Buffer Minimization of Real-Time Streaming Applications Scheduling on Hybrid CPU/FPGA Architectures

Jun Zhu, Ingo Sander, Axel Jantsch
Royal Institute of Technology, Stockholm, Sweden
{junz, ingo, axel}@kth.se

Abstract

We address the problem of real-time streaming applications scheduling on hybrid CPU/FPGA architectures. The main contribution is a two-step approach to minimize the buffer requirement for streaming applications with throughput guarantees. A novel declarative way of constraint based scheduling for real-time hybrid SW/HW systems is proposed, while the application throughput is guaranteed by periodic phases in execution. We use a voice-band modem application to exemplify the scheduling capabilities of our method. The experimental results show the advantages of our techniques in both less buffer requirement and higher throughput guarantees compared to the traditional PAPS method.

I. Introduction

The current trend toward systems-on-chip (SoCs) consisting of several modules of processor, custom circuit and memory (e.g., the hybrid CPU/FPGA architecture [1]) makes the global analysis of heterogeneous software/hardware (SW/HW) systems essential. Meanwhile, the real-time streaming applications on such platforms always have stringent demands on non-functional properties (e.g., timing, design cost, energy dissipation), besides functional correctness. To capture all the design concerns globally, while making optimal design decisions at an early system level, is still a big challenge.

Synchronous data flow (SDF) has been widely used to model and analyze streaming applications on single-/multi-processors [2], [3]. An example of such a model is illustrated in the upper part of Figure 1, which is used as a tutorial example in this paper. Nodes denote the computation processes. Edges associated with FIFOs denote the communication channels with finite storage, which decouples the input and output data streams of each communication channel. For instance, $FIFO_{i,j}$ decouples the input data stream s_1 from the output data stream s_2 of

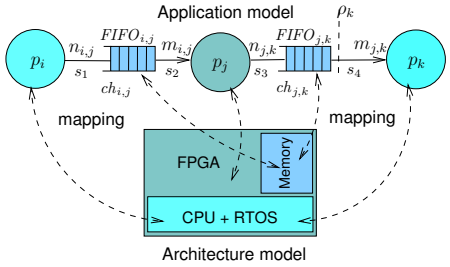


Fig. 1: An example application mapped onto the hybrid CPU/FPGA architecture.

the communication channel $ch_{i,j}$ between process p_i and p_j. Processes read tokens from the input-side FIFOs, and emit the result data tokens to the output-side FIFOs at the end of the computation. The input/output token numbers are fixed at each execution [2] and denoted as symbols at each side of the communication channels (e.g., process p_j with $m_{i,j}$ input tokens and $n_{j,k}$ output tokens).

The application model is mapped onto the hybrid CPU/FPGA architecture below. Process p_i and p_k are implemented as SW and scheduled sequentially by the real-time operating system (RTOS) on the same CPU, and process p_j is implemented as HW custom circuit. The computation modules (both SW and HW) communicate via tightly coupled memory for data operations. Needless to mention, the general architecture platform considered may have multiple CPU or custom circuit modules.

Due to the static nature of SDF models, we can afford to use sophisticated algorithms to compute optimized schedules at compile time. In this paper, we propose our constraint based scheduling methodology for real-time streaming applications with minimal buffer requirements on such a hybrid SW/HW platform.

A. Related Work

Lee and Messerschmitt [2] present techniques to construct periodic admissible sequential schedules (PASS) on single-processors or periodic admissible parallel schedules (PAPS) on multi-processors. Later, Bhattacharyya et al. [4]

have taken buffer minimization into consideration using heuristics in PASS (but not PAPS) construction.

Govindarajan et al. [5] address the parallel scheduling techniques for SDF applications to obtain maximal throughput with minimized buffer requirement without computation resource constraints, i.e., with unlimited number of processors. Furthermore, Stuijk et al. investigate the buffer minimization of applications with different specified throughput requirements with the same assumption on computation resources [6]. In our previous work, buffer dimensioning has been addressed on reconfigurable FPGA HW [7]. Nevertheless, none of these techniques can handle the global optimization of both sequential SW (RTOS) and parallel HW scheduling on our hybrid CPU/FPGA architecture.

In [8], Madsen el at. validate the sanity of several existing scheduling policies (i.e., rate monotonic and earliest deadline first) of the multi-processor RTOS on a SystemC model. However, a systematic way to explore and find out an optimal schedule according to the required throughput is still an open issue.

In this paper, we aim to provide such kind of buffer minimization schedules for real-time SDF streaming applications on hybrid CPU/FPGA architectures. Different from all the previous (operational) work mentioned, we focus on describing the constraint based scheduling problems (a declarative way), but apply the existing successful optimization techniques [9] for problem solving.

B. Paper overview

The rest of the paper is organized as follows. Section II motivates our work and Section III proposes our scheduling work flow. We formalize our problem as constraint based scheduling in Section IV. Section V shows the experimental results. Finally, Section VI concludes the paper.

II. Motivation

Here, we introduce the streaming applications execution semantics and motivate our work by several schedules with varying buffer requirements and throughput guarantees.

A. Execution semantics

In this paper, we only consider SDF models which can run infinitely with bounded buffer and are said to be consistent [2]. Given the communication channel $ch_{i,j}$ between process p_i and p_j with the input data rate $n_{i,j}$ and output data rate $m_{i,j}$, for consistent SDF models, p_i and p_j can run in a repetitive pattern with non-trivial (non-zero) firing times r_i and r_j, where r_i and r_j are the minimum integer solutions of a set of balance equations $r_i \cdot m_{i,j} = r_j \cdot n_{i,j}$ for all the communication channels.

To quantify the process computation and FIFO storage capabilities of the application model, a process computation *latency* list T contains computation time $t_{C,x}$ to execute each process p_x once and 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 FIFOs have sufficient data tokens and the output-side FIFOs have enough vacant space. A process *executes* only when it is enabled and allowed by the scheduling policy. While a process is computing, the data tokens remain on the input-side FIFOs until the computation is completed [6]. At the end of each execution, the output results are available in the output-side FIFOs. We capture the data stream s as a time indexed set of events, $s = \{e_0, e_1, \cdots, e_n, \cdots\}$. Each event $e_n = (n, v_n)$ represents the number $v_n \in \mathbb{N}_0$ of data tokens present during the time slot n.

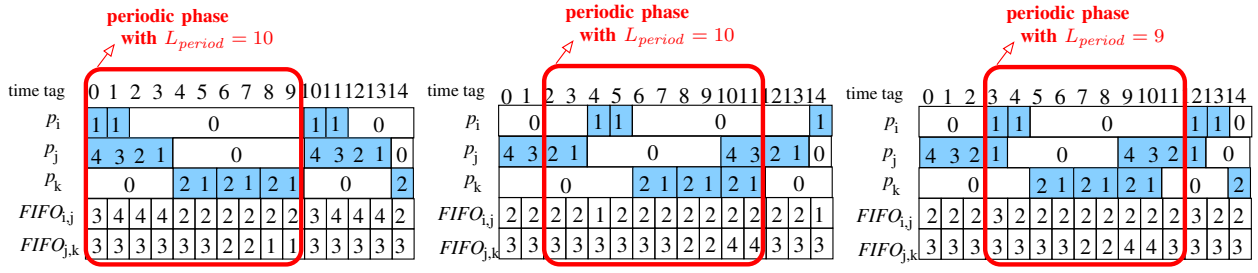
B. Schedules with varying buffer requirements

Here, the example application is instantiated with computation latency list $T = [1, 4, 2]$ and process input/output token numbers $n_{i,j} = 1$, $m_{i,j} = 2$, $n_{j,k} = 3$ and $m_{j,k} = 1$. Thus, the process firing times vector is $< r_i, r_j, r_k > = < 2, 1, 3 >$ for the example application in Figure 1. We assume there are some initial tokens in buffers $FIFO_{i,j}$ and $FIFO_{j,k}$, which are denoted as $B^0_{i,j} = 2$ and $B^0_{j,k} = 0$ respectively.

Using the application to architecture mapping as shown in Figure 1, three valid periodic schedules are illustrated in Figure 2. Figure 2a is the PAPS [2] with unroll factor[1] $J = 1$. 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 n, a process p_x in $executing$ (shadowed) state has a number to denote the remaining execution time slots, a $stalling$ (non-shadowed) process status is denoted as 0, and a buffer $FIFO_y$ status is denoted as the occupied storage space (existing tokens plus space reservation) in tokens. At time tag 0, the process status list is $T'_0 = [1, 4, 0]$, in which p_i and p_j are executing with 1 and 4 time slots left respectively and p_k is stalled; in the meantime the FIFO status list is $\Gamma'_0 = [3, 3]$, with 3 tokens space used each FIFO. As the schedule advances to time tag 10, the application encounters the same status lists as at time tag 0 (i.e., $T'_{10} = T'_0$ and $\Gamma'_{10} = \Gamma'_0$), and enters a periodic phase. The periodic phase has length $L_{period} = 10$, in which the sink process p_k always runs 3 times. Consequently, the schedule guarantees an average output data rate $\rho_k = \frac{3 \cdot m_{j,k}}{L_{period}} = \frac{3}{10}$ at process p_j and requires buffer storage $\Gamma = [4, 3]$, which are the maximum buffer usages at each FIFO.

However, a periodic parallel schedule (not the PAPS in in [2]) with minimized buffer (i.e., $\Gamma = [2, 4]$), which

[1]The iteration number of the minimal repetitive pattern (Section II-A).



(a) PAPS with $\rho_k = \frac{3}{10}$ (b) A minimal buffer schedule with $\rho_k = \frac{3}{10}$ (c) A minimal buffer schedule with $\rho_k = \frac{3}{9}$

Fig. 2: Comparison of different schedules of an instance of the example application, in which p_i and p_k are mapped onto the same processor and can only be scheduled sequentially.

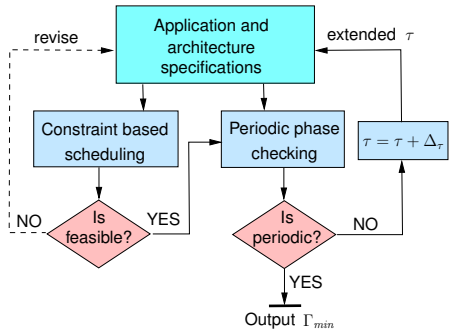


Fig. 3: Scheduling work flow with minimal buffer requirement and guaranteed throughput.

guarantees the same throughput $\rho_k = \frac{3}{10}$, does exist in Figure 2b. Furthermore, a schedule with 10% higher throughput guarantee $\rho_k = \frac{3}{9}$ can still be achieved in Figure 2c, which has the same buffer cost $\Gamma = [3, 4]$.

Although the application throughput can be improved by increasing J in PAPS, the implementation cost of the periodic phase and buffer requirement both increase accordingly (see the case study in Section V), and a systematic way is still lacking [2]. In this paper, we intend to provide a systematic way to construct optimal schedules with minimal buffer requirement and higher throughput guarantees.

III. Work flow

In Figure 3, we propose a two-step scheduling work flow. The work flow inputs (in shadowed box) are the application and architecture specifications, e.g., the application model, specified application and architecture mapping, required throughput, a time period τ considered. The work flow can be described as follows.

Step 1: When the constraint based scheduling problem is feasible, a pending schedule with minimized buffer is got; otherwise, the specifications need to be revised (which is out of the scope of this paper).

Step 2: The throughput guarantees are checked for the pending schedule (whether a periodic phase could be found). If the throughput guarantee or maximum execution time is met, it stops and outputs the valid schedule with minimal buffer sizes Γ_{min}; otherwise, it increases the considered τ with Δ_τ and goes back to **Step 1**.

Apparently, only when the throughput guarantees are met in Step 2, the output results are valid. The initial values of τ and Δ_τ are application dependent and are given empirically.

IV. Streaming application scheduling

In this section, we first illustrate our event model for streaming data flows. Subsequently, we formalize our constraint based scheduling problems.

A. Event model

We construct our event model as cumulative functions [10], [11] on streaming data flows. Each time slot equals to an abstract clock cycle.

Without loss of generality, we adopt the example application in Figure 1, and characterize the input/output workloads of each communication channel and the processing capabilities of the output side processes 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, $R_{i,j}(t) = \int_0^t s_1$ 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, $R'_{i,j}(t) = \int_0^t s_1 = R_{i,j}(t)$ in Figure 1.

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 $[0, t], t \in \mathbb{N}_0$.*

For instance, $C_{i,j}(t) = \int_0^t s_2$ in Figure 1, which forms the basis for compositional analysis.

B. Buffer properties

While a process is executing, the extra buffer space reservation in the scheduling (see Section II-A) can be modelled 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, i.e., $D_{i,j}(t) = R'_{i,j}(t) + d_{i,j}(t), d_{i,j}(t) \in \{0, n_{i,j}\}$.*

For instance,

$$D_{i,j}(t) = \begin{cases} \int_0^t s_1 + n_{i,j} & \text{if } p_i \text{ is executing} \\ \int_0^t s_1 & \text{if } p_i \text{ is stalling} \end{cases}$$

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 4, which is consistent with the schedule in Figure 2a. $R'_{i,j}(t)$ is ignored for its equivalence to $R_{i,j}(t)$ (see Definition 2).

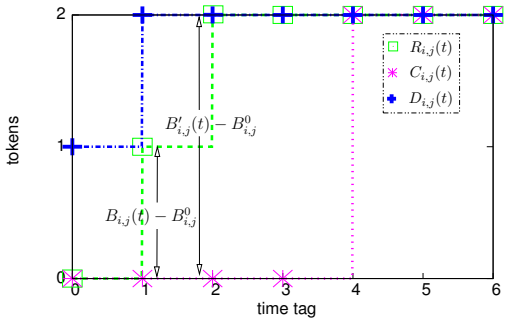


Fig. 4: Cumulative functions and buffer properties for the PAPS in Figure 2a.

Consequently, we derive the following 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.*

$$B_{i,j}(t) = R_{i,j}(t) - C_{i,j}(t) + B^0_{i,j}, \quad \forall t \in \mathbb{N}_0 \qquad (1)$$

Property 2. *(Buffer usage) 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.*

$$B'_{i,j}(t) = D_{i,j}(t) - C_{i,j}(t) + B^0_{i,j}, \quad \forall t \in \mathbb{N}_0 \qquad (2)$$

Based on the definitions and properties above, a full list of constraints to formalize the execution semantics of streaming applications are given in Appendix.

C. Resource limits and throughput requisites

Instead of describing the actual algorithms (the how) used to find the solution, our declarative method formalizes the properties (the what) of the desired solution as constraints.

Constraint 1. *(Sequential execution) In a set of processes $\boldsymbol{P}_a$ mapped onto the same processor CPU_a, at any time at most one can execute (sequentially) according to:*

$$\sum_{p_j \in \boldsymbol{P}_a} W_j(t) \in \{0, 1\}. \quad \forall t \in \mathbb{N}_0 \qquad (3)$$

where $W_j(t) = max(L_j(t), L_j(t + \Delta_t)),\ \forall \Delta_t \in [1, t_{C,j}]$

$$L_j(t+1) = \frac{C_{i,j}(t+1) - C_{i,j}(t)}{m_{i,j}} \in \{0, 1\}$$

in which $W_j(t)$ denotes the computing or stalling 0-1 status of process p_j (different from the fine-grained process status exemplified in Figure 2), $L_j(t+1)$ denotes the incremental properties (step) of the service function $C_{i,j}(t+1)$.

Constraint 2. *(Application throughput) After some start-up time period τ_0 ($\tau_0 > 0$) with no stable output tokens, a specified throughput ρ_k should be met to sustain the required output rate at the application sink process p_k.*

$$C_k(\tau_0 + c \cdot \Delta_t) \geqslant \rho_k \cdot c \cdot \Delta_t, \quad \forall c \in \mathbb{N}_0, \exists \Delta_t \in \mathbb{N} \qquad (4)$$

Scheduling objective. The scheduling objective is to find the minimal total buffer sizes as follows.

$$\min: \sum_{\forall FIFO_{y,z} \in \mathbf{F}} \gamma_{y,z} \qquad (5)$$

in which **F** is the set of the buffers being considered and $\gamma_{y,z}$ is the size of buffer $FIFO_{y,z}$, subject to Constraint A-1 - A-5 (in Appendix) and Constraint 1 - 2.

D. Application throughput guarantees

Proposition 1. *(Throughput guarantees) For a consistent SDF streaming application (see Section I), a periodic phase (see Section II-B) in its schedule always exists. The required application throughput is guaranteed by the output data rate during this period.*

Proof: A consistent SDF streaming application could run infinitely. However, the application scheduling status (process and FIFO status, see Section II-B) space is always finite. Thus, some scheduling status will be re-visited in a non-terminating schedule. As we consider deterministic scheduling, the application schedule enters a periodic phase when a repeated scheduling status is met. The output data rate during this period could sustain infinitely, which meets the required application throughput guarantees. ■

E. Extension to MIMO and cyclic model

The extension of our methodology to multiple input and multiple output (MIMO) models, as shown in Figure 5[2],

[2]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.

is intuitive. Without loss of generality, we use the MIMO process p_j in Figure 5 for illustration.

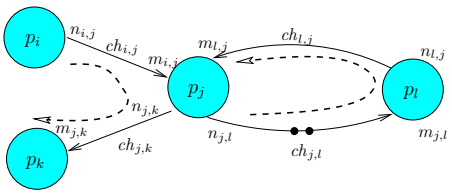


Fig. 5: A cyclic MIMO application model.

According to Definition 1 and 3, each input channel has their individual arrival and service functions (i.e., $R_{i,j}(t)$, $R_{l,j}(t)$, $C_{j,k}(t)$, and $C_{j,l}(t)$), which have the linear relations (static firing rates) between them as follows.

Constraint 3. *(MI linear relation)*

$$\frac{R_{i,j}(t)}{n_{i,j}} = \frac{R_{j,l}(t)}{n_{j,l}}, \frac{C_{j,k}(t)}{m_{j,k}} = \frac{C_{j,l}(t)}{m_{j,l}}, \forall t \in \mathbb{N}_0 \quad (6)$$

Similarly, the output channels have the linear relations on the output and demand functions as follows.

Constraint 4. *(MO linear relation)*

$$\frac{R'_{j,k}(t)}{n_{j,k}} = \frac{R'_{j,l}(t)}{n_{j,l}}, \frac{D_{j,k}(t)}{n_{j,k}} = \frac{D_{j,l}(t)}{n_{j,l}}, \forall t \in \mathbb{N}_0 \quad (7)$$

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, e.g., the paths ("$ch_{i,j} \rightarrow ch_{j,k}$" and "$ch_{j,l} \rightarrow ch_{l,j}$") in dashed lines in Figure 5. Based on the complete set of paths, our scheduling methodology fits the MIMO application models well.

Furthermore, for directed cyclic graphs as shown in Figure 5, the data tokens required for loop initialization can be explicitly modeled as the initial token offsets in Equation 1 and 2 directly.

V. Case study

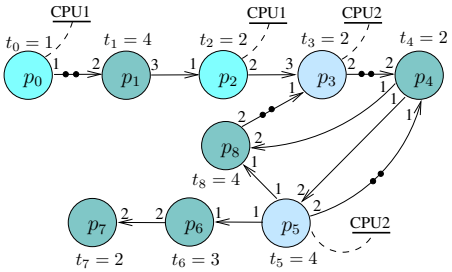


Fig. 6: The modem application partitioned for a multiprocessor CPU/FPGA architecture.

To evaluate the potential of our methodology, we use it on an application of voice-band data modem [3], which has 9 processes and 11 FIFOs. The application model with customized specification parameters is illustrated in Figure 6.

TABLE I: Comparison of scheduling methods.

	PAPS				CMBS		
	J	throughput	buffer	time[a]	throughput	buffer	time[a]
#1	1	4.8e-2	26	13	4.8e-2	23	270
#2	2	6.1e-2	29	16	6.3e-2	23	230
#3	3	6.7e-2	31	21	6.7e-2	23	240
#4	6	7.4e-2	32	26	7.4e-2	23	195
#5	8	7.6e-2	35	34	7.7e-2	23	190
#6	22	8.1e-2	49	61	8.3e-2	24	190
#7	100	8.3e-2	127	4202	9.1e-2	24	170

[a] It is the execution time (ms) in solutions finding.

We start from a manual mapping from the application to an multi-processor CPU/FPGA architecture. The labeled processes in the application model are partitioned and mapped onto multi-processors, i.e., p_0 and p_2 mapped onto CPU_1, and p_3 and p_5 mapped onto CPU_2. The rest processes are mapped onto parallel executing custom hardware.

We implement our constraint based minimal buffer scheduling (CMBS) methodology with the public domain constraint solving toolkit *Gecode* [9], which is a library written in C++. We compare our CMBS with the reference scheduling method, a trivially customized PAPS [3]. To make this comparison more reasonable, we implement the reference PAPS method in C++ as well and run both methods on HP xw4600 workstation to solve the scheduling problems for the modem application.

The experimental results are shown in Table I, which compares and quantifies the buffer requirement and the experimental execution time achieved by the schedules using PAPS and CMBS.

For PAPS, the application throughput may be improved by increasing the unroll factor J, i.e., the cases from $\#1$ to $\#6$. Correspondingly, we report the results of CMBS with some competitive throughput (no less than in PAPS).

However, the implementation cost of the periodic schedule increases with higher J and it is still in lack of a systematic way to find a finite J [2] yielding an optimal schedule. We simply increase J by 1 each time until the throughput improvement is negligible (i.e., less than 1e-4 to the throughput at $J = 1$), which is the 'optimal' case $\#7$ of PAPS with $J = 100$. In case $\#7$ of CMBS, we report the results achieved by a schedule with the maximal throughput guarantees.

From our experimental results, we summarize the observations made as follows.

- Our CMBS always requires less buffer storage space upon the equivalent application throughput guarantees.

[3] Instead of each process being scheduled onto any of the computation resource, a fixed mapping is adopted, i.e., a process can only be scheduled onto a particular CPU or custom circuit. This customization makes PAPS (proposed in [2]) fit our hybrid CPU/FPGA platform.

- In some case (when required throughput is high), our CMBS can achieve higher throughput guarantees than PAPS with much less buffer requirement. For instance, in case #7 CMBS requires 20% of buffer storage demanded by PAPS but gets 8% higher throughput guarantees.
- Our CMBS is more flexible to meet the vary required throughput guarantees. However, the throughput guarantees of PAPS are determined by the chosen J, which has quite limited options.
- The execution time in our CBMB methods is not sensitive to different throughput guarantees. In fact, when the throughput requirement is higher, the timing of constraint based analysis is shorter and might lead to less execution time. On the contrary, the execution time of PAPS increases fast when J and throughput are relatively higher.
- PAPS is faster in execution time when J is relatively small. However, our CMBS surpasses PAPS upon higher throughput requirement, e.g., in case #7.

VI. Conclusion

We have studied the problem of constructing schedules for real-time streaming applications with minimal buffer requirement on hybrid CPU/FPGA architectures. The problem has been formalized declaratively as constraint base scheduling, and can be effectively solved by constraint solvers. The experimental results show that our methodology performs significantly better than the traditional PAPS method in terms of buffer requirement. It is also flexible in the sense that it can be used to construct schedules to guarantee the required (feasible) throughput.

In the future, we plan to further reduce the storage space requirement by memory sharing between different buffer modules. We also plan to verify our methodology with an implementation of a multi-processor system using industrial RTOS on FPGA platforms.

APPENDIX

List of constraints on execution semantics

Using the application model in Figure 1 as example, we formalize the constraints below, in which $\forall t \in \mathbb{N}_0$. We assume the designer will specify some initial values (e.g., $C_{i,j}(0) = 0$ in this paper), which are thus not constrained (considered).

Constraint A-1. *(Token ratios) For process p_j, the $R'_{j,k}(t)$ and $C_{i,j}(t)$ follow the static input/output tokens ratio.*

$$R'_{j,k}(t) \cdot m_{i,j} = C_{i,j}(t) \cdot n_{j,k} \quad \text{(A-1)}$$

Constraint A-2. *(Computation latency) The incoming tokens in buffer $FIFO_{i,j}$ takes at least $t_{C,j}$ slots to be served by process p_j.*

$$R_{i,j}(t) - C_{i,j}(t + \Delta_t) \geqslant 0, \quad \forall \Delta_t \in [1, t_{C,j}] \quad \text{(A-2)}$$

Constraint A-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.*

$$D_{j,k}(t) = R_{j,k}(t + t_{C,j}) \quad \text{(A-3)}$$

Constraint A-4. *(Buffer size) The buffer size $\gamma_{i,j}$ of buffer $FIFO_{i,j}$ meets the maximum buffer space requirement.*

$$\gamma_{i,j} \geqslant B'_{i,j}(t) \quad \text{(A-4)}$$

Constraint A-5. *(Latency and tokens) Process p_j has computation latency $t_{C,j}$ and input/output data tokens $m_{i,j}$ and $n_{j,k}$.*

$$C_{i,j}(t + t_{C,j}) - C_{i,j}(t) = m_{i,j} \cdot K_j(t + t_{C,j}) \quad \text{(A-5)}$$

$$D_{i,j}(t + t_{C,j}) - D_{i,j}(t) = n_{j,k} \cdot K_j(t + t_{C,j}) \quad \text{(A-6)}$$

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

in which $K_j(t + t_{C,j})$ denotes the incremental properties of $C_{i,j}(t + t_{C,j})$ and $D_{i,j}(t + t_{C,j})$ in a period of $t_{C,j}$.

Acknowledgments

Thanks to Mikael Lagerkvist for the help on Gecode, and Dr. Zonghua Gu and anonymous reviewers for helpful comments to improve the content of the paper.

References

[1] D. Andrews, D. Niehaus, R. Jidin, M. Finley, W. Peck, M. Frisbie, J. Ortiz, E. Komp, and P. Ashenden, "Programming models for hybrid FPGA-CPU computational components: A missing link," *IEEE Micro*, vol. 24, no. 4, pp. 42–53, 2004.

[2] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers*, vol. C-36, no. 1, pp. 24–35, January 1987.

[3] ——, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, September 1987.

[4] P. K. M. Shuvra S. Bhattacharyya and E. A. Lee, *Software Synthesis from Dataflow Graphs*. Norwell, MA, USA: Kluwer Academic Press, 1996.

[5] R. Govindarajan, G. R. Gao, and P. Desai, "Minimizing buffer requirements under rate-optimal schedule in regular dataflow networks," *Journal of VLSI Signal Processing*, vol. 31, no. 3, pp. 207–229, July 2002.

[6] S. Stuijk, M. Geilen, and T. Basten, "Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs," in *DAC '06*, CA, USA, July 2006, pp. 899–904.

[7] J. Zhu, I. Sander, and A. Jantsch, "Performance analysis of reconfiguration in adaptive real-time streaming applications," in *Proceedings of IEEE workshop on ESTIMedia*, Atlanta, USA, October 2008.

[8] J. Madsen, K. Virk, and M. J. Gonzalez, "A SystemC-based abstract real-time operating system model for multiprocessor system-on-chip," in *Multiprocessor System-on-Chip*. Morgan Kaufmann, 2004.

[9] "Generic Constraint Development Environment (Gecode)," http://www.gecode.org/.

[10] R. L. Cruz, "Quality of service guarantees in virtual circuit switched networks," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 6, pp. 1048–1056, 1995.

[11] S. Chakraborty, S. Kunzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," in *DATE '03*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 190–195.