<http://www.diva-portal.org>

Postprint

This is the accepted version of a paper presented at *9th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, RAPIDO 2017, Stockholm, Sweden, 25 January 2017*.

Citation for the original published paper:

Rosvall, K., Khalilzad, N., Ungureanu, G., Sander, I. (2017)

Throughput propagation in constraint-based design space exploration for mixed-criticality systems

In: *ACM International Conference Proceeding Series*, 3023977 Association for Computing Machinery (ACM)

<https://doi.org/10.1145/3023973.3023977>

N.B. When citing this work, cite the original published paper.

Permanent link to this version:

<http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-208450>

Throughput Propagation in Constraint-Based Design Space Exploration for Mixed-Criticality Systems

Kathrin Rosvall, Nima Khalilzad, George Ungureanu, Ingo Sander
KTH Royal Institute of Technology
Stockholm, Sweden
{krosvall, nkhal, ugeorge, ingo}@kth.se

ABSTRACT

When designing complex mixed-critical systems on multi-processor platforms, a huge number of design alternatives has to be evaluated. Therefore, there is a need for tools which systematically find and analyze the ample alternatives and identify solutions that satisfy the design constraints. The recently proposed design space exploration (DSE) tool DeSyDe uses constraint programming (CP) to find implementations with performance guarantees for multiple applications with potentially mixed-critical design constraints on a shared platform. A key component of the DeSyDe tool is its throughput analysis component, called a throughput *propagator* in the context of CP. The throughput propagator guides the exploration by evaluating each design decision and is therefore executed excessively throughout the exploration. This paper presents two throughput propagators based on different analysis methods for DeSyDe. Their performance is evaluated in a range of experiments with six different application graphs, heterogeneous platform models and mixed-critical design constraints. The results suggest that the MCR throughput propagator is more efficient.

CCS Concepts

•**Computer systems organization** → *System on a chip; Embedded systems; Real-time systems*; •**Hardware** → *Methodologies for EDA*;

Keywords

Constraint Programming, Correct-by-Construction, Design Space Exploration, Performance Analysis

1. 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 multi-processor architectures do not provide guaranteed QoS, and are very difficult to analyze due to shared resources and caches [24]. 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) [15] with predictable architectures, which can give guaranteed QoS with respect to timing [16, 11, 19]. Especially, there has been great interest in synchronous data flow (SDF) [14] thanks to 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 [23, 18, 9]. Recently, approaches based on constraint programming (CP) [25, 1, 20, 17] have demonstrated their potential as complete methods for finding optimal mappings, mainly concerning throughput.

This paper uses the CP-based DeSyDe (**D**esign space exploration for **S**ystem **D**esign) tool [8, 20]. DeSyDe can provide system implementations with performance guarantees for multiple applications that do not require any special operating system or run-time scheduler. It provides order-based schedules that can be implemented as bare-metal solution. Apart from mapping, scheduling and throughput analysis, DeSyDe's CP model also contains interprocessor-communication, buffer sizes, energy consumption, memory consumption and a range of platform-related cost metrics. Due to the fact that the CP model captures the problem as a whole instead of decomposing it into its subproblems, trade-offs as e.g. between throughput and energy consumption are comprehended by the model. To the best of our knowledge, DeSyDe is currently the only CP-based DSE tool for mixed-critical systems that supports this range of performance metrics.

Throughput analysis plays a key role in DSE for real-time and mixed-critical systems. It is a computation-intensive task that is essential to evaluate the quality of potential solutions repetitively throughout the exploration. Therefore, it is crucial that this analysis, implemented by means of a *propagator* in a CP context, is performed efficiently. The contribution of this paper is the comparison of two alternative throughput propagators, one based on maximum cycle ratio analysis (MCR) and the other based on state space

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

RAPIDO '17, January 23 - 25, 2017, Stockholm, Sweden

© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4840-9/17/01...\$15.00

DOI: <http://dx.doi.org/10.1145/3023973.3023977>

exploration (SSE), in a CP-based design space exploration for multiple applications on an MPSoC platform and design constraints of mixed-criticality. A number of experiments with SDF graphs of real applications and a synthetic cyclic graph evaluate the efficiency in terms of run-time of the two propagators in comparison.

2. RELATED WORK

There are three commonly used methods to compute the throughput of SDFGs: maximum cycle ratio (MCR) analysis, state space exploration and max-plus algebra. Maximum cycle ratio analysis [5, 12], commonly also referred to as maximum cycle *mean* (MCM), requires the conversion of an SDFG to its homogeneous equivalent. This is not a draw-back in the presented CP-approach since the mapping and scheduling is done on the unfolded, homogeneous graph in order to take advantage of data-parallelism, potentially yielding higher throughput.

Alternatively, throughput, as inverse of the iteration period of an SDFG, can be determined without conversion of the graph by state-space exploration (SSE) [10]. In [10] it is experimentally shown that SSE often outperforms MCR analysis in practice. However, this applies for the theoretically achievable throughput of an SDFG, not a mapped and scheduled graph that is unfolded to achieve optimal throughput on a given platform, shared among multiple applications. Hence, the problem setting in this paper is essentially different, and the results can differ significantly.

A more recently proposed method [7] uses max-plus algebra and operates on a similar unfolded graph representation as the method in this paper. Due to this and the fact that it has shown promising results in terms of performance, a throughput propagator based on max-plus algebra may be considered for future work.

There are a number of related CP-based approaches that take throughput into account. The work in [25, 26] maps and schedules SDFGs on a heterogeneous multiprocessor platform with buffer and throughput constraints, but in an iterative process, meaning that the throughput analysis is not part of the CP model, but performed off-line. Further, the CP model is essentially different in that the representation of the schedules in the CP model is time-based, not order-based as in the model in this paper.

Bonfietti et al. [2, 1] apply an MCR propagator as throughput constraint in a CP model. However, interprocessor communication and buffer sizes are not considered in that work.

The approach in [17] considers a network-on-chip platform, including the configuration of a routing scheme, in the CP model. However, it maps applications with SDF granularity and therefore does not exploit data-parallelism for throughput optimization as the DeSyDe tool in this paper. Further, their model does not support energy consumption. Most significantly, none of the above mentioned approaches consider a state space exploration propagator or support for simultaneous scheduling and throughput analysis of multiple applications. The approach presented in this paper can map and schedule multiple applications on a shared platform with or without individual or global performance and cost constraints. The framework also offers support for mixed-criticality, in the sense that constraints on performance and cost metrics can be hard for some applications, while other applications are provided with best-effort service on the remaining resources.

Figure 1: Example for a cyclic SDF graph

3. DESIGN SPACE EXPLORATION

This section introduces the DSE problem for which the throughput propagators of this paper have been developed. In addition, constraint programming (CP) in combination with dataflow analysis is presented as the method chosen to tackle the problem.

3.1 The DSE Problem

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 system metrics like energy consumption and area cost.

This 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.

3.1.1 Application Model

Two application models are currently supported: synchronous dataflow graphs (SDFGs) and independent period tasks. SDF graphs [14] are widely used for modeling streaming applications, while periodic tasks [4] are typically used for modeling feedback control tasks. The throughput propagators discussed in this paper only concern SDFGs. For further information about the support of periodic tasks see [13].

An example SDFG is given in Figure 1. A finite set of actors A and a finite set of channels C constitute an SDFG $G(A, C)$. An actor $a \in A$ produces a fixed rate p of tokens on output op and actor b consumes a fixed rate q of tokens from input ip via a channel $c = (a, op, p, b, ip, q, tok)$. A channel can hold initial tokens, tok . Initial tokens are depicted with a dot on the arc for a channel, optionally with an adjacent integer signifying multiple initial tokens. The DSE maps and schedules multiple application graphs simultaneously on a shared platform, i.e. it takes a set of graphs $G_z(A_z, C_z)$ as input.

For consistent SDFGs, the fixed production and consumption rates allow to derive a fixed number of repetitions for each actor, which together constitute one iteration of the graph. During one iteration, all actors execute or *fire* as many times as given by function $\gamma : A \rightarrow \mathbb{N}$, which in the end leads back to the initial token distribution.

All consistent SDFGs can be converted into single-rate, homogeneous dataflow graphs (HSDFGs), with the traditional approach from [22] or a more recent method described in [6]. The conversion to HSDF is a common step when dealing with timing analysis of SDFGs and used for example in [12] and [14]. Since the conversion can lead to a massive increase in size of the graph, it has been identified as

Figure 2: Target Platform Model

a problem and methods working directly on SDFGs have been proposed as an alternative, e.g. in [10] and [23]. Yet, the homogeneous representation has the advantage to expose data-parallelism, which can be exploited when the goal is to find a mapping and scheduling with optimal or sufficient throughput for an application. Therefore, for the mapping and scheduling of application graphs in this paper, one iteration of the homogeneous-rate version is used.

The throughput of an SDFG is the inverse of its iteration period. The self-timed execution of the order-based schedules yields a pipelined operation with initial latency followed by the periodic phase. The length of the periodic phase for each application graph is its iteration period.

3.1.2 Platform Model

An illustration of the considered target platform model is provided in Figure 2. In order to allow for the analysis of communication delays on a shared platform at compile-time, the underlying platform model must provide predictable service. Without a predictable interconnect, possible contention occurring at run-time makes worst-case communication time (WCCT) analysis overly conservative or even impossible. We assume a predictable bus platform, based on time-division multiple-access (TDMA). The bus is divided into a round of time slots and each processing unit receives one or more slots. With this model, each processor has a dedicated share of the bus. The assignment of TDMA slots to processors is also part of the DSE model.

The communication buffers for the edges of the SDFGs are placed into local memory of the receiving actor’s processor. The platform must provide blocking send- and receive-primitives for order-based self-timed schedules, e.g. through a dedicated communication block as a network interface. The communication block is assumed to have a limited amount of buffer space for tokens waiting for transfer over the bus. By this means, computation and communication can run in parallel on each processing node. The DSE model also ensures that the buffers, both on the sending and receiving side, do not exceed the available space.

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. A processor can for example be instantiated in three different modes: economy, standard or performance. Apart from general purpose processors, the MPSoC-platform can also host dedicated hardware accelerators for actor functions. 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. As part of fu-

ture work, the platform model is planned to be extended by shared memory and network-on-chip communication.

3.1.3 Design Constraints

In addition to the application graphs and platform model, a set of constraints can be specified to configure the exploration performed by the DSE. These constraints are referred to as *design constraints* throughout this paper in order to avoid confusion with the constraints that constitute a CP model.

Most importantly, the design constraints are used to specify the goal of the exploration. There are two compliance levels to choose from: *satisfy* or *optimize* (i.e. maximize or minimize, respectively). By this means, the DSE can support exploration with mixed-critical performance constraints in the sense that satisfy-constraints are hard constraints that must be fulfilled, while optimize-constraints allow the exploration of best-effort solutions. It is also possible to specify both kind of constraints for the same performance metric, e.g. an upper bound for energy consumption, but also request to minimize it further if possible. The design constraints are a powerful tool which allows to use the same CP model for the exploration of diverse design goals. They can be combined in any arbitrary way to fit the design requirements and hence provide for a framework with great flexibility.

3.2 The Method: CP + Dataflow Analysis

The problem of multi-processor scheduling under performance constraints is a typical application for combinatorial optimization. Constraint Programming (CP) is a well-known technique for solving combinatorial problems, with and without optimality requirements. The core of a CP model is a set of *decision variables*, which capture a solution to the problem at hand. Each variable has a *domain* of possible values. Relationships between the variables are expressed in terms of *constraints*, e.g. the values of two integer variables x and y may have to satisfy the constraint $x > y$.

The first step of CP is to create a model capturing a solution to the problem with variables, their domains, constraints and optionally an optimization criterion. Then, a constraint solver 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.

A fundamental advantage of CP over the, due to the complexity of the problem, commonly used heuristic algorithms is that the problem is captured as a whole, instead of decomposing it into sub-problems and hence disregarding the interdependencies between them. This way, the approach remains complete, optimal and trade-off-aware. Yet, due to the separation of concerns in CP, i.e. detaching the description of the problem from the way it is solved, a heuristic search method can be used to operate on the *same CP model*, e.g. if exhaustive search is impracticable due to the scale of the problem.

By means of constraints, the CP model comprehends the relations between different variables of the model. A constraint is implemented through a propagator that operates on the constraint’s subscribed variables. A propagator has

to fulfill a number of obligations in order to correctly function in a CP environment. Specifically, it must be correct in the sense that it never prunes values that could still be part of a valid solution in the current search tree branch. A propagator must also be contracting, i.e. it is only allowed to remove values from, not add to, the variables' domains.

The remains of this section introduce the variables of the CP model that are relevant for throughput propagation. The problem to solve is the mapping and scheduling of n SDF application graphs $G_z(A_z, C_z)$ onto a platform model with a set of processing nodes P . Each processing node p is associated with a set of processor modes $\mathcal{M}_p$ containing at least one mode. A mode $M \in \mathcal{M}_p$ determines the parameters of power consumption, area cost, monetary cost, local memory size and computation speed factor. We denote the union sets of application graph actors and channels as $\mathcal{A} = \bigcup_{z \in [0, n-1]} A_z$ and $\mathcal{C} = \bigcup_{z \in [0, n-1]} C_z$, respectively.

At the core of the CP model for the DSE, dataflow modeling is used to capture the mapping and scheduling decisions and their implications. 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. Also throughput analysis is performed on the MSAG. The different CP variables forming the MSAG represent actors, channels, tokens and actor properties, respectively. An example with two simple application graphs and a possible resulting mapping and scheduling captured in an MSAG is illustrated in Figure 3.

Figure 3: Example of a mapping- and scheduling-aware graph (MSAG) in the CP model

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. With this type of schedules and blocking read and write primitives, the system can be implemented as bare-metal solution, without any special kind of scheduler or operating system. Making scheduling decisions potentially involves adding channels to the input application graphs in the MSAG. The order-based schedules are captured through the variables $next_a$ (1), pointing to a successor actor or, in case a is the last actor in the schedule, to a special reserved value. Figure 3b shows how the decision $next_{src_a} = src_b$ adds an edge between the two actors.

$$\forall a \in \{0 \dots |\mathcal{A}| + |P| - 1\} : next_a \in \{0 \dots |\mathcal{A}| + |P| - 1\} \quad (1)$$

Mapping of actors affects their WCET and can result in communication delays. The processor and processor mode assignments are captured by variables $proc_a$ (2) and $proc_mode_p$ (3), respectively. The actors' WCETs are assigned based on this mapping (4).

$$\forall a \in \mathcal{A} : proc_a \in P \quad (2)$$

$$\forall p \in P : proc_mode_p \in \{0 \dots |\mathcal{M}_p| - 1\} \quad (3)$$

$$\forall a \in \mathcal{A} : wct_a(proc_a, proc_mode_{proc_a}) \in \mathbb{N}^+ \quad (4)$$

If for any channel, the source and destination actors are mapped onto different processing nodes, additional actors reflecting the communication process and delays are added to the MSAG, as can be seen in Figure 3b for both channels of the input graphs. Channels for which both source and destination actor are mapped onto the same processing node are implemented with the local memory, which is assumed to not inflict any delay. Since the data dependency is respected by the schedule, no change is done to the MSAG for this case. For communication delays, the CP model contains a set of constraints that performs worst-case communication time (WCCT) analysis, based on the model of the TDMA bus. It involves blocking (5), sending (6) and potentially receiving time (7) as well as communication buffers and order-based scheduling of messages send from (8) or received at (9) the same node.

$$\forall c \in \mathcal{C} : wct_b_c \in \mathbb{N}^0 \quad (5)$$

$$\forall c \in \mathcal{C} : wct_s_c \in \mathbb{N}^0 \quad (6)$$

$$\forall c \in \mathcal{C} : wct_r_c \in \mathbb{N}^0 \quad (7)$$

$$\forall c \in \{0 \dots |\mathcal{C}| + |P| - 1\} : sendnext_c \in \{0 \dots |\mathcal{C}| + |P| - 1\} \quad (8)$$

$$\forall c \in \{0 \dots |\mathcal{C}| + |P| - 1\} : recnext_c \in \{0 \dots |\mathcal{C}| + |P| - 1\} \quad (9)$$

The model also captures memory consumed by actors instantiated on processing nodes, as well as communication buffers placed into the local memory. This ensures that the mapping is valid even in terms of sufficient memory on all processing nodes. The available buffer locations on the sending (10) and receiving (11) side are initial tokens in the MSAG. The non-filled circles in Figure 3b indicate that the buffer sizes are only bounded and not fixed, and part of the exploration.

$$\forall c \in \mathcal{C} : buffer_s_c \in \mathbb{N}^+ \quad (10)$$

$$\forall c \in \mathcal{C} : buffer_r_c \in \mathbb{N}^+ \quad (11)$$

Lastly, specialized throughput propagators that operate on the MSAG have been implemented as a means to enable throughput analysis in the CP model, as detailed in Section 4.

4. THROUGHPUT PROPAGATION

This section demonstrates the usefulness of throughput propagation in a CP-based DSE approach, and introduces two throughput propagators, using different analysis methods.

Figure 4 illustrates a possible partial search tree for the simple example in the root node: an SDF application graph with four actors, a two-processor platform and a design constraint on throughput, in terms of the scheduled graph's iteration period. In order to maintain traceability, the example is simplified with fixed WCETs, denoted in the gray

Figure 4: Example of a search tree with throughput propagation

circles attached to the actors, and without consideration of the details of WCCT analysis.

The search tree example shows how branching on actor mappings affects the minimal achievable iteration period, as inverse of maximal achievable throughput, in the current location of the search. Following the mapping decisions in the left branch, the design constraint is violated after the third step, which means the sub-tree with the failed node at its root is pruned from the tree, and search is continued in a different direction. Without throughput analysis of partially mapped and scheduled graphs by means of a throughput propagator inside the CP model, it would not be possible to discard all potential solutions of the sub-tree already at this stage. Considering that there are several additional variables to branch on after the mapping, such as scheduling of actors and channels, the impact of pruning sub-trees is significant.

After the failed node, search backtracks to the last still-feasible partial solution and continues with mapping actor C to processing element PE₁. This mapping adds the block, send and receive actors for the TDMA bus communication to the MSAG. Depending on the delays of these actors, which are only bounded at this stage and will be assigned later when branching on TDMA slot allocation, the throughput design constraint is still satisfiable. Thus, search will continue from this node.

Both throughput propagators are subscribed to the same variables of the CP model, which are used to determine the MSAG for the current state of the search. In addition to the variables illustrated in Figure 3: next_a , sendNext_c , recNext_c , buffer_s and buffer_r , the actor delays for application actors wcet_a and potential communication actors wcet_b , wcet_s and wcet_r are needed. An additional argument is the set of channels $\mathcal{C}$, for information about initial tokens and source and destination actors. The result of the analysis is propagated to CP variables period_g for the length of the iteration period, as the inverse of throughput, of each mapped and scheduled application graph g .

The maximum achievable throughput, i.e. minimal iteration period, of the MSAG is determined by the heaviest

cycle in the graph. Note that the CP model prohibits cycles without initial tokens since they are equivalent to deadlock. Cycles occur for different reasons, as demonstrated in Figure 3b. The scheduling creates a cycle of actors on each processing node. For communication on the TDMA bus, there are cycles combining communication and buffer sizes, i.e. source actor and block actor on the sending side and send actor and receive actor on the receiving side, respectively. The scheduling of channels with source actors on the same processing node creates a cycle of block and send actors. For cyclic input graphs, cycles across processors result in the MSAG if actors in the input graph cycle are mapped onto different processing nodes. The simplest cycles in the MSAG are created through self-edges on each actor, prohibiting auto-concurrency. These edges are omitted in Figure 3b for readability.

As mentioned in Section 3.2, a propagator must be correct and contracting. For the throughput propagators this means that for a partial mapping and scheduling, where not all CP variables are assigned a fixed value yet, it must propagate the lower bound on the achievable iteration period. In other words, the analysis must be performed in such a way that further decisions taken along the search tree branch cannot result in a smaller lower bound on the iteration period. This is achieved by using the minimum value of the domains for all variables that impose a delay, i.e. WCET and WCCT, and the maximum value for all variables that carry potential for parallelization, i.e. buffer sizes. For scheduling, only the assigned next_a variables are added as edges to the MSAG. The remainder of this section presents both throughput propagators developed for this paper, including their different representations of the MSAG.

4.1 MCR Propagator

This propagator calculates the minimum achievable iteration period of each application graph as the maximum cycle ratio (MCR) of the MSAG. The cycle ratio cr of a cycle c

of edges is defined as:

$$cr(c) = \frac{\sum_{e \in c} W_1(e)}{\sum_{e \in c} W_2(e)} \quad (12)$$

The boost graph library’s [3] `maximum_cycle_ratio` function is used to perform the computation of the MCR. Therefore, the representation of the MSAG for this propagator is a `boost::adjacency_list` with actor delays as `edge_weight` (W_1) and initial tokens, representing buffers, schedules and initial tokens from the input graphs, as `edge_weight2` (W_2).

The MSAG can be a disconnected graph consisting of several connected sub-graphs. This is the case if at least one application graph does not share any processing element with actors from another application graph. Therefore, the MCR propagator first analyses which application graphs are covered by which sub-graph of the MSAG. Then, MCR analysis is performed for each of the sub-graphs and the results are propagated to the `periodg` variables. As long as there are unassigned variables that can affect the period, i.e. the variables used to create the MSAG, the propagator removes all values from the `periodg`’s domains that are lower than the minimal achievable period determined by the analysis. Once all relevant variables are assigned, the propagator assigns the iteration period to its fixed value.

The experiments in Section 5 show that the MCR propagator is efficient. However, it can only propagate the minimal achievable iteration period. It may be possible to add further analyses that can use the same representation of the MSAG. For example, if relevant, a longest path analysis may be used to determine the initial latency of the pipelined schedule, after which the iteration period is reached. This would of course impose additional delay for analysis.

4.2 SSE Propagator

The second propagator uses state space exploration, as described in [10]. The MSAG is represented as a matrix that contains the initial token distribution of the MSAG, in combination with a vector with the minimal delay imposed by each MSAG actor, i.e. input application graph actors and communication actors. State space exploration is the simulation of the self-timed behavior of an SDFG. Self-timed means that actors execute as soon as all input tokens are available. This fits very well with the order-based schedules of the CP model.

The state space of the self-timed execution of a consistent SDFG is made up of a transient phase, followed by the periodic phase. The length of the periodic phase divided by the number of completed iterations of the graph is its iteration period, and the inverse of its throughput. The length of the transient phase is the initial latency, after which the throughput is reached. The SSE-based throughput propagator propagates the result for the iteration period to the `periodg` variables, and in addition the initial latency to variables `init_latencyg`. In the same manner as the MCR propagator, the SSE propagator propagates the lower bound by removing all smaller values from these variables’ domains, unless all variables that are used to create the MSAG have a fixed value, in which case it assigns the variables’ values.

Due to the fact that the state space exploration visits all states of self-time execution of the MSAG, it gathers more knowledge about the system that could be propagated back to the CP model. E.g., it is aware of the start and end times of all actor executions, which can be necessary information

Figure 5: Overview of the DeSyDe Framework

to capture in the CP model to enable its extension with an additional application model with a timed model of computation. Most importantly, this information can be extracted from the same SSE analysis as performed by the propagator in the current version and does not add further delay to the analysis. Even though the propagation of additional information is not needed in the current version of the DSE, it can be exploited as part of future work.

5. DeSyDe AND EXPERIMENTS

Figure 5 gives an overview of the DeSyDe DSE framework. The DeSyDe tool, including all files for the experiments presented in this paper, are publicly available on the DeSyDe github repository [8]. The framework takes the application graphs with their design constraints, the 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. A CP solver can be configured in many ways, e.g. using different, possibly heuristic, search techniques or guide the search to promising areas of the search tree through clever branching techniques. The work in this paper is focused on the comparison of throughput propagators and employs a standard depth-first search (DFS) for satisfy problems, and branch-and-bound (BAB) search for the optimization problems in the following experiments. The solver used in this paper [21] also supports parallel search through work-stealing, where different threads operate on different parts of the search tree. This feature is enabled for all of the experiments. All experiments were carried out on a system with an Intel Xeon CPU E3 running at 3.6GHz with 32GB RAM, and Ubuntu 14.04.

For the experiments, the six graphs of Figure 6a are used: five SDFGs of real applications and one synthetic graph as an example of a cyclic SDFG. A summary of the performed experiments is contained in Table 6b. Each column corresponds to a different experiment. The upper half of the table shows which applications graphs are part of each experiment, and with which kind of design constraint on throughput. The letter S means that a hard throughput constraint is specified that needs to be satisfied. The letter O indicates that the throughput for the application is subject to optimization. Experiments 4 to 6 use mixed-critical design

(a) The six application graphs used for the experiments

Exp.	1	2	3	4	5	6	7
App.							
Sobel			S		S		
SUSAN	S		S				
RASTA	S						S
JPEG enc.					O		
VAD						S	S
Synthetic		S		O		O	
Platform	4+2	4+2	4+2	4+4	4+4	4+4	4+2
Time SSE	16:52	56:06	01:33	00:17	01:34	16:09	720:02
Time MCR	02:35	13:55	00:18	00:07	00:51	03:05	720:05
#Sol. SSE	3,219,992	18,456,544	688,416	1	1	1	27,189,020
#Sol. MCR	3,219,992	18,456,544	688,416	1	1	1	49,295,301

(b) Experiment setup and results. The exploration times are shown using the following format: minutes:seconds. S: satisfy a specified throughput constraint. O: optimize throughput.

(c) Comparison of exploration times

Figure 6: Experiments setup and results.

constraints on throughput, combining hard and best-effort constraints.

For the target platform, two different heterogeneous platform models are considered, consisting of a number of *small* and *large* processors. The large processors run 30% faster than the small processors. Each processor can be initiated in two modes: standard mode or economy mode. In the economy mode the computation speed is slower than in the standard mode. The row in Table 6b stating the platform configuration used for each experiment indicates two numbers. The first number corresponds to the count of small processors, while the second number is the count of large processors on the platform.

The amount of time spent for exploration using each of the two propagators is provided in two separate rows in Table 6b. The last rows of the table show the number of designs found for each experiment. DeSyDe finds the same number of designs regardless of the choice of the propagator in experiments where the entire design space is explored, i.e. Experiments 1 to 6. Note that for Experiments 4 to 6, which involve optimization, a single solution is provided, since the BAB search engine is configured to only look for better, not equally good solutions, in the experiments. Experiment 7 reached the specified time-out of 12 hours with both versions of the throughput propagator. Therefore, the experiment was stopped before the entire design space was explored. However, the number of designs found by the propagators

within the 12 hour time-out differs significantly. DeSyDe with the SSE propagator found merely 55% of the designs found by the MCR propagator within the same time.

The MCR propagator outperformed the SSE propagator with respect to exploration time in all of the experiments. This can be attributed to the fact that the MCR propagator is analytical, while the SSE propagator performs a simulation of the entire state space. In order to compare the exploration times conveniently, the normalized exploration times are visualized in Figure 6c. The figure shows that the improvement was more significant in Experiments 1, 2 and 6 which have larger design spaces than the other experiments.

6. CONCLUSIONS AND FUTURE WORK

This paper presented two different throughput propagators in a CP-based design space exploration. The performance of the propagators has been compared in a range of experiments, partially with mixed-critical design constraints. The experiments show that the MCR propagator is more efficient in terms of run-time, while the SSE propagator has the potential to propagate further information, e.g. bounds for time-based schedules, without adding further analyses steps.

Directions for future work include the extension of the platform model with shared memories and networks-on-chip architectures and the consideration of throughput propagation based on max-plus algebra.

7. ACKNOWLEDGMENTS

This work was partially funded by the EU FP7 project CONTREX (FP7-611146) and the EU H2020 project SAFE-POWER (687902).

8. REFERENCES

- [1] A. Bonfietti, L. Benini, M. Lombardi, and M. Milano. An efficient and complete approach for throughput-maximal SDF allocation and scheduling on multi-core platforms. In *DATE*, pages 897–902, 2010.
- [2] A. Bonfietti, M. Lombardi, M. Milano, and L. Benini. Throughput constraint for synchronous data flow graphs. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 26–40. Springer Berlin Heidelberg, 2009.
- [3] Boost Graph Library. <http://www.boost.org>. Accessed: 2016-11-20.
- [4] G. C. Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications*. Springer, Santa Clara, CA, USA, 2011.
- [5] A. Dasdan and R. Gupta. Faster maximum and minimum mean cycle algorithms for system-performance analysis. *IEEE TCAD*, 17(10):889–899, 1998.
- [6] R. de Groote, P. K. F. Hölzenspies, J. Kuper, and H. Broersma. Back to basics: Homogeneous representations of multi-rate synchronous dataflow graphs. In *MEMCODE’13*, pages 35–46, 2013.
- [7] R. De Groote, J. Kuper, H. Broersma, and G. J. Smit. Max-plus algebraic throughput analysis of synchronous dataflow graphs. In *EUROMICRO-SEAA*, pages 29–38, 2012.
- [8] DeSyDe. <https://github.com/forsyde/DeSyDe>. Accessed: 2016-11-10.
- [9] M. Fakhri, 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. H. Ghamarian, M. Geilen, S. Stuijk, T. Basten, A. Moonen, M. J. Bekooij, B. D. Theelen, and M. Mousavi. Throughput analysis of synchronous data flow graphs. In *ACSD*, pages 25–36, 2006.
- [11] A. Hansson, K. Goossens, M. Bekooij, and J. Huisken. CoMPSoC: A template for composable and predictable multi-processor system on chips. *ACM TODAES*, 14(1):2:1–2:24, Jan. 2009.
- [12] K. Ito and K. K. Parhi. Determining the minimum iteration period of an algorithm. *VLSI Signal Processing*, 11(3):229–244, 1995.
- [13] N. Khalilzad, K. Rosvall, and I. Sander. A modular design space exploration framework for multiprocessor real-time systems. In *FDL*, September 2016.
- [14] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. In *Proceedings of the IEEE*, volume 75 of 9, pages 1235–1245, September 1987.
- [15] E. A. Lee and A. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE TCAD*, 17(12):1217–1229, Dec. 1998.
- [16] 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*, pages 137–146, New York, NY, USA, 2008. ACM.
- [17] U. M. Mirza, F. Gruian, and K. Kuchcinski. Mapping streaming applications on multiprocessors with time-division-multiplexed network-on-chip. *CEE*, 40(8):276–291, 2014.
- [18] 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*, pages 332–341, 2005.
- [19] C. Pitter and M. Schoeberl. A real-time Java chip-multiprocessor. *ACM TECS*, 10(1):9:1–9:34, Aug. 2010.
- [20] K. Rosvall and I. Sander. A constraint-based design space exploration framework for real-time applications on MPSoCs. In *DATE*, pages 1–6, 2014.
- [21] C. Schulte, G. Tack, and M. Z. Lagerkvist. *Modeling and Programming with Gecode*, 2016. <http://www.gecode.org/>.
- [22] S. Sriram and S. S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker, Inc., New York, NY, USA, 2000.
- [23] S. Stuijk, T. Basten, M. Geilen, and H. Corporaal. Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In *DAC*, pages 777–782, 2007.
- [24] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, et al. The worst-case execution-time problem—overview of methods and survey of tools. *ACM TECS*, 7(3):1–53, 2008.
- [25] J. Zhu, I. Sander, and A. Jantsch. Buffer minimization of real-time streaming applications scheduling on hybrid CPU/FPGA architectures. In *DATE*, pages 1506–1511, 2009.
- [26] J. Zhu, I. Sander, and A. Jantsch. Constrained global scheduling of streaming applications on MPSoCs. In *ASP-DAC*, pages 223–228, 2010.