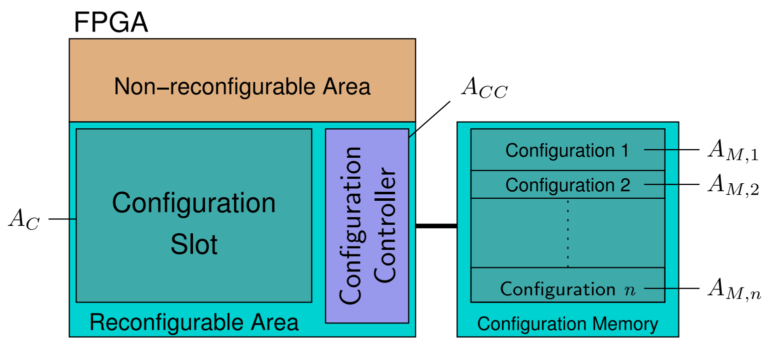


Fig. 2. Overview of a RTR FPGA using just-in-time (JIT) reconfiguration with a single configuration slot.

at the same time. However, here we focus on a "just-in-time" approach (JIT), in which a single[5] *configuration slot* is shared by different configurations at runtime. Every time a new system function is needed, the *configuration controller* enables the reconfiguration, and the bit stream of the new hardware implementation is loaded from the configuration memory into this reconfiguration slot.

Since the prespecified application throughput needs to be sustained during the whole time period of reconfigurations, the consequence may be that there is a need for extra buffer space, which includes the output buffer to sustain the output throughput during reconfiguration transitions and possibly the buffer to store input data tokens. The only unit we use for area is logic elements (LEs). Area requirements in form of memory elements are converted into LEs. Given the hardware implementations for each mode of the process, the cost of configuration controller A_{CC}, configuration slot A_C, and configuration memory $\sum_{i=1}^{i=n} A_{M,i}$ are static (fixed). For the JIT configuration on RTR FPGA in Figure 2, the total design cost in terms of area is

$$A_{JIT} = A_{CC} + \sum_{i=1}^{i=n} A_{M,i} + \underbrace{k_C \cdot max(A_{M,1}, \ldots, A_{M,n})}_{A_C} + A_{Buffer}, \qquad (3)$$

in which A_{Buffer} is the area cost of buffers, that is, $FIFO_{i,j}$ and $FIFO_{j,k}$ in Figure 1.

To design JIT reconfigurable systems efficiently, it is critical to dimension the minimal conservative buffer size and the corresponding cost A_{Buffer}, and to explore the implementation trade-offs of different design options.

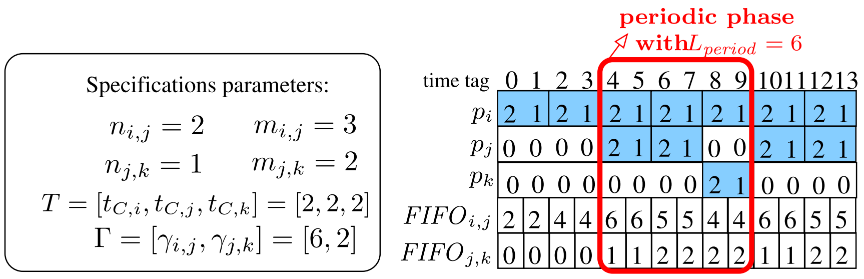
4. MODEL OF COMPUTATION

We adopt the synchronous model [Zhu et al. 2008a] to capture the timing analysis of streaming applications, which preserves the static execution input/output data tokens of SDF models. Based on the simulation mechanisms of the synchronous model, the periodic phases in deterministic scheduling are revealed and utilized to provide application throughput guarantees.

4.1 Synchronous MoC

In our synchronous model, time slots are numbered with $n \in \mathbb{N}_0$, and the data stream s is a time indexed set of events, $s = \{e_0, e_1, \cdots, e_n, \cdots\}$. Each event $e_n = (n, v_n)$ represents

[5]For some applications with predictable MCR, it may be possible to use multiple configuration slots and preload the useful configurations to overcome the JIT reconfiguration stall. We call such a mechanism *prefetch* adaptation, which remains to be addressed in our future work (see Section 8).


Specifications parameters:

$n_{i,j} = 2 \qquad m_{i,j} = 3$

$n_{j,k} = 1 \qquad m_{j,k} = 2$

$T = [t_{C,i}, t_{C,j}, t_{C,k}] = [2, 2, 2]$

$\Gamma = [\gamma_{i,j}, \gamma_{j,k}] = [6, 2]$

periodic phase with $L_{period} = 6$

time tag	0	1	2	3	4	5	6	7	8	9	10	11	12	13
p_i	2	1	2	1	2	1	2	1	2	1	2	1	2	1
p_j	0	0	0	0	2	1	2	1	0	0	2	1	2	1
p_k	0	0	0	0	0	0	0	0	2	1	0	0	0	0
$FIFO_{i,j}$	2	2	4	4	6	6	5	5	4	4	6	6	5	5
$FIFO_{j,k}$	0	0	0	0	1	1	2	2	2	2	1	1	2	2

Fig. 3. A self-timed schedule (right) of the example application (Figure 1) with specified specification parameters (left).

the number $\upsilon_n \in \mathbb{N}_0$ of data tokens present during the time slot n. Especially, we restrict a fixed constant time distance between the events of data streams for timing properties analysis and simulation.[6]

To quantify the process computation and FIFO storage capabilities of the application model, a process computation *latency* list T contains the worst case execution time (WCET)[7] $t_{C,x}$ in time slots to execute each process p_x once. A time slot equals to an abstract clock cycle. A FIFO *size* list Γ contains the storage capacity $\gamma_{y,z}$ in data tokens for each FIFO $FIFO_{y,z}$. For instance, the example application in Figure 1 has $T = [t_{C,i}, t_{C,j}, t_{C,k}]$ and $\Gamma = [\gamma_{i,j}, \gamma_{j,k}]$.

A process is enabled and ready for execution when both the input-side FIFO(s) has sufficient data tokens and the output-side FIFO(s) has enough vacant space. A process *executes* only when it is enabled and assigned the priority by scheduling policies. While a process is computing, the data tokens remain on the input-side FIFO(s) until the computation is completed [Stuijk et al. 2006]. At the end of each execution, the output results are available in the output-side FIFO(s).

4.2 Simulation Based on Synchronous MoC

Here, the example application is instantiated with computation latency list $T = [2, 2, 2]$, storage size list $\Gamma = [6, 2]$, and process input/output token numbers $n_{i,j} = 2$, $m_{i,j} = 3$, $n_{j,k} = 1$ and $m_{j,k} = 2$, as illustrated on the left of Figure 3. Thus, the application process firing times vector is $< r_i, r_j, r_k > = < 3, 2, 1 >$.

In the simulation based on the synchronous MoC, we use self-timed scheduling [Sriram and Bhattacharyya 2000], which means a process *executes* as soon as it is enabled; otherwise, it *stalls*. The corresponding self-timed schedule is illustrated on the right side, in which the process and FIFO status are listed in separated rows. The time evolution is depicted in corresponding columns and advances 1 per column. At each time tag, a process in *executing* (shadowed) state has a number to denote the remaining execution time slots, a *stalling* (nonshadowed) process status is denoted as 0, and a FIFO status is denoted as the occupied storage space (existing tokens plus scheduling reservation) in number of tokens. At each time tag x, the process and FIFO status list are denoted as T'_x and Γ'_x respectively.

At time tag 0, the process status list is $T'_0 = [2, 0, 0]$, in which p_i is executing with 2 time slots left and p_j and p_k are stalled; in the meantime the FIFO status list is $\Gamma'_0 = [2, 0]$, with only 2 tokens space reserved at $FIFO_{i,j}$. At time tag 2, p_i finishes

[6]In general synchronous languages, there is no time metrics associated with the interval between ticks [Lee and Zheng 2007].

[7]The WCET of a process is the maximum possible running time on a certain hardware platform. It is usually derived by worst case analysis of a process program flow [Chen et al. 2001].

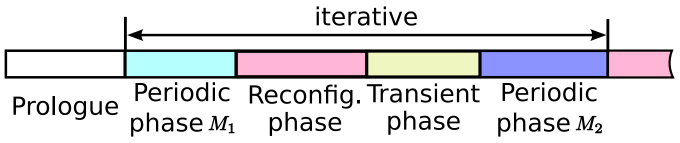


Fig. 4. Timing phases of reconfiguration analysis. In this graph, only the particular mode transition from M_1 to M_2 is shown.

the previous computation, emits 2 tokens into $FIFO_{i,j}$, and reserves another 2 tokens space from $FIFO_{i,j}$ for a new execution; thus, $T'_2 = [2, 0, 0]$ and $\Gamma'_2 = [4, 0]$. At time tag 4 ($T'_4 = [2, 2, 0]$ and $\Gamma'_4 = [6, 1]$), besides the similar status changes in p_i and $FIFO_{i,j}$, p_2 is enabled and starts to compute.

As the schedule advances to time tag 10, the application encounters the same process and FIFO status list as at time tag 4 (i.e., $T'_{10} = T'_4 = [2, 2, 0]$ and $\Gamma'_{10} = \Gamma'_4 = [6, 1]$), and enters a periodic phase. The periodic phase extends from time tag 4 to 9, with length $L_{period} = 6$, in which the process p_i, p_j, and p_k are guaranteed to run 3, 2, and 1 times respectively. Consequently, the schedule guarantees an average output throughput $\rho_{out} = \frac{1 \cdot m_{j,k}}{L_{period}} = \frac{1}{3}$ to process p_k with buffer storage requirement $\Gamma = [6, 3]$, which are the maximum buffer usages at each FIFO.

4.3 Discussions

Using deterministic scheduling policies, such as self-timed scheduling, the above simulation based way can construct schedules for SDF applications at design time with periodic phases and guaranteed throughput (see the schedule in Figure 3 and the proof of Proposition 1). Later, similar techniques based on periodic phases will be exploited in our proposed analysis method as encoded constraints (see Section 6.4).

However, for adaptive systems with reconfiguration requests known at runtime, the optimal scheduling policies are usually not specified in advance. Thus, simulation based methods are lacking systematic ways and exact scheduling policies to dimension the minimal buffer requirement of adaptive systems. Furthermore, they have inherent limitations to cover the corner cases not being considered, which can lead to too optimistic buffer dimensioning.

5. RECONFIGURATIONS ANALYSIS FRAMEWORK

Here, we present a reconfiguration analysis framework based on iterative timing phases and the declarative execution semantics of streaming applications. Instead of describing the actual algorithms (the how) used to solve the problem, we formalize the properties (the what) of the desired solutions in different phases as composable constraints (to be introduced in Section 6), and exploit a constraint solver [Gecode 2009] in solutions finding.

The staged timing phases in reconfiguration analysis, as illustrated in Figure 4, are the following.

Prologue. This is the start-up phase with no throughput guarantees. The length of the prologue phase can be specified by τ_0 in Constraint 7 (Section 6).

Periodic phase $M_1(M_2, \cdots, M_N)$. These are phases with guaranteed throughput in working mode $M_1(M_2)$ of the adaptive process p_j. While the length L_{period} is throughput relevant, the sustainable throughput requirements can be distinct in different working modes and can be specified in Constraint 7 and 8.

Reconfiguration phase. This phase starts with an initial working mode M_1 upon the reconfiguration request MCR. The mode transmission starting time tag t' (determined by a reconfiguration decision variable $\xi(t)$ in Section 6.5) is explored in a specified periodic phase L_{period}[8] to find the optimal reconfiguration with minimized *objective function* (to be given in Equation (4)). The reconfiguration stall (working mode transmission to M_2) takes $t_{R,j}$ time slots (Constraint 10). The length of this phase is specified to be the worst case $L_{period} + t_{R,j}$.

Transient phase. This phase has a length τ_1, in which throughput is met but with no periodic properties in scheduling yet.

Caused by the periodic properties in the scheduling of each working mode, the buffer requirement can be analyzed in a finite length of time, without everlasting analysis. To make the phases in gray (colored) iterative, we can use the timing analysis to traverse all reconfiguration scenarios. The conservative size of each buffer should adopt their worst case dimension respectively. Our design objective is to find the minimal total buffer sizes as follows.

$$\min : \sum_{FIFO_{x,y} \in \mathbf{F}} \gamma_{x,y}, \tag{4}$$

in which $\mathbf{F}$ is the set of the buffers being considered and $\gamma_{x,y}$ is the size of each buffer $FIFO_{x,y}$.

6. CONSTRAINT BASED ANALYSIS

In this section, we first illustrate our event model on data streams. Subsequently, we formalize a constraint based analysis approach which fits well to capture both the streaming application execution semantics and the varying design concerns in different timing phases proposed for reconfiguration analysis. Without loss of generality, we adopt the example application in Figure 1 for illustration.

6.1 Event Models

We construct our event model as cumulative functions on data streams, similar to Cruz [1995] and Chakraborty et al. [2003]. The input/output workloads of each communication channel and the processing capabilities of the computation processes are characterized as follows.

Definition 1 (*Arrival Function*). The arrival function $R_{i,j}(t)$ of the communication channel $ch_{i,j}$ is defined as the sum of tokens arriving from the input data stream during the time interval $[0, t]$, $t \in \mathbb{N}_0$.

For instance, $R_{i,j}(t) = \sum_0^t s_1$ in Figure 1.

Definition 2 (*Output Function*). The output function $R'_{i,j}(t)$ from process p_i to the communication channel $ch_{i,j}$ equals to the arrival function $R_{i,j}(t)$ of $ch_{i,j}$.

For instance, $R'_{i,j}(t) = \sum_0^t s_1 = R_{i,j}(t)$ in Figure 1, which forms the basis for compositional analysis.

Definition 3 (*Service Function*). The service function $C_{i,j}(t)$ of the communication channel $ch_{i,j}$ by process p_j is defined as the sum of tokens served and removed from the buffer $FIFO_{i,j}$ via the data stream by p_j during the time interval $[0, t]$, $t \in \mathbb{N}_0$.

[8]It is based on the assumption that the worst case reconfiguration response (starting) time is L_{period} after the runtime MCR.

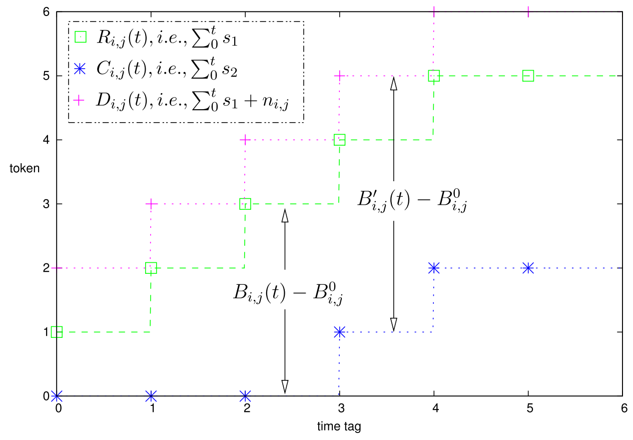


Fig. 5. Cumulative event models and derived buffer properties, which are consistent with the schedule in Figure 3.

For instance, $C_{i,j}(t) = \sum_0^t s_2$ in Figure 1.

6.2 Buffer Properties

While a process is executing, the extra buffer space reservation in scheduling test (see the schedule in Figure 3) can be modeled with the *demand function*:

Definition 4 (*Demand Function*). The demand function $D_{i,j}(t)$ of the communication channel $ch_{i,j}$ is defined as the sum of $R'_{i,j}(t)$ and the demanding space $d_{i,j}(t)$ at time tag t on $FIFO_{i,j}$ from the input side process p_i, that is, $D_{i,j}(t) = R'_{i,j}(t) + d_{i,j}(t)$, $d_{i,j}(t) \in \{0, n_{i,j}\}$.

For instance,

$$D_{i,j}(t) = \begin{cases} \sum_0^t s_1 + n_{i,j} & \text{if } p_i \text{ is executing} \\ \sum_0^t s_1 & \text{if } p_i \text{ is stalling} \end{cases}$$

in Figure 1.

A graphical interpretation of the definitions of $R_{i,j}(t)$, $C_{i,j}(t)$ and $D_{i,j}(t)$ is illustrated in Figure 5, which is consistent with the schedule in Figure 3 for the example application. Since process p_i is always executing, $D_{i,j}(t)$ always demands $n_{i,j}$ tokens above $R_{i,j}(t)$. $R'_{i,j}(t)$ is not shown since it is equivalent to $R_{i,j}(t)$ (see Definition 2).

Consequently, we derive the following buffer properties.

Property 1 (*Backlog*). The backlog $B_{i,j}(t)$ (tokens arrived but not yet served) in buffer $FIFO_{i,j}$ is the vertical distance between $R_{i,j}(t)$ and $C_{i,j}(t)$ plus an offset of the initial buffer tokens $B^0_{i,j}$ at time tag 0.

$$B_{i,j}(t) = R_{i,j}(t) - C_{i,j}(t) + B^0_{i,j}, \quad \forall t \in \mathbb{N}_0 \tag{5}$$

Property 2 (*Buffer Status*). In scheduling, the buffer space in use $B'_{i,j}(t)$ for $FIFO_{i,j}$ (equals to $B_{i,j}(t) + d_{i,j}(t)$) is the vertical distance between $D_{i,j}(t)$ and $C_{i,j}(t)$ plus an offset of the initial buffer tokens $B^0_{i,j}$ at time tag 0.

$$B'_{i,j}(t) = D_{i,j}(t) - C_{i,j}(t) + B^0_{i,j}, \quad \forall t \in \mathbb{N}_0 \tag{6}$$

6.3 Streaming Application Execution Semantics

Although an operational semantics of SDF is used in simulation [Stuijk et al. 2006], the simulation based approaches are argued lacking scheduling policies for adaptive systems as discussed in Section 4.3. Here, based on the definitions and properties above, a full list of constraints to formalize the declarative execution semantics of SDF streaming applications are formalized as the start point of our constraint based analysis. These constraints hold during the whole lifetime of streaming applications ($\forall t \in \mathbb{N}_0$). Meanwhile, the designers can specify some specification dependent parameters, for instance, the initial (tokens) offset $B^0_{i,j}$ for $B'_{i,j}$.

We assume that the data tokens remain on the input-side FIFO(s) when a process is computing. At the end of execution, data tokens are removed from input-side FIFO(s) and output result tokens are emitted into output-side FIFO(s) as well. Thus, the input and output token ratios of each process can be formalized as the following constraint.

Constraint 1 (*Token Ratios*). For process p_j, the input and output token ratios can be formalized by $R'_{j,k}(t)$ and $C_{i,j}(t)$ as the following.

$$R'_{j,k}(t) \cdot m_{i,j} = C_{i,j}(t) \cdot n_{j,k} \tag{7}$$

Constraint 2 (*Computation Latency*). Process p_j has computation latency $t_{C,j}$ in each execution instance.

$$C_{i,j}(t + t_{C,j}) - C_{i,j}(t) = m_{i,j} \cdot K_j(t + t_{C,j}) \tag{8}$$

$$D_{j,k}(t + t_{C,j}) - D_{j,k}(t) = n_{j,k} \cdot K_j(t + t_{C,j}) \tag{9}$$

$$\text{where} \quad K_j(t + t_{C,j}) \in \{0, 1\}$$

in which $K_j(t + t_{C,j})$ denotes the incremental properties of $C_{i,j}(t + t_{C,j})$ and $D_{j,k}(t + t_{C,j})$, that is, $K_j(t + t_{C,j})$ has value '1' if process p_j finishes one instance of execution exactly at time tag $t + t_{C,j}$, otherwise it has value '0'.

Constraint 3 (*Space Reservation*). In the communication channel $ch_{j,k}$, the demand function of process p_j reserves vacant space $t_{C,j}$ slots in advance, which corresponds to the semantics that the process can only execute when there are enough space in output-size FIFO(s).

$$D_{j,k}(t) = R_{j,k}(t + t_{C,j}) \tag{10}$$

Constraint 4 (*Asynchronous Buffer*). The incoming tokens in buffer $FIFO_{i,j}$ takes at least $t_{C,j}$ slots to be served by process p_j, which models the buffering behavior determined by the consumer process(es).

$$R_{i,j}(t) - C_{i,j}(t + \Delta_t) \geqslant 0, \quad \forall \Delta_t \in [1, t_{C,j}] \tag{11}$$

Constraint 5 (*Buffer Requirement*). The buffer size $\gamma_{i,j}$ of buffer $FIFO_{i,j}$ meets the maximum buffer space requirement, which guarantees a conservative buffer dimensioning.

$$\gamma_{i,j} \geqslant B'_{i,j}(t) \tag{12}$$

6.4 Application Throughput Guarantees

For the input data stream s_1 to process p_j, a peak or average throughput ρ_{in} is described as follows.

Constraint 6 (*Source Input*). For the source signal process p_i with computation latency $t_{C,i}$, the data stream with a peak throughput ρ_{in} is constrained according to

$$R_i(t + t_{C,i}) - R_i(t) \leqslant t_{C,i} \cdot \rho_{in} \tag{13}$$

Otherwise, a stable average throughput ρ_{in} can be verified at specified time instances as follows

$$R_i(t_0 + k \cdot \Delta_{t_0}) - R_i(t_0) = k \cdot \Delta_{t_0} \cdot \rho_{in}, \quad \forall k \in \mathbb{N} \tag{14}$$

in which the given t_0 specifies the starting time instance for average throughput checking, and Δ_{t_0} determines the interval between different time instances.

The application output throughput ρ_{out} is subject to the following constraint.

Constraint 7 (*Application Output Throughput*). After some start-up time period τ_0 ($\tau_0 > 0$) with no stable output tokens, a specified output throughput ρ_{out} should be sustained at the application sink process p_k.

$$C_k(\tau_0 + c \cdot L_{period}) \geqslant \rho_{out} \cdot c \cdot L_{period}, \quad \forall c \in \mathbb{N}_0 \tag{15}$$

in which L_{period} is the length of the periodic phase in the schedule (see Figure 3).

However, the problem to determine the length of L_{period} that can provide optimal buffer cost in this formulation is NP-complete itself. Empirically, we choose its length as $L_{period} = q \cdot \lceil \frac{r_k}{\rho_{out}} \rceil, q \in \mathbb{N}\backslash\{\infty\}$, in which the incremental q leads to an increasing L_{period} and r_k is the *repetition firing rate* of process p_k (see Equation (1)). Thus, Constraint 7 can only guarantee that the buffer cost is minimized (optimal) in implementation using the given length of L_{period}. On the other hand, the length of L_{period} can be prespecified (if validly) by the designer, which corresponds to the cost to implement a periodic static schedule.

PROPOSITION 1 (THROUGHPUT GUARANTEES). *For a consistent SDF streaming application (see Section 3.2), a periodic phase (see the schedule in Figure 3) in its schedule always exists. The required application throughput is guaranteed by the output properties during this period.*

PROOF. A consistent SDF streaming application could run infinitely. However, the state space of application scheduling status (process and FIFO status, see Section 4.2) is finite. Thus, some scheduling status will be revisited in a nonterminating schedule. As we consider deterministic scheduling, the application schedule enters a periodic phase when a repeated scheduling status is met. The output throughput during this period could sustain infinitely, which guarantees the application throughput. □

Furthermore, the periodic properties in the scheduling of SDF applications can be modeled as the following.

Constraint 8 (*Periodic Phase*). We specify that, at time tag t' after $t_{interR,i}$ of the previous mode change, the process and FIFO status are repeatable at time tag $t' + L_{period}$, and the schedule enters a periodic phase with length L_{period}.

$$B'_{i,j}(t') = B'_{i,j}(t' + L_{period}), \quad \forall FIFO_{i,j} \tag{16}$$

$$W'_i(t') = W'_i(t' + L_{period}), \quad \forall p_i \tag{17}$$

$$\text{where} \quad W'_i(t') = \sum_{k=1}^{t_{C,i}} k \cdot C_i(t' + k)$$

in which the variable $W'_i(t')$ and $W'_i(t' + L_{period})$ is the encoding of the process status for process p_i in scheduling.

For instance, the process status of p_i at time tag 4 and 10 have the equivalence $W'_i(4) = W'_i(10) = 2$ in scheduling as illustrated in Figure 3.

6.5 Reconfiguration Relevant Constraints

To capture the runtime *reconfiguration* of the reconfigurable process, we use a Boolean function $\xi(t)$ to denote the working mode at time tag t of the reconfigurable process p_j, that is, $\xi(t) = 0$ indicates that p_j works in mode M_1, otherwise p_j is in (or being reconfigured to) mode M_2. Thus, the computation latency $t_{C,j}$ for the adaptive process p_j can be formulated.

Constraint 9 (*Computation Latency - Adaptive*). To be aware of the reconfiguration decision $\xi(t)$, the computation latency $t_{C,j}$ of the adaptive process p_j is

$$t_{C,j} = \neg\xi(t) \cdot t_{C,j}^{M_1} + \xi(t) \cdot t_{C,j}^{M_2} \tag{18}$$

in which $t_{C,j}^{M_1}$ and $t_{C,j}^{M_2}$ denote the computation latency in working mode M_1 and M_2 for p_j respectively,

However, Equation (18) can not be implemented in Gecode directly, since $t_{C,j}$ is not an explicit variable in our model. Accordingly, the constraints for adaptive process p_j containing $t_{C,j}$ need to be rewritten correspondingly. For instance, Equation (8) is equivalent to the following constraint applicable in Gecode.

$$\begin{aligned}\neg\xi(t) \cdot C_{i,j}(t + t_{C,j}^{M_1}) + \xi(t) \cdot C_{i,j}(t + t_{C,j}^{M_2}) - C_{i,j}(t) =& \\ m_{i,j} \cdot (\neg\xi(t) \cdot K_j(t + t_{C,j}^{M_1}) + \xi(t) \cdot K_j(t + t_{C,j}^{M_2}))&\end{aligned} \tag{19}$$

Especially, the multiplication of two variables in Equation (19) can be captured by a nonlinear arithmetic constraint (*mult*) in Gecode. Similarly, other SDF execution semantics on computation and buffer resources can be extended to be reconfiguration aware (for the reconfigurable process), which are omitted for clarity in this article.

Furthermore, we need to consider its computation stall during the reconfiguration transition stage. Such a stall takes $t_{R,j}$ time, and can be described as a constraint as follows.

Constraint 10 (*Reconfiguration Stall*). The computation of the adaptive process p_j stalls for $t_{R,j}$ time period after the reconfiguration starts at time tag t'.

$$C_j(t' + \Delta_t) - C_j(t') = 0, \quad \forall \Delta_t \in [1, t_{R,j}] \tag{20}$$

6.6 Extensions of Constraint Based Formulation

The proposed formulation does not take into account models with multiple input and multiple output (MIMO) processes or cyclic models. However, the extension of our reconfiguration analysis methodology to such models is intuitive. Without loss of generality, we use the MIMO process p_j in Figure 6[9] for illustration.

For MI channels $ch_{i,j}$ and $ch_{l,j}$ of process p_j, the service functions are associated with each other caused by the static input data token rate $m_{i,j}$ and $m_{l,j}$ of the same consumer process p_j. They have the linear relations as follows.

Constraint 11 (*MI Linear Relation*).

$$\frac{C_{i,j}(t)}{m_{i,j}} = \frac{C_{l,j}(t)}{m_{l,j}} \tag{21}$$

[9]For clarity in the graph, the FIFO modules on communication channels are omitted. Instead, a number of dots are used to denote the initial buffer token numbers.

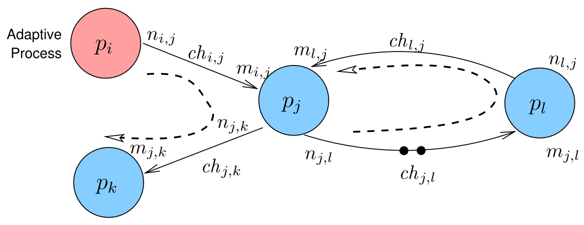


Fig. 6. A cyclic MIMO application model.

Similarly, the MO channels $ch_{j,k}$ and $ch_{j,l}$ have the linear relations on the output and demand functions according to output data token rate of the same producer process p_j as follows.

Constraint 12 (*MO Linear Relation*).

$$\frac{R'_{j,k}(t)}{n_{j,k}} = \frac{R'_{j,l}(t)}{n_{j,l}}, \quad \frac{D_{j,k}(t)}{n_{j,k}} = \frac{D_{j,l}(t)}{n_{j,l}} \tag{22}$$

A MIMO model can be analyzed by traversing it with a set of paths, where each path is a sequence of communication channels such that the output channels of a process always succeed its input channels. A set of paths are complete only when all the communication channels are covered. For instance, the paths ("$ch_{i,j} \rightarrow ch_{j,k}$" and "$ch_{j,l} \rightarrow ch_{l,j}$") in dashed lines complete the MIMO model in Figure 6. Based on the complete set of paths, our methodology fits the MIMO application models well.

For directed cyclic graphs, the data tokens required for loop initialization can be explicitly captured as the initial token offsets in Equations (5) and (6). For instance, two initial tokens of communication channel $ch_{j,l}$ are denoted as $B^0_{j,l} = 2$ and the actual backlog of $ch_{j,l}$ is $B_{j,l}(t) + B^0_{j,l}$.

7. EXPERIMENTAL RESULTS

Inspired by the success of NP-complete constraint solver *Gecode* [Gecode 2009], which is a C++ library, we implement our constraint base analysis framework on it to exploit the state-space exploration techniques in constraint programming domain. Both the example application of Figure 1 and an industrial application from Thales Communications are considered for experiments. While both applications are pipelined models, we refer to our work in [Zhu et al. 2009] for case studies on the scheduling of cyclic applications with MIMO processes, in which the architecture platform is hybrid CPU/FPGAs with no reconfigurations.

For different design options of the adaptive process(es), for instance, with different specifications on $t_{C,j}$ and $t_{R,j}$ for process p_j in Figure 1, the minimum buffer sizes of the application are dimensioned without losing the application throughput guarantees. All experiments are carried out on a HP xw4600 Linux workstation with a Quad-Core[10] 2.40GHZ processor and 4GB of RAM.

7.1 The Example Application

We assume the adaptive process p_j in the example application has two different modes, and both configurations have the same properties (i.e., the same input and output

[10]Only one core is actually utilized, since Gecode 2.1 used for this article does not support multithread searching yet (see Section 8).

Table I. Design Options for the Adaptive Process p_j

options	#1	#2	#3	#4	#5	#6	#7	#8
$t_{R,j}$	**2**	**3**	**4**	**5**	**10**	**15**	**20**	**40**
$t_{C,j}$	16	13	12	**10**	8	6	5	4
A_M	0.5	0.8	1.0	1.3	2.5	3.8	5.0	10

data tokens, computation time and reconfiguration time). Thus, two iterations of the working mode transitions (Figure 4) in reconfiguration analysis can traverse all the reconfiguration scenarios.

We elaborate the model shown in Figure 1 with concrete specification parameters (i.e., $n_{i,j} = 2$, $m_{i,j} = 3$, $n_{j,k} = 1$ and $m_{j,k} = 2$) and assume a minimum interval $t_{interR,j} = 50$ time slots between the two consecutive reconfigurations of p_j.

First, we assume that different design options have varying reconfiguration time $t_{R,j}$ but with a fixed latency $t_{C,j} = 10$ (the bold numbers listed out in Table I), and evaluate the FIFO sizes requirement. Figure 7(a) shows the minimal FIFO sizes needed upon different $t_{R,j}$, corresponding to different output throughput ρ_{out}. To consider the two concerns $t_{R,j}$ and ρ_{out} separately, apparently, higher ρ_{out} demands larger FIFO sizes, so do the design options with higher $t_{R,j}$.

In the following scenario, instead, we choose the design options according to each column listed in Table I. These design options show different implementation strategies in the speed and area trade-offs, for instance, an adder can be implemented as carry-lookahead adder (optimized for speed) or a ripple-adder (optimized for area). Empirically, they conform to $t_{R,j} = k_R \cdot A_{M,j}$ with $k_R = 10.0$ and an assumed relation $t_{C,j} \propto \lceil \frac{1}{\sqrt{t_{R,j}}} \rceil$. Although, higher ρ_{out} ($\rho_{out} = 1/8$) still demands larger FIFO sizes, the FIFO sizes are not monotonic to $t_{R,j}$ any more, as both $t_{C,j}$ and $t_{R,j}$ can affect the buffer requirement to sustain ρ_{out} during reconfiguration. We can see that the design options with $t_{R,j}$ close to 5 need less buffer.

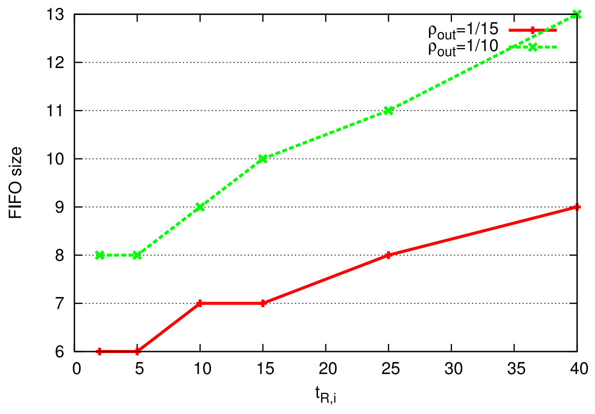
With a given compression ratio $k_C = 4.0$, the design costs are evaluated. As the design options after #5 simply show a fast monotonically increasing cost, we only present the cost of design option #1-5 for clarity in Figure 7(c). We see higher throughput requirement still leads to larger design costs. However, the design costs heavily depend on the $t_{C,j}$ and $t_{R,j}$ trade-off, that is, the speed and time trade-off, and #2 with $t_{R,i} = 3$ shows the minimum cost.

For this example application, the solutions finding time for each design option is 26–55ms, with peak memory 2.5–5.8MB.

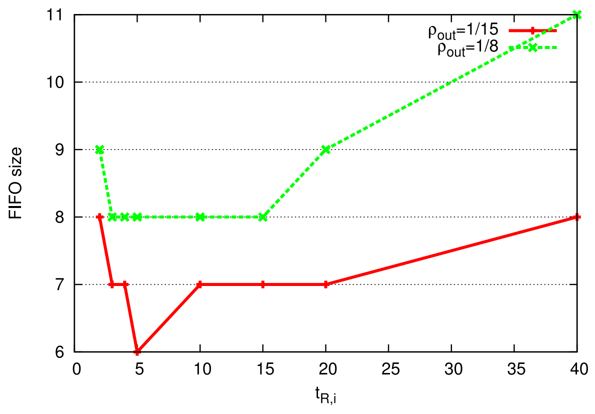
7.2 An Industrial Application

To evaluate the potential of our methodology in adaptive systems, we use it on an industrial coding and modulation application from Thales communication, as illustrated in Figure 8. The diagram shows a mix of digital and analogue modules, with a channel Coder preceded by a Cipher block, and followed by a digital modulator, a digital up converter, a digital to analogue converter and an analogue filter. We focus on the reconfigurable part, that is, the Coder with 3 modes of bursts BR, BL or BT and the Cipher with 3 algorithms 1–3. The source of the Cipher algorithm is the input stream, and the output stream from the Coder needs to sustain a stable throughput. The design objective in performance analysis is to minimize the buffer requirement without losing the stable output transmission, when either or both of the two modules are in reconfiguration.

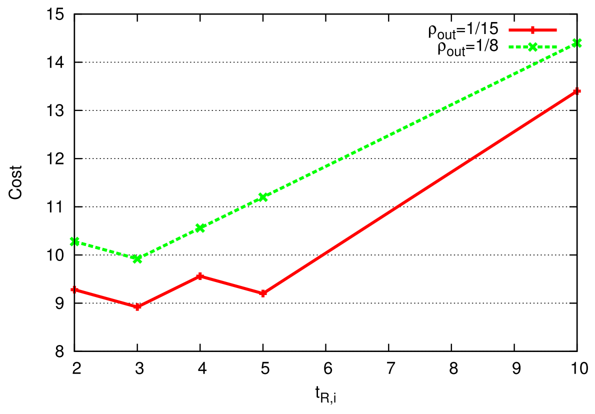
The adaptive part of the abstract application model is illustrated in Figure 9(a), in which the specification parameters are omitted for clarity. There are two



(a) FIFO size upon varying $t_{R,j}$ & fixed $t_{C,j}$



(b) FIFO size upon varying $t_{R,j}$ & $t_{C,j}$



(c) Design cost upon varying $t_{R,j}$ & $t_{C,j}$

Fig. 7. Experimental results of the example application.

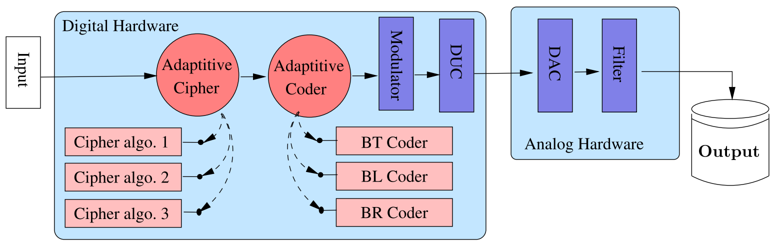


Fig. 8. The coding and modulation case study synopsis.

adaptive processes (modules): the Cipher p_j and the Coder p_k. Each of them receives the adaptation control signal $s_{M,j}$ or $s_{M,k}$ from the environment, and can change the working modes among three possibilities (i.e., algorithm 1–3 for the Cipher, and BT/BL/BR coder for the Coder). The input date stream has a peak throughput ρ_{in}, and an average output throughput ρ_{out} is demanded.

We assume the reconfigurations of two adaptive Cipher and Coder are independent of each other. To decouple the analysis, the buffer $FIFO_{j,k}$ between p_j and p_k has been partitioned into two disjoint logic FIFOs $FIFO'_{j,k}$ and $FIFO''_{j,k}$, as shown in the dashed box, to be analyzed individually. From the static data token ratios of the SDF model and ρ_{out}, we derive the average output throughput requirement $\rho_{out'}$ of the Cipher, which is also the average input throughput to the Coder. For the Cipher, the peak input throughput and average output throughput are ρ_{in} and $\rho_{out'}$ respectively. In this way, we decouple the analysis of two reconfigurable modules as follows.

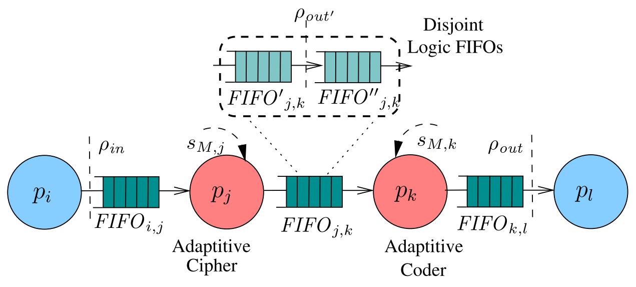
(1) To find the minimum buffer sizes for the Cipher buffers $FIFO_{i,j}$ and $FIFO'_{j,k}$ to meet the average output throughput requirement $\rho_{out'}$ upon the peak input throughput ρ_{in} (subject to Equation (13) in Constraint 6).
(2) To find the minimum play-out buffer $FIFO''_{j,k}$ and $FIFO_{k,l}$ for the Coder module to meet the average output requirements ρ_{out} upon the average input throughput $\rho_{out'}$ (subject to Equation (14) in Constraint 6).

Although the decoupled analysis (based on disjoint logic FIFOs) has the possibility to over-dimension the physical FIFO size, it is a conservative approach without restrictions on the reconfiguration of different adaptive modules, that is, independent of reconfiguration protocols. For instance, using this analysis approach, the Cipher and Coder modules can be even reconfigured at the same time, when protocols of the reconfiguration of consecutive modules are still lacking.

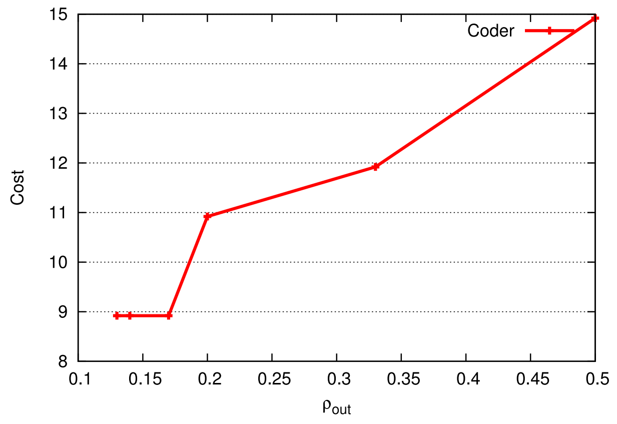
For both the Cipher and Coder, all the reconfiguration transition possibilities are traversed (i.e., $3 \times (3-1) = 6$), and we choose the worst case as the conservative buffer requirement.

In both Coder and Cipher modules, the design costs increase with the output throughput, as shown in Figures 9(b) and 9(c). The design costs of different implementation strategies for the Cipher with varying $t_{R,j}$ are also shown in Figure 9(c), and we can see the design costs are not monotonic to $t_{R,j}$. It exemplifies that our framework suits design trade-offs analysis in design options exploration, and can be applied on a series of compositional adaptive processes (modules) as well.

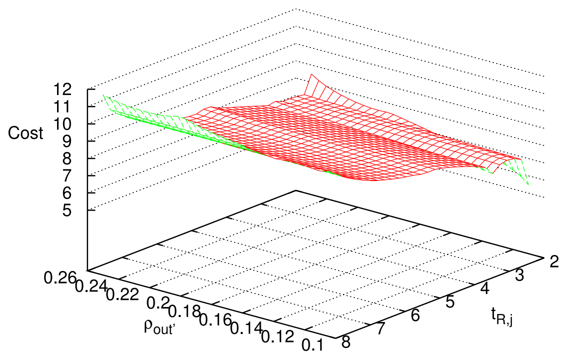
Furthermore, the complexity of solution finding increases exponentially with problem size (compared with the example application). For the Coder, the solutions finding time is 247–708ms and the peak memory is 22.4–54.4MB. For the Cipher, they are 352–408ms and 27.7–36.4MB respectively.



(a) Abstract model of industrial application (with disjoint logic FIFOs in the dashed box).



(b) Design cost of JIT adaptation on the Coder.



(c) Design cost of JIT adaptation on the Cipher.

Fig. 9. Industrial application and experimental results.

8. CONCLUSIONS AND FUTURE WORK

We present a constraint based performance analysis framework for adaptive real-time streaming applications, with the implementation on a public domain constraint solver; Gecode [2009]. The experimental results show that our framework suits reconfiguration analysis and design trade-offs analysis well. It can be used to exploit the reconfigurability of adaptive real-time streaming applications, without losing efficiency. Especially, the industrial case study illustrates the capability of our methodology to cope with the sequential composition of adaptive systems.

Future work. In this article, we have considered JIT adaptation on a single configuration slot. On the other hand, for some applications, the prefetch adaptation with multiple configuration slots might overcome the reconfiguration stall and further reduce the design cost. We plan to compare these two approaches using some industrial applications, and verify the result with Virtex-4 FPGA implementations. So far, the buffer dimensioning has been conducted without considering memory sharing between different logic buffers. However, using some buffer merging algorithms, as proposed in Murthy and Bhattacharyya [2004] and Govindarajan et al. [2002], the memory sizes can be further reduced when different logic buffers are sharing the same physical memory module. Such buffer sharing algorithms complement our analysis approach and remain to be our future work. Furthermore, Gecode starts to support parallel searching with multiple threads from version 3.1 [Gecode 2009]. Facing the increasing problem size and complexity, we plan to adopt multiple threads and exploit heuristic techniques in Gecode during the state-space exploration as well.

ACKNOWLEDGMENTS

We thank anonymous reviewers for very helpful comments and Professor Twan Basten for useful discussions to improve the techniques and content of this article.

REFERENCES

BENVENISTE, A., CASPI, P., EDWARDS, S. A., HALBWACHS, N., LE GUERNIC, P., AND SIMONE, R. D. 2003. The synchronous languages 12 years later. *Proc. IEEE 91*, 1, 64–83.

BILSEN, G., ENGELS, M., LAUWEREINS, R., AND PEPERSTRAETE, J. A. 1996. Cyclo-static dataflow. *IEEE Trans. Signal Proces. 2*, 44, 397–408.

BOUDEC, J.-Y. L. AND THIRAN, P. 2001. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer.

CHAKRABORTY, S., KUNZLI, S., AND THIELE, L. 2003. A general framework for analysing system properties in platform-based embedded system designs. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'03)*. IEEE Computer Society, 190–195.

CHAKRABORTY, S., PHAN, L. T. X., AND THIAGARAJAN, P. S. 2005. Event count automata: A state-based model for stream processing systems. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS'05)*. IEEE Computer Society, 87–98.

CHAUDHURI, S., GUILLEY, S., FLAMENT, F., HOOGVORST, P., AND DANGER, J.-L. 2008. An 8x8 run-time reconfigurable FPGA embedded in a SoC. In *Proceedings of the 45th Annual Conference on Design Automation (DAC'08)*. ACM, 120–125.

CHEN, K., AUGUST, D. I., AND MALIK, S. 2001. Retargetable static timing analysis for embedded software. In *Proceedings of the 14th International Symposium on Systems Synthesis*. 39–44.

CRUZ, R. L. 1995. Quality of service guarantees in virtual circuit switched networks. *IEEE J. Select. Areas Comm. 13*, 6, 1048–1056.

GECODE. 2009. Generic Constraint Development Environment. http://www.gecode.org/.

GEILEN, M. AND BASTEN, T. 2004. Reactive process networks. In *Proceedings of the 4th ACM International Conference on Embedded Software (EMSOFT'04)*. ACM, 137–146.

GHAMARIAN, A. H., GEILEN, M. C. W., BASTEN, T., THEELEN, B. D., MOUSAVI, M. R., AND STUIJK, S. 2006. Liveness and boundedness of synchronous dataflow graphs. In *Proceedings of the Formal Methods in Computer-Aided Design (FMCAD'06)*. IEEE Computer Society, 68–75.

GOVINDARAJAN, R., GAO, G. R., AND DESAI, P. 2002. Minimizing buffer requirements under rate-optimal schedule in regular dataow networks. *J. VLSI Signal Proces. 31*, 3, 207–229.

KIRISCHIAN, V., GEURKOV, V., AND KIRISCHIAN, L. 2008. A multi-mode video-stream processor with cyclically reconfigurable architecture. In *Proceedings of the Conference on Computing Frontiers (CF'08)*. ACM, 105–106.

LEE, E. A. AND MESSERSCHMITT, D. G. 1987. Static scheduling of synchronous dataflow programs for digital signal processing. *IEEE Trans. Computers C-36*, 1, 24–35.

LEE, E. A. AND SANGIOVANNI-VINCENTELLI, A. 1998. A framework for comparing models of computation. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 17*, 12, 1217–1229.

LEE, E. A. AND ZHENG, H. 2007. Leveraging synchronous language principles for heterogeneous modeling and design of embedded systems. In *Proceedings of the 7th ACM-IEEE International Conference on Embedded Software (EMSOFT'07)*. ACM, 14–123.

LP_SOLVE. 2009. lp_solve reference guide. http://lpsolve.sourceforge.net/5.5/.

LUBLINERMAN, R. AND TRIPAKIS, S. 2008. Translating dataflow to synchronous block diagrams. In *Proceedings of the 6th Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia'08)*. 101–106.

MOREIRA, O., BASTEN, T., GEILEN, M., AND STUIJK, S. 2009. Buffer sizing for rate-optimal single-rate dataflow scheduling revisited. *IEEE Trans. Comput.*

MURTHY, P. K. AND BHATTACHARYYA, S. S. 2004. Buffer merging—a powerful technique for reducing memory requirements of synchronous dataow specifications. *ACM Trans. Des. Autom. Electron. Syst. 9*, 2, 212–237.

PHAN, L. T., CHAKRABORTY, S., AND THIAGARAJAN, P. 2008. A multi-mode real-time calculus. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS'08)*. IEEE.

REAL, J. AND CRESPO, A. 2004. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Syst. 26*, 2, 161–197.

RICHTER, K., ZIEGENBEIN, D., JERSAK, M., AND ERNST, R. 2002. Model composition for scheduling analysis in platform design. In *Proceedings of the 39th Annual Conference on Design Automation (DAC'02)*. ACM, 287–292.

SANDER, I. AND JANTSCH, A. 2008. Modelling adaptive systems in ForSyDe. *Electron. Notes Theor. Comput. Sci. 200*, 2, 39–54.

SHIN, Y., KIM, D., AND CHOI, K. 2000. Schedulability-driven performance analysis of multiple mode embedded real-time systems. In *Proceedings of the 37th Conference on Design Automation (DAC'00)*. 495–500.

SRIRAM, S. AND BHATTACHARYYA, S. S. 2000. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker, Inc., New York, NY.

STUIJK, S., GEILEN, M., AND BASTEN, T. 2006. Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. In *Proceedings of the 43rd Annual Conference on Design Automation. (DAC'06)*. 899–904.

WIGGERS, M. H., BEKOOIJ, M. J. G., AND SMIT, G. J. M. 2008. Computation of buffer capacities for throughput constrained and data dependent inter-task communication. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'08)*. ACM, 640–645.

XILINX LTD. http://www.xilinx.com.

ZHU, J., SANDER, I., AND JANTSCH, A. 2008a. Energy efficient streaming applications with guaranteed throughput on MPSoCs. In *Proceedings of the International Conference on Embedded Software (EMSOFT'08)*. 119–128.

ZHU, J., SANDER, I., AND JANTSCH, A. 2008b. Performance analysis of reconfiguration in adaptive real-time streaming applications. In *Proceedings of the 6th Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia'08)*. 53–58.

ZHU, J., SANDER, I., AND JANTSCH, A. 2009. Buffer minimization of real-time streaming applications scheduling on hybrid CPU/FPGA architectures. In *Proceedings of the Conference on Design Automation and Test in Europe (DATE '09)*. 1506–1511.

Received February 2009; revised October 2009; accepted November 2009