A Constraint-Based Design Space Exploration Framework for Real-Time Applications on MPSoCs

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

Abstract—**Design space exploration (DSE) is a critical step in the design process of real-time multiprocessor systems. Combining a formal base in form of SDF graphs with predictable platforms providing guaranteed QoS, the paper proposes a flexible and extendable DSE framework that can provide performance guarantees for multiple applications implemented on a shared platform. The DSE framework is formulated in a declarative style as interprocess communication-aware constraint programming (CP) model. Apart from mapping and scheduling of application graphs, the model supports design constraints on several cost and performance metrics, as e.g. memory consumption and achievable throughput. Using constraints with different compliance level, the framework introduces support for mixed criticality in the CP model. The potential of the approach is demonstrated by means of experiments using a Sobel filter, a SUSAN filter, a RASTA-PLP application and a JPEG encoder.**

I. Introduction

Although multiprocessor embedded platforms have been available for many years, there is still a considerable lack of systematic design methods that can guarantee real-time performance for the final implementation. Current industrial multiprocessor platforms are extremely difficult to analyze due to shared resources and caches [1], and provide very little support for a correct-by-construction design flow. As a consequence, the verification costs are exploding and are already dominating the overall design costs. To overcome this situation, many predictable platforms, like PRET [2], CoMPSoC [3], or JOP [4], have been proposed in recent years. Together with models of computations (MoCs) [5], providing a well-defined semantics for application models, these predictable platforms form a solid base for a big step towards a correct-by-construction design flow.

A key component of a correct-by-construction design flow is the design space exploration (DSE) framework which performs the interdependent steps of mapping, scheduling and performance analysis with the goal to provide multiprocessor schedules with performance guarantees. This paper introduces a flexible and extendable DSE framework, combining a formal MoC with predictable platform services. The framework is formulated in a declarative style as constraint satisfaction problem. The main contribution is a constraint-based DSE framework which

- maps and schedules multiple applications on a shared platform for computation and communication, providing cost and performance guarantees for the individual applications and the system as a whole,
- supports mixed-criticality and performance metrics in terms of allocated resources, memory consumption, buffer dimensioning, processor utilization and achievable throughput and can be conveniently extended with further metrics without modifying the existing model,
- integrates a specialized constraint for interprocessor communication aware throughput analysis,
- solves the DSE problem as a whole and thereby avoids the disadvantages of decomposing it into the interdependent sub-problems, and
- has illustrated its potential through a case study with typical streaming applications and mixed-critical design constraints.

II. Related Work

In order to handle the complexity of the problem, mapping and scheduling of SDFGs onto multiprocessor platforms under performance constraints is often decomposed into its three sub-problems, as in [6], [7], [8] and [9]. First, a mapping is established based on an approximate cost function, e.g. balancing the workload on the nodes in [7]. In a subsequent step, a schedule is derived for the obtained mapping, typically resorting to heuristic techniques such as list scheduling [6]. Finally, the performance of the resulting mapping and scheduling is analyzed.

The CP-based approach used in this paper has the advantage that mapping, scheduling and performance analysis are performed simultaneously, i.e. the problem is not decomposed. The impact of each step in the mapping and scheduling process is evaluated immediately and as soon as a partial solution proves unsatisfactory in terms of performance or cost metrics, the search is continued in a different direction. Due to avoided decomposition, the solution space of the model contains all possible solutions and the chosen search technique, as described in Section III-B, can either guarantee completeness and optimality or, for larger scale problems, balance search time and quality of found solutions.

Another advantage of the CP approach is that the same model supports DSE problems with different goals, i.e. maximizing for throughput or minimizing the number of allocated processors while satisfying memory constraints. In addition, new cost or performance metrics can be added without changing the existing model. By this means, a CP-based DSE framework offers flexibility and extendibility, as opposed to problem-restricted heuristic approaches.

There are a number of related CP approaches. The work presented in [10] proposed a CP approach to map and schedule acyclic task graphs on multiple resources. The method presented in this paper can handle both acyclic and cyclic graphs. The work in [11], [12] maps and schedules SDFGs on a heterogeneous multiprocessor platform with buffer and throughput constraints, but in an iterative process. The approach of 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.

The work of Bonfietti et al. [13], [14] is most similar to the approach used in this paper. However, that work does not consider interprocessor communication, buffer dimensioning, memory consumption, resource utilization or a specific target platform. Also the throughput computation for the DSE framework in this paper has been extended to enable interprocessor communication-aware throughput analysis. Most significantly, none of the above mentioned approaches, heuristic or CP-based, supports simultaneous scheduling of multiple applications. The work in [7] considers multiple applications, but each application is scheduled individually and seperated by means of a TDMA time wheel. 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.

III. The DSE Framework

Figure 1 illustrates the proposed design space exploration (DSE) framework. It is designed to support three interdependent activities: *mapping*, *scheduling* and *performance analysis*. The objective of the design space exploration framework is to find an efficient implementation of a set of SDF-applications with individual design constraints on a shared multiprocessor platform which can provide predictable performance. The framework supports cost and performance analysis in terms of allocated resources, memory consumption, buffer dimensioning, processor utilization and minimum guaranteed throughput. The following sections describe the framework from input specifications to the resulting output.

A. Input

As illustrated in Figure 1, four different types of information serve as input to the framework: One or more *application graphs* with an associated set of *design constraints*, a description of the *target platform* for the implementation and the processor-specific *application properties* in terms of worst-case execution times and memory consumption.

Application Model: In order to allow for accurate performance analysis at compile time, it is required that the application description has an underlying formal model of computation (MoC) which provides a sufficient level of analyzability. In the current framework version SDF graphs (SDFGs) [15] are used to describe the applications. The SDF MoC is commonly used for streaming applications. It has the advantage of high analyzability at compile time, e.g. static scheduling, buffer dimensioning and throughput computation.

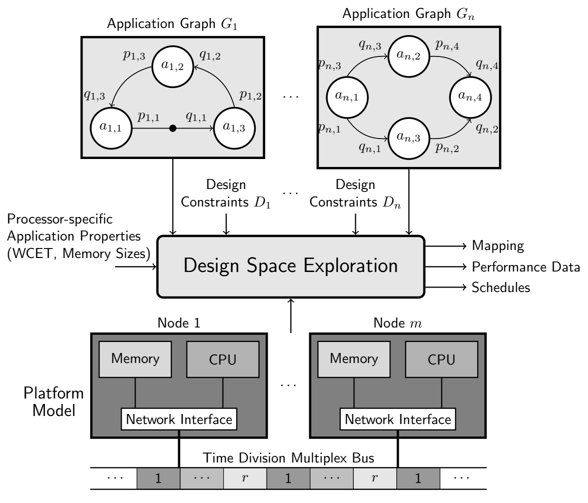


Figure 1. Overview of the DSE Framework

An SDFG $G(A, C)$ consists of a finite set of actors A and a finite set of channels C. An actor $a \in A$ produces a fixed rate of p tokens on outport op and sends them to an actor b that consumes a fixed rate of q tokens from inport ip via a channel $c = (a,\ op,\ p,\ b,\ ip,\ q,\ tok)$. A channel can contain initial tokens, denoted by tok. Initial tokens are visualized by a dot on the channel, optionally with an adjacent integer indicating the presence of more than one token.

The fixed production and consumption rates allow for static scheduling of SDFGs. For all consistent SDFGs, a function $\gamma : A \to \mathbb{N}$ gives the number of repetitions of each actor during one iteration of the graph. At the end of one iteration, the state of the channels is equal to the initial token configuration.

When using techniques as CP, it is important that the constraint model is not built on restricting assumptions. In case of the mapping and scheduling problem, this means that the full potential of parallelism needs to be exposed to the model. The design constraints, as described in a following part of this section, can then be used to limit the solution space if desired, e.g. by specifying that all instances of an SDF actor shall be mapped onto the same node.

The CP model is therefore supplied with the most parallel representation of one iteration of the SDFGs, which contains all $\gamma(a)$ repetitions of each actor a and the dependencies, i.e. tokens exchanged, between them. Algorithm 1 shows how to convert an SDFG into the most parallel representation. As a result, every channel will have homogeneous rates, as can be seen in the example in Figure 2. Note that the conversion is different from the conversion to HSDF in [16], since the rates can be greater than 1. In the following, the term *actor* will refer to the actors of the SDFG with homogeneous rates, unless otherwise stated.

Platform Model: In order to provide real-time guarantees, the underlying platform needs to be predictable with respect to timing behavior. To enable worst-case communication time analysis in the current version of the DSE framework, we assume a TDM bus-based multiprocessor architecture as illustrated in Figure 1. A TDM round consists of several time

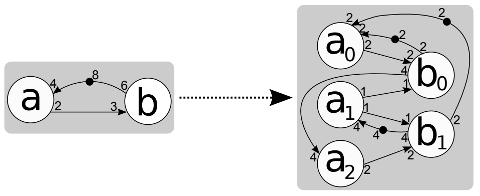


Figure 2. SDFG conversion example with $\gamma(a,\ b) = (3,\ 2)$

Algorithm 1 SDFG conversion

```
Input: SDF graph $G = (A, C)$ with repetition vector $\gamma(A)$
Output: rate-homogeneous SDF graph $G' = (A', C')$
for each actor $a$ in $A$, add $\gamma(a)$ actors $a_0, ..., a_{\gamma(a)-1}$ to $A'$
for each actor $a$ in $A'$, set $outports_a = 0$ and $inports_a = 0$
for each channel $(a, op, p, b, ip, q, tok)$ in $C$ do
    $i,\ k,\ r,\ t := 0$
    $j := outports_{a_i}$
    $l := inports_{b_k}$
    $x := \gamma(a) \cdot p$
    while $x > 0$ do
        if r==0 then
            $p' := min(p, q)$
            $x := x - p'$
            $r := |p - q|$
        else
            $p' := min(r, p, q)$
            $x := x - p'$
            $r := max(x\ mod\ p, x\ mod\ q)$
        if $tok > 0$ and $tok > x$ then $t := p'$;
        add channel $(a_i, j, p', b_k, l, p', t)$ to $C'$
        $outports_{a_i} := outports_{a_i} + 1$
        $inports_{b_k} := inports_{b_k} + 1$
        if $(\gamma(a) * p - x)\ mod\ p == 0$ then
            $i := (i + 1)\ mod\ \gamma(a)$
            $j := outports_{a_i}$
        else $j := (j + 1)$
        if $(\gamma(b) * q - x)\ mod\ q == 0$ then
            $k := (k + 1)\ mod\ \gamma(b)$
            $l := inports_{b_k}$
        else $l := (l + 1)$
```

slots and each processing node which sends data over the interconnect will be assigned one or more dedicated slots.

For interprocessor communication by means of token exchange, it is assumed that the buffers are placed into the local memory of the processing node hosting the receiving actor. The DSE framework can provide both self-timed or static schedules, as described in the following section. To accommodate for self-timed schedules, the platform needs to provide blocking send and receive primitives which are provided by a dedicated communication block, e.g. a network interface. Static schedules require a time-based runtime scheduler.

A platform can have different kinds of processing nodes, e.g. a general purpose CPU or a dedicated hardware accelerator for a specific SDF actor function. For all valid combinations of actors and processing nodes, the resulting properties in terms of worst-case execution time (WCET) and memory consumption is an additional input to the framework. At a later stage, the platform model is planned to be extended, e.g. by integrating shared memory and network-on-chip communication.

Design Constraints: The proposed framework allows the designer to specify constraints that the generated mapping and scheduling have to satisfy. These types of constraints will be referred to as *design constraints* to avoid confusion with the constraints that constitute a constraint programming model.

There are two classes of design constraints. They can either concern performance and cost metrics, as for example throughput or memory consumption, or the mapping and scheduling decisions itself. For the performance metrics, the level of compliance needs to be specified as either *satisfy* or *optimize* (i.e. maximize or minimize). By this means, the DSE framework supports mixed criticality applications in the sense that satisfy-constraints are hard constraints that *must* be fulfilled. Optimize-constraints allow for best-effort solutions, using the remaining resources after fulfilling all hard constraints. Furthermore, the design constraints can have a local or global scope, concerning either individual applications, actors and processing nodes or the entire system. An example for a local constraint can be *"Application 1 must produce at least one sample every 500 cycles."*. A global constraint could e.g. minimize the number of allocated processing nodes.

The set of design constraints concerning the mapping and scheduling decisions give the designer the opportunity to directly influence the process if desired. An example for such a design constraint is *"Actor A shall be mapped onto processing node X (or: a processing node of type X)."*, where X could e.g. be a specific hardware accelerator for the function of actor A.

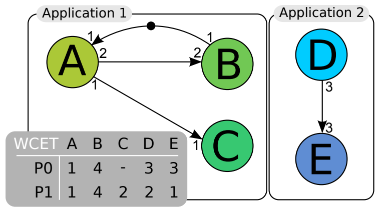
The use of design constraints is a very powerful method which allows to use the same DSE model for mapping and scheduling SDFGs on a shared multiprocessor platform for problems with diverse design goals. Design constraints can be combined in any thinkable way to fit the design requirements. This provides for a framework with great flexibility.

B. Method

The problem of mapping and scheduling is a typical application for combinatorial optimization. Constraint Programming (CP) is a well-proven technique for solving combinatorial problems, with and without optimality requirements. In CP, the components of a problem are modeled in terms of *decision variables*. Each variable has a *domain* of possible values. The relationship between variables are described in form 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 of the solution 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 by trying the remaining alternatives from the variable domains after propagation. Lastly, the search operates on the created tree to find solutions that satisfy all constraints.

A fundamental advantage of CP over the commonly used heuristic algorithms is the separation of the description of the problem from the way it is solved. Due to this separation of concerns, different solvers and solving techniques for finding a solution to the same constraint model can be applied, e.g. with respect to the scale of the problem. If exhaustive search is applicable, the approach is optimal and complete, i.e. finding the optimal solution is guaranteed and finding no solution means that no solution exists. For problems of larger scale, it may be preferred to sacrifice optimality and completeness through the application of a heuristic search method which



WCET	A	B	C	D	E
P0	1	4	-	3	3
P1	1	4	2	2	1

Figure 3. Example Application Graphs with WCET table

can find a solution in reasonable time. Whichever search method is chosen to find a solution, they can all operate on the *same CP model*. Furthermore, a CP model is modular and therefore easily extendable due to the declarative nature of the models. New components, e.g. additional performance metrics, can be conveniently integrated without modifying the existing constraints of the CP model.

The decision variables in the current version of the CP model are described in Section IV.

C. Output

As final result, the framework provides *1*) an allocation of platform resources, *2*) a mapping of actors to processing nodes, *3*) a schedule ordering the actors on each node, *4*) a schedule for interprocess communication for each node on the TDM bus with derived worst-case communication times (WCCTs), and *5*) cost and performance data for the system and all applications, currently in terms of throughput, memory consumption, minimum buffer sizes and processor utilization.

The framework maps and schedules one iteration of the SDFG. The created schedules are pipelined and not blocked, which means that the execution of different iterations of the graph can overlap in time on different nodes.

Within the CP model, the representation of schedules is order-based (as opposed to time-based). As final output of the framework, both order-based, i.e. self-timed according to the terminology of [17], or time-based fully static schedules can be provided. For self-timed schedules, only the order and not the exact start times of actor executions are specified. Instead, exact start times result from the arrival of the required data tokens at run time. As support for self-timed execution, the platform must provide blocking write and read mechanisms for interprocessor communication, since the actor executions synchronize by means of the exchanged data tokens. For fully static schedules, all start times are fixed. A runtime scheduler is needed to support this type of schedules.

In either case, the schedule consists of a transient phase and a periodic phase. By transient phase, we refer to the time it takes to execute the first iteration of the graph when the system is started. The length of the transient phase, the *initial latency*, is individual for each application, as is the throughput. The throughput is the inverse of the length of the periodic phase, i.e. the time it takes to complete one iteration of the SDF application graph after the transient phase.

Figure 4 shows a schedule, in form of the fully static version, which the framework finds for the example application graphs in Figure 3 and a platform with 2 processing nodes. The schedule contains computation performed on the processing nodes $P0$ and $P1$ as well as communication in form of

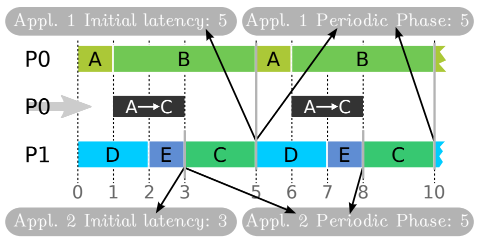


Figure 4. Possible mapping and schedule on two nodes for the application graphs in Figure 3

messages sent over the TDM bus (in the row marked with a gray arrow). In the presented solution, actors A and B are mapped onto $P0$, while C, D and E are mapped onto $P1$. For WCCT analysis, all local communication is assumed to have zero delay. Only the token exchange from actor A to actor C suffers a communication delay due to the bus transfer. Based on the bus bandwidth, number of TDM slots, length of the TDM round and token rate and size, the WCCT for this example has been determined to be 2 cycles. In order to capture the worst case for the communication time, it has to be assumed that a message becomes available to the network interface for sending exactly when its last TDM slot ends.

As illustrated in Figure 4, the first iteration of Application 1 is completed after 5 cycles. Hence, the initial latency is 5 cycles. The second iteration completes at cycle 10, yielding a throughput of $\frac{1}{5}\frac{samples}{cycles}$. Application 2 also reaches a throughput of $\frac{1}{5}\frac{samples}{cycles}$, after an initial latency of 3 cycles. All buffers need to hold one token each. The utilization for each processing node is the sum of the actors' WCETs on that node divided by the length of the periodic phase of the applications allocated to that node. In the presented solution, both $P0$ and $P1$ have a utilization of 1.

IV. The CP Model

This section describes the decision variables which constitute the CP model. 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 and a bus divided into m TDM slots. 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. The WCET and memory consumption of actors on processing nodes is given by the functions $t : \mathcal{A} \times P \rightarrow \mathbb{N}$ and $mem : \mathcal{A} \times P \rightarrow \mathbb{N}$. In the following, we list all variables in the CP model and explain how they are constrained.

$\texttt{proc}_a \in P, \forall a \in \mathcal{A}$: captures the mapping of actors.

$\texttt{next}_a \in \mathcal{A} \cup \{-1\}, \forall a \in \mathcal{A}$: captures the order-based schedules for the processing nodes. The value of $\texttt{next}_a$ identifies the direct successor of an actor a in the schedule for node $\texttt{proc}_a$, whereas -1 indicates that a is the last actor on that node. For every pair of actors $(a,\ b)$, if b depends on a, i.e. a series of channels with no initial tokens leads from a to b, then a may not be scheduled after b on the same processing node. A `no_cycle` constraint ensures that cyclic dependencies on and across processing nodes are avoided in order to prevent deadlock.

$\texttt{sendOrder}_c$ and $\texttt{recOrder}_c \in |\mathcal{C}|, \forall c \in \mathcal{C}$: describe the communication schedule by ordering the messages sent from

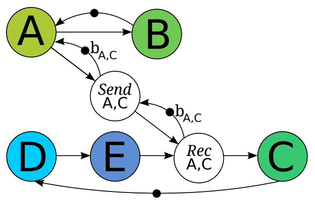


Figure 5. Mapping- and scheduling-aware graph representing the mapping and schedule on two processors from Figure 4

and received on each processing node. The order depends primarily on the actor schedule, i.e. the $\texttt{next}_a$ variables, but for cases were one actor has several outgoing (incoming) channels for which the receiving (sending) actors are mapped on another processing node, the order can be set arbitrarily and will therefore generate branches in the search tree.

$\texttt{tdmAlloc}_p \in [0, m], \forall p \in P$: allocates a number of TDM slots to each processing node. The sum of allocated slots may not exceed m. A processing node without any interprocessor communication is assigned 0 slots. If m is greater than the number of processing nodes that need to access the bus, then the additional slots can be distributed arbitrarily. Hence, the variables will generate branches in the search tree.

$\texttt{wcet}_a \in \mathbb{N}$ *($\forall a \in \mathcal{A}$):* assigns the WCET to each actor, depending on the mapping. I.e. $\texttt{wcet}_a := t(a, \texttt{proc}_a)$.

$\texttt{wcct}_c \in \mathbb{N}$ *($\forall c \in \mathcal{C}$):* determines the WCCT of each channel. If the sending and receiving actor are mapped onto the same node, the WCCT is 0. Otherwise it is derived from token rate and size, allocated TDM slots, bus bandwidth and communication schedule, considering potentially preceding, blocking messages. Since, in the current version of the model, the token buffers are assumed to be placed into the local memory of the receiving actors' processing node, the WCCT only comprises the sending time. Once shared memories are integrated into the platform model, the WCCT analysis will also include the receiving time.

$\texttt{initLatency}_z \in \mathbb{N}, \forall z \in [0, n-1]$: captures the length of the first iteration of each application graph, derived from schedule and WCETs.

$\texttt{period}_z \in \mathbb{N}, \forall z \in [0, n-1]$: gives the length of the periodic phase, i.e. the inverse of the throughput, for each application. For the presented model, a specialized constraint has been implemented which performs a maximum cycle mean (MCM) analysis on a graph representation of the mapped and scheduled application graphs with actor delays corresponding to WCETs for the application actors and WCCT for communication actors. For details on MCM analysis, we refer to [18]. The MCM analysis is performed for each application, starting from its first actor. An example for a mapping- and scheduling-aware graph (MSAG) is shown in Figure 5, representing the mapping and scheduling from Figure 4.

The schedule on each node, given by the $\texttt{next}_a$ variables, is represented by the cycles of actors (A,B) and (D,E,C). For each channel c with $\texttt{wcct}_c > 0$, two communication actors are added to the MSAG, one representing the sending delay and the other representing the receiving delay (currently 0). The communication actors are ordered according to $\texttt{sendOrder}_c$ and $\texttt{recOrder}_c$. Initial tokens, if present, are placed on the channel from sending to receiving actor. The back-edges with initial tokens from the communication actors represent free spaces in the buffer.

$\texttt{bufferSz}_c \in \mathbb{N}$ *($\forall c \in \mathcal{C}$):* gives the buffer size for each channel required in order to achieve the predicted throughput. Two dependent actors can be executed in parallel in the periodic phase (in a pipelined fashion) if sufficient tokens for the connecting channel are produced in the transient phase. The amount of tokens produced determines the buffer size.

$\texttt{memLoad}_p \in \mathbb{N}, \forall p \in P$: is calculated from the sum of actor sizes $mem(a, \texttt{proc}_a)$ with $\texttt{proc}_a = p$, i.e. all actors allocated to the node p, and the sum of buffer sizes that are placed in to the nodes local memory. $\texttt{memLoad}_p$ must not exceed the local memory size of processing node p.

$\texttt{utilization}_p \in \mathbb{Q}[0,1], \forall p \in P$: gives the processor utilization as the ratio between sum of WCETs of actors on node p and the length of the period of the application(s) which have actors mapped onto p.

$\texttt{procsAlloc} \in |P|$: gives the number of processing nodes with $\texttt{utilization}_p > 0$

In addition to the constraints described above, the specified design constraints are also integrated into the CP model as constraints on the decision variables. The variables $\texttt{proc}_a$, $\texttt{next}_a$, $\texttt{sendOrder}_c$, $\texttt{recOrder}_c$ and $\texttt{tdmAlloc}_p$ are used for branching to construct the search tree. The values of the remaining variables result directly from assignments of the aforementioned.

V. Experiments

In order to evaluate the potential of the presented approach, we use four typical streaming applications for experiments, as depicted in Figure 6. Because the applications have been gathered from various sources, both WCETs and actor sizes for data and memory have been adapted and normalized to cycles for time units and data units (du) for data and memory sizes. The target platform consists of 8 homogeneous processing nodes that are connected via a TDMA bus with 8 slots per round and a bandwidth of 32 $\frac{du}{cycle}$. The length of the TDMA round is 1 cycle. All tokens have a size of 8 du and the local memory of the processing nodes is 160 du. The actor's WCETs and sizes are listed in Table I. The experiments were conducted on a Quad-core running at 2.5 GHz with 8 GB of RAM, using the Gecode constraint solver.

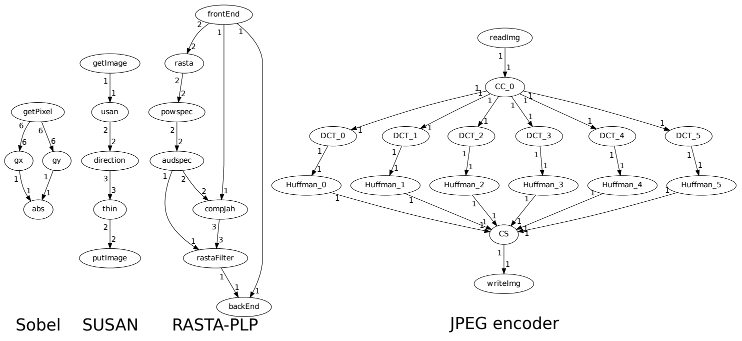


Figure 6. The SDF applications graphs used in the experiments

The DSE process has been evaluated with all possible combinations from one up to all four applications sharing the same platform for computation and communication. The

Table I
ACTOR WCETS [CYCLES] AND SIZES [DU]

Actor	WCET	size	Actor	WCET	size
getPixel	320	2	powspec	235	6
gx	77	4	audspec	108	5
gy	77	4	compJah	170	4
abs	123	5	rastaFilter	194	7
getImage	20	2	backEnd	133	5
usan	1177	4	getImg	413	8
direction	833	4	CC	1101	4
thin	32	5	DCT	252	6
putImage	15	5	Huffman	340	5
frontEnd	141	2	CS	2524	4
rasta	31	4	writeImg	132	5

Table II
EXPERIMENTAL RESULTS. RUNTIME AND GUARANTEED THROUGHPUT^{-1} $[\frac{cycles}{sample}]$. GRAY CELLS: BEST-EFFORT CANDIDATE.

Scenario	**First Solution**				**Best Solution**			
	So	*Su*	*Ra*	*Jp*	*So*	*Su*	*Ra*	*Jp*
Sobel	41ms				41ms			
	320				320			
Susan	38ms				38ms			
		1177				1177		
Rasta	212ms				212ms			
			235				235	
JPEG	1467ms				66311ms			
				2656				2524
SoSu	251ms				3402ms			
	320	1197			320	1177		
SoRa	1155ms				19059ms			
	364		364		320		278	
SoJp	4761ms				13539ms			
	320			3248	397			2656
SuRa	1884ms				7612ms			
		1561	1561			1197	235	
SuJp	8731ms				26390ms			
		1197		3248		1197		2524
RaJp	9126ms				46670ms*			
			364	3248			364	2996
SoSuRa	3522ms				47790ms*			
	397	2030	515		397	2030	343	
SoSuJp	13688ms				29751ms*			
	397	2030		4432	397	2030		3290
SoRaJp	10378ms				173186ms*			
	397		515	4432	397		515	2656
SuRaJp	7594ms				1229642ms*			
		2030	515	4432		2030	515	2786
SoSuRaJp	10644ms				253305ms*			
	397	2030	540	5606	397	2030	540	4432

framework was configured to satisfy real-time constraints in terms of required throughput for all but one application. The remaining application was mapped and scheduled after best-effort with the goal to maximize the throughput to reflect mixed-criticality optimization. Table II lists all evaluated scenarios, i.e. combinations of applications. The cells with gray background indicate which application was chosen for best-effort in each scenario. The required throughput bounds were set as follows: Sobel $\frac{1}{400}\frac{sample}{cycles}$, SUSAN $\frac{1}{1050}\frac{sample}{cycles}$ and RASTA $\frac{1}{550}\frac{sample}{cycles}$, considering that one iteration of the SDFG produces one *sample*.

In order to satisfy the hard real-time constraints, the search was in all cases continued until a satisfying solution was found. As a means to balance the effort spent in terms of search time and the quality of the solution in terms of minimized throughput, the optimizing search was interrupted if no better solution was found within 30 minutes (1800000ms).

Table II contains two columns with results. The first column shows the time after which a first satisfying solution was found and also the quality of the solution in terms of throughput. Note that the results are given as inverse of the throughput, i.e. length of the periodic phase. The second column shows the best solution that could be found within the time limit of 30 minutes after a previous solution. For the first three cases, the optimal solution was found at once. For all other scenarios better solutions were found. For the first 9 scenarios, the search for an optimal solution was completed within the time limit. For the remaining 6 scenarios, marked with an asterix, no guarantee for optimality can be given, because the search has not completed within the time limit.

We want to emphasize that a simple branch-and-bound search engine without elaborated branching techniques was used for the experiments, which is reasonable since the focus of the paper lies in the modeling of the DSE problem. Nonetheless, the framework has already in this stage proven capable of finding good-quality solutions within a reasonable amount of time. As a next step, we plan to investigate how different search techniques can be exploited to efficiently find solutions for larger scale problems. We also want to clarify that the goal of the framework is not to find the global optimum, but to spent reasonable effort to find good quality solutions.

VI. CONCLUSIONS AND FUTURE WORK

The paper has presented a flexible and extendable DSE for real-time multiprocessor systems based on constraint programming. As demonstrated in the experiments, the framework can give guarantees for individual applications with specific design constraints of mixed-criticality sharing the same predictable platform. In future work we plan to extend the framework to integrate shared memories and networks-on-chip into the platform model, and to explore efficient solving techniques.

REFERENCES

[1] R. Wilhelm *et al.*, “The worst-case execution-time problem—overview of methods and survey of tools,” *ACM TECS*, vol. 7, no. 3, 2008.
[2] B. Lickly *et al.*, “Predictable programming on a precision timed architecture,” in *CASES*, 2008.
[3] A. Hansson *et al.*, “CoMPSoC: A template for composable and predictable multi-processor system on chips,” *ACM TODAES*, 2009.
[4] C. Pitter and M. Schoeberl, “A real-time Java chip-multiprocessor,” *ACM TECS*, vol. 10, no. 1, 2010.
[5] E. A. Lee and A. Sangiovanni-Vincentelli, “A framework for comparing models of computation,” *IEEE TCAD*, vol. 17, no. 12, 1998.
[6] S. S. Bhattacharyya *et al.*, “Optimized software synthesis for synchronous dataflow,” in *ASAP*, 1997.
[7] S. Stuijk *et al.*, “Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs,” in *DAC*, 2007.
[8] O. Moreira *et al.*, “Multiprocessor resource allocation for hard-real-time streaming with a dynamic job-mix,” in *IEEE RTAS*, 2005.
[9] M. Fakih *et al.*, “Towards performance analysis of SDFGs mapped to shared–bus architectures using model–checking,” in *DATE*, 2013.
[10] K. Kuchcinski, “Constraints-driven scheduling and resource assignment,” *ACM Trans. Design Autom. Electr. Syst.*, vol. 8, no. 3, 2003.
[11] J. Zhu *et al.*, “Buffer minimization of real-time streaming applications scheduling on hybrid CPU/FPGA architectures,” in *DATE*, 2009.
[12] ——, “Constrained global scheduling of streaming applications on MPSoCs,” in *ASP-DAC*, 2010.
[13] A. Bonfietti *et al.*, “Throughput constraint for synchronous data flow graphs,” in *CPAIOR*, 2009.
[14] ——, “An efficient and complete approach for throughput-maximal SDF allocation and scheduling on multi-core platforms,” in *DATE*, 2010.
[15] E. A. Lee and D. G. Messerschmitt, “Synchronous data flow,” in *Proceedings of the IEEE*, ser. 9, vol. 75, 1987.
[16] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*, 1st ed. Marcel Dekker, Inc., 2000.
[17] E. Lee and S. Ha, “Scheduling strategies for multiprocessor real-time DSP,” in *GLOBECOM ’89*, 1989.
[18] A. Dasdan and R. Gupta, “Faster maximum and minimum mean cycle algorithms for system-performance analysis,” *IEEE TCAD*, 1998.