ADFG: a scheduling synthesis tool for dataflow graphs in real-time systems

A. Honorat
INRIA
alexandre.honorat@inria.fr

H. N. Tran
INRIA
hai-nam.tran@inria.fr

L. Besnard
CNRS
loic.besnard@irisa.fr

T. Gautier
INRIA
thierry.gautier@inria.fr

J.-P. Talpin
INRIA
jean-pierre.talpin@inria.fr

A. Bouakaz
ANSYS
adnan.bouakaz@ansys.com

ABSTRACT

This paper presents a synthesis tool of real-time system scheduling parameters: ADFG computes task periods and buffer sizes of systems as signal processing applications, resulting in a trade-off between throughput maximization and buffer size minimization. ADFG synthesizes systems modeled by ultimately cyclo-static dataflow (UCSDF) graphs, an extension of the standard CSDF model. Two new synthesis algorithms are also introduced and evaluated.

ACM Reference format:
A. Honorat, H. N. Tran, L. Besnard, T. Gautier, J.-P. Talpin, and A. Bouakaz. 2017. ADFG: a scheduling synthesis tool for dataflow graphs in real-time systems. In *Proceedings of RTNS '17, Grenoble, France, October 4–6, 2017,* 10 pages.
DOI: 10.1145/3139258.3139267

1 INTRODUCTION

Real-time systems can be found in a wide range of domains, to ensure safety (e.g. in medical devices such as pacemakers [26]) or throughput (e.g. in digital signal processors (DSP) such as video decoders [24]). These systems can be abstracted as a set of tasks that must be run on the available processors with at most one task running at a time on each processor. Tasks have deadlines and exchange data (their inputs/outputs) through communication buffers. The main real-time challenge is to schedule all tasks: i.e. to execute them while respecting deadlines, processors availability, data exchange constraints, and eventually while respecting a throughput constraint.

In this paper we introduce the tool ADFG (Affine DataFlow Graph) which synthesizes real-time system scheduling parameters: it computes a safe periodic schedule ensuring throughput maximization and buffer size minimization.

The problem discussed in this work and solved by ADFG is the synthesis of periodic scheduling parameters for real-time systems modeled as ultimately cyclo-static dataflow (UCSDF) graphs. This synthesis aims for a trade-off between throughput maximization and total buffer size minimization. The synthesizer inputs are: a UCSDF graph which describes tasks by their Worst Case Execution Time (WCET), and directed buffers connecting tasks by their data production and consumption rates; the number of processors in the target system; the real-time scheduling synthesis algorithm to be used. The outputs are the synthesized scheduling parameters: the tasks periods, offsets, processor bindings and priorities, and the buffers initial marking and maximum sizes. The task and scheduling models are formally introduced in section 2, UCSDF graph model is formally introduced in section 4.

ADFG was originally the implementation of Bouakaz's work [8]. However the tool had not been packaged yet to be easily installed and used. Code refactoring led to improve the theory, and also to add some features. Therefore contributions presented in this paper cover theory, implementation and evaluation. Firstly, more accurate bounds and Integer Linear Programming (ILP) formulations have been used. Besides, dataflow graphs need no more to be weakly connected [1] for EDF policy on multiprocessor systems. The new implementation also avoids to use a fixed parameter for some multiprocessor partitioning algorithms, now an optional strategy enables to compute it. Finally implementation has been adapted to standard technologies to be more easily installed and used. As the synthesizer evolved a lot, new evaluations have been made.

The remainder of the paper is organized as follows: assumptions and related work are presented first in sections 2 and 3. The main theory used by ADFG is outlined in section 4.

[1] A directed graph is said *weakly connected* when its undirected induced graph is connected (i.e. a path exists between each vertices pair).

Then the most important scheduling synthesis improvements are presented in section 5. Section 6 provides an overview of the ADFG tool. Finally some evaluation results are presented in section 7 and lead to a conclusion and to a future work discussion in section 8.

2 SYSTEM MODEL AND ASSUMPTIONS

We consider hard real-time systems composed of m homogeneous processors where to execute n periodic tasks. Each task τ_i is released every T_i unit of times; τ_i has a worst-case execution time C_i and an implicit deadline D_i $(= T_i)$. Tasks may start with a positive offset O_i and are mapped on a processor referenced by $v_i \in [\![1; m]\!]$.

Tasks data exchanges are modeled thanks to a dataflow directed multi-graph: tasks—vertices of the graph—produce tokens that they send to other tasks, and symmetrically consume some of the received tokens. These tokens are sent instantaneously through unidirectional First In First Out (FIFO) buffers—edges e of the graph—, which have a limited size δ_e and an initial number of present tokens θ_e. The numbers of tokens produced and consumed at each task firing (i.e. execution) are parameters of each edge, they can be static (always the same amount) or more generally cyclo-static (finite sequence of different amounts). One main dataflow graph model is considered: the UCSDF model detailed in section 4.1, which is an extension of the Cyclo-Static Dataflow (CSDF) [5] model, itself an extension of the Static Dataflow (SDF) [22] one. Moreover we consider multiprocessor partitioned preemptive scheduling context under either an Earliest Deadline First (EDF) or Fixed Priority (FP) scheduler.

Scheduling this model requires to compute the task periods and buffer sizes such that all deadlines are met and such that there is no buffer underflow or overflow, while maximizing the throughput and minimizing the total buffer size. Then the total buffer size and the throughput are the main metrics to compare different scheduling parameter valuations. The comparison must be done on a dataflow graph with the same inherent properties (WCET, production and consumption rates) but with different task periods, priorities, offsets, processor bindings, and with different buffer sizes or buffer initial numbers of tokens.

We assume the following definition of the throughput Θ of a periodic system. Firstly the throughput of an actor $\Theta(\tau_i)$ is defined by its average number of firings per time unit: $\Theta(\tau_i) = \frac{1}{T_i}$ for periodic scheduling. Then it is possible to extend this definition to the whole system, independently of a task: in the case of periodic scheduling $\Theta = \frac{1}{H}$ with H being the least common multiple of all task periods, called the hyperperiod ($H = \mathrm{lcm}_{1 \leq i \leq n}(T_i)$). However, as ADFG synthesizes the task and buffer parameters under a maximum number of processors constraint (which is not present in the throughput definition), it is more accurate to say that ADFG maximizes the processor utilization factor $U = \sum \frac{C_i}{T_i}$ instead of the throughput; both values evolving in the same way, the terms are used equivalently in the paper.

3 RELATED WORK

The DARTS [2] tool allows to compute the strictly periodic scheduling parameters achieving the best throughput under EDF or Rate Monotonic (RM) policies, with a maximum total buffer size as a constraint. DARTS considers only acyclic CSDF graphs and a non-constrained number of available processors on the target system. ADFG and DARTS differ in the input constraints: ADFG considers cyclic and acyclic UCSDF graphs, constrained and non-constrained buffer sizes, and at the opposite, a constrained number of available processors in the target system. The buffer size versus number of processors constraint opposition is inherent to the scheduling problem since it is not possible to achieve both throughput maximization and buffer minimization at the same time [30].

Other researches on the scheduling synthesis often relax the available processors constraint (their number is considered as potentially infinite) and focus on the maximum achievable throughput and the buffers. Moreover these works use most likely either static periodic scheduling [7, 19] or self-timed scheduling [18, 23] (each task is fired whenever available resources allow it, regarding to the consumed and produced tokens to avoid buffer underflows and overflows, and regarding to the processors availability). Like ADFG, several synthesizers use ILP problems especially to approximate the buffer sizes. Previous ILP formulations for CSDF graphs have been made in order to maximize the throughput and minimize the total buffer size: under self-timed scheduling [33], under static periodic scheduling with maximum number of processors constraint [32], or under static periodic scheduling with minimum throughput constraint [3], but not under EDF and FP scheduling. Another synthesis tool [16] exists for a mix of the periodic and self-timed scheduling policies, with priorities.

4 ADFG SCHEDULING SYNTHESIS THEORY

This section presents briefly the core theory of ADFG [8], also used by the new algorithm presented in section 5.3. The UCSDF extension of the CSDF graph model is presented first, then an abstraction of the tasks relation in this model, and finally the common steps of the different synthesis algorithms.

4.1 UCSDF model

In the CSDF model, each buffer can be seen as a quadruple $(\tau_p,\ \tau_c, v_p^{\omega},\ v_c^{\omega})$ where τ_p and τ_c are respectively the source task (producer) and the destination task (consumer) whose token production/consumption rates are defined by the infinite periodic sequence v_p^ω and v_c^ω (ω denotes the infinite periodic extension of the finite sequences v_p and v_c).

The Ultimately Cyclo-Static Dataflow (UCSDF) model adds a non periodic production/consumption sequence to each rate in order to model an initialization part at the beginning of the execution. When this sequence is empty for all rates, such UCSDF graph is exactly a CSDF graph. At the opposite when the initialization sequence is not empty, a UCSDF graph is *ultimately* a CSDF one (when reaching the periodic part of all rates: from this point the execution is in the *stationary state*). The quadruple defining each buffer then becomes $(\tau_p, \tau_c, u_p v_p^\omega, u_c v_c^\omega)$, u being the finite initialization sequence of the production/consumption rates.

When finding a solution to the synthesis problem, the initialization part u only impacts on the buffer sizes (and their initial number of tokens), they may be needed to be greater than during the stationary state of the execution.

Several metrics can be defined upon the buffer rates. Firstly $|v|$ denotes the length of a finite sequence and we let $x = u_p v_p^\omega$. The j integer variable represents in this section the j-th firing of a task, hence $x_p(j)$ is the production rate of τ_p j-th firing. Then it is possible to compute the average token production per firing a_x for the periodic sequence[2]:

$$a_x = \frac{||v_p||}{|v_p|}, \; with \; ||v|| = \oplus v(|v|) = \sum_{1 \le j \le |v|} v(j) \qquad (1)$$

and to compute the lower affine bound λ^l and the upper one λ^u for the complete sequence:

$$\lambda_x^l = \min_{1 \le j \le |u_p|+|v_p|} \{\oplus x(j) - a_x j\} \qquad (2)$$

$$\lambda_x^u = \max_{1 \le j \le |u_p|+|v_p|} \{\oplus x(j) - a_x j\} \qquad (3)$$

(in a SDF graph both λ_x^l and λ_x^u would be equal to 0). These equations remain the same from the consumption point of view, with the only difference to take the consumption rates and so with $y = u_c v_c^\omega$ instead of x. Figure 1a gives an example of a buffer and its computed metrics.

Hence the number of produced/consumed tokens by a task over time can each be bounded by two affine equations in j, that lead to the name of the tool *Affine* DataFlow Graph:

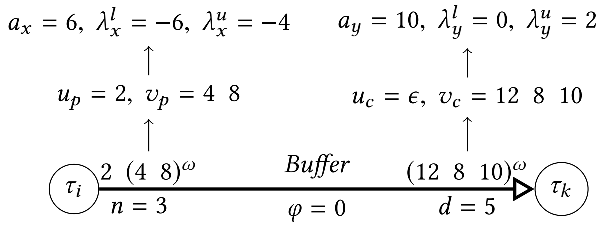
$$\lambda_x^l + a_x j \le \oplus x(j) \le \lambda_x^u + a_x j$$

The affine equations are then used to derive clock relations between producers and consumers.

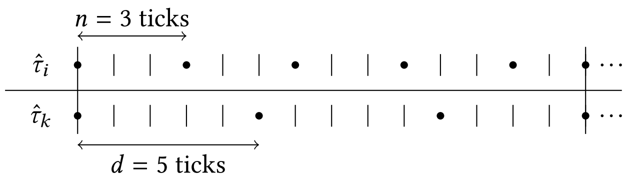
4.2 Clock relations

Clock relations between two tasks give an abstraction of their relative firing times. The buffer average token production a_x and consumption a_y are used to deduce a clock relation between the firings of the producer and the consumer. For example if τ_p produces 6 tokens (in average) destined for τ_c consuming only 3, τ_c must be two times faster (i.e. two times more fired) than τ_p in order to avoid any buffer underflow or overflow. Underflow (resp. overflow) occurs when a consumer (resp. producer) fires while there are not enough (resp. too many) tokens in one of its incoming (resp. outgoing) buffers. To avoid that, constraints must be respected by all buffers and undirected cycles of buffers.

[2]Note that a_x is equal to the constant rate in the SDF model.


$a_x = 6, \; \lambda_x^l = -6, \; \lambda_x^u = -4$
$a_y = 10, \; \lambda_y^l = 0, \; \lambda_y^u = 2$
$u_p = 2, \; v_p = 4 \; 8$
$u_c = \epsilon, \; v_c = 12 \; 8 \; 10$
τ_i
$2 \; (4 \; 8)^\omega$
Buffer
$(12 \; 8 \; 10)^\omega$
τ_k
$n = 3$
$\varphi = 0$
$d = 5$

(a) Buffer between τ_i and τ_k, its affine properties are above the buffer arrow and its derived clock relation is below it



(b) Activation clock representation over lcm$(n, d) = 15$ ticks, the two tasks start on the same tick because $\varphi = 0$

Figure 1: Clock relation and corresponding task clocks

A clock relation also needs a phase to be fully specified by a triple (n, φ, d). The n and d variables correspond to a_x and a_y divided by their greatest common divisor (so $\frac{n}{d} = \frac{a_x}{a_y}$), but the first firing of one of the two tasks can be delayed: this delay is abstracted by the phase φ. The computation of φ is done when solving the following constraints presented in this section. These constraint equations have already been proved in previous work [11, 13].

Figure 1 provides an example for two tasks τ_i and τ_k which are (n, φ, d)-related. The clock relation is intuitively represented by ticks (small vertical bars) and task firings (filled dots) in fig. 1b, while the metrics (presented in section 4.1) used to retrieve the n and d are detailed in fig. 1a.

The clock relations help to formalize the speed constraint of the first paragraph of this section. Considering τ_i sending tokens to τ_l related by (n, φ, d), their speeds can be defined relatively by the equation:

$$dT_i = nT_l \qquad (4)$$

So the most simple constraint is that for all buffers $e_{i \leftrightarrow l}$ between τ_i and τ_l, all induced relative speed equations must be the same: hence all ratio $\frac{n_{e_{i\leftrightarrow l}}}{d_{e_{i\leftrightarrow l}}}$ must be the same, and only one clock relation connecting τ_i to τ_l is stored. The clock relations are directed (as the buffers) so this comparison must be done with all relations oriented to the same direction (the reverse of (n, φ, d) is $(d, -\varphi, n)$). In a cycle composed of k clock relations, this constraint becomes $\prod_{i=0}^{k-1} n_i = \prod_{i=0}^{k-1} d_i$.

The consistency constraints above do not involve φ and can be checked early. Those involving φ are described below.

Underflow. The underflow constraint needs the variable θ_e to represent the initial number of tokens in the buffer e.

$$\theta_e + \frac{a_y}{d}\varphi \geq \lambda_y^u - \lambda_x^l + a_y C_{under} \tag{5}$$

Overflow. The overflow constraint is very similar to the underflow one, but necessitates also the variable δ_e to represent the maximum size of the buffer e.

$$\theta_e + \frac{a_x}{n}\varphi - \delta_e \leq \lambda_y^l - \lambda_x^u - a_x C_{over} \tag{6}$$

C_{under} and C_{over} values are depending on the scheduling policy and other parameters as the producer and consumer priorities or the fact they are on the same processor or not.

Cycles. One constraint is created per cycle of a computed cycle basis. The cycle constraint can be explained as the fact that a task τ_i cannot have delay with itself. Indeed all k tasks in a cycle including τ_i are related by successive clock relations with phases between them, which leads to a phase between τ_i and itself. The following constraint ensures that this delay is equal to 0 for each task in the cycle.

$$\sum_{i=1}^{k}(\prod_{l=1}^{i-1} d_l)(\prod_{l=i+1}^{k} n_l)\varphi_i = 0 \tag{7}$$

All the constraints with φ are linear and can be expressed as an Integer Linear Programming (ILP) problem. In the prototype version, the objective of this problem was to minimize the sum of all buffers. This objective is modified in ADFG and the new one is presented in section 5.1.

The clock relations are not sufficient to compute directly the tasks periods T_i. However in a weakly connected component (WCC) of the dataflow graph, they allow, thanks to eq. (4), to express each task period relatively to the period of a basis task. If *basis* is the reference actor in a WCC, α coefficients can be defined such that $T_i = \alpha_i T_{basis}$ (and $\alpha_{basis} = 1$) for every task in this component. Then computing the periods for the scheduling synthesis necessitates to only find the basis one (with the extra constraint on all periods that $T_i \geq C_i$). Standard scheduling test algorithms, like Quick Processor Demand Analysis (QPA) [34, 35] for EDF, and Response Time Analysis (RTA) [1, 20] (especially the RTA lower and upper bounds [6, 28]) for FP, have been adapted to work with these *symbolic* periods. Their symbolic version are respectively called SQPA and SRTA in ADFG.

The period synthesis algorithms in ADFG usually start by testing the schedulability of the system with the minimum acceptable T_{basis} and gradually increase it up to the maximum acceptable one, or perform a dichotomy between them.

The maximum acceptable T_{basis} is given by the implementation as the maximum integer stored by an `int` in the programming language. On the contrary the minimum acceptable T_{basis} can be derived from a last constraint that can be expressed using the symbolic periods: the processor utilization factor $U = \sum \frac{C_i}{T_i}$ indeed becomes $U = \sum \frac{C_i}{\alpha_i T_{basis}} = \frac{\sigma}{T_{basis}}$ with $\sigma = \sum \frac{C_i}{\alpha_i}$, so the necessary standard scheduling constraint $U \leq m$ becomes:

$$T_{basis} \geq \frac{\sigma}{m} \tag{8}$$

with m as the number of processors, and considering that the whole dataflow graph is weakly connected. An algorithm able to schedule UCSDF graphs with several WCCs will be presented in section 5.3.

Once T_{basis} and all φ have been found, it is possible to compute all periods T_i and offsets O_i. The solution is unique when T_{basis} is known precisely but several synthesis algorithms may return a minimum value T_{basis}^l instead, so an ILP is used with the following constraints:

$$\max(\frac{\sigma}{m}, T_{basis}^l) \leq T_{basis} \leq \texttt{int.MAX\_VALUE} \tag{9}$$

$$T_i = \alpha_i T_{basis} \tag{10}$$

$$O_l - O_i = \frac{\varphi}{n} T_i \tag{11}$$

with the objective to minimize T_{basis} in the prototype.

4.3 Main steps of the synthesis algorithms

All implemented periodic scheduling synthesis algorithms respect the following steps:

(1) decompose the graph: compute all clock relations (with undefined φ) and the WCCs;
(2) compute the priorities (if the scheduling policy is fixed priorities): note that the priorities are computed without knowing yet the periods;
(3) partition tasks across available processors: this step may need to call internally an algorithm of step 5;
(4) compute all φ: thanks to an ILP solver with the constraints from eqs. (5) to (7);
(5) perform symbolic synthesis: thanks to SRTA or SQPA for example, computing only the WCC basis period;
(6) compute all task periods and offsets: thanks to an ILP solver with the constraints from eqs. (9) to (11);
(7) precise buffer initial tokens and buffer size computation: each buffer clock relation (regarding to its producer and consumer) is used in conjunction to the ultimately cyclo-static rates to compute the minimum and maximum needed tokens [9].

When an ILP problem is not feasible or when the basis period is too large, the algorithms automatically state that the system is not schedulable.

5 SYNTHESIS IMPROVEMENTS

Modifications have been made on the original equations and algorithms used in the ADFG prototype, new algorithms have also been added. Main modifications concern the two ILP problems—section 5.1—(steps 4 and 6 of the workflow detailed in section 4.3). A new task to processor pre-mapping algorithm—section 5.2—(step 3) has been added, as well as a synthesis algorithm (step 5) for EDF—section 5.3.

5.1 ILP problems reformulations

The two reformulations concern each ILP problem objective function, they help to have quicker and unique results. The first ILP problem computes clock relation phases (step 4 presented in the previous section). The second one computes the periods and the offsets (step 6).

Phases computation ILP. To compute the best phases, the objective of this ILP was to minimize the total buffer sizes (each buffer size is a variable in the problem), with the idea that if buffer sizes are minimized, the phases will be indirectly minimized too since a phase delays the execution of the producer or consumer and thus increases the number of tokens stored in the buffer. However a null phase is not always the best solution regarding to eqs. (5) and (6), especially because nullifying the phase can lead to increase the initial number of tokens (also a variable of the problem) and thus the buffer size. This former objective is sufficient but is complex to handle for the ILP solver as the phases are free variables (they can be negative). So the new formulation tries to minimize the absolute value of the phases *and* to minimize the total buffer size. As the two variables have not the same unit, the same coefficient as in eqs. (5) and (6) is used to multiply the phases. The new objective has the following formulation (e denotes buffers):

$$\sum_{e} \delta_e + \sum_{e} \frac{a_y(e)}{d(e)} |\varphi(e)| \tag{12}$$

Note that there can be several occurrences of the same φ in this objective because there can be several buffers between two tasks, which necessarily share the same clock relation.

Periods and offsets computation ILP. The initial objective of this ILP was to minimize the basis task period, in order to achieve the best throughput. However, the offsets were not in the objective so the ILP solver could choose to start all tasks with 100 units of time of delay for example. The new objective is now to minimize the basis period and its offset (as offsets are linearly related in the weakly connected UCSDF graph by eq. (11), it will also minimize all offsets). The new formula is just the sum of the two variables: $T_{basis} + O_{basis}$ (since they have the same time unit).

5.2 Partitioning imbalance ratio

The task placement on the processors influences both the processor utilization factor and the buffer sizes. One dataflow graph partitioning strategy might be balancing the total utilization factor between the available processors. However the most balanced partitioning (if each processor has the same utilization factor) is not always the one minimizing the buffer sizes (intuitively because the producer and the consumer can execute simultaneously). Thus there is a trade-off between buffer size minimization and processor utilization maximization, and it depends on the partitioning of the UCSDF graph, or in other terms, on the mapping of tasks to the processors.

The SCOTCH graph partitioner [25] is used by ADFG in several algorithms to perform the task mapping. SCOTCH uses the *imbalance ratio* metric to control the maximum accepted load imbalance between the processors, and tries to minimize the total node separation cost while respecting the imbalance ratio. The total node separation cost is the sum of the weight of arcs going from a partition to another. In ADFG, this cost is the buffer size gain of putting two communicating tasks on the same partition. The buffer size is approximated thanks to eqs. (5) and (6) whose coefficients C_{under} and C_{over} depend, among other properties, on the producer and consumer processor placement. The gain is the difference between the two values of the buffer size approximation: when producer and consumer are on the same processor or not. The buffer size is *under*-approximated thanks to the equation: $\lambda_x^u + \lambda_y^u - (\lambda_x^l + \lambda_y^l) + a_x\, C_{over} + a_y\, C_{under} \leq \delta_e$.

The imbalance ratio is the sum of absolute differences between computation loads on each processor and the average per processor, divided by the total load. In the partitioning algorithm, task periods are not known yet so their symbolic load (SL) $SL_i = \frac{C_i}{\alpha_i}$ is used instead. Several metrics can be derived from the SL:

- total SL per processor, $SL_{Proc_q} = \sum_{\tau_i | v_i = q} SL_i$
- average SL per processor, $SL_{ProcAvg} = \frac{\sum_{1 \leq q \leq m} SL_{Proc_q}}{m}$
- total SL, $SL_{TOT} = \sum_{1 \leq i \leq n} SL_i$

Considering that $\mathcal{M}$ denotes a task to processor mapping (more formally it corresponds to a valuation of all v_i in $[\![1; m]\!]$), the imbalance ratio $imb(\mathcal{M})$ of a mapping can be defined[3] as follows:

$$imb(\mathcal{M}) = \frac{\sum_{1 \leq q \leq m} |SL_{Proc_q} - SL_{ProcAvg}|}{SL_{TOT}} \tag{13}$$

[3] The original definition can be found on page 10 of the SCOTCH user guide (version 6.0) with the variable name δ_{map}; eq. (13) is a simpler definition for the case of homogeneous processors used in this work.

SCOTCH is used in three ADFG algorithms, so ADFG must compute an imbalance ratio to give to SCOTCH, ensuring that partitioning is doable with this ratio. The minimum of this ratio ensures the maximum processor utilization, while the maximum of this ratio ensures the minimum total buffer size. In the ADFG prototype the imbalance ratio was a quite small constant, leading sometimes to unfeasible partitioning.

Heuristics have been integrated to ADFG in order to quickly compute the maximum allowed imbalance ratio. Three heuristics have been implemented: one to compute the best balanced partitioning, another to compute the worst one, and a trade-off between the two (the default one). Each heuristic corresponds to an ADFG optional *strategy* input parameter, enabled for algorithms using SCOTCH. Then ADFG computes the imbalance ratio of the heuristic partitioning, and sets it as the maximum allowed for SCOTCH.

Well balanced partitioning. All actors are sorted in the descending order regarding to their SL_i, and then are successively added to the current less loaded partition. This is a naive greedy algorithm; it is quadratic [4] in the actors (to sort them), and since it is not optimal it still gives some leeway to SCOTCH. This is the THROUGHPUT_MAX strategy: it ensures the highest U for a weakly connected graph.

Worst balanced partitioning. It is possible to compute the worst balanced partitioning imbalance ratio imb_{single} by placing all tasks on a single processor. Following the imbalance ratio definition, this leads to the result $imb_{single} = \frac{2m-2}{m}$, depending only on the number of available processors. Note that since SCOTCH is free to put the tasks wherever in order to minimize the total buffer size, this can lead to use less processors than available. This is the BUFFER_MIN strategy.

Trade-off partitioning. The average SL per task $SL_{TaskAvg} = \frac{SL_{TOT}}{n}$ is computed first and then the SL_i are sorted in the ascending order according to the metric $\frac{|SL_i - SL_{TaskAvg}|}{SL_i}$. The m first symbolic loads are placed each one on a different processor, and finally all SL_i left are added to the most loaded processor. This gives a certainly reachable maximum imb_{max} imbalance ratio, still using each processor (at the opposite of the worst balanced partitioning which uses only one processor). The minimum imbalance ratio imb_{min} is also computed, using the algorithm described in the above paragraph for the well balanced partitioning. Finally the balanced imbalance ratio for the trade-off partitioning is computed as follows:

$$imb_{bal} = imb_{min} + (imb_{max} - imb_{min})\frac{\min\{m, n\}}{n} \qquad (14)$$

which tends to the minimum one when the number of tasks increases. This is the BALANCED strategy.

[4]A better ADFG implementation could achieve a linearithmic complexity, but the (at worst) quadratic insertion sort is used for now.

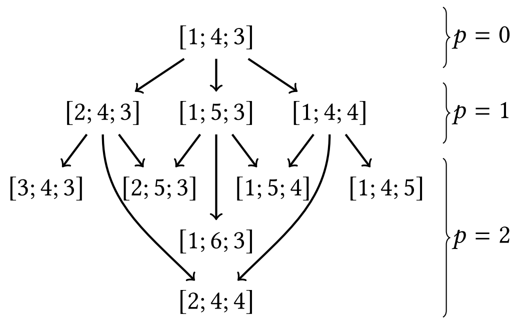


Figure 2: Weakly connected component (WCC) basis period vector tree as constructed by PQPA (example for three WCCs and depth $p = 0–2$)

5.3 Unconnected graphs EDF scheduling

The Parametric QPA (PQPA) algorithm relying on the ADFG theory had been proposed (but not implemented) to handle the EDF scheduling synthesis of unconnected UCSDF graphs on homogeneous multiprocessor architecture [10]. This algorithm has been implemented and adapted in the latest version of ADFG.

The main idea of PQPA is to start with the lowest possible basis period (respecting the constraint of eq. (8)) of each weakly connected component, and to increment each one until finding the best schedulable point (with the greatest total processor utilization factor U_{TOT}). The original algorithm performs a depth-first search: the first WCC basis period is incremented successively until the first schedulable point, then the second WCC, and so on. The search in a direction stops in two cases: when the period becomes larger than the maximum allowed, or when the system is schedulable (regarding to the QPA test).

If $\mathcal{L}$ represents the number of WCCs in the UCSDF graph, the starting point of PQPA is then a vector of $\mathcal{L}$ dimensions (containing the basis periods). Iterations of PQPA construct a tree of vectors (all being incremented from the first one). Each level p (starting from 0) of this tree has basis period vectors which have been incremented exactly p times by 1 (not necessarily all increments on the same row) relatively to the root. Thus the number of vectors per level is the binomial coefficient $\binom{p+\mathcal{L}-1}{\mathcal{L}-1}$; in terms of combinatorics this is the number of *weak compositions* of p in $\mathcal{L}$ parts (e. g. $5 = 1 + 4$ and $5 = 4 + 1$ are two different weak compositions of 5 in two parts). Figure 2 gives a tree example for three WCCs for depth 0 to 2, starting with the T_{basis} vector $[1; 4; 3]$.

This algorithm has two drawbacks. Firstly it favors a direction, i.e. a weakly connected component, by doing a depth-first search: a high U_{TOT} (almost equal to m) can be found with a very low U_C on the first component C while the others are equal. Secondly, if it starts from a point where U_{TOT} is far greater than m, numerous increments will be needed to reach the first schedulable point; that is problematic since the number of nodes at a specific level is combinatorial. Hence two aspects of the algorithm have been modified: the search strategy and the starting point.

Search strategy. The search strategy is now breadth-first search. This ensures fairness between the processor utilization factor of each component (as long as the starting point is also fair). As iterations are made level by level, there are two possible strategies to generate nodes of the next level $p+1$: enumerate directly all nodes corresponding to the weak compositions of $p+1$, or generate all the $\mathcal{L}$ possible children of each node at level p and check each time if this node has not yet been generated by another node ($[2;4;3]$ and $[1;5;3]$ can both generate $[2;5;3]$ for example). The second solution has been chosen in ADFG, considering that there is not so many nodes if the starting point is near a schedulable point. Moreover both solutions need a check method since the schedulable nodes must not have children anyway.

Starting point. The starting point must be near a schedulable point, and must ensure fairness between the different components execution. To do this, PQPA starts from a point where each WCC has a processor utilization factor equal to $\frac{m}{\mathcal{L}}$, the average one. Then the constraint of eq. (8) becomes:

$$T_{basis}(C) \geq \frac{\mathcal{L}}{m}\sigma(C) \tag{15}$$

The starting basis period of the component C is then $\frac{\mathcal{L}}{m}\sigma(C)$.

As PQPA works for multiprocessor systems, a new mapping algorithm has been written. The principle is a standard best fit heuristic: tasks are successively added to the partition which ensures the best U_{TOT}. PQPA has also been slightly modified to start with the last best basis period vector found, in order to avoid useless iterations (when tasks are added to a processor, the basis period of their WCC cannot be lesser than before). Thus the mapping is not related to the graph topology (it is not one WCC per processor for example).

6 ADFG SCHEDULING TOOL

ADFG synthesizes strictly periodic scheduling parameters: the tasks periods, priorities (if FP scheduling), offsets, processor bindings, and the buffer sizes and initial number of tokens. This section firstly describes the different ADFG inputs and outputs, then the available algorithms are briefly presented, and finally an implementation overview is provided.

6.1 Inputs and outputs

The inputs are those described in the problem statement: a UCSDF graph, the number of processors in the target system, and a scheduling algorithm. The scheduling algorithm and the number of processors are simple parameters; however the UCSDF graph can be stored in different formats, and can contain problem specifications (tasks WCET, consumption and production rates) as well as problem results (tasks periods, etc.). Moreover the results can be used independently to the dataflow graph to perform simulations for example, that use a simpler format. The different formats used in ADFG or exported by it are described below.

Inputs. In the prototype version, ADFG reads the UCSDF graphs in the SDF3 [29] format, slightly modified to support UCSDF and not only CSDF. This format is still supported but it lacks the storage of some results as the tasks periods, offsets and priorities. A new `XML` format (with the extension `.adfg`) has been designed to handle all the UCSDF graph inherent specifications and the results of the synthesizing. This format has been designed within Eclipse, and comes with a minimal graphical editor plugin generated by the Epsilon EuGENia [21] project within the Eclipse Modeling Framework. Concretely this format enables to store all tasks and buffers separately, buffers having specific attributes to identify their producer and consumer tasks. Ports, mandatory in SDF3, are not present in this model.

Outputs. The output is mainly textual in Eclipse and in the CLI. The synthesized parameters can be exported in the input model file, but some information easily recomputable from the results (as the total utilization factor) is not exported. In order to simulate the scheduling, other exports are possible: into the Yartiss [14] and Cheddar [27] input file formats. It is also possible to generate a DOT representation of the input UCSDF graph in order to have visual prettier representations.

Options. Several synthesized parameters can also be fixed by the user while performing the analysis. The user can choose to reuse the periods, the initial numbers of tokens or the buffer sizes. If all these parameters are fixed, ADFG behaves more like an analyzer than like a synthesizer.

6.2 Scheduling algorithms

Sixteen algorithms have been implemented in order to synthesize the scheduling parameters, for EDF and FP scheduling policies. They differ by their inputs restrictions (several are reserved for uniprocessor systems, most accept only weakly connected dataflow graphs), by their objectives (maximize the throughput, minimize the total buffer size, minimize the number of preemptions, and trade-off between them) and by their complexities and hence precision.

Table 1 summarizes the available algorithm behaviors, the abbrevation *W. C. G.* stands for weakly connected graph in the *Restriction* column. The *Test* column specifies which internal algorithm synthesizes the task periods. The first and last columns contain respectively the ADFG synthesis *Name* and a short *Complement*.

Name	Restriction	Partitioning	Test	Complement
EDF_UNI	—	—	$U \leq 1$	Does not aim at equality between the WCCs processor utilization [13] (alg. 1, sec. IV)
EDF_UNI_DA	—	—	SQPA	P. C.
EDF_UNI_HDA	—	—	SQPA	P. C. with threshold
EDF_UNI_PQPA	—	—	PQPA	Algorithm presented in section 5.3
EDF_MULT_MBS_HDA	W. C. G.	SCOTCH	PSQPA	Optional strategy and P. C. with threshold
EDF_MULT_MBS_ID	W. C. G.	SCOTCH	$U \leq m$	Optional strategy
EDF_MULT_BF_UF_ID	W. C. G.	Best Fit + heuristic	$U \leq m$	—
EDF_MULT_BF_SQPA_CD	W. C. G.	Best Fit + SQPA	PSQPA	Adapted for fractional deadlines
EDF_MULT_BF_PQPA	—	Best Fit + PQPA	PQPA	Algorithm presented in section 5.3
GEDF_MULT	W. C. G.	—	ffdbfSQPA	For global scheduling, adapted from [4]
SP_UNI	W. C. G.	—	SRTA	Deadline Monotonic
SP_UNI_LOP	W. C. G.	—	SRTA	Priorities for total approximate buffer size minimization solved by ILP [12]
SP_UNI_UDH	W. C. G.	—	SRTA	As SP_UNI_LOP but solved by combinatorial search with utilization distance heuristic
SP_MULT_MBS	W. C. G.	SCOTCH	PSRTA	Optional strategy
SP_MULT_BF_FBBFFD	W. C. G.	Best Fit + heuristic	PSRTA	Adapted from [17]
SP_MULT_BF_SRTA	W. C. G.	Best Fit + SRTA	PSRTA	—

Table 1: Uni- and multi-processor EDF and SP (fixed priorities) algorithm

Some EDF algorithms try to enforce precedence constraints [15] between the tasks in order to avoid preemptions [11], the *Complement* column of such algorithms uses the abbreviation *P. C.* to notify it. As these constraints may induce a poor processor utilization, a variant with a threshold $\rho \in [0, 1]$ is sometimes used: the precedence is not enforced if it would lead to a deadline less than ρT_i. The precedence constraints are enforced, when possible, thanks to the task deadlines also computed by ADFG in this case. The computed deadlines are *fractional*: $D_i = \beta_i T_i$ with $\beta \in \mathbb{Q} \restriction_{[0;1]}$. Some other algorithms can handle fractional deadlines, but it cannot be specified yet in the ADFG input file formats.

Finally several multiprocessor algorithms use best fit partitioning (step 3): they successively map each task on each processor, and leave it on the processor ensuring the best fit metric. The "Best Fit + SRTA/SQPA/PQPA" algorithms perform their schedulability test each time a task is mapped in order to maximize U_{TOT}, which increases the complexity; at the opposite the "Best Fit + heuristic" algorithms use a quick heuristic (schedulability is checked only at step 5).

6.3 Implementation

The ADFG tool is free and can be downloaded online[5] as a pre-compiled binary for Linux. The implementation is mainly in Java, and uses only open-source dependencies. The ADFG tool can be used from a Command Line Interface (CLI) within a Linux terminal, from the Eclipse framework, or as a Java `jar` library. The software is built thanks to Maven.

Dependencies. The Java library JGraphT provides structures and methods to respectively represent graphs and compute some of their properties. Graphs in some algorithms are partitioned with SCOTCH. Finally all ILP problems are solved by LpSolve. As SCOTCH and LpSolve are written in C, they are accessed through the Java Native Interface (JNI).

Determinism. Several code parts were not deterministic in the prototype version, depending on the order of tasks in lists for example, itself depending on the input UCSDF graph file parsing order. To avoid this, all tasks and buffers are separately sorted in lexicographic order after the parsing. At the same time they are given an increasing unique identifier; this identifier can be used to ensure determinism. For example SP_UNI_DM assigns increasing priorities to tasks with equal deadlines, according to their identifiers order.

7 EVALUATION

The evaluated applications come from the *StreamIT* [31] benchmark and from the SDF3 and DARTS examples. Eight applications are studied in this paper: `cd2dat` uses a pure CSDF representation (with production and consumption rate sequence length $|v|$ up to 7) whereas the others use the SDF model (each $|v| = 1$). They have 6 to 120 tasks and 5 to 146 buffers. More data on these applications and the minimum

[5] http://polychrony.inria.fr/ADFG/

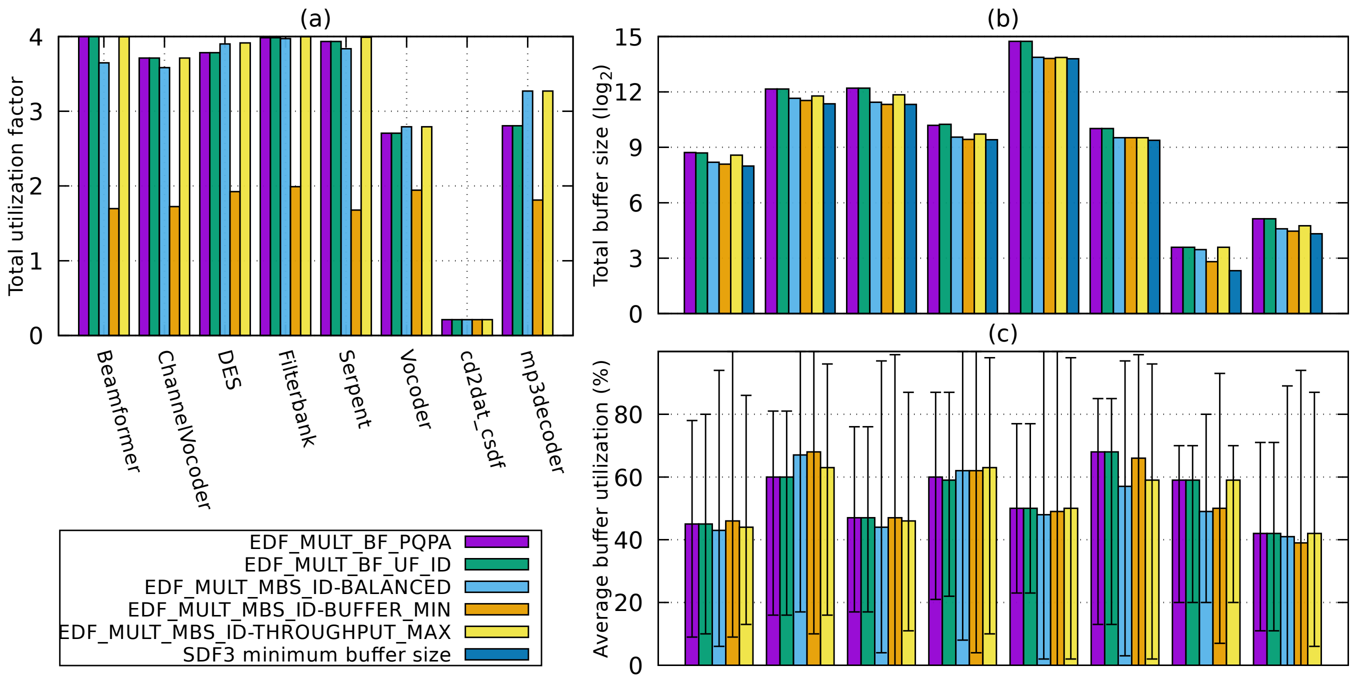


Figure 3: Evaluation of 8 models with several synthesis algorithms for EDF on 4 processors. All charts share the same abscissa. (a) Total processor utilization U_{TOT}. (b) Total buffer size computed by ADFG (and by SDF3 for the minimum). (c) Simulated average buffer utilization.

achievable total buffer size computed by SDF3 (with infinite number of available processors) can be found in [8, p. 114].

Results of the applications evaluation with 3 different multiprocessor (for $m = 4$) synthesis algorithms for EDF are presented in fig. 3; one algorithm using SCOTCH partitioning, it is evaluated for the three strategies seen in section 5.2. All charts are histograms clustered by the application names, which appear in the same order when abscissa is not specified. The evaluated metrics are: the total buffer size computed by ADFG and the minimum one of SDF3 (a); the total processor utilization factor U_{TOT} (b); and the average buffer usage over time and over all buffers as simulated by Cheddar [27] (c), with bars representing minimum and maximum utilization. All presented applications synthesized by ADFG in fig. 3 took only around 500 ms each to execute (with an Intel i7-3740QM @2.70GHz processor).

It is difficult to compare the results of the new PQPA algorithm seen in section 5.3 since it is designed to handle unconnected UCSDF graphs whereas all other algorithms do not support this possibility (all 8 applications are weakly connected). The total buffer size found with PQPA (first column of each cluster on fig. 3, columns order is the same as in the legend) is the highest for all applications but Filterbank, however U_{TOT} is quite good: it is in the average for all applications and reaches at least 3.5 (over 4) for 5 of them. Concerning the three new strategies for SCOTCH partitioning, the buffer size versus throughput trade-off is clear for the BUFFER_MIN strategy which has the lowest U_{TOT} and total buffer size for all applications. Notice that the total buffer size chart uses a log scale, so the difference is not as visible as in the U_{TOT} chart. At the opposite the THROUGHPUT_MAX strategy always reaches the best U_{TOT} compared to the other algorithms, but it is surprisingly not the worst one for buffer sizes. BALANCED strategy is, as expected, a good trade-off between the two.

The simulation made by Cheddar confirmed that all synthesized systems were indeed schedulable, and that no buffer underflow or overflow occurred. The average buffer utilization computed by Cheddar, chart (c) in fig. 3, does not highlight any difference between the synthesis algorithms. However it shows that the buffers are completely used (to 100%) in only few cases (see the upper bar for maximum buffer usage), and the same observation can be made for minimum usage (to 0%, see the lower bar). It means that ADFG overestimates respectively the buffer sizes and the initial number of tokens on it.

Finally the evaluation confirms that ADFG computes safe scheduling parameters and provides efficient strategies to choose different trade-offs between processor utilization factor maximization and buffer size minimization.

8 CONCLUSION AND FUTURE WORK

Given an application modeled with an UCSDF graph, ADFG enables to synthesize parameters to schedule the application with a limited number of processors. ADFG provides different algorithms for EDF and FP policies, and enables to choose different trade-offs between throughput maximization and buffer sizes minimization. As all the ADFG syntheses in the presented evaluation were performed within a second, this can help real-time system designers to reduce the prototyping time of their applications.

Yet several metrics are not taken into account in the synthesis: particularly communication and preemption costs. Moreover ADFG considers the task WCET disregarding to the number of tokens consumed and produced by this task, whereas a task consuming or producing less data will probably take less time to execute. Adapting the presented algorithms to handle cyclo-static WCET and communication costs constitutes the next steps of this work.

REFERENCES

[1] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. 1993. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. In *Software Engineering Journal*, Vol. 8.

[2] M. Bamakhrama and T. Stefanov. 2011. Hard-real-time scheduling of data-dependent tasks in embedded streaming applications. In *International Conference on Embedded Software (EMSOFT)*.

[3] M. Benazouz, O. Marchetti, A. Munier-Kordon, and T. Michel. 2010. A new method for minimizing buffer sizes for Cyclo-Static Dataflow graphs. In *8th IEEE Workshop on Embedded Systems for Real-Time Multimedia*.

[4] M. Bertogna and S. Baruah. 2011. Tests for Global EDF Schedulability Analysis. In *J. Syst. Archit.*, Vol. 57.

[5] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. 1996. Cycle-static Dataflow. In *Trans. Sig. Proc.*, Vol. 44.

[6] E. Bini, T. Huyen Chau Nguyen, P. Richard, and S. K. Baruah. 2009. A Response-Time Bound in Fixed-Priority Scheduling with Arbitrary Deadlines. In *IEEE Transactions on Computers*, Vol. 58.

[7] B. Bodin, A. Munier-Kordon, and B. D. de Dinechin. 2013. Periodic schedules for Cyclo-Static Dataflow. In *The 11th IEEE Symposium on Embedded Systems for Real-time Multimedia*.

[8] A. Bouakaz. 2013. *Real-time scheduling of dataflow graphs*. Ph.D. Dissertation. Université Rennes 1.

[9] A. Bouakaz and T. Gautier. 2014. An abstraction-refinement framework for priority-driven scheduling of static dataflow graphs. In *Twelfth ACM/IEEE Conference on Formal Methods and Models for Codesign*.

[10] A. Bouakaz, T. Gautier, and J.-P. Talpin. 2014. Earliest-deadline first scheduling of multiple independent dataflow graphs. In *IEEE Workshop on Signal Processing Systems (SiPS)*.

[11] A. Bouakaz and J.-P. Talpin. 2013. Buffer Minimization in Earliest-deadline First Scheduling of Dataflow Graphs. In *14th ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES ’13)*.

[12] A. Bouakaz and J.-P. Talpin. 2013. Design of Safety-Critical Java Level 1 Applications Using Affine Abstract Clocks. In *International Workshop on Software and Compilers for Embedded Systems*. St. Goar, Germany.

[13] A. Bouakaz, J.-P. Talpin, and J. Vitek. 2012. Affine Data-Flow Graphs for the Synthesis of Hard Real-Time Applications. In *International Conference on Application of Concurrency to System Design*. Hamburg, Germany.

[14] Y. Chandarli, F. Fauberteau, D. Masson, S. Midonnet, and M. Qamhieh. 2012. YARTISS: A Tool to Visualize, Test, Compare and Evaluate Real-Time Scheduling Algorithms. In *WATERS*. Italy.

[15] H. Chetto, M. Silly, and T. Bouchentouf. 1990. Dynamic Scheduling of Real-time Tasks Under Precedence Constraints. In *Real-Time Syst.*, Vol. 2.

[16] A. Dkhil, X. Do, P. Dubrulle, S. Louise, and C. Rochange. 2014. Self-timed Periodic Scheduling for a Cyclo-static DataFlow Model. In *Procedia Computer Science*, Vol. 29.

[17] N. Fisher, S. Baruah, and T. P. Baker. 2006. The Partitioned Scheduling of Sporadic Tasks According to Static-Priorities *(ECRTS ’06)*.

[18] A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, B. D. Theelen, M. R. Mousavi, A. J. M. Moonen, and M. J. G. Bekooij. 2006. Throughput Analysis of Synchronous Data Flow Graphs *(ACSD ’06)*.

[19] R. Govindarajan, G. R. Gao, and P. Desai. 2002. Minimizing Buffer Requirements under Rate-Optimal Schedule in Regular Dataflow Networks. In *Journal of VLSI signal processing systems for signal, image and video technology*, Vol. 31.

[20] M. Joseph and P. Pandya. 1986. Finding Response Times in a Real-Time System. In *The Computer Journal*, Vol. 29.

[21] D. S. Kolovos, L. M. Rose, S. B. Abid, R. F. Paige, F. A. C. Polack, and G. Botterweck. 2010. Taming EMF and GMF Using Model Transformation. In *MODELS*.

[22] E. A. Lee and D. G. Messerschmitt. 1987. Synchronous data flow. In *Proceedings of the IEEE*, Vol. 75.

[23] O. M. Moreira and M. J. G. Bekooij. 2007. Self-Timed Scheduling Analysis for Real-Time Applications. In *EURASIP Journal on Advances in Signal Processing*.

[24] H. Oh and S. Ha. 2002. Fractional Rate Dataflow Model and Efficient Code Synthesis for Multimedia Applications. In *SIGPLAN Not.*, Vol. 37.

[25] F. Pellegrini. 2012. Scotch and PT-Scotch Graph Partitioning Software: An Overview. In *Combinatorial Scientific Computing*.

[26] R. Pellizzoni, P. Meredith, M.-Y. Nam, M. Sun, M. Caccamo, and L. Sha. 2009. Handling Mixed-criticality in SoC-based Real-time Embedded Systems. In *Seventh ACM International Conference on Embedded Software (EMSOFT ’09)*.

[27] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. 2004. Cheddar: A Flexible Real Time Scheduling Framework *(SIGAda ’04)*.

[28] M. Sjodin and H. Hansson. 1998. Improved response-time analysis calculations. In *19th IEEE Real-Time Systems Symposium*.

[29] S. Stuijk, M.C.W. Geilen, and T. Basten. 2006. SDF^3: SDF For Free. In *Application of Concurrency to System Design, 6th International Conference (ACSD ’06)*.

[30] S. Stuijk, M. Geilen, and T. Basten. 2008. Throughput-Buffering Trade-Off Exploration for Cyclo-Static and Synchronous Dataflow Graphs. In *IEEE Transactions on Computers*, Vol. 57.

[31] W. Thies, M. Karczmarek, and S. P. Amarasinghe. 2002. StreamIt: A Language for Streaming Applications *(CC ’02)*.

[32] W. F. J. Verhaegh, E. H. L. Aarts, P. C. N. van Gorp, and P. E. R. Lippens. 2001. A two-stage solution approach to multidimensional periodic scheduling. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 20.

[33] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit. 2007. Efficient Computation of Buffer Capacities for Cyclo-Static Dataflow Graphs. In *44th ACM/IEEE Design Automation Conference*.

[34] F. Zhang and A. Burns. 2009. Improvement to Quick Processor-Demand Analysis for EDF-Scheduled Real-Time Systems. In *21st Euromicro Conference on Real-Time Systems*.

[35] F. Zhang and A. Burns. 2009. Schedulability Analysis for Real-Time Systems with EDF Scheduling. In *IEEE Trans. Comput.*, Vol. 58.