Technische Universiteit
Eindhoven
University of Technology

IDEA 2016: Integrating Dataflow, Embedded Computing, and Architecture

http://caes.ewi.utwente.nl/idea2016

Proceedings

Waheed Ahmad, Twan Basten, Robert de Groote, Alok Lele, and Orlando Moreira (eds.)

ES Reports
ISSN 1574-9517
ESR-2017-01
31 January 2017

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems

http://www.es.ele.tue.nl/esreports
esreports@es.ele.tue.nl

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems
PO Box 513
NL-5600 MB Eindhoven
The Netherlands

Contents

Foreword
Waheed Ahmad, Twan Basten, Alok Lele, Robert de Groote, and Orlando Moreira 1

Keynote: High Performance Embedded Computing Using Heterogeneous Computational Fabrics - The ALMARVI Vision and Beyond
Zaid Al-Ars 3

Multiprocessor Scheduling of a Multi-mode Dataflow Graph Considering Mode Transition Delay
Hanwoong Jung, Hyunok Oh and Soonhoi Ha 5
ACM ToDAES full paper doi: 10.1145/2997645

Worst Case Response Time Analysis of a Synchronous Dataflow Graph in Multiprocessor Real-time Systems
Junchul Choi and Soonhoi Ha 7
ACM ToDAES full paper doi: 10.1145/2997644

Flexible and Trade-Off-Aware Constraint-Based Design Space Exploration for Streaming Applications on Heterogeneous Platforms
Kathrin Rosvall and Ingo Sander 9

Performance Estimation of Template-based Gesture Recognition on Multi-Core Architectures Using Scenario-Aware Dataflow Graphs
Florian Grützmacher, Benjamin Beichler, Bart Theelen and Christian Haubelt 11

Towards State-Based RT Analysis of FSM-SADFGs on MPSoCs with Shared Memory Communication
Ralf Stemmer, Maher Fakih, Kim Grüttner and Wolfgang Nebel 13

A Model-Driven Framework for Hardware-Software Co-design of Dataflow Applications
Waheed Ahmad, Bugra Mehmet Yildiz, Arend Rensink and Marielle Stoelinga 15

Extended Abstract: Process Networks for Reactive Streaming with Timed-automata Implementation
Peter Poplavko, Dario Socci, Rany Kahil, Marius Bozga and Saddek Bensalem 17
IDEA 2016 best interactive presentation award

Analysis and Visualization of Execution Traces of DataFlow Applications
Hadi Alizadeh Ara, Amir Behrouzian, Marc Geilen, Martijn Hendriks, Dip Goswami and Twan Basten 19

Mode-controlled Dataflow based Buffer Allocation for Real-time Streaming Applications Running on a Multi-processor without Back-pressure
Hrishikesh Salunkhe, Alok Lele, Orlando Moreira and Kees van Berkel 21

Symbolic computation of the latency for dataflow graphs
Pascal Fradet, Alain Girault and Adnan Bouakaz 23
ACM ToDAES full paper doi: 10.1145/3007898

Probabilistic Model Checking for Uncertain Scenario-Aware Data Flow
Joost-Pieter Katoen and Hao Wu 25
IDEA 2016 Best paper award
ACM ToDAES full paper doi: 10.1145/2914788

Foreword

Welcome to the proceedings of abstracts of IDEA 2016, the workshop on Integrating Dataflow, Embedded computing and Architecture. IDEA 2016 was held at CPS Week 2016, Vienna, Austria on April 11th 2016, as the second workshop in a series, following the first edition held in Amsterdam, the Netherlands in 2015.

Over the years, dataflow has been gaining popularity among Embedded Systems researchers around Europe and the world. The goal of the workshop series is to provide a platform to researchers and practitioners to present work on modelling and analysis of present and future high performance embedded computing applications and architectures using dataflow.

For 2016, we decided to broaden the impact of research on dataflow by publishing mature work submitted to IDEA in a special section of ACM Transactions on Design Automation of Embedded Systems, ACM ToDAES. For this purpose, we introduced two tracks, full paper submissions for ToDAES, and interactive presentation (IP) submissions for presentation at IDEA only. This was well-received by the dataflow community, and 9 full papers and 7 IPs were submitted. Out of all submissions, 11 were selected for presentation at IDEA 2016. All submissions were reviewed by three reviewers. The program was further greatly enriched with the keynote talk of Zaid Al-Ars (Delft University of Technology, Netherlands).

These conference proceedings contain 2-page abstracts for the 11 submissions accepted for IDEA 2016. The proceedings were originally published as part of the electronic ES week proceedings provided to ES week participants on USB stick. To make these abstracts available to a wider community, we now have collected the abstracts in this ES Report published by Eindhoven University of Technology.

Several of the abstracts contained in these proceedings have been further extended to full papers for the ACM ToDAES special section. The special section papers have been published in Volume 17, Issues 1 and 2, of ACM ToDAES. The special section is the result of an open call for papers, followed by a thorough review process. From ten submissions, in four review rounds, five submissions were accepted. Four of those submissions originate from IDEA workshop papers. The table of contents of these abstract proceedings provides pointers to these full papers.

At the end of the workshop, two awards were given, one for best full paper and one for best IP. The best IP award was won by Peter Poplavko for the abstract “Process Networks for Reactive Streaming with Timed-automata Implementation” and its presentation. The paper ”Probabilistic Model Checking for Uncertain Scenario-Aware Data Flow” by Joost-Pieter Katoen and Hao Wu was selected as the best full paper submission. This paper is available in ACM ToDAES 17(1).

The IDEA workshop and the resulting ACM ToDAES special section could not have become a reality without the help of many others. First of all, the Program Committee (PC) did the hard work to give many detailed and very helpful reviews for both the workshop and the ToDAES special section. Both the IDEA and the CPS week organizing committee contributed to the organization of the workshop. The ACM ToDAES editor-in-chief Naehyuck Chang and Annie Yu supported the realization of the ToDAES special section. Financial support for the IDEA 2016 workshop keynote and the best paper and best IP awards was provided by 3TU NIRICT, https://www.3tu.nl/nirict. The workshop and the special section were organized by the ALMARVI project under the ARTEMIS grant agreement no. 621439 and by the FP7 project SENSATION, FP7-ICT-2011-8, grant agreement no. 318490. We take this opportunity to thank everyone who contributed in making the workshop and the special section a success.

January 2017

Waheed Ahmad
Twan Basten
Alok Lele
Robert de Groote
Orlando Moreira

IDEA 2016 KEYNOTE

Zaid Al-Ars, TU Delft, NL

High Performance Embedded Computing Using Heterogeneous Computational Fabrics — The ALMARVI Vision and Beyond

Abstract:

In the past decade, demand for high performance computing has been steadily increasing throughout the computing spectrum, all the way from high-end supercomputers, down to small handheld devices. To facilitate this need in the field of embedded computing, various concepts have been proposed to bring high performance architectures to the embedded domain. However, the diverse and in many cases stringent application requirements in this domain (such as low power, portability, form-factor limitations, etc.) have made it difficult to come up with a single design paradigm that satisfies the wide variation of available applications. This results in the need to develop custom-made solutions for each application, with an associated high cost of design, debug and verification of the hardware and application software. In this talk, we discuss the current trends in high performance embedded systems, where heterogeneous computing plays an important role in satisfying the computational requirements of the system on the one hand, while software abstraction ensures easy programmability and portability of the developed application software. We also present the ALMARVI project vision and ongoing activities as an example of this trend. Finally, we show how this trend into heterogenous computing is also penetrating the high performance computing field, resulting in the convergence of various aspects of the computational spectrum from the high-end to the embedded world.

Multiprocessor Scheduling of a Multi-mode Dataflow Graph Considering Mode Transition Delay

Hanwoong Jung
Department of Computer Science and Engineering
Seoul National University
Seoul 151-744, Republic of Korea
Email: jhw7884@iris.snu.ac.kr

Hyunok Oh
Department of Information System
Hanyang University
Seoul 133-791, Republic of Korea
Email: hoh@hanyang.ac.kr

Soonhoi Ha
Department of Computer Science and Engineering
Seoul National University
Seoul 151-744, Republic of Korea
Email: sha@snu.ac.kr

***Abstract*—Several extensions of a dataflow model have been proposed to support dynamic behavior changes while preserving static analyzability. While there exist several scheduling techniques for these extensions, to our best knowledge, no one allows task migration between different execution modes. By observing that the resource requirement can be reduced if task migration is allowed, we propose a multiprocessor scheduling technique of a multi-mode dataflow graph considering task migration between modes. To satisfy the throughput constraint, the proposed technique calculates the actual throughput requirement of each mode and the output buffer size for tolerating throughput jitter.**

I. Introduction

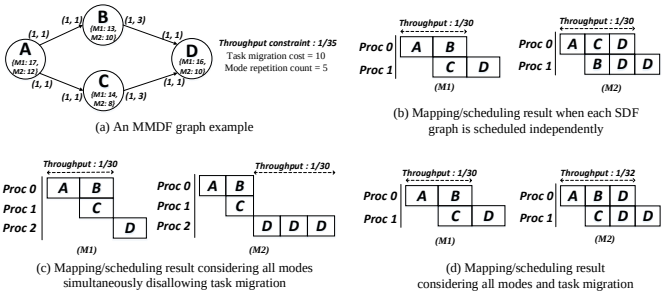
Synchronous dataflow model [1] is widely used for model-based design methodology, because it enables us to construct a static schedule of actors onto a given hardware platform so that we can estimate the performance and resource requirement of an application. But the SDF model has a severe restriction which cannot express the dynamic behavior of an application.

To overcome this limitation, several extensions have been proposed to the SDF model, including FSM-based scenario-aware dataflow (FSM-SADF) [2], parameterized SDF (PSDF) [3], MTM-SADF [4], mode-aware dataflow (MADF) [5], and so on. They all assume that an application has a finite number of behaviors (or modes) and each behavior (mode) can be represented by an SDF graph. We denote those MoCs (Models of Computation) as multi-mode dataflow (MMDF) graphs in this paper and define a representative MMDF model that can be implemented by any specific extension.

In this paper, we propose a multiprocessor scheduling technique of a multi-mode dataflow graph allowing task migration between modes.

II. Motivational Example

Because an MMDF graph is composed of SDF graphs in multiple modes, the static schedule of the SDF graph on each mode can be constructed under a given throughput constraint. Since the mode transition delay degrades the average throughput of an MMDF application, however, the throughput constraint can be violated even if the throughput of each mode is higher than the throughput constraint. So, the mode transition delay should be taken into account.



(a) An MMDF graph example

(b) Mapping/scheduling result when each SDF graph is scheduled independently

(c) Mapping/scheduling result considering all modes simultaneously disallowing task migration

(d) Mapping/scheduling result considering all modes and task migration

Fig. 1: Motivational example of task migration

While there exist several techniques to schedule a multi-mode dataflow graph with considering the mode transition delay, the existing approaches do not allow task migration between modes [5][6]. Figure 1 (c) shows a scheduling result without task migration. It requires more processors than those that allow task migration. However, we observe that the resource requirement for a given throughput constraint can be reduced if task migration is allowed. Figure 1 (d) shows a mapping and scheduling result produced by the proposed technique. It requires 2 processors and only 10 additional time units for task migration, while Figure 1 (b) requires 30 time units when each mode is scheduled independently.

III. Throughput requirement analysis

For simple coordination of task migration and conservative estimation of the mode transition overhead, we assume that mode transition and task migration is performed in a *blocking* fashion: the task schedule is *blocked* at the mode transition boundary. Then we can compute the mode transition delay from each mode, $MaxTransDelay(m)$, conservatively and the effective throughput of an MMDF graph with a given static scheduling results of modes. We can formulate the throughput requirement of each mode, $TR(m)$, to satisfy a given throughput constraint, $ThrConst$, as follows:

$$TR(m) = \frac{ThrConst \times MRC(m)}{MRC(m) - (MaxTransDelay(m) \times ThrConst)}$$

where $MRC(m)$ is the minimum number of iterations executed in mode m.

Since the mode transition delay causes the jitter of output production in an MMDF application, an output buffer is adopted to tolerate the jitter. To determine the buffer size, we compute the arrival curves of the input and the output streams in the buffer as shown in Figure 2. The black solid line represents the minimum arrival curve of the input stream which presents the number of generated samples to the output buffer. From the arrival curves, we obtain the minimum output buffer size which is the maximum difference between the curves in every time interval (Δt).

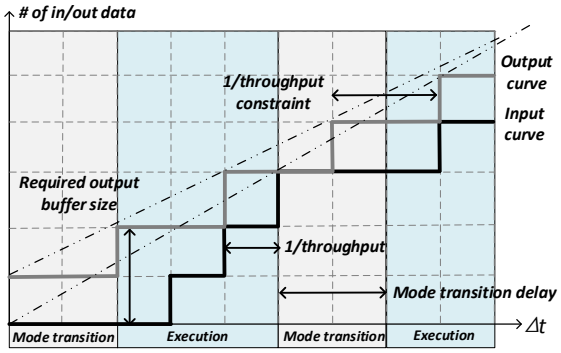


Fig. 2: Arrival curves for input and the output streams

IV. Design Space Exploration

For MMDF scheduling problem, we adopt a multi-objective genetic algorithm(GA). The main objective is to minimize the number of used processors in all modes. The GA framework also aims to minimize the overall task migration cost as the secondary objective. The reduction of task migration will save energy consumption of the system, and reduce the network traffic in an NOC architecture. Therefore, it is very desirable to reduce the total task migration cost in an MMDF graph considering all mode transition scenarios.

V. Experimental results

To prove the viability of the proposed framework, we conduct experiments with four synthetic examples and three real applications: H.264 decoder, vocoder and LTE receiver algorithms. We compare the proposed technique with two different heuristics. The first heuristic schedules SDF graphs independently and performs the processor renaming heuristic. We denote this technique as *Base*. The second approach fixes task mapping in all modes disallowing task migration as the existing approaches usually assume. This technique is denoted as *Fixed*.

Figure 3 shows the experimental results for all applications. The results show that the *Proposed* method requires no more processors than *Base* and *Fixed* approaches in all cases. The *Fixed* approach requires more processors in most cases than *Base* and *Proposed* approaches. Since the *Fixed* approach does not allow task migration, the number of required processors to meet the given throughput constraint is independent of the task migration cost. Also, as task migration cost ($MC(t)$) increases, the *Base* approach requires more processors than

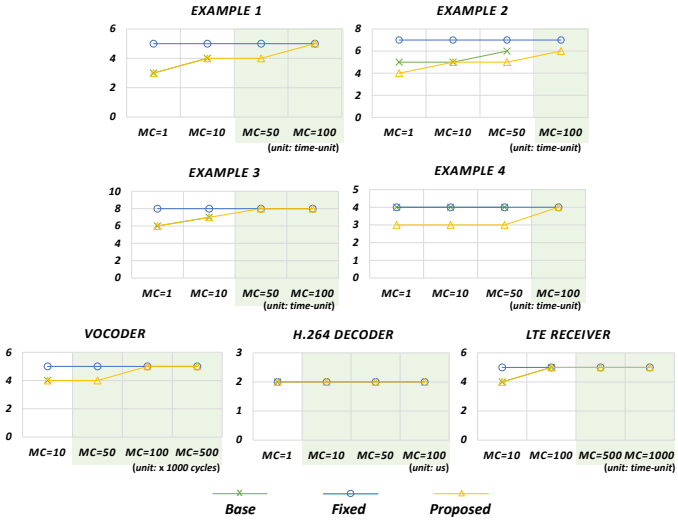


Fig. 3: Comparison results: *Base*, *Fixed* and *Proposed*

the *Proposed* approach or could not find a feasible solution due to large task migration cost values as shown in Figure 3.

VI. Conclusion

In this paper, we address the multiprocessor scheduling problem of a multi-mode dataflow (MMDF) graph allowing task migration. First we derive a formula to compute a conservative bound on the mode transition delay between two modes and compute the required throughput performance of each mode to satisfy the given throughput constraint. We also compute the required output buffer size to tolerate the output jitter. Finally we propose a mapping/scheduling framework based on a genetic algorithm to minimize the number of processors while keeping the throughput constraint. Experimental results prove the superior performance of the proposed technique over the related work.

References

[1] E. A. Lee *et al.*, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[2] S. Stuijk *et al.*, "Scenario-aware dataflow: modeling, analysis and implementation of dynamic applications," in *Embedded Computer Systems (SAMOS), 2011 International Conference on*, 2011, pp. 404–411.

[3] B. Bhattacharya *et al.*, "Parameterized dataflow modeling for dsp systems," *Signal Processing, IEEE Transactions on*, vol. 49, no. 10, pp. 2408–2421, 2001.

[4] H. Jung *et al.*, "Dynamic behavior specification and dynamic mapping for real-time embedded systems: hopes approach," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 4s, 135:1–135:26, Apr. 2014.

[5] J. T. Zhai, "Adaptive streaming applications : analysis and implementation models," *ser. PHD Thesis, Universiteit Leiden*, 2015.

[6] M. Geilen *et al.*, "Predictable dynamic embedded data processing," in *Embedded Computer Systems (SAMOS), 2012 International Conference on*, 2012, pp. 320–327.

Worst Case Response Time Analysis of a Synchronous Dataflow Graph in Multiprocessor Real-time Systems

Junchul Choi and Soonhoi Ha
Department of Computer Science and Engineering, Seoul National University, Seoul, Republic of Korea
{hinomk2, sha}@iris.snu.ac.kr

***Abstract*—In this paper, we propose a novel technique that estimates a tight upper bound of worst case response time (WCRT) of a synchronous dataflow (SDF) graph when the SDF graph shares processors with other real time tasks. The proposed technique combines two techniques in a novel way: schedule time bound analysis (STBA) and response time analysis (RTA). In addition, we propose a genetic algorithm (GA) based framework using the proposed WCRT analysis technique to accommodate the maximum number of real time tasks in a given hardware platform.**

I. Introduction

Synchronous dataflow (SDF) model [1] enables us to determine the mapping and scheduling of tasks at compile-time so that we can estimate the throughput and latency of the application. If an SDF application is executed under a self-timed or static-assignment policy that does not preserve the timing information, the performance estimated from static mapping and scheduling at compile-time is not guaranteed to be met at run-time. The problem addressed in this paper is to estimate the worst case response time of an SDF graph when it shares the processors with other real-time tasks.

The approach based on the response time analysis (RTA) [2] has been proposed to check the schedulability of dependent tasks but supports only preemptive processor. Compositional analysis [3] achieves scalability, but it sacrifices estimation accuracy. A holistic WCRT analysis approach, called schedule time bound analysis (STBA) [4], computes the conservative schedule time bound for each task considering all possible scheduling patterns. But it suffers from the scalability problem.

In this paper, we propose a novel WCRT estimation technique that combines the STBA for the interference between task instances in the same SDF application and the RTA for the interference from the other real time tasks. Using the proposed WCRT estimation technique, we propose a genetic algorithm (GA) based framework to accommodate the maximum number of other real-time tasks for a given hardware platform.

II. Motivational Example

Fig. 1 (a) shows an example SDF graph and a schedule of the SDF graph is given in Fig. 1 (b). As shown in Fig. 1 (c), The schedule is first transformed into a directed acyclic graph, called task instance dependency graph (TIDG) that expresses data dependency and also the scheduling order made under the self-timed scheduling policy. Assume the schedule is executed under a static assignment policy on non-preemptive processors. If node E finishes with 1 execution time, the execution order of C3 and F2 is switched and the overall response time becomes larger than expected at compile time as shown in Fig. 1 (d). This example confirms the need of WCRT analysis when a self-timed or static assignment policy is used.

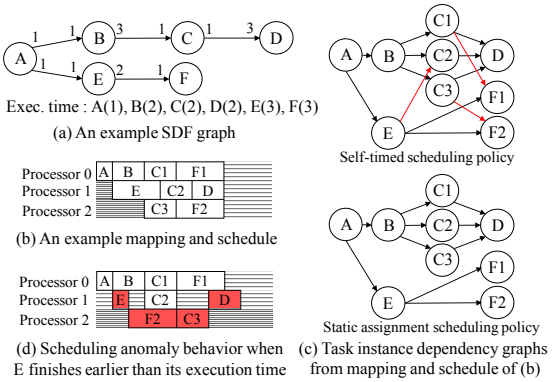


Fig. 1. Task instance dependency graphs from the given scheduling results and scheduling anomaly behavior at run-time

The second problem is defined to accommodate real-time tasks into the system as many as possible by changing task mapping and priorities. Suppose that we have two real-time tasks τ_0 and τ_1 that have 2 execution times in addition to the SDF graph. Schedulability of the system varies according to the mapping and priority settings as shown in Fig. 2. The system becomes schedulable when mapping of τ_0 and τ_1 is selected as illustrated in Fig. 2 (c).

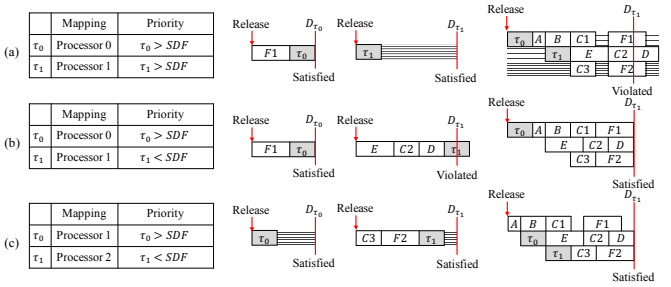


Fig. 2. Three DSE solutions and corresponding worst case execution scenarios of applications

III. Proposed Technique

We first compute the schedule time bound of each TIDG task τ conservatively applying the STBA technique that finds the schedule time bounds of tasks. Release time bound

(τ^{minR}, τ^{maxR}) is determined from the finish time bounds of predecessors. If a higher priority task possesses the processor or a lower priority task runs on the non-preemptive processor at the release time of τ, τ is scheduled after it finishes. τ is further delayed if any higher priority task is released before τ starts. Start time bound (τ^{minS}, τ^{maxS}) is computed considering the best case and worst case schedule. We determine the finish time bound (τ^{minF}, τ^{maxF}) by computing the minimum and maximum preemptions after the start time.

Next, we use the RTA method to conservatively estimate the interference from other real time tasks with dynamic offset. If we compute the maximum interference independently per each task, the total interference along the SDF execution can be highly overestimated. For a tight estimation, we compute the relative distance from the release time of a target task to the next possible request appearance of the preemptor task and use the computed distance as a task offset in the RTA method. We conservatively compute the minimum distance considering all possible execution paths in the TIDG graph so that the interference along the entire SDF execution is tightly estimated. Then the schedule time bounds are modified accordingly. The worst case response time of the SDF graph is computed as the maximum among τ^{maxF} of all tasks.

For analyzing the schedulability of real-time tasks, we transform dependent TIDG tasks to a set of independent tasks with release jitter $\tau^{maxR} - \tau^{minR}$ and use the conventional RTA method that supports only independent tasks.

Based on the proposed WCRT estimation technique, we present a GA framework that solves a scheduling problem whose objective is to accommodate as many real time tasks as possible into the system while satisfying all deadline constraints of applications by determining mapping and priorities of real-time tasks.

IV. Experiments

We generate 100 synthetic examples that parameters are randomly selected. We compare the WCRT estimation of the SDF graph with Y&W method [2], pyCPA [3], and STBA [4]. Fig. 3 shows the WCRT estimation gap ratio between the proposed technique and other approaches. The proposed technique gives tighter bounds than other approaches in almost all examples. A looser bound than the Y&W method was found for two examples only with very small estimation gap, at most 2.820%. pyCPA provides on average 3.58 times WCRT results.

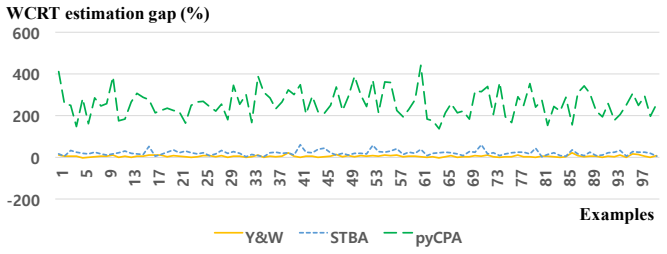


Fig. 3. WCRT estimation gap between the proposed technique and the reference techniques Y&W, STBA, and pyCPA for 100 synthetic examples

The performance of the proposed GA framework is verified with H.263 encoder benchmark from SDF3 [5] and synthetically generated real-time tasks. Fig. 4 shows the GA result for three techniques while increasing the deadline. From the results, we can notify that the number of accommodated tasks is converged to 40 as the deadline increases. It is because processor utilization becomes the bottleneck to schedule the real-time tasks eventually while latency constraints are the bottleneck initially. It shows that the STBA approach accommodates the smallest number of tasks as the deadline increases, since it tends to overestimate the WCRT when there are may real-time tasks in the system. Compared to the other approach, the proposed technique shows better performance.

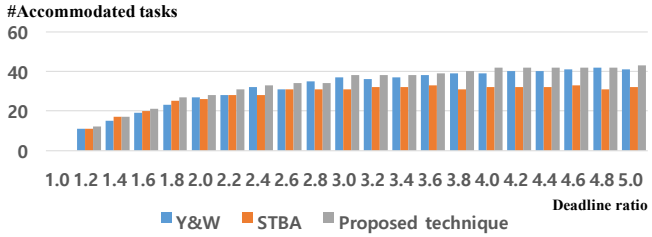


Fig. 4. The number of accommodated real time tasks while increasing the deadline of H.263 benchmark

V. Conclusion

In this paper, we propose a novel technique that estimates a tight upper bound of worst case response time of an SDF graph when the SDF graph shares processors with real time tasks and it is executed at run-time with a self-timed or a static-assignment policy. we compute the schedule time bounds for all TIDG tasks, taking into account the interference from other real time tasks by performing response time analysis. Based on the proposed WCRT estimation technique, we present a GA framework that solves a scheduling problem to accommodate a maximal set of real time tasks in a given hardware platform. With experimental results, it is verified that the proposed technique estimates the WCRT more tightly. In addition, tighter estimation allows us to accommodate more real time tasks than other techniques. It remains as a future work to consider multiple SDF graphs that share processors with each other as well as real-time tasks.

References

[1] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sept 1987.

[2] T.-Y. Yen and W. Wolf, "Performance estimation for real-time distributed embedded systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 11, pp. 1125–1136, Nov. 1998. [Online]. Available: http://dx.doi.org/10.1109/71.735959

[3] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the symta/s approach," *Computers and Digital Techniques, IEE Proceedings* -, vol. 152, no. 2, pp. 148–166, Mar 2005.

[4] J. Kim, H. Oh, J. Choi, H. Ha, and S. Ha, "A novel analytical method for worst case response time estimation of distributed embedded systems," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, May 2013, pp. 1–10.

[5] S. Stuijk, M. Geilen, and T. Basten, "Sdf3:sdf for free," 2006. [Online]. Available: http://www.es.ele.tue.nl/sdf3/download/examples.php

Flexible and Trade-Off-Aware Constraint-Based Design Space Exploration for Streaming Applications on Heterogeneous Platforms

Kathrin Rosvall and Ingo Sander
KTH Royal Institute of Technology, School of ICT, Electronics and Embedded Systems
Stockholm, Sweden
{krosvall, ingo}@kth.se

I. Introduction

Design space exploration (DSE) is a critical step in the design process of real-time multiprocessor systems. To provide performance guarantees, DSE methods need to be based on formal models and platform architectures with guaranteed quality of service (QoS). However, current industrial multiprocessor architectures do not provide guaranteed QoS, and are very difficult to analyze due to shared resources and caches [1]. Furthermore, current industrial practice lacks suitable formal application and platform models, and as a consequence DSE in industry is often conducted manually or cannot give performance guarantees.

A promising approach to overcome the present situation is to combine formal analyzable models based on the theory of models of computation (MoC) [2] with predictable architectures, which can give guaranteed QoS with respect to timing [3], [4]. Especially, there has been a lot of interest in synchronous data flow (SDF) [5] and cyclo-static data flow [6] because of well-developed analysis techniques.

Due to its complexity, the problem of mapping and scheduling streaming applications on heterogeneous MPSoCs under real-time and performance constraints has traditionally been tackled by incomplete heuristic algorithms [7]–[9]. Recently, approaches based on constraint programming (CP) [10]–[12] have demonstrated their potential as complete methods for finding optimal mappings, mainly concerning throughput. However, so far none of the available CP approaches considers the trade-off between throughput and buffer requirements or throughput and energy consumption.

The DSE problem addressed in this paper is to find implementations with performance guarantees for a set of applications on a shared multiprocessor platform. This involves the interdependent activities of *mapping*, *scheduling* and *performance analysis*. The analyses concern performance metrics of individual applications, such as for example throughput and latency, as well as global, system-wide metrics like energy consumption and area cost.

The presented DSE is a pure compile-time method, providing static schedules. In order to enable compile time analyses, the method relies on formal and analyzable models of computation (MoCs) for the applications and predictable target platform models.

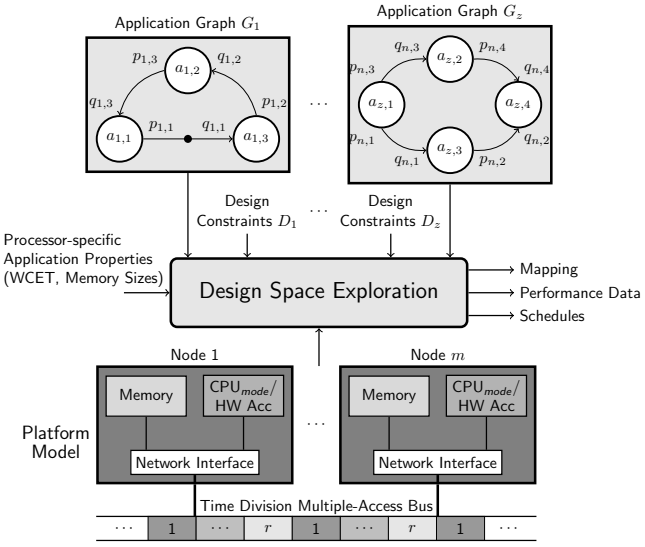


Figure 1. Overview over the DSE Framework

Apart from the aforementioned integration of buffer sizing and energy consumption into the CP model, this work extends the current state-of-the-art in the field by presenting a novel two-step approach for the solver that combines the advantages of heuristics with the completeness property of constraint programming to enable trade-off aware DSE.

II. Constraint-Based Design Space Exploration

Figure 1 gives an overview of the DSE framework developed for the work in this paper. It takes applications as SDF graphs with design constraints, a platform description, and WCET and memory size figures as input parameters, generates the corresponding CP model for the DSE problem, invokes the CP solver and outputs the results in terms of mapping, schedules and performance data.

The design constraints serve different important purposes. Most importantly, they specify the goal of the exploration by combining constraints of mixed-critical compliance levels on chosen performance metrics. The goal of the exploration can involve multiple performance metrics for optimization,

that can either be combined into a, optionally weighted, cost function or used for multi-step exploration. Satisfy-constraints can be specified independently of and in addition to optimization goals. Currently, the DSE framework supports design constraints on latency and throughput of individual applications, number of processors allocated for the mapping and utilization, energy consumption, area cost and monetary cost of individual processors and the system as a whole. Apart from performance metrics, design constraints can also address mapping and scheduling decisions.

Considering the platform model, a predictable bus platform, based on time-division multiple-access (TDMA), is assumed for this paper. That means the bus is divided into a round of time slots and each processing unit receives one or more dedicated slots. With this model, each processor has an own share of the bus and hence accesses from different processors do not interfere with each other. The processors of the target platform can be instantiated in different modes, creating a trade-off between factors like power consumption, area cost, monetary cost and performance, e.g. in terms of worst-case execution times (WCETs) and hence throughput. The model is flexible and the modes can be designed arbitrarily, currently considering power consumption, area cost, monetary cost, local memory size and computation speed. Apart from general purpose processors, the MPSoC-platform can also host dedicated hardware accelerators for actor functions, as e.g. an FFT. As additional input to the DSE framework, the WCET and memory consumption for all valid combinations of actors and processing elements, and modes if applicable, need to be provided.

The core of the DSE framework is a CP model that combines the ideas and concepts of constraint programming with dataflow modeling and analysis to solve the design space exploration problem. The CP variables reflect a *mapping- and scheduling-aware graph* (MSAG), which captures all input applications and the implications of mapping and scheduling decisions taken during the exploration. The different CP variables forming the MSAG represent actors, channels, tokens and actor properties, respectively. The schedules described by the CP model are static order-based schedules, i.e. only the order in which actors execute on each node is fixed, but not the start or end times. Notably, a specialized throughput propagator has been implemented as a means to enable throughput analysis in the CP model.

The CP model is passed to a constraint solver, which performs the intertwined steps of *propagation*, *branching* and *search*. Propagation removes values from the variable domains that are in conflict with the constraints. Branching builds a search tree from the remaining alternatives in the variable domains after propagation. The search operates on the created tree to find solutions that satisfy all constraints. Finally, the framework outputs a, or, depending on the design constraints, a number of mappings with schedules and corresponding performance data.

III. Experiments and Results

In order to evaluate the potential of the DSE model, four streaming applications have been used: a Sobel filter, a SUSAN filter, a RASTA-PLP application and a JPEG encoder. Four sets of experiments have been conducted, based on four different variations of the platform model and variations of the optimization goals. A novel two-step approach has been applied, which utilizes the advantage of heuristics, while still keeping the completeness property of CP. This approach also enables the combination of multiple optimization goals, as e.g. throughput and energy consumption. A standard, parallel branch-and-bound (BAB) search engine was used by the solver. In summary, the framework was able to provide optimal solutions for several cases and good, potentially optimal solutions for the remaining cases, where optimality could however not be proven within a specified time limit.

IV. Conclusion

The presented trade-off aware design space exploration combines the advantages of heuristics with the completeness property of constraint programming. The potential of the method has been illustrated by a case study with four typical signal processing applications and individual constraints sharing the same multiprocessor platforms. The results of the experiments illustrate that the novel two-stage approach reduces analysis time without sacrificing completeness.

Directions for future work include the integration of additional models of computation into the DSE framework and the extension of the platform model with shared memories, networks-on-chip architectures and TDM-scheduling.

References

[1] R. Wilhelm *et al.*, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM TECS*, vol. 7, 2008.

[2] E. A. Lee and A. Sangiovanni-Vincentelli, "A framework for comparing models of computation," *IEEE TCAD*, vol. 17, 1998.

[3] B. Lickly, I. Liu, S. Kim, H. D. Patel, S. A. Edwards, and E. A. Lee, "Predictable programming on a precision timed architecture," in *CASES*, 2008.

[4] A. Hansson, K. Goossens, M. Bekooij, and J. Huisken, "CoMPSoC: A template for composable and predictable multi-processor system on chips," *ACM TODAES*, vol. 14, 2009.

[5] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," in *Proceedings of the IEEE*, vol. 75, 1987.

[6] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cyclo-static data flow," in *ICASSP*, vol. 5, 1995.

[7] S. Stuijk, T. Basten, M. Geilen, and H. Corporaal, "Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs," in *DAC*, 2007.

[8] O. Moreira, J.-D. Mol, M. Bekooij, and J. Van Meerbergen, "Multiprocessor resource allocation for hard-real-time streaming with a dynamic job-mix," in *IEEE RTAS*, 2005.

[9] M. Fakih, K. Grüttner, M. Fränzle, and A. Rettberg, "Towards performance analysis of SDFGs mapped to shared–bus architectures using model–checking," in *DATE*, 2013.

[10] A. Bonfietti, M. Lombardi, M. Milano, and L. Benini, "Throughput constraint for synchronous data flow graphs," in *Integr. of AI and OR Techn. in Constr. Progr. for Combin. Optim. Problems*, 2009, vol. 5547.

[11] A. Bonfietti *et al.*, "An efficient and complete approach for throughput-maximal SDF allocation and scheduling on multi-core platforms," in *DATE*, 2010.

[12] K. Rosvall and I. Sander, "A constraint-based design space exploration framework for real-time applications on MPSoCs," in *DATE*, 2014.

Performance Estimation of Template-based Gesture Recognition on Multi-Core Architectures

Florian Grützmacher, Benjamin Beichler, Christian Haubelt
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18051 Rostock, Germany
Email: florian.gruetzmacher2@uni-rostock.de

Bart Theelen
Embedded Systems Innovation by TNO
PO Box 6235, 5600 HE Eindhoven, Netherlands
Email: bart.theelen@tno.nl

***Abstract*—In our work, we present a Scenario-Aware Dataflow model of a template-based hand gesture recognition system based on Dynamic Time Warping and Exponential Moving Average signal filtering. Our model enables us to estimate the important characteristics of the system like real-time capabilities of the online recognition system, response times in different scenarios, and the average utilization of the computing platform, which directly influences the power consumption of the system.**

I. Introduction

Gesture recognition is a promising way for Human-Computer Interaction especially for mobile platforms. As multi-core implementations easily get very complex, design decisions have to be taken carefully in order to meet performance and energy constraints. Formal model-based development can dramatically improve the development process.

Our work is based on Scenario-Aware Dataflow (SADF) graphs, which were introduced in [1] and have already been applied for MP3 and MPEG-4 decoders [2] [3]. In our work we use SADF graphs to select proper parallelization configurations of an online gesture recognition system regarding real-time requirements and latency when performed on multi-core processors. Scenarios are used to distinguish between different processing modes which depend on the average fluctuation of the sensor signals, calculated by the Exponential Moving Average (EMA) of the difference between two samples.

II. SADF model of the Gesture Detection Algorithm

In [4] an extended dataflow model based on Enable/Invoke Dataflow (EIDF) graphs has been introduced to formally describe a gesture detection algorithm. However, this model has no capabilities to capture timing information. In order to substantiate design decisions as early as possible in the development process of gesture recognition systems, we used Scenario-Aware Dataflow (SADF) as model of computation, since it can capture both timing information of actors in form of execution times and variations in execution times, production, and consumption rates of actors based on its scenarios. We use two different scenarios, namely an enabled and a disabled recognition process, based on the EMA of the preceding sliding window. To model the characteristics of input signals, i.e., the occurrence of high fluctuating sensor signals, we adjust the transition probabilities of the Markov chain in the SADF model. Depending on the scenario, the processing of a sliding window can have different response times.

As described in [4], the gesture detection is based on a sliding window approach. Distances of each sliding window to recorded references of gestures are calculated and the smallest is selected. This calculations can be distributed over all cores of a tile, while multiple tiles can process consecutive sliding windows in parallel. In the dataflow model, distance calculations performed on the same core are modeled by chained actors which fire sequentially, whereas different chains model the execution on different processor cores. Based on this, we introduce two scenarios in which tiles can act, namely the gesture recognition is done for a sliding window because high fluctuations occur in the sensors signals, and no gesture recognition is done and only the EMA is calculated. The probabilities for scenarios can be calibrated to real situations in which gestures are performed or not.

In order to obtain a cyclo-static behavior sending sliding windows to tiles and receiving their results, additional scenarios are introduced. Those control the consumption and production rates of actors, that distribute the sliding windows to particular tiles (`WindowDist`) and evaluate the distances calculated by the cores of a particular tile (`Eval`).

In Fig. 1 the concept is shown for a configuration of 3 tiles. In scenario *SentTo3* actor `WindowDist` only produces a token on those input edges, which correspond to the third tile. Equivalently, actor `Eval` consumes a token from this tile in its scenario *GetFrom3* which is synchronized with scenario *SentTo3* of `WindowDist`. For the sake of clarity, the detector is shown as a separate actor, while in our model the actor *Eval* in fact is the detector which contains the Markov chain and controls all actors, including itself. Due to the cyclo-static activation of tiles, actors belonging to a tile fire fewer times than the `Eval` actor. Since detectors must per definition produce at least one control token to each control channel and actors must consume exactly one control token, this would lead to buffer overflows on control channels to those actors in configurations with more than one tile. To avoid this, an *inactive* scenario is introduced to all actors belonging to a tile. In this scenario, no tokens are consumed/produced

with an execution time of 0 time units. This enables those actors to consume a control token, by neither delaying any other following firings of those actors nor producing any tokens to synchronization edges. While the tile processing the next sliding window receives *highEMA* or *lowEMA* control tokens, according to the transition probabilities of the Markov chain, all other tiles actors receive *inactive* control tokens. This concept is shown in Fig. 1. For the sake of clarity, the states and transition probabilities for *highEMA* and *lowEMA* scenarios are not shown. Our model has been annotated with the worst-case execution times based on measured values of the corresponding functions in our implementation.

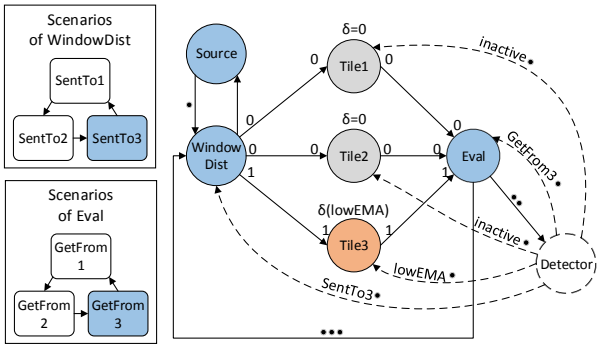


Fig. 1. Illustration of Tiles 1 and 2 being inactive, while Tile 3 is performing *lowEMA*.

A. Processor Utilization

We have extended the performance analysis tool for SADF in SDF^3 [3] with the ability to compute the scenario occurrence probabilities for each actor. This feature enables us to calculate the ratio of computation time of a worker core within an observed time interval to the length of that interval, which corresponds to the processor utilization. We deduce this time interval from the difference of the arrival times of two consecutive sliding windows at a particular core. Since all actors mapped to a core always fire consecutively synchronized by the feedback channel from `Eval` to `WindowDist`, the computation time of the processor core is the cumulative average execution (CAE) time of those actors. The CAE is calculated with equation Eq. 1 with M being the number of actors a_{ic} mapped onto a specific core c and S_i being the number of scenarios of that specific actor. The execution time of actor a_{ic} in scenario s_j is denoted as $\delta_i(j)$ and $p_i(j)$ is the occurrence probability of scenario s_j for actor a_{ic} which we get from the extension we introduced.

$$CAE = \sum_{1<i\leq M} \sum_{1<j\leq S_i} p_i(j) * \delta_i(j) \qquad (1)$$

In a configuration with N tiles, those actors CAE is one N-th of the CAE just considering scenarios *lowEMA* and *highEMA*, since only the N-th firing of that actors is not in a *inactive* scenario with an execution time of 0. Because the CAE is implicitly divided by N, the *inter firing latency* (IFL) of the `Source` actor, which is used to model the input frequency of sensor samples, can be used as the time interval for the utilization calculation, since its IFL is also the N-th part of time between two sliding windows arriving at a particular core. Therefore the processors average utilization equals its CAE divided by IFL_{Source}.

III. Experiments and Results

We implemented the hand gesture recognition system introduced in [5] and [4]. In order to improve predictability, we disabled the caches of the individual cores. As a consequence the execution times increased approximately by a factor of two. Because of this we doubled the period between sliding windows in comparison to earlier publications to 40 ms by shifting it by two samples instead of one.

Comparing the measured *response times* and the analytically acquired *response times* using our SADF model shows, that there is a small deviation for scenario *highEMA* of approximately 0.85 % for the configuration with one tile and 0.79 % for all other configurations. In scenario *lowEMA* there is a deviation of approximately 14 %, which is much higher than in scenario *highEMA*. It leads to the assumption that there is an additional absolute error apart from a relative error between model and implementation. However, since the real-time requirement in scenario *highEMA* is the important one, this overestimation is acceptable to deduce the real-time ability of different configurations. In order to test our model for real-time requirements, we evaluated the IFL of the `Source` actor. This actor is synchronized with the `WindowDist` actor and its delay equals the time period between two sliding windows. The IFL of `Source` equals its execution time, if the real-time requirement is met, i.e. processing a sliding window needs less than 40 ms. If the IFL is higher than its execution time, the next sliding has to be skipped. From the analytically acquired properties we could predict that the configuration with one tile does not meet real-time requirements in scenario *highEMA*, while all other configurations do. This was verified by our experiments. Moreover, the analytically acquired utilizations from our SADF model for the real-time enabled configurations show a deviation of less than 2 % in the *highEMA* scenario and less than 8 % for the *lowEMA* scenario to the task load of our implementation.

In conclusion, our SADF model of the gesture recognition system is suitable to predict real-time capabilities, as well as response times and processor utilization with less than 2 % error for the high computation scenario.

References

[1] B. D. Theelen, M. C. Geilen, T. Basten, J. P. Voeten, S. V. Gheorghita, and S. Stuijk, "A scenario-aware data flow model for combined long-run average and worst-case performance analysis," in *Proc. of MEMOCODE'06*. IEEE Computer Society, pp. 185–194.

[2] B. Theelen, M. Geilen, and J. Voeten, "Performance model checking scenario-aware dataflow," in *FORMATS'11*. Springer, pp. 43–59.

[3] B. D. Theelen, "A performance analysis tool for scenario-aware streaming applications," in *Proc. of QEST'07*. IEEE, pp. 269–270.

[4] F. Grützmacher, J.-P. Wolff, and C. Haubelt, "Sensor-based online hand gesture recognition on multi-core dsps," in *Prof. of GlobalSIP'15*. IEEE, pp. 898–902.

[5] F. Grützmacher, J.-P. Wolff, and C. Haubelt, "Exploiting thread-level parallelism in template-based gesture recognition with dynamic time warping," in *Proc. of the iWoAR'15*.

Towards State-Based RT Analysis of FSM-SADFGs on MPSoCs with Shared Memory Communication

Ralf Stemmer†, Maher Fakih*, Kim Grüttner* and Wolfgang Nebel†
*OFFIS – Institute for Information Technology, Oldenburg, Germany
†Carl von Ossietzky Universität Oldenburg, Germany

I. Introduction

Scenario-aware-Data-Flow Graphs (SADFGs), first introduced in [5], achieve a good trade-off between expressiveness (allowing the expression of more dynamic behaviors than Synchronous Data-Flow [2] graphs (SDFGs)) and analyzability. Finite-State-Machine Scenario-Aware-Data-Flow (FSM-SADF) [4] graphs are a further simplification of the general SADF graphs. Both extend SDFGs with scenarios. In an FSM-SADF MoC a set of typical scenarios are pre-defined (through a finite state machine) for a specific SDF application. The SDF application reacts to every scenario in a different manner leading to more efficiency and better throughput. Fig. 1a shows an example FSM-SADF of an MPEG decoder as introduced by [5]. The vertices of the graph can be either kernels (`VLD`, `MC`, `IDCT` and `RC`) or detectors (`FD`). The solid edges are called *data* channels while the dashed edges are called *control* channels. Every input port of the kernels/detector in an FSM-SADF has a consumption rate and likewise each output port has a production rate. In each scenario, kernels can have different production and consumption rates. While general SADF graphs enable multiple scenario changes during one *iteration*[1] by allowing multiple scenario detectors, FSM-SADF graphs support only one detector per graph which simplifies the analysis of such models. Initial tokens are visualized by dots on the edges. For example the data channel from the kernel `RC` to the detector `FD` is initialized by three tokens, so that the detector can be executed three times before the kernel `RC` can execute to generate new tokens.

In this work, we extend our previous model-checking based real-time analysis approach in [1] for the analysis of timing bounds for FSM-SADFGs mapped on a shared memory multi-core architecture (1a). In our previous work only analysis of SDFGs was supported. We utilize timed automata (TA) as a common semantic model to represent worst-case execution times (WCET) of kernels, detectors and shared communication resource. The analysis model furthermore supports analysis of access protocols for buses, DMA, private local and shared memories of the MPSoC. For given FSM-SADFGs, MPSoC architecture, FSM-SADFG to MPSoC mapping, execution & communication times, and scheduling a network of TA can be generated. Using the UPPAAL model-checker, safe timing bounds of the FSM-SADFG MPSoC implementation can be provided. The closest work to ours was published recently in [3] where a translation from FSM-SADF graphs to TA was presented. In difference to our approach, the authors concentrated in their translation and analysis only on the FSM-SADF MoC and did not consider MPSoC mapping and resulting resource sharing aspects (i.e. contention on communication resources) in their work.

[1] An *iteration* is the minimum non-zero execution (i.e. at least one kernel has been executed) such that the initial state of the graph is obtained [1].

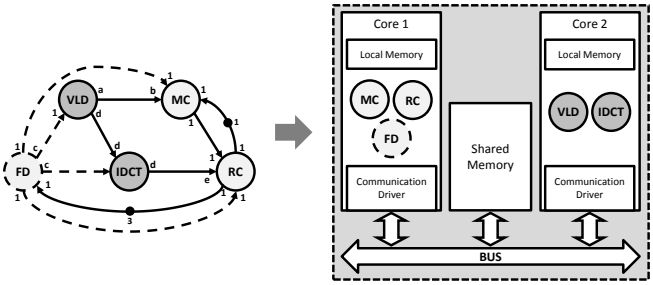
II. Translating FSM-SADF to Timed Automata

Model-checking SDFGs running on MPSoCs with shared communication resources was already described in [1]. In the following, only the relevant extensions will be described to enable model-checking FSM-SADF graphs. To support FSM-SADF, the detector and the FSM, determining the scenario in which the current SDF is executing, must be modeled as TA. The TA template of the detector and kernel (also called *actor* in the context of SDFGs) are similar, see Fig. 2a. Both TA can be split into three phases, the *read phase* in which all tokens are read from the channels, the *computation phase* where the read data gets processed, and the *write-phase* in which new tokens are written to the output ports. In addition to a kernel, the detector triggers a scenario switch at the end of the computation phase. In this case, an additional synchronization channel (Fig. 2a `updateScenario`) is used to trigger a state transition of the FSM TA, modeling the scenario switching, as shown in Fig. 2b. The global variable `scenario`, shown in Fig. 2b serves as an index for an array that holds the token rate configuration for each kernel in a specific scenario. Since the FSM is part of the *computation phase* of the detector, the execution time of the FSM is covered by the best-case and worst-case execution time of the detector. During the *write phase*, the detector produces control tokens but may also produce data tokens.

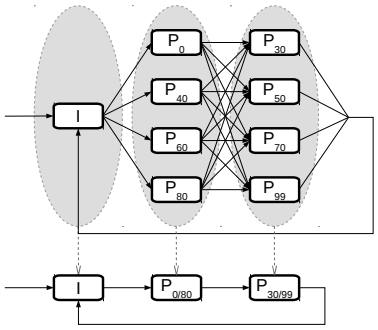
In model-checking, non-determinism can rapidly lead to a model with unmanageable state space (known as state-explosion problem). To avoid non-determinism in the FSM of an FSM-SADF graph, multiple possible successor states of an initial state can be abstracted to a single state as shown

Table I: I, P_0, P_x are the states of the non-deterministic FSM [4] and become I, $P_{0/80}$, $P_{30/99}$ after abstracting to a deterministic FSM in Fig. 1b with best-case (BC) and worst-case (WC) rates.

Rate	*Nondet. Scenarios*			*Det. Scenarios (BC/WC)*		
	I	P_0	P_x	I	$P_{0/80}$	$P_{30/99}$
a	0	0	1	0	0 / 1	1 / 1
b	0	0	x	0	0 / 80	30 / 99
c	99	1	x	99	1 / 80	30 / 99
d	1	0	1	1	0 / 1	1 / 1
e	99	0	x	99	0 / 80	30 / 99

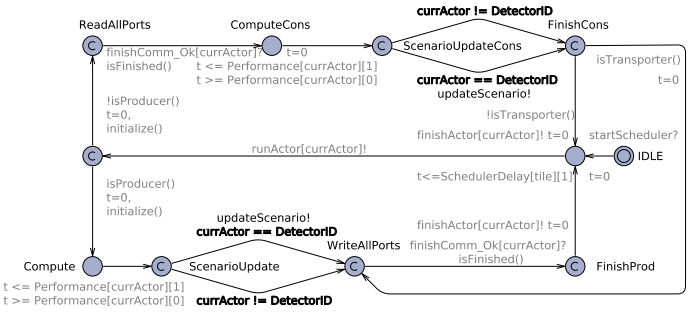


(a) FSM-SADF graph mapped on a dual core MPSoC. Kernels on the same core use local FIFOs, kernels mapped on different cores use FIFOs in shared memory.

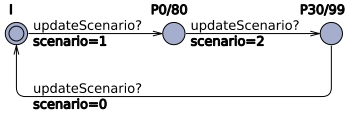


(b) Top: original FSM [3], Bottom: deterministic abstracted FSM

Figure 1: FSM-SADF model of an MPEG-4 decoder [4] with an abstracted deterministic FSM



(a) A simplified TA template for the kernel (denoted by actor) and detector. `DetectorID` defines whether or not this TA is a detector.



(b) A TA template of the scenario FSM

Figure 2: Extensions to the TA representation of SDFGs

in Fig. 1b. This abstraction is valid since we are interested in getting the worst-case value of some timing constraint (such as *Period* or *end-to-end latency*). That is why the group of successor states of the current state in the non-deterministic FSM can be abstracted into a single state representing worst-case-scenario state among this group. The same procedure could be done for the best-case scenario state. This lead to a new deterministic FSM which describes the token rates in a more coarse way. To estimate which original scenario is the worst and which one is the best, each one has to be analyzed individually as a SDF graph with a static token rate for this specific scenario. Note that also different execution times for every kernel/detector are used according to the current scenario (the execution time values were taken from [4]). This was done with the SDF2TA tool described in [1].

Table I shows that the channel *b* can be either 0 or 80 after the abstraction, while it had multiple rates ranging between $0, 40, 60$ and 80 (see FSM in Fig. 1b) before. In Fig. 1b the state $P_{0/80}$ is equivalent to the original state P_{80} (see Tab. I) because in this scenario, the kernels `RC` and `MC` (see Fig. 1a) have to be executed most frequently and the highest amount of macro blocks need to be transferred between kernels. To get the best-case values, $P_{0/80}$ becomes equivalent to the original state P_0 (Tab. I) where most data channels transfer zero tokens and kernels are activated at most once per iteration. Kernels of the FSM-SADF that can work in different scenarios get notified of their scenario by control tokens produced by the detector. In case a kernel needs to be executed multiple times during one scenario iteration, it needs the proper amount of control tokens at its input port that can be consumed during each read phase. Control tokens are propagated from the detector to other kernels using the same shared resources that data tokens use. Therefore control tokens are modeled just like the data tokens only with a different token size.

In our example, we mapped the kernels, the detector and their communication channels implemented as FIFOs on a dual core architecture. The detector `FD` and the kernels `RC` and `MC` are mapped onto *Core 1* while the kernels `VLD` and `IDCT` are mapped on *Core 2* as shown in Fig. 1a. The communication channels between kernels on the same core are mapped to a local memory, FIFOs for inter-core communication are mapped to shared memory.

In a first evaluation, we analyzed the worst-case end-to-end latency for the worst and the best scenario as show in Tab. I for $P_{0/80}$ and $P_{30/99}$. The end-to-end latency of our example was 8928 cycles for the best case scenario selection and 8969 cycles for the worst case scenario.

ACKNOWLEDGEMENT

This work has been partially supported by the *CONTREX* project, funded by the European Commission under grant agreement FP7-611146.

REFERENCES

[1] Fakih M, Grüttner K, Fränzle M, Rettberg A (2015) State-based real-time analysis of sdf applications on mpsocs with shared communication resources. J Syst Archit 61(9):486–509, DOI 10.1016/j.sysarc.2015.04.005

[2] Lee E, Messerschmitt D (1987) Synchronous data flow. Proceedings of the IEEE 75(9):1235–1245, DOI 10.1109/PROC.1987.13876

[3] Skelin M, Wognsen ER, Olesen MC, Hansen RR, Larsen KG (2015) Model checking of finite-state machine-based scenario-aware dataflow using timed automata. In: Industrial Embedded Systems (SIES), 2015 10th IEEE International Symposium on, IEEE, pp 1–10

[4] Stuijk S, Ghamarian AH, Theelen BD, Geilen MCW, Basten T (2008) FSM-based SADF. Tech. rep., Eindhoven University of Technology, Department of Electrical Engineering

[5] Theelen BD, Geilen MCW, Basten T, Voeten JPM, Gheorghita SV, Stuijk S (2006) A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In: Proceedings of the Fourth ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2006. MEMOCODE '06. Proceedings., IEEE Computer Society, Washington, DC, USA, MEMOCODE '06, pp 185–194, DOI 10.1109/MEMCOD.2006.1695924, URL http://dx.doi.org/10.1109/MEMCOD.2006.1695924

A Model-Driven Framework for Hardware-Software Co-design of Dataflow Applications

Waheed Ahmad, Bugra M. Yildiz, Arend Rensink, Mariëlle Stoelinga
University of Twente, The Netherlands
Email: {w.ahmad, b.m.yildiz, arend.rensink, m.i.a.stoelinga}@utwente.nl

I. Introduction

Hardware-software (HW-SW) co-design is an engineering practice that allows to meet system-level objectives by exploiting the synergy of hardware and software through their simultaneous design. However, current tools and approaches for HW-SW co-design have difficulties coping with the concurrency and increasing complexity of modern-day systems. As a result, the time and effort needed for modeling and validating such designs are negatively affected. Therefore, an automated modeling approach is required which supports:

- *Modularity*: The modeling approach should separate various aspects -such as hardware, software and their mapping- to keep different concerns modular. Modules targeting different concerns are reusable and easier to maintain.
- *Extensibility*: To allow the implementation of possible future requirements in the co-design process, the modeling approach should have convenient extension mechanisms.
- *Completeness*: The modeling approach should consider and include all the basic elements to cover the domain concepts in HW-SW co-design.
- *Interoperability*: The modeling approach should support interoperability between various tools, such as for model designers and verifiers.

Model-Driven Engineering (MDE) is an approach that helps to fulfill the aforementioned requirements [5]. In MDE, the important concepts of the target domain are formally captured by a so-called *metamodel*. Separate metamodels for the domains of interest keeps the design modular. All models are instances of a metamodel, or possibly an integrated set of metamodels. Furthermore, models can be transformed to the other via *model transformations*, defined at the metamodel level.

Synchronous Dataflow (SDF) graphs are well-known computational models for real-time streaming and dataflow applications. Current SDF analysis tools, e.g., SDF^3 lack support for energy optimization. To bridge this gap, the SDF graphs are often translated to other domains [1] [3] [4].

Even though these approaches propose novel solutions for energy optimization, they do not fulfill the criteria set earlier. In particular, the following issues are not taken into account:

- *Modularity:* The hardware platform, software application and mapping of the application to the platform are not represented separately in these approaches. The final model is a combined representation of all these three aspects. So, any changes that occur in any of these aspects require modifications in the combined representation.
- *Interoperability:* These approaches are not fully automated. The construction of the final models from SDF graphs is achieved manually, which is a labor-intensive and error-prone task.
- *Extensibility:* The lack of automation also leads to the fact that there is no systematic extension mechanism for satisfying future requirements. Furthermore, an extension will affect every single part of the approach, due to the lack of modularity.

To address these issues, we propose a model-driven framework for HW-SW co-design of energy-optimal dataflow applications. Our framework automates the approach of [1] by applying state-of-the-art techniques from MDE.

The work in [1] proposes an approach to compute energy-optimal schedules for dataflow applications running on multiprocessor platforms. This approach encodes the problem of finding such schedules (while satisfying minimal throughput requirements) as an optimization problem, defined as a reachability property over priced timed-automata models. This is then checked by the model checker UPPAAL CORA [2].

To achieve our aims, we define three metamodels in our framework: (1) a metamodel for SDF graphs; (2) a metamodel for Platform Application Models (PAMs), which describes the power levels of a processor type and the cost of switching between them; and (3) a metamodel for expressing allocations of the tasks in an SDF graph to the processor types in a PAM. The PAM metamodel allows to represent the most commonly used power reduction techniques like Dynamic Power Management (DPM, switching to low power state) and Dynamic Voltage and Frequency Scaling (DVFS, throttling processor frequency). Moreover, the PAM metamodel allows to express the state-of-the-art notion of voltage frequency islands (VFIs), which are clusters of processors that run on a common clock frequency/voltage. These three metamodels target different concerns modularly. In addition, the approach in [1] considers model checker UPPAAL CORA for generating energy-optimal schedules. Therefore, for supporting the generation of UPPAAL CORA models, we also use an existing UPPAAL metamodel. The models conforming to three metamodels explained earlier, are transformed to UPPAAL CORA models via *model transformations* in the framework.

II. The Model-Driven Framework

The overview of the framework is given in Figure 1. The HW-SW co-design of the application consists of the first four steps:

- In step 1, a SDF model of the software application is created using the SDF^3 tool in an XML format specific to the tool.
- In step 2, the SDF model is transformed to an SDF model that conforms to the metamodel we defined for SDF graphs.
- In step 3, the hardware platform model is created using *PAM Visual Editor* that is a graphical editor for specifying Platform Application Models (PAMs). This model conforms to the PAM metamodel we defined for PAMs.
- In step 4, an allocation model is created for specifying the mapping of the tasks in the SDF model to the processor types in the PAM.

The analysis of the co-design for energy-optimal schedules is conducted using the UPPAAL CORA model checker. This is achieved in the last three steps:

- In step 5, the co-design is transformed to a priced-timed-automata model that conforms to the UPPAAL metamodel.
- In step 6, the priced-timed-automata model is transformed to the format accepted by the model checker.
- In step 7, we analyze the resulting model to compute the energy-optimal schedule using the UPPAAL CORA model checker.

III. Evaluation

A. Interoperability

In our case, we utilize SDF^3 for creating SDF graphs and UPPAAL CORA for deriving energy-optimal schedule via model checking. These tools have their own domains and focus areas. For example, SDF^3 is widely used for SDF graph analysis. However, it does not support cost optimization and model-checking. We have implemented model transformations in our framework to achieve interoperability between these tools.

B. Completeness

All domain knowledge of SDF graphs and PAM is captured fully by corresponding metamodel elements.

C. Modularity

The modularity of our framework allows system designers to model hardware platforms and software components separately. In this way, if any change is required either in software or hardware models, only the related part needs to be updated independently from the other.

D. Extensibility

Due to the adoption of MDE techniques, our framework provides systematic ways for extensions for future requirements. One can extend the framework using the following mechanisms: introducing new models with related metamodels and new transformations; extending existing transformations or metamodels.

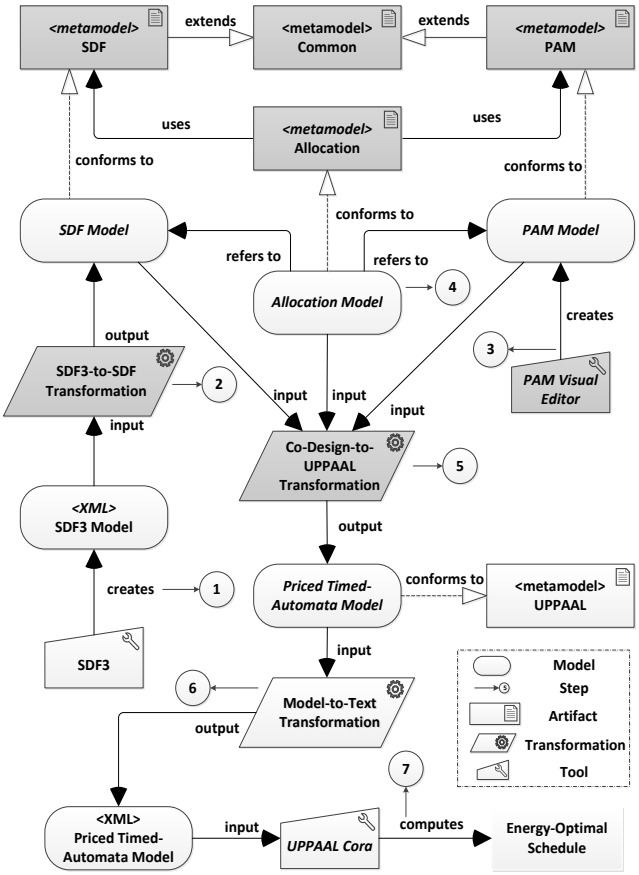


Fig. 1: Overview of our framework. The elements with dark background color represent the new contributions.

As an example, suppose we want to extend our hardware platform models with the concept of "battery". This extension can be achieved through the following steps:

- Developing a battery metamodel.
- Transforming the concepts in the battery metamodel to the concepts in the priced timed-automata domain. This is achieved through extending the Co-Design-to-UPPAAL model transformation in step 5.

References

[1] W. Ahmad, P.K.F. Hölzenspies, M.I.A. Stoelinga, and J. van de Pol. Green computing: Power optimisation of VFI-based real-time multiprocessor dataflow applications. In *DSD'15*, pages 271–275, Aug 2015.

[2] G. Behrmann, K. G. Larsen, and J. I. Rasmussen. Optimal scheduling using priced timed automata. *SIGMETRICS Perform. Eval. Rev.*, 32(4):34–40, Mar. 2005.

[3] P. Huang, O. Moreira, K. Goossens, and A. Molnos. Throughput-constrained voltage and frequency scaling for real-time heterogeneous multiprocessors. In *SAC '13*. ACM, 2013.

[4] A. Nelson, O. Moreira, A. Molnos, S. Stuijk, B. Nguyen, and K. Goossens. Power minimisation for real-time dataflow applications. In *DSD'11*, pages 117–124, Aug 2011.

[5] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2013.

Extended Abstract: Process Networks for Reactive Streaming with Timed-automata Implementation

Peter Poplavko, Dario Socci, Rany Kahil, Marius Bozga, Saddek Bensalem
VERIMAG Lab (CNRS,University of Grenoble), France

Most of modern academic tool flows for embedded real-time systems support either the *streaming* or the *reactive-control* class of application programming models. These two classed have historically developed two different design methodologies. The former, such as CompSoc [9], are typically dataflow-related and is based on the analysis and optimization of timing properties in system steady state. The latter, such as Prelude [4], are based on synchronous language compilation and classical real-time schedulability analysis. However, when implementing modern complex applications (such as avionics, satellite and robotics control systems) on many-core platforms we encounter disadvantages of the separation of systems into two classes. Focusing on only one of them imposes certain undesirable methodological restrictions that are not necessarily present in the other one. We present our current ideas towards unifying these two classes.

To this end, in this abstract we discuss a recently developed [11] model of computation: Fixed-Priority Process Networks (FPPN). FPPNs extend streaming models by support of time-dependent (yet deterministic) behavior and real-time task properties (*e.g.,* sporadic/periodic activations with deadlines) for the processes and channels that are not necessarily FIFO's. These extensions are possible due to decoupling between the process blocking from the inter-process channel accesses. Our public design flow [14], [10] compiles FPPN's to executable component-based model with timed automata components. Timed automata is thus used as a 'meta-model' to define the semantics of FPPN and to provide a basis for simulation and deployment. Moreover, automata are useful means for adding the system middleware components that cannot be expressed in higher-level models of computation, such as run-time management, *e.g.,* QoS control [1], and custom scheduling policies [13]. We demonstrate combining such automata with FPPN models in [14] and [12].

An instance of FPPN is composed of four main entities: *Processes* (tasks), *Data Channels* (communication buffers), *Event Generators* and *Functional Priorities*. The process network example in Fig. 1 represents an imaginary signal processing application with input sample period 200 ms, reconfigurable filter coefficients and a feedback loop. The filter coefficients are reconfigured by sporadic process CoefB.

We see several periodic processes, annotated by their periods, and a sporadic process, annotated by minimal inter-arrival time. This process also has a non-default burstyness value $m_e = 2$. We also see inter-process channels – the blackboards

This research received funding from **MoSaTT-CMP** – European Space Agency project, and from **CERTAINTY** – European FP7 project

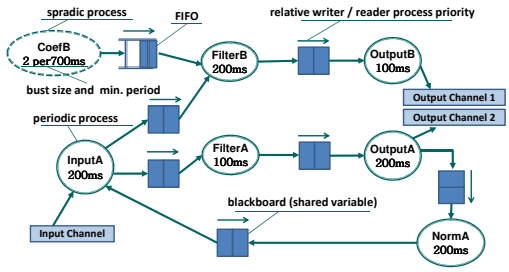


Fig. 1: Fixed Priority Process Network Example

and a FIFO, annotated by relative functional priority, FP. Also the environment input/output channels are shown.

A *Process* in FPPN represents a software subroutine that operates with internal variables and input/output channels connected to it through ports. Formally, a process is defined as an automaton extended with data variables x, and input/output channels. At its transitions the automaton can either read data from an input channel '$x?c$' and write data to an output channel $x!c$ or perform internal computations on variables. A minimal execution run that brings the process automaton back to its initial state is called a *job execution* or simply a job, by analogy to real time systems where a job is the atomic execution of a real-time task. In terms of streaming models, a job is analogous to a 'firing' and a process is analogous to an 'actor'. Between the job executions, the process waits to be activated by a (time stamped) event, which can occur following a periodic, sporadic, or any other user-defined pattern. This is analogous to 'task release' events in real-time systems. Note that in our framework [14], a process automaton does not have explicit timing constraints, but gets synchronized with another system component (the event-generator automaton) that ensures the real-time properties of the process. Note also that the user does not have to program a process directly as an automaton, instead it is programmed in software language (C/C++) and gets automatically 'compiled' into the equivalent automaton.

Data Channels ensure *read and write operations* for communication. These operations are *non-blocking*, which means that reading from an empty channel will not block the reader. Therefore, next to the data value, the read operation returns the so-called *validity flag*, *i.e.,* a Boolean indicator of whether the data is valid. There are inter-process and external (environment) channels. The former join a pair of processes and are protected from data races by functional priorities between the reader and writer process. An external channel is joined to a unique process and is coupled to its event generator. The main reason why the jobs of a process have to execute their activation times and deadlines is to ensure a timely and safe

access to external channels.

By default, our tools support two internal channel types: FIFO (*i.e.,* a bounded queue) and blackboard (*i.e.,* a shared variable that keeps the last written value). Other types can be potentially introduced in our framework by defining their automata models [14].

An *event generator* e is defined by the set of possible sequences of time stamps $\{\tau_k\}_e$ that it can possibly produce at run time. By default, we define two types of event generators: *periodic* and *sporadic* with possible burstyness m_e (by default 1) but other arrival patterns can be potentially defined as timed automata components. Every event generator is associated with a unique process and determines whether the given process is periodic or sporadic. Every process p has a deadline d_p.

An FPPN network can be described by two directed graphs. The first graph is the default process network graph (P, C), whose nodes are processes P and the edges are channels C. This graph can be cyclic and defines the communicating pairs of processes and the direction of dataflow: from writer to reader. The second graph is the functional priority DAG: (P, FP). No cyclic paths are allowed in this graph. The edges define functional priority relation between the processes. It is however, not a partial order relation, as it is not necessarily transitive. We require that:

$$(p_1, p_2) \in C \implies (p_1, p_2) \in FP \vee (p_2, p_1) \in FP$$

i.e., a functional priority should either follow the direction of the data flow or the opposite direction.

For a given pair of processes, the relation FP defines the sequential precedence order of their execution in the case of their synchronous activation at the same time. If, on the contrary, two processes related by FP are not activated at the same time online, then the FPPN model disregards the order defined by FP but instead imposes the order that gives the precedence to the job that was activated earlier. By requiring every channel to imply an FP relation we ensure deterministic dependence of precedence constraints on the synchronous activation times, which leads to deterministic functional dependencies of external outputs on external inputs (and their time stamps).

For a given process network and functional code in C/C++ our design tools [10] automatically instantiate a component-based timed automaton model in BIP [1], which we have extended for support of non-instantaneous transitions (to model job execution steps), thus making this model in a certain sense a 'task automaton' model as defined in TIMES tool [2]. The automatically derived BIP model is compiled into C++ classes and then can be either executed in simulation mode on a desktop or deployed on a multi-core embedded platform to be executed in real-time mode. To support multi-core parallelism we use a multi-thread BIP RTE engine, an extended version of [15]. Furthermore, from a certain quite general restriction of FPPN one can derive offline task graphs and do static task graph scheduling offline [11]. The derived task-graph models are similar to HSDF graphs derived from SDF for scheduling purposes but in our case we also calculate arrival time and deadline of every graph node.

Using our tools, we have modeled and implemented on MPPA multi-core platform [5] an industrial application case study Flight Management System [6], the results in [11] are encouraging. As shown there, our design flow demonstrates certain features that combine important elements from both streaming and reactive control methodologies.

FPPN combines certain concepts of synchronous and streaming languages. While practiced for a long time in streaming, derivation and scheduling precedence-constrained multi-tasking models from synchronous languages has received attention only recently; *e.g.,* [3], [4] propose task graph derivation and scheduling algorithms, which do not support, however, sporadic events and multi-processors. [8] is one of the few streaming languages supporting external events with deadlines, though only periodic ones. In [7] reactive process networks (RPNs) are proposed. We believe that by simple model transformation (merging the fixed-priority and process-automata models to re-introduce blocking behavior) FPPNs can be translated to a restriction of RPNs. Compared to RPN, FPPN is adapted to real-time tasks by introducing the priority and explicit timing of events.

REFERENCES

[1] T. Abdellatif, J. Combaz, and J. Sifakis. Model-based implementation of real-time applications. In *Proceedings of the tenth ACM international conference on Embedded software*, EMSOFT '10. ACM, 2010.

[2] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times — a tool for modelling and implementation of embedded systems. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems*, pages 460–464. Springer, 2002.

[3] S. Baruah. Semantics-preserving implementation of multirate mixed-criticality synchronous programs. In *RTNS'12*, pages 11–19. ACM, 2012.

[4] M. Cordovilla, F. Boniol, J. Forget, E. Noulard, and C. Pagetti. Developing critical embedded systems on multicore architectures: the prelude-schedmcore toolset. In *RTNS*, 2011.

[5] B. D. de Dinechin, D. van Amstel, M. Poulhiès, and G. Lager. Time-critical computing on a single-chip massively parallel processor. In *Proc. DATE'14*, pages 97:1–97:6, 2014.

[6] G. Durrieu, M. Faugère, S. Girbal, D. G. Pérez, C. Pagetti, and W. Puffitsch. Predictable flight management system implementation on a multicore processor. In *ERTSS'14*, 2014.

[7] M. Geilen and T. Basten. Reactive process networks. In *EMSOFT'04*, pages 137–146, New York, NY, USA, 2004. ACM.

[8] S. J. Geuns, J. P. H. M. Hausmans, and M. J. G. Bekooij. Sequential specification of time-aware stream processing applications. *ACM Trans. Embed. Comput. Syst.*, 12(1s):35:1–35:19, Mar. 2013.

[9] A. Hansson, K. Goossens, M. Bekooij, and J. Huisken. Compsoc: A template for composable and predictable multi-processor system on chips. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 14(1):2, 2009.

[10] P. Poplavko, P. Bourgos, D. Socci, S. Bensalem, and M. Bozga. Multicore code generation for time-critical applications, http://www-verimag.imag.fr/multicore-time-critical-code,470.html.

[11] P. Poplavko, D. Socci, S. Paraskevas Bourgos, and M. B. Bensalem. Models for deterministic execution of real-time multiprocessor applications. In *DATE'15*, 2015.

[12] D. Socci. *Scheduling of Certifiable Mixed-Criticality Systems*. PhD thesis, Unvirsity Grenoble Alpes, 2016.

[13] D. Socci, P. Poplavko, S. Bensalem, and M. Bozga. Modeling mixed-critical systems in real-time bip. In *ReTiMiCs'2013*, 2013.

[14] D. Socci, P. Poplavko, S. Bensalem, and M. Bozga. A timed-automata based middleware for time-critical multicore applications. In *SEUS*, 2015.

[15] A. Triki, J. Combaz, S. Bensalem, and J. Sifakis. Model-based implementation of parallel real-time systems. In *FASE'13*. Springer, 2013.

Analysis and Visualization of Execution Traces of Dataflow Applications

Hadi Alizadeh Ara*, Amir Behrouzian*, Marc Geilen*, Martijn Hendriks†*, Dip Goswami*, Twan Basten*†,
*Eindhoven University of Technology, Eindhoven, The Netherlands
†TNO ESI, Eindhoven, The Netherlands

I. Objectives

Embedded applications are an integral part of modern embedded systems. These applications typically have real-time constraints on throughput or latency. Temporal analysis techniques are required at the design stage to ensure that the applications meet their constraints and determine the required amount of resources (processors or memories for the application).

The Synchronous Dataflow Graph (SDFG) [4] model of computation is a popular method for temporal analysis of applications. SDFG represents the application by a graph consisting of *actors* and *channels*. Actors represent the individual computations within the application. Each actor has an execution time which usually indicates the upper bound on the time it requires to complete its *firing* (execution), once it is started. Channels model the dependencies of individual computational tasks on the resources, on input data, and on the data produced by other tasks. When an actor starts and completes firing, it consumes and produces a fixed amount of *tokens* on the FIFO channels respectively. An actor is able to fire if its consumption rates are not greater than the number of tokens on the channels from which it consumes. Figure 1 shows two SDFGs. Each SDFG models a working mode of a single application. Each mode has three actors. The execution time of actors x, y and z are assumed to be $101, 73$ and 125 milliseconds in both modes respectively. The numbers near the channel endings indicate the consumption and production rates. The actors are bound to different processors in different modes. This property is modeled by having a self edge on each actor which contains a token labelled with the name of the processor to which it is bound (e.g. x is bound to processors $P1$ and $P2$ in modes A and B respectively).

The temporal analysis techniques developed for SDFGs can analyse throughput and latency for data-driven execution of the application or when the application is mapped to a predictable platform with limited resources. These techniques output the latency, throughput and possibly the critical path within a graph that models the task and resource dependencies. However, they do not give much tangible information about the system such as when and where the tasks are being processed or in which time intervals which one of the resources are busy or idle. We therefore develop a technique to visualize execution traces of dataflow applications. Visualization gives developers a more detailed understanding of the temporal behaviour of the application. For example visualization of the critical data and resource dependencies, makes it easier to recognize the system's bottlenecks. TRACE [1] is a powerful Gantt chart visualization tool capable of presenting tasks on resources and dependencies between them as a function of time. Moreover, it can run performance analysis such as critical path analysis, latency and throughput analysis based on execution trace

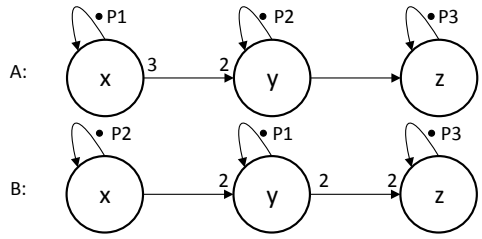


Fig. 1: An example application having two different modes

information and it can compare traces and visualize their differences for model validation and design-space exploration.

We have integrated TRACE visualization into the SDF^3 tool. SDF^3 is a tool for temporal analysis of SDFG and various types of dynamic dataflow models such as Cyclo Static Dataflow (CSDF) [2], Senario Aware Dataflow (SADF) [6] and Finite State Machine-SADF (FSM-SADF) [5]. SDF^3 also supports automated mapping of applications onto predictable multi-processor platforms. This integration enables us to visualize execution traces of application tasks and track the resource usage associated with these executions. Moreover, we can utilize the various analysis and visualization options in TRACE such as critical path analysis to visualize the critical dependencies.

II. Methods

In dataflow theory, the set of actor firings that brings the tokens back to the initial token distribution, is called an *iteration*. According to [3], the temporal behaviour of an SDFG can be captured by finding the *time differences* between the production times of tokens at the end and start of an iteration. In this method the production times of each token and consequently the start and the completion times of each actor firing can be represented by a *symbolic time stamp* of the form $t = \max_i(t_i + g_i)$ where t_i are the symbolic availability times of initial tokens ($P1, P2, P3$ for the example). g_i are suitable constants that indicate the time difference between t and t_i. It is $-\infty$ if there is no dependency between t and t_i. In $(max, +)$ algebra we can represent a symbolic time stamp with the vector dot-product $t = \bar{t}.\bar{g}$ where $\bar{t} = [t_{P1}; t_{P1}; t_{P2}]$ for the example. Assuming $\bar{t} = [t_{P1}; t_{P1}; t_{P2}] = [0; 0; 0]$, the start and completion time of the first firing of actor x in mode A of the example can be represented as $[0; -\infty; -\infty]$ and $[101; -\infty; -\infty]$ respectively. This vector representation of the start (completion) time of firings is called the *symbolic start (completion)* time. The time differences g_i are determined by *symbolically simulating* the application graph for one iteration. Symbolic simulation characterizes the time differences of the application by a matrix in $(max, +)$ algebra. This matrix allows to compute the completion times of any iteration using

the following equation in $(max, +)$ algebra

$$\gamma_{k+1} = G_m \times \gamma_k \tag{1}$$

where G_m is called the *characterization matrix* of the application when it is in mode m and γ_k is a vector that determines the start of iteration k. Using Equation 1 we can capture the evolution of the application in time. However, this equation cannot be directly used for tracing purposes, because, it only records the completion time of firings that produce tokens at the end of the iteration. Individual firings inside the iteration have been abstracted into a single matrix multiplication. To overcome this problem, we construct another matrix that allows us to reconstruct detailed information about the individual actor firings from the starting point of an iteration. During the symbolic simulation we store the symbolic start time and completion times. We do this for all firings involved in one iteration of the graph in the order they get enabled and completed. All stored vectors are vertically concatenated to construct a matrix. With this matrix, all firings within the iteration can be calculated from the initial token time stamps γ_k as follows.

$$\tau_k = H_m \times \gamma_k \tag{2}$$

where H_m is the *tracing matrix* in mode m and τ_k is a vector that contains the start and completion times of all firings within iteration k in the order they are stored during the symbolic simulation. It is important to realize that tracing only requires extra calculations for the iterations for which we want to observe the traces. In the worst-case, the number of firings in a single iteration is exponential in the size of the graph. Nevertheless, the tracing function is computationally efficient in many practical cases.

III. Results

We implemented this tracing algorithm for FSM-SADF in the SDF^3 tool. We generate two files as input to the TRACE tool, a configuration file and the actual trace file. The configuration file defines specific properties of the FSM-SADF executions as actor names, scenario names and iteration indexes. These properties are called *attributes* in the TRACE tool and used to colour and group the traces for different visualization purposes. TRACE considers an individual task execution as a single *claim* which is made on a resource from the start time of the execution to its completion time. The trace file then lists all claims and their attributes for the requested tracing interval. TRACE outputs Gantt charts of *activities*, i.e. task executions, and resource claims for a given trace file. These Gantt charts then can be grouped together and coloured, based on the selected attributes.

Consider the application in Figure 1. Assume it executes iterations in the given mode order: A, B, B, A, A, B. TRACE outputs, by default, a Gantt chart in which every task execution has a different colour. For a better visualization, we group the executions by actor names and use different colours for different iterations with the same colour for task executions within an iteration. Figure 2 shows the Gantt chart of activities and the corresponding processor claims for the given order respectively. By applying the critical path analysis to the traces, the dependencies between the different executions are analysed and critical dependencies are found. Then the colouring of the Gantt charts are changed for better visualization of the critical path, as shown in Figure 3 (with the critical path coloured in red). By visualizing the critical path, we can easily verify that

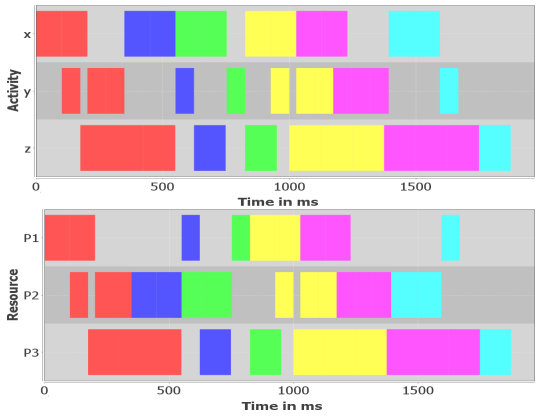


Fig. 2: Gantt charts of actor firings and processor claims

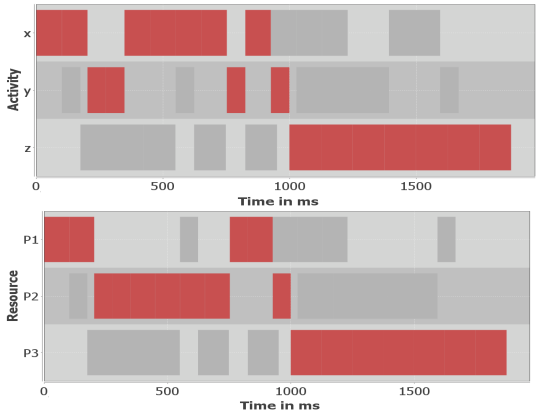


Fig. 3: Gantt charts of actor firings and processor claims, visualizing the results of critical path analysis

for example increasing the frequency of processors $P1$ and $P2$ during the first second and $P3$ during the next second will reduce the system latency; also the throughput will improve if the sequence is being repeated.

IV. Conclusion

We provided an efficient method to symbolically compute the start and end time of all actor firings of a dataflow graph. This information may be used to visualize dataflow execution traces to obtain a better understanding of the temporal behaviour of the system.

Acknowledgment

This research is supported by the ARTEMIS joint undertaking through the ALMARVI project (621439).

References

[1] Embedded Systems Innovation by TNO. website http://trace.esi.nl.

[2] G. Bilsen, M. Engels, et al. Cyclo-static dataflow. *IEEE Transactions on Signal Processing*, 44, 1996.

[3] M. Geilen and S. Stuijk. Worst-case performance analysis of synchronous dataflow scenarios. In *Proc. ISSS+CODES*, 2010.

[4] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. *In Proc. IEEE*, 75, 1987.

[5] S. Stuijk, M. Geilen, et al. Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications. In *Proc. SAMOS*, 2011.

[6] B. Theelen, M. Geilen, et al. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *Proc. Memocode*, 2006.

Mode-controlled Dataflow based Buffer Allocation for Real-time Streaming Applications Running on a Multi-processor without Back-pressure

Hrishikesh Salunkhe*, Alok Lele*, Orlando Moreira†, Kees van Berkel*
* Eindhoven University of Technology, The Netherlands; † Intel Corporation, The Netherlands
{h.l.salunkhe, a.lele, c.h.v.berkel}@tue.nl; orlando.moreira@intel.com

I. Introduction

Current smartphones support multiple radio standards such as GSM and LTE [1] running simultaneously. The transceivers for these standards are real-time streaming applications that process a potentially infinite sequence of input data streams and have strict timing requirements [2]. Moreover, to enable low-cost high-volume market, they are often mapped on a heterogeneous multiprocessor platform having severely constrained on-chip memory resources. Therefore, it is essential to minimize memory consumption by these applications.

A real-time streaming application consists of multiple computational tasks, mapped on a hardware, that communicate with each other by producing (or consuming) data values through finite First-In First-Out (FIFO) buffers, which are mapped on a on-chip memory. Buffer allocation for these applications involve the minimization of total memory consumption by buffers while reserving sufficient space for each data value production without overwriting any live (not consumed) data values and guaranteeing the satisfaction of real-time constraints. Systems may prevent overwriting of live data by implementing a back pressure mechanism in which a producer task is suspended until there is a sufficient space available on its output FIFO buffer [3]. However, in systems without back-pressure, a producer executes as soon as it is enabled without checking for the availability of space on its output buffers. This may result in buffer overflow, where a producer task overwrites a live data value. Moreover, systems without back-pressure [4] are not uncommon, since back-pressure incurs extra processing and synchronization overheads. We focus on buffer allocation for systems without back-pressure.

Dataflow is a well-known model of computation that can be used to analyze real-time streaming applications [2]. A dataflow graph consists of nodes called actors and edges. Actors communicate through edges, which represent FIFO buffers, using tokens (a data transfer unit). Buffer allocation techniques exist for applications modeled as dataflow graphs running on a hardware platform with [5], [3], [6] and without back-pressure [7].

Static dataflow such as Single-rate Dataflow (SRDF) supports rigorous timing analysis, however, it cannot conveniently express the dynamic behavior of a real-time streaming application [8]. In such applications, actor firings and dependencies change dynamically. Moreover, when back-pressure is not supported both the best- and worst-case timing behavior must be considered. However, modeling the best- and worst-case [7] behavior in static dataflow can be very pessimistic, leading to the overestimation of the necessary buffer sizes.

Dynamic dataflow [9] can easily capture dynamic behavior, however, it cannot be subjected to temporal analysis. Mode-controlled Dataflow (MCDF, [2]) is a restricted form of boolean dataflow, that allows temporal analysis and is described in Section II. In an MCDF graph, a specific subgraph is chosen per iteration, depending on a mode of its execution. We observe that MCDF not only can model dynamic behavior of these applications but also their best- and worst-case timing behavior more accurately.

In this paper, we propose a buffer allocation solution for real-time streaming applications modeled as MCDF graphs running on a heterogeneous multiprocessor platform without back-pressure. We consider MCDF graphs with a class of mode sequence called Recurrent-choice Mode Sequence (RCMS) which consists of mode sequences of equal length; RCMS not only allows to model practically relevant applications but also provides tractable analysis. We show that our technique provides up to 36% reduction in memory consumption compared to the existing SRDF-based technique for an LTE Advanced receiver.

II. Mode-controlled dataflow

In static dataflow, e.g. Single-rate Dataflow (SRDF) [13], actors have fixed execution times and communicate with each other using tokens through edges (FIFO buffers). In each firing, an actor consumes/produces a single token from/to its input/output edges. The initial state of such graphs is specified by initial tokens, visualized as a dot(s) on some graph edges. Dataflow graphs are iterative, they run continuously processing virtually infinite input sequence in a pipelined manner.

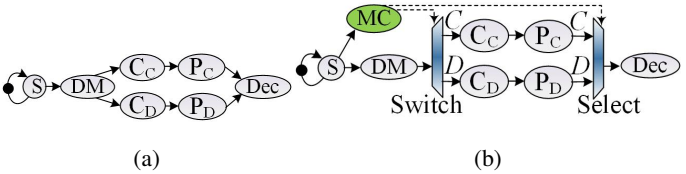


Fig. 1: A typical radio application in (a) SRDF and (b) MCDF: modes C (control) and D (data)

Mode-controlled Dataflow (MCDF) [2] is a restricted form of boolean dataflow [11] that supports run-time mode switching as well as design-time temporal analysis. In an MCDF

graph, in each iteration, based on a mode value produced by a so-called mode controller, actors belonging to a specific sub-graph are fired. A typical MCDF graph is comprised of multiple switch, select, static dataflow actors and a single mode controller actor as shown in Fig. 1b. Modal actors C_C and P_C (C_D and P_D) fire only in the iterations of mode C (D). Amodal actors S, DM and Dec fire in every iteration. In each iteration, a switch (select) consumes a control token from the control input and a data token from the amodal (modal) input, and produces the data token on the modal (amodal) output. The modal output (input) is determined by the mode of the control token. In each iteration, the mode controller passes a mode value through a control token to all switch and select actors. Note that a modal port of a switch (select) can be unconnected, in that case, it fires without producing (consuming) any token. A Recurring) Mode Sequence (RMS), a static and self-repeating sequence of modes, defines an order of modes in which the graph will execute. Generally, the dynamic behavior of an application can be split into one or more static behaviors, where switching from one to another static behavior is performed at runtime. Each such static behavior can be modeled as an RMS. In a dynamic behavior, after the last mode in the current RMS, the graph execution may non-deterministically switch to the first mode of any of the associated RMSs and must continue executing until it reaches the end of the RMS. We introduce a choice (non-determinism) in selecting RMSs using Recurrent-choice Mode Sequence (RCMS); it is captured in regular expression as $(c_1|c_2|...|c_n)^*$, where c_1, c_2, ...c_n are the RMSs.

III. MCDF-based buffer allocation

The optimal buffer allocation algorithm has exponential time complexity [7]. Therefore, we use a heuristic method, where we extract the SRDF equivalent graphs from the MCDF model to compute the sufficient buffer sizes. We construct an SRDF graph termed as RMS graph that is equivalent to the MCDF graph for a given RMS. We demonstrate that two separate graphs are required to capture best- and worst-case execution dependencies among the RMSs of an RCMS (recall Section II) associated with the application. Consequently, we construct the best- and worst-case RCMS graphs; we add best- and worst-case dependencies among the RMS graphs to obtain the best- and worst-case RCMS graphs respectively. These graphs conservatively model the timing behavior of the RCMS for the application. We then simulate these two graphs separately to obtain the earliest production (using the best-case RCMS graph) and latest consumption times (using the worst-case RCMS graph) of tokens. Buffer computation algorithm uses these times to build token lifetimes. For each edge in a graph, a sufficient buffer size is given by the maximum number of tokens whose lifetimes overlap with each other.

IV. Experiments & results

In this section, we benchmark our technique using an LTE Advanced receiver [1]. LTE-Advanced marks the next major step towards the evolution of LTE. It consists of 5 single LTE carriers. We compute buffer sizes for each combination of such LTE carriers from 1 to 5 carriers.

The buffer sizes for LTE Advanced receiver computed using the SRDF and MCDF-based techniques are shown in

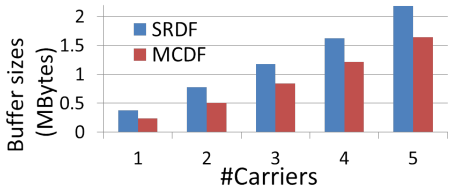


Fig. 2: Buffer sizes: SRDF Vs. MCDF

Fig. 2. The MCDF-based technique achieves from 24% to 36% reduction in memory consumption compared to the SRDF-based technique. This is because, the best- and worst-case modeling in MCDF reduce the estimated token lifetimes, resulting into smaller buffer sizes.

V. Conclusion

In this paper we provide a buffer allocation solution for applications modeled as Mode-controlled Dataflow (MCDF) graphs running on hardware without back-pressure. We consider MCDF graphs with Recurrent-choice Mode Sequence (RCMS) that consists of the mode sequences of equal length; RCMS not only allows to model practically relevant applications but also provides tractable analysis. We capture the best- and worst-case timing behavior of an MCDF graph using two Single-rate Dataflow (SRDF) graphs that provide more accurate estimation of token lifetimes, which results in up to 36% reduction in the memory expenditure compared to the existing SRDF-based buffer allocation technique for an LTE Advanced receiver.

References

[1] E. Dahlman et al., *4G: LTE/LTE-Advanced for Mobile Broadband*. Academic Press, 2nd ed., 2014.

[2] O. Moreira et al., *Scheduling Real-Time Streaming Applications onto an Embedded Multiprocessor*. Springer International Publishing, 2014.

[3] O. Moreira et al., "Buffer sizing for rate-optimal single-rate data-flow scheduling revisited," *IEEE Transactions on Computers*, 2010.

[4] D. Nadezhkin et al., "Realizing fifo communication when mapping kahn process networks onto the cell," in *SAMOS*, (Berlin), Springer, 2009.

[5] S. Stuijk et al., "Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs," in *DAC*, (NY, USA), ACM, 2006.

[6] M. H. Wiggers et al., "Buffer capacity computation for throughput-constrained modal task graphs," *ACM TECS.*, Jan. 2011.

[7] H. Salunkhe et al., "Buffer allocation for real-time streaming on a multi-processor without back-pressure," in *ESTIMedia*, Oct 2014.

[8] H. Salunkhe et al., "Mode-controlled dataflow based modeling & analysis of a 4g-lte receiver," in *DATE*, EDAA, 2014.

[9] S. Sriram et al., *Embedded Multiprocessors: Scheduling and Synchronization*. NY, USA: Marcel Dekker, Inc., 1st ed., 2000.

[10] S. Stuijk et al., "Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications," in *SAMOS*, July 2011.

[11] J. Buck, "Static scheduling and code generation from dynamic dataflow graphs with integer-valued control streams," in *SSC*, 1994.

[12] G. Bilsen et al., "Cyclo-static data flow," in *ICASSP*, 1995.

[13] E. A. Lee and et al., "Synchronous data flow," 1987.

[14] H. Salunkhe et al., "Mode-controlled dataflow based buffer allocation for real-time streaming applications running on a multi-processor without back-pressure," Tech. Rep. 15-03, Dept. of Comp. Sci. TU/e, The Netherlands, July 2015.

[15] M. Breschel et al., "10.8 a multi-standard 2g/3g/4g cellular modem supporting carrier aggregation in 28nm cmos," in *IEEE International ISSCC*, Feb 2014.

Symbolic computation of latency for dataflow graphs

(abstract)

Adnan Bouakaz Pascal Fradet Alain Girault

INRIA; Univ. Grenoble Alpes

first.last@inria.fr

I. MOTIVATION

We present the symbolic computation of data-flow graphs latency, with two variants: the multi-iteration latency and the input-output latency. These are important timing constraints that are usually used in the design of real-time control systems. The input-output latency is particularly useful for real-time control systems since it is the maximum delay between sampling data from sensors and sending control commands to the actuators.

Latency analysis can either be performed at compile time, for design space exploration, or at run-time, for resource management and reconfigurable systems. However, this analysis has an exponential time complexity, which may cause a huge run-time overhead or make design space exploration unacceptably slow. We propose to compute the latency *symbolically*, i.e., as a function of parameters of the given data-flow graph. By parameters, we mean the input and output rates of the data-flow actors, as well as their execution times. Such functions can be quickly evaluated for each different configuration or checked w.r.t. different quality-of-service requirements.

II. DEFINITIONS AND BASIC NOTIONS

We are given a data-flow graph G made of edges of the form $A \xrightarrow{p \quad q} B$ with two *actors* A and B such that A produces p tokens each time it *fires* while B consumes p tokens each time it fires. Besides, we are given the execution time t_A of A and t_B of B. Essentially, G is a parameterized version of an SDF graph [7] where rates and execution times can be formal parameters. As with any SDF graph, we can solve the system of balance equations to find the iteration of the graph [7]. *E.g.,* for the simple graph $A \xrightarrow{p \quad q} B$, there is a single balance equation $z_A p = z_B q$, where z_X denotes the number of firings of actor X in the iteration.

The repetition vector of this simple graph is:

$$[z_A = q/\gcd(p,q), z_B = p/\gcd(p,q)]$$

Finally, the *load* imposed by actor X is the product $z_X t_X$.

In this work, we focus on as soon as possible (ASAP) scheduling of consistent graphs without auto-concurrency. In such self-timed executions, an actor fires as soon as it becomes idle (no auto-concurrency) and has enough tokens on its input channels. We assume that there are sufficient processing units, *e.g.,* there are as many processors as actors or all actors are implemented in hardware. ASAP scheduling allows the graph to reach its maximal throughput. Such schedules are naturally pipelined and composed of a prologue followed by a steady state that repeats infinitely.

The *multi-iteration latency* $\mathcal{L}_G(n)$ of the first n iterations of a graph G is equal to the finish time of the last firing of its first n iterations (time is counted from the very first firing).

The *period* $\mathcal{P}_G$ of the execution of a graph G is the average length of an iteration, formally defined as:

$$\mathcal{P}_G = \lim_{n\to\infty} \frac{\mathcal{L}_G(n)}{n} \tag{1}$$

The *input-output latency* $\ell_G(n)$ of the n^{th} iteration of a graph G is equal to the time between the start time of the first firing and the finish time of the last firing of the n^{th} iteration. The definition given in [6] is slightly different but in our context (graphs with initially empty channels) the two definitions are equivalent. The input-output latency of the complete execution ℓ_G is defined as the maximal latency over all iterations:

$$\ell_G = \max_{n=1..\infty} \ell_G(n) \tag{2}$$

III. SYMBOLIC COMPUTATION

We first derive analytic formulas for the multi-iteration latency of the first n iterations (i.e., $\mathcal{L}_G(n)$) of graph $G = A \xrightarrow{p \quad q} B$. Since we are interested in the (approximation of) the minimum achievable latency, we assume that buffers are unbounded. There are two cases depending on whether A or B imposes the highest load.

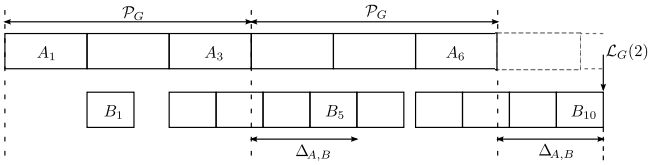


Fig. 1: ASAP schedule and multi-iteration latency $\mathcal{L}_G(2)$ of graph $G = A \xrightarrow{p \quad q} B$ in the case $z_A t_A \geq z_B t_B$ ($p = 5$, $q = 3$, $t_A = 14$, $t_B = 8$). Each box represents the firing and execution time of one actor.

• **Case** $z_A t_A \geq z_B t_B$, *i.e.,* A imposes a higher load than B. As illustrated in Fig. 1, actor A never gets idle and $\mathcal{P}_G = z_A t_A$. Therefore, we have:

$$\mathcal{L}_G(n) = n\mathcal{P}_G + \Delta_{A,B} \tag{3}$$

such that $\Delta_{A,B}$ is the remaining execution time for actor B after actor A has finished its firings of the n^{th} iteration ($\Delta_{A,B}$ is constant over all iterations). The formulas for $\Delta_{A,B}$ can be found in [2].

• **Case** $z_A t_A < z_B t_B$: see [2].

For a chain $A \xrightarrow{p_1 \quad q_1} B \xrightarrow{p_2 \quad q_2} C \to \cdots \to Z$ of actors, we compute an *upper bound* of the multi-iteration latency of the first n iterations, denoted $\hat{\mathcal{L}}_{A\to Z}$ (we omit n for the sake of conciseness). We first compute exactly $\mathcal{L}_{A\to B}$. However, since this computation assumes that the producer can run consecutively, it cannot be applied between B and C.

We compute an upper bound linearization of the firings of B such that they are consecutive and $\forall j \leq nz_B.\ f_{B^u}(j) \geq f_B(j)$ (details are in [2]). The intuition is to transform each actor for which the execution has gaps (*e.g.,* B in Fig. 1) into a virtual actor having the same load but without any gap, so that we can compute the latency for each edge of the chain.

We actually use two upper bound linearization methods to make the firings of B consecutive: **(i)** The *Push* method pushes *all* firings of B to the right end to get rid of all the gaps; **(ii)** The *Stretch* method increases the execution time of B in order to fill the gaps over an infinite execution. They are incomparable and there are graphs for which either *Push* or *Stretch* is better. Since the two methods are not costly to try, we apply both and take the minimum.

For a general acyclic SDF graph G, we represent it as a set of *maximal chains* $\mathcal{G}(G)$, that is, chains from a source actor to a sink actor. We then compute the multi-iteration latency of each such chain g, and we finally have: $\mathcal{L}_G(n) = \max_{g \in \mathcal{G}(G)} \{\mathcal{L}_g(n)\}$.

Regarding the input-output latency, we can compute the maximum input-output latency of the n^{th} iteration of a chain G, denoted $\ell_G(n)$, from its multi-iteration latency: $\ell_G(n) = \mathcal{L}_G(n)$ minus the start time of the first firing of the source actor in the n^{th} iteration. If the source actor A imposes the highest load among all actors of the graph or if all the channels are unbounded, then the source actor never gets idle and achieves the maximal throughput (otherwise, buffer sizes must be taken into account, see [2]). It follows that:

$$\ell_G(n) = \mathcal{L}_G(n) - (n-1)z_A t_A \tag{4}$$

For an arbitrary chain G, we also make use of a backward linearization technique to compute a safe upper bound of ℓ_G.

IV. Results

We have evaluated our approach for computing the multi-iteration latency using millions of randomly generated chains. The experiments show that the average over-approximation is negligible when the number of firings per iteration of the graph is small. Indeed, if there are many harmonious rates (recall that, when p divides q or q divides p, the computed latency for $A \xrightarrow{p\ \ q} B$ is exact), then the computed latency remains close to the exact value. Then, the average over-approximation increases to reach its peak (approximately 2.5%) at around fifty firings per iteration. This is because the exact values of latency at these points are small and hence the over-approximation is more noticeable. Then, the average over-approximation decreases and converges to zero for graphs with large latencies.

Table I presents our results for five real applications: the H.263 decoder, the data modem and sample rate converter from the SDF^3 benchmarks [8], the Fast Fourier Transform (FFT), and the time delay equalizer (TDE) from the StreamIt benchmarks [9]. All these graphs have a chain structure. Table I shows that our approach gives exact results for most of these benchmarks. Production and consumption rates of channels of these graphs are quite harmonious (p divides q or q divides p), for which our approach performs very well.

Finally, we evaluate our approach for computing the input-output latency using 10^5 randomly generated chains. The experiment shows that our analysis over-approximates the exact computation, on average, by at most 13%. The over-approximation is less noticeable for graphs with large input-output latencies.

TABLE I: Multi-iteration latency computation for real benchmarks.

graph	$\mathcal{P}_G$	$\mathcal{L}_G(1)$	$\hat{\mathcal{L}}_G(1)/\mathcal{L}_G(1)$	$\hat{\mathcal{L}}_G(2)/\mathcal{L}_G(2)$
(a) modem	32	62	1	1
(b) sample con.	960	1000	1.022	1.011
(c) H.263 dec.	332046	369508	1	1
(d) FFT	78844	94229	1	1
(e) TDE	17740800	19314069	1	1

V. Related work

Few symbolic results about SDF graphs can be found in the literature. The results reported here and in [2], [3] on symbolic latency are the first of their kind.

For the symbolic throughput computation, [4] consider the *token timestamp vector* $\vec{s}_i$, where each entry corresponds to the production time of tokens in the i^{th} iteration of the graph. Then, the authors use the max-plus algebra to express the evolution of the token timestamp vector: $\vec{s}_i = M\vec{s}_{i-1}$. They have proved that the eigenvalue of matrix M is equal to the period of the graph.

[5] presents a parametric throughput analysis for SDF graphs with *bounded* parametric execution times of actors but constant rates. Since rates and delays are non-parametric, the SDF-to-HSDF transformation is possible and the throughput analysis is based on the MCM of the resulting HSDF graph.

A different analytic approach to estimate lower bounds of the maximum throughput is to compute strictly periodic schedules instead of ASAP schedules (*e.g.,* [1]). This approach is similar to our *Stretch* linearization method to compute the latency of the graph. We have however found that using both *Push* and *Stretch* methods usually gives better results.

References

[1] B. Bodin, A. Munier-Kordon, and B. de Dinechin. Periodic schedules for cyclo-static dataflow. In *Symposium on Embedded Systems for Real-time Multimedia*, pages 105–114, 2013.

[2] A. Bouakaz, P. Fradet, and A. Girault. Symbolic analysis of dataflow graphs (extended version). Technical Report 8742, INRIA, 2016.

[3] A. Bouakaz, P. Fradet, and A. Girault. Symbolic buffer sizing for throughput-optimal scheduling of dataflow graphs. In *Proceedings of the 2016 IEEE 22nd Real-Time and Embedded Technology and Applications Symposium*, 2016.

[4] M. Geilen. Synchronous dataflow scenarios. *ACM Trans. Embed. Comput. Syst.*, 10(2):16:1–16:31, 2011.

[5] A. H. Ghamarian, M. C. W. Geilen, T. Basten, and S. Stuikj. Parametric throughput analysis of synchronous data flow graphs. In *Conf. on Design, Automation and Test in Europe*, pages 116–121, 2008.

[6] A. H. Ghamarian, S. Stuijk, T. Basten, M. C. W. Geilen, and B. D. Theelen. Latency minimization for synchronous data flow graphs. In *Euromicro Conf. on Digital System Design Architectures, Methods and Tools*, pages 189–196, 2007.

[7] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. In *Proceedings of the IEEE*, pages 1235–1245, 1987.

[8] S. Stuijk, M. Geilen, and T. Basten. SDF^3: SDF for free. In *Int. Conf. on Application of Concurrency to System Design*, pages 276–278, 2006.

[9] W. Thies and S. Amarasinghe. An empirical characterization of stream programs and its implications for languages and compiler design. In *Int. Conf. on Parallel Architectures and Compilation Techniques*, pages 365–376, 2010.

Probabilistic Model Checking for Uncertain Scenario-Aware Data Flow

Joost-Pieter Katoen and Hao Wu
Software Modelling and Verification Group, RWTH, Aachen, Germany
{*katoen, hao.wu*}*@cs.rwth-aachen.de*

***Abstract*—The Scenario-Aware Dataflow (SADF) model is based on concurrent actors that interact via channels. It combines streaming data and control to capture scenarios while incorporating hard and soft real-time aspects. To model data-flow computations that are subject to uncertainty, SADF models are equipped with random primitives. We propose to use probabilistic model checking to analyse uncertain SADF models. We show how measures such as expected time, long-run objectives like throughput, as well as timed reachability—can a given system configuration be reached within a deadline with high probability?—can be automatically determined. The crux of our method is a *compositional* semantics of SADF with exponential agent execution times combined with *automated abstraction* techniques akin to partial-order reduction. We present the semantics in detail, and show how it accommodates the incorporation of execution platforms enabling the analysis of energy consumption. The feasibility of our approach is illustrated by analysing several quantitative measures of an MPEG-4 decoder and an industrial face recognition application.**

This paper considers *exponentially timed* SADF (called eSADF), a version of SADF in which the duration of all firings of actors are governed by negative exponential probability distributions. An eSADF model which represents an MPEG-4 decoder can be found in Figure 1. eSADF can be considered as an extension of exponentially timed SDF as originally proposed in [1].) Although the restriction to exponential distributions seems a severe restriction at first sight, there are (at least) three good reasons to consider it. First, this assumption is a rather adequate abstraction when considering that actor firings are typically subject to random fluctuations (e.g., in hardware) and only mean durations are known. Technically speaking, the exponential distribution maximises the entropy under these assumptions. That is to say, if only mean values are known, the statistically most neutral assumption is to have these phenomena exponentially distributed. Secondly, series-parallel combinations of exponential distributions (so-called phase-type distributions) can approximate any arbitrary continuous probability distribution with arbitrary precision. Our semantic model and analysis algorithms support the analysis of these phase-type distributions. Finally, the use of exponential distributions enables the usage of modern probabilistic model-checking tools for the quantitative analysis of SADF models.

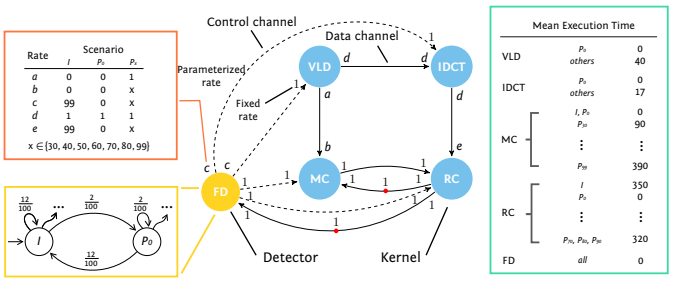
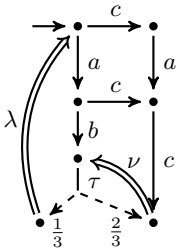


Figure 1. An eSADF for an MPEG-4 decoder

As eSADF is based on asynchronously communicating actors, firings have exponential durations, and sub-scenario selection is based on discrete-time Markov chains, *Markov automata* (MA) [2], [3] are a natural choice for capturing the semantics of eSADF. These automata are transition systems in which the target of an interactive transition is a distribution over states (rather than a single state), and that incorporates random delay transitions to model firing durations. Non-determinism occurs if several interactive transitions emanate from a given state. This paper provides a *compositional* semantics of eSADF using Markov automata. The compositional aspect naturally reflects the logical structure of the eSADF graph, is easily amenable to single actor replacements—as just the semantics of that actor is to be adapted while the remaining automata remain the same—and finally enables component-wise reduction. The latter technique is important in case the state space of the eSADF graph is too large to be handled. Our compositional semantics allows for replacing the automaton for a few actors by an equivalent, but much smaller, automaton. This technique has e.g., been exploited in [4]. Compositionality in SDF has recently also been exploited for modular code generation [5]. Our semantics is defined using a succinct process algebra for describing MA [6] in a textual way. As a result, the semantics is relatively easy to comprehend and modular.



A sample MA

Markov automata of realistic, industrially-sized eSADF graphs can be huge and too large to be handled. One of the main causes for this state space exposition is the highly concurrent character of typical data-flow computations in which many agents run in parallel. To diminish this effect on the state space growth, we use *confluence reduction* [7]—a technique akin to partial-order reduction—that allows for an on-the-fly reduction of the state space. The key of confluence reduction is to detect independent concurrent transitions such that for the analysis only one ordering of concurrent transitions needs to be considered (instead of all possible orderings). We show that all non-determinism in the MA-semantics of eSADF arises from the independent execution of actors, and can (in theory) be eliminated using confluence reduction. As confluence reduction is performed at the language level (i.e., the process algebra) using conservative heuristics, non-determinism may persist after reduction, but if so, it does not influence quantitative measures. An simple sample of eSADF (cf. Figure 2 (left)) consisting of detector A and kernel B with scenarios I and P together with its corresponding MA is shown in Figure 2.

To analyse eSADF graphs we exploit recently developed algorithms and software tool-support for verifying Markov automata. The main challenge in MA analysis is the intricate interplay between exponential durations and non-determinism. The latter arises naturally by the concurrent execution of the different actors. It was recently shown that several measures-of-interest such as expected time and long-run average objectives can be reduced to efficient analysis techniques for

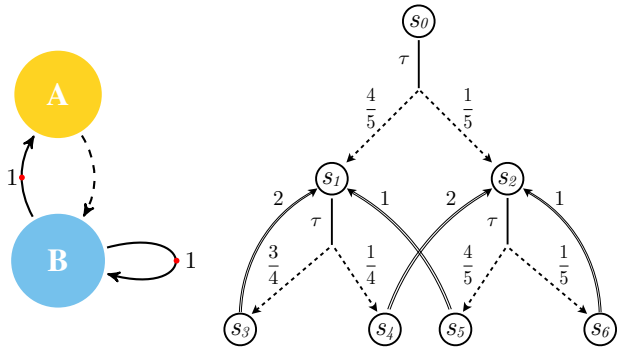


Figure 2: A sample eSADF graph and its (somewhat simplified) MA

Markov decision processes [8]. In addition, timed reachability objectives—can a certain system configuration be reached within a deadline with a high probability?—were shown to be computable by appropriate discretisation techniques [8], and extensions towards the treatment of costs (modelling energy consumption) in Markov automata have become available [9]. Our MA semantics facilitates the quantitative analysis of eSADF graphs using these novel and efficient analysis techniques. To sum up, our semantics is conceptually simple, compositional, and yields a rigorous framework for quantitative analysis of eSADF.

We have developed a prototypical implementation of our approach that maps eSADF graphs (expressed in the XML-format as supported by the SDF^3 tool [1]) onto Markov automata. These MA can then be analysed by the analysis tool presented in [8]. An extension with confluence reduction enables the minimisation of MA. This paper shows the feasibility of our approach by presenting two case studies. The original and optimized (after applying confluence reduction described above) state spaces of the generated MAs of these case studies and the sample eSADF introduced above are shown in Table I.

		before red.	with conf. red
The sample eSADF	#states	19	7
	#transitions	19	7
MPEG-4 Decoder	#states	61918	20992
	#transitions	81847	40910
	#non-det. state	4	1
Face recognition	#states	106784	29440
	#transitions	154688	77344

Table I. MA size for sample eSADF graphs

The MPEG-4 decoder is a benchmark SADF case from the literature [10]. We show the effect of confluence reduction and analyse several measures of interest of the MPEG-4 decoder such as buffer occupancy, throughput, and probability to reach certain buffer occupancies within a given deadline. We compare the results to analysis results using the SDF^3 tool for SADF and to [4]. Some experimental results, such as the throughput (also compared with SDF^3) and the buffer occupancy are shown in Table II and Figure 3, respectively.

As a second case study we present the analysis of an industrial face recognition application. This model is substantially larger than the MPEG-4 decoder as it exhibits a high degree of parallelism. We study the quantitative effect

[1] See http://www.es.ele.tue.nl/sadf/xml.php.

	p8	p9	p10	p11
Throughput	IDCT	VLD	MC	RC
Our approach	0.0423732	0.0423732	0.000746128	0.000746128
SDF^3	0.0437919	0.0437919	0.000745268	0.000745268

Table II. Throughput of each kernel in MPEG-4 decoder

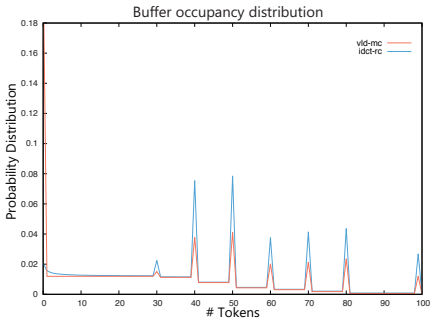


Figure 3. Token distribution in three MPEG-4 channels

of including auto-concurrency, and analyse several metrics. We conclude by presenting an extension of our framework by incorporating execution platforms on which the eSADF agents are supposed to be executed. This allows for studying the quantitative effect of exploiting multi-core platforms, as well as the quantitative impact of dynamic voltage and frequency scaling (DVFS) and dynamic power management (DPM). We present an energy analysis of the MPEG-4 decoder for a simple execution platform based on the Samsung Exynos 4210 processor in Table III.

#states	#tran.		Power cons.	Thr. IDCT	Thr. RC	Exp. energy 1 iter.
215851	314609	Min	0.746337 (W)	21.975093	0.415335	1.581342 (mJ)
		Max	1.528917 (W)	30.279699	0.563065	2.423743 (mJ)

Table III: Energy consumption of deploying the MPEG-4 decoder on the Samsung Exynos 4210 processor

References

[1] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*. CRC Press, 2009.

[2] Y. Deng and M. Hennessy, "On the semantics of Markov automata," *Information and Computation*, vol. 222, pp. 139–168, 2013.

[3] C. Eisentraut, H. Hermanns, and L. Zhang, "On probabilistic automata in continuous time," in *IEEE Symp. on Logic in Computer Science (LICS)*. IEEE, 2010, pp. 342–351.

[4] B. D. Theelen, J.-P. Katoen, and H. Wu, "Model checking of scenario-aware dataflow with CADP," in *DATE*. IEEE, 2012, pp. 653–658.

[5] S. Tripakis, D. N. Bui, M. Geilen, B. Rodiers, and E. A. Lee, "Compositionality in synchronous data flow: Modular code generation from hierarchical SDF graphs," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 3, pp. 83:1–83:26, 2013.

[6] M. Timmer, J.-P. Katoen, J. van de Pol, and M. Stoelinga, "Efficient modelling and generation of Markov automata," in *Int. Conf. on Concurrency Theory (CONCUR)*, ser. LNCS, vol. 7454. Springer, 2012, pp. 364–379.

[7] M. Timmer, J. van de Pol, and M. Stoelinga, "Confluence reduction for Markov automata," in *FORMATS*, ser. LNCS, vol. 8053. Springer, 2013, pp. 243–257.

[8] D. Guck, H. Hatefi, H. Hermanns, J.-P. Katoen, and M. Timmer, "Analysis of timed and long-run objectives for Markov automata," *Logical Methods in Computer Science*, vol. 10, pp. 1–29, 2014.

[9] D. Guck, M. Timmer, H. Hatefi, E. Ruijters, and M. Stoelinga, "Modelling and analysis of markov reward automata," in *Int. Symp. on Automated Technology for Verification and Analysis (ATVA)*, ser. LNCS, vol. 8837. Springer, 2014, pp. 168–184.

[10] B. D. Theelen, M. Geilen, T. Basten, J. Voeten, S. V. Gheorghita, and S. Stuijk, "A scenario-aware data flow model for combined long-run average and worst-case performance analysis," in *MEMOCODE*. IEEE, 2006, pp. 185–194.