

Simultaneous Budget and Buffer Size Computation for Throughput-Constrained Task Graphs*

Maarten H. Wiggers¹, Marco J.G. Bekooij^{2,3}, Marc C.W. Geilen¹, and Twan Basten^{1,4}

¹Eindhoven University of Technology, ²NXP Semiconductors, ³University of Twente,

⁴Embedded Systems Institute. The Netherlands

{m.h.wiggers, m.c.w.geilen, a.a.basten}@tue.nl; marco.bekooij@nxp.com

Abstract—Modern embedded multimedia systems process multiple concurrent streams of data processing jobs. Streams often have throughput requirements. These jobs are implemented on a multiprocessor system as a task graph. Tasks communicate data over buffers, where tasks wait on sufficient space in output buffers before producing their data. For cost reasons, jobs share resources. Because jobs can share resources with other jobs that include tasks with date-dependent execution rates, we assume run-time scheduling on shared resources. Budget schedulers are applied, because they guarantee a minimum budget in a maximum replenishment interval. Both the buffer sizes as well as the budgets influence the temporal behaviour of a job. Interestingly, a trade-off exists: a larger buffer size can allow for a smaller budget while still meeting the throughput requirement. This work is the first to address the simultaneous computation of budget and buffer sizes. We solve this non-linear problem by formulating it as a second-order cone program. We present tight approximations to obtain a non-integral second-order cone program that has polynomial complexity. Our experiments confirm the non-linear trade-off between budget and buffer sizes.

I. INTRODUCTION

Current embedded multimedia systems, such as car-entertainment systems and smart-phones, process multiple concurrent streams of data. Each stream of data is processed by a job, thereby resulting in a so-called multi-job system. In our multi-job system we have that (1) users start and stop jobs, and (2) jobs have different types of real-time requirements, i.e. hard, firm, soft, or best-effort.

Jobs are implemented as task graphs on a multiprocessor system to satisfy their performance requirements. In these task graphs, tasks communicate data over fixed capacity FIFO buffers. Tasks start based on the presence of sufficient data in their input buffers and sufficient space in their output buffers.

For reasons of cost, jobs share resources. In our multiprocessor system, we apply run-time schedulers that, on each resource, guarantee each job a minimum resource budget in a maximum replenishment interval. These run-time schedulers form the class of budget schedulers [10]. We apply run-time scheduling because there can be jobs in our system with data-dependent execution rates and furthermore run-time scheduling is an attractive option to support that users start and stop jobs. Budget schedulers are applied because they provide resource budgets that are independent of the behaviour of other jobs, which simplifies the performance analysis and increases robustness.

As tasks wait on the presence of space in output buffers, both the buffer sizes as well as the resource budgets influence

the temporal behaviour of a job. Interestingly, a trade-off exists between buffer sizes and budgets, e.g. a larger buffer size can allow for smaller resource budgets while still satisfying the real-time constraints. However, despite this trade-off no existing work is known to us that looked at the simultaneous computation of run-time scheduler settings and buffer sizes.

Instead, budget and buffer sizes are until now determined in two separate phases of the mapping flow [5], [8]. The mutual dependence of these two phases either causes (1) false negatives, i.e. a solution is not found even though a solution exists, or (2) iterations through these two phases. We are not aware of suitable heuristics to guide these iterations, and the non-linear trade-off between budget and buffer sizes and the dependence on the topology of the graph complicate the development of these heuristics.

In this paper, we present a method to simultaneously compute minimal budget and buffer sizes that are sufficient to guarantee the satisfaction of a throughput constraint in polynomial time. This method allows to make different trade-offs between budget and buffer sizes by changing the coefficients of the optimised cost function.

Run-time scheduling often causes highly non-linear effects. However, recently it has been shown that budget schedulers allow for conservative dataflow models that (1) abstract from multiprocessor scheduling anomalies and (2) have monotonic temporal behaviour [10]. Over the past few years it has been shown that buffer sizes that satisfy a throughput constraint can be computed using similar linear programming formulations, for important classes of dataflow models [9] that can also capture data-dependent inter-task communication. For reasons of space and to focus on the core contribution of this work, we restrict ourselves to task graphs that can be modelled with a single-rate dataflow graph [6].

The model that captures the effects of budgets schedulers from [10] with the linear constraints to compute buffer sizes do not together form a linear programming problem. We do not see an option to arrive at a reasonable linearised approximation. Instead, we formulate a second-order cone programming (SOCP) problem, which is a convex optimisation problem that can be efficiently solved, with polynomial complexity, using interior-point methods. This SOCP problem simultaneously computes sufficient budget and buffer sizes such that for each considered task graph its throughput constraint is satisfied.

The outline is as follows. Section II presents our application and analysis model. Section III combines the model to capture the effects of budget schedulers and the linear constraints to compute buffer sizes. Section IV presents and discusses

*This work was supported in part by the EC through FP7 IST project 216224, MNEMEE.

our SOCP formulation, which we evaluate in a number of experiments in Section V. Section VI concludes.

II. APPLICATION MODELLING

We define \mathbb{N} as the natural numbers $0, 1, \dots$, we define \mathbb{N}^* as $\mathbb{N} \setminus \{0\}$, and we define \mathbb{R}^+ as the non-negative real numbers.

A. Input and Output Descriptions of Mapping

1) *Input Description*: The input is a configuration that determines the set of constraints, and two functions that determine the coefficients in the objective function.

A configuration C is a tuple $C = (Q, P, M, \mu, \varrho, o, \varsigma, g)$ that consists of a finite set Q of task graphs, a finite set P of processors, a finite set M of memories, a function $\mu : Q \rightarrow \mathbb{R}^+$ that specifies the throughput requirement of a task graph, a function $\varrho : P \rightarrow \mathbb{N}$, that associates a replenishment interval with a processor, a function $o : P \rightarrow \mathbb{R}^+$ that specifies the maximum scheduling overhead per replenishment interval of a processor, and a function $\varsigma : M \rightarrow \mathbb{N}$ that specifies the maximum storage capacity for each memory. Further, the constant $g \in \mathbb{N}^*$ specifies the budget allocation granularity in this multiprocessor system. We define a task graph $T \in Q$ as a directed multi-graph $T = (W, B, \pi, \chi, \nu, \zeta, \iota)$. We have that W is a finite set of vertices, and that B is a finite set of labelled directed edges. The vertices are tasks and the edges are FIFO buffers. Task w executes on processor $\pi(w)$, with $\pi : W \rightarrow P$. On this processor, the task executions of task $w \in W$ have a worst-case execution time of $\chi(w)$, with $\chi : W \rightarrow \mathbb{R}^+$. Tasks synchronise on containers in FIFO buffers. FIFO buffer $b \in B$ is placed in memory $\nu(b)$, with $\nu : B \rightarrow M$. The size of a container of b is given by $\zeta(b)$, with $\zeta : B \rightarrow \mathbb{N}^*$, while the number of initially filled containers on b is $\iota(b)$, with $\iota : B \rightarrow \mathbb{N}$. Budget and buffer sizes are traded off with the weight functions $\mathbf{a} : W \rightarrow \mathbb{R}$ and $\mathbf{b} : B \rightarrow \mathbb{R}$, respectively.

2) *Output Description*: The output is a mapped configuration. In a mapped configuration, each task graph is augmented with two functions. Each task w is allocated a budget $\beta(w)$, with $\beta : W \rightarrow \mathbb{N}^*$, and each FIFO buffer b has a capacity given by a fixed number of containers $\gamma(b)$, with $\gamma : B \rightarrow \mathbb{N}^*$.

3) *Notation*: For convenience, we define W_Q as the union of the sets of tasks from all task graphs in Q of configuration C . Furthermore, we define B_Q as the union of the sets of buffers from all task graphs in Q of configuration C .

B. Single-Rate Dataflow

A Single-Rate Dataflow (SRDF) Graph G , also known as a computation graph [6] or as a homogeneous synchronous dataflow graph [4], [7] or as a marked graph [2], is a directed multi-graph $G = (V, E, \rho, \delta)$. The set V is a finite set of vertices, and E is a finite set of labelled directed edges. The vertices in an SRDF graph represent actors and the edges represent queues of tokens. The number of tokens on the queue represented by edge e is given by $\delta(e)$, with $\delta : E \rightarrow \mathbb{N}$. In an SRDF graph, an actor v has a single actor firing duration that is given by $\rho(v)$, with $\rho : V \rightarrow \mathbb{R}^+$. Furthermore, in an SRDF graph, in every firing, actors produce a single token on all adjacent output queues and consume a single token from all adjacent input queues.

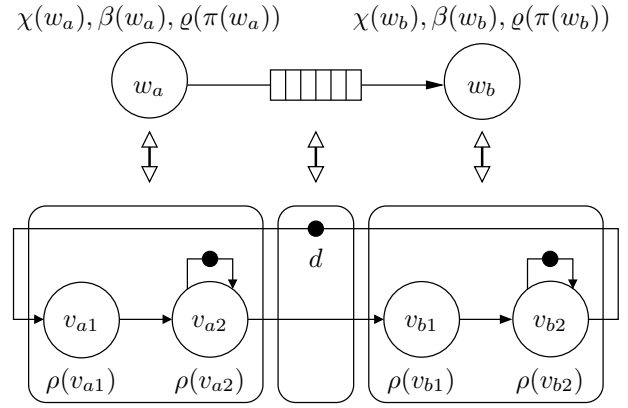


Fig. 1. Producer-consumer task graph with corresponding SRDF graph, where the buffer has a capacity of d containers that are all initially empty.

1) *Periodic Admissible Schedule (PAS)*: Let $\sigma(v_i, k)$ be the start time of the k -th firing of actor v_i , with $\sigma : V \times \mathbb{N}^* \rightarrow \mathbb{R}$. A schedule σ is admissible if for all actors v_i and for any k there is at least one token on all input queues of v_i at time $\sigma(v_i, k)$. An admissible schedule σ of SRDF graph G is periodic, i.e. a PAS, with period $\varphi(G) > 0$, if there exist real numbers $s(v_i)$ such that $\sigma(v_i, k) = s(v_i) + (k - 1)\varphi(G)$, for $k \geq 1$. The initial start times s determine a PAS with period $\varphi(G)$ if

$$s(v_j) \geq s(v_i) + \rho(v_i) - \delta(e_{ij})\varphi(G) \quad (1)$$

for each queue $e_{ij} \in E$ from actor v_i to v_j [6].

2) *Temporal Monotonicity*: An essential property of SRDF graphs is that they are temporally monotonic [9]. This both means that a smaller firing duration can never lead to later token arrival times, and that an increase in the number of initial tokens can never lead to later token arrival times.

C. Modelling Run-Time Scheduling

Following [10], we construct the SRDF graph as follows for each task graph. Each task $w_a \in W$ is modelled by a two-actor dataflow component consisting of actors $v_{a1}, v_{a2} \in V$, with $v_{a1} \neq v_{a2}$ and queues $e_{a1a2}, e_{a2a2} \in E$, with e_{a1a2} from actor v_{a1} to actor v_{a2} and e_{a2a2} from actor v_{a2} back to itself. Further, queue e_{a2a2} has one initial token, i.e. $\delta(e_{a2a2}) = 1$. Queues from actors that are part of other dataflow components to this dataflow component are all input queues of actor v_{a1} . Queues from this dataflow component to actors that are part of other dataflow components are all output queues of actor v_{a2} . Every buffer b_{ab} from task w_a to task w_b is modelled by two queues e_{a2b1} and e_{b2a1} in opposite direction between the two dataflow components that correspond with these tasks. Queue e_{a2b1} models the flow of data and has the initially filled containers as its initial number of tokens, i.e. $\delta(e_{a2b1}) = \iota(b_{ab})$. Queue e_{b2a1} models the flow of empty containers and has the initially empty containers as its initial number of tokens, i.e. $\delta(e_{b2a1}) = \gamma(b_{ab}) - \iota(b_{ab})$. For task w_a , we now have the two corresponding actors v_{a1} and v_{a2} . With v_{a1} we associate a firing duration equal to the difference between the replenishment interval and the budget of the task, i.e. $\rho(v_{a1}) = \varrho(\pi(w_a)) - \beta(w_a)$. With v_{a2} we associate a firing duration $\rho(v_{a2}) = \varrho(\pi(w_a)) \cdot \chi(w_a) / \beta(w_a)$.

Given the just specified one-to-one relation between task graph T and SRDF graph G and monotonicity of the SRDF graph, if a PAS exists with $\varphi(G) \leq \mu(T)$, then the throughput constraint of task graph T is satisfied.

From the set of task graphs Q , we obtain a set of SRDF graphs H_Q . Let the set V_Q be the union of all sets of actors in H_Q and let E_Q be the union of all sets of queues in H_Q .

III. PROBLEM STATEMENT

In general, we have the problem of finding firing durations and numbers of initial tokens that are sufficient to allow a PAS with a given period μ , given Constraint (1) for all edges.

As described in Section II-C, in our case a task w_i is modelled by a dataflow component consisting of two actors: v_{i1} and v_{i2} , with $\rho(v_{i1}) = \varrho(\pi(w_i)) - \beta(w_i)$ and $\rho(v_{i2}) = \varrho(\pi(w_i)) \cdot \chi(w_i) / \beta(w_i)$. We partition the set of queues E_Q in two subsets E_1 and E_2 , where all output queues of actors v_{i1} are in queue set E_1 and all output queues of actors v_{i2} are in queue set E_2 . By construction of the SRDF graph each actor v_{i1} only has a single output edge e_{i1i2} on which there are no initial tokens, i.e. $\delta(e_{i1i2}) = 0$. This implies that Constraint (1) is instantiated to Constraint (2) for edges in E_1 .

$$\forall e_{ij} \in E_1 : s(v_j) \geq s(v_i) + \varrho(\pi(w_i)) - \beta(w_i) \quad (2)$$

Constraint (1) is instantiated to Constraint (3) for edges in E_2 , where $\mu(e_{ij})$ is the throughput constraint of the task graph corresponding to queue e_{ij} .

$$\forall e_{ij} \in E_2 : s(v_j) \geq s(v_i) + \frac{\varrho(\pi(w_i))}{\beta(w_i)} \chi(w_i) - \delta(e_{ij}) \mu(e_{ij}) \quad (3)$$

For Time-Division Multiplex scheduling the replenishment interval of the scheduler equals the interval over which the budgets are guaranteed, which means that we can express this constraint as follows. In a configuration C , let $\tau(p)$ be the set of tasks that execute on processor p . Constraint (4) states that on a processor p the sum of the budgets allocated to the tasks in $\tau(p)$ needs to be at most the replenishment interval on p .

$$\forall p \in P : \varrho(p) \geq \sum_{w_i \in \tau(p)} \beta(w_i) \quad (4)$$

We do not see any opportunity to linearise this set of constraints in terms of the budgets, i.e. while keeping the budget a variable that remains to be determined. Therefore, we resort to formulate the problem as a second-order cone program (SOCP) [1], which is a generalisation of linear programming. The next section presents our SOCP formulation.

IV. PROGRAM FORMULATION

Algorithm 1 presents our SOCP formulation of the constraints discussed in Section III augmented with Constraint (10), which specifies that the FIFO buffers need to fit in their assigned memories. Here $\psi(m)$ is the set of queues that correspond with the buffers placed in memory m , and $\zeta(e_i)$ is the container size of the buffer modelled by e_i . We approximate the integer number of tokens $\delta(e)$ of queue e by a real valued number of tokens $\delta'(e)$, with $\delta' : E \rightarrow \mathbb{R}^+$ and $\delta(e) = \lceil \delta'(e) \rceil$. We also approximate the integer budget $\beta(w)$

of task w by a real valued budget $\beta'(w)$, with $\beta' : W \rightarrow \mathbb{R}^+$ and $\beta(w) = g \lceil \beta'(w) / g \rceil$. The correctness of both non-integral approximations is discussed in the next subsection.

We approximate $1/\beta(w_i)$ with $\lambda(w_i)$. In this way changing Constraint (3) to become Constraint (7). Because $\varrho(\pi(w_i))$ and $\chi(w_i)$ are constants in our formulation, Constraint (7) is an affine constraint. The relation between $\beta'(w_i)$ and $\lambda(w_i)$ is specified in Constraint (8). One would ideally have specified that the relation between $\beta'(w_i)$ and $\lambda(w_i)$ is given by the equation $\lambda(w_i)\beta'(w_i) = 1$. However, this is a non-convex constraint. Instead, an approximation is made in Constraint (8) in order to obtain a valid SOCP formulation. The chosen approximation is conservative, since $\lambda(w_i) \geq 1/\beta'(w_i)$ implies that a larger than or equal firing duration is taken into account for actors v_{i2} , which by monotonicity of SRDF is conservative.

The objective function aims to minimise a weighted sum of budgets and tokens. Because of the relation between the SRDF graph and the task graph, this objective is equivalent to the minimisation of a weighted sum of budget and buffer sizes. The weights can be freely chosen depending on the specific multiprocessor instance that is under consideration.

Algorithm 1

Minimise

$$\sum_{w_i \in W_Q} \mathbf{a}(w_i) \beta'(w_i) + \sum_{e_i \in E_Q} \mathbf{b}(e_i) \zeta(e_i) \delta'(e_i) \quad (5)$$

Subject to

$$\forall e_{ij} \in E_1 : s(v_j) \geq s(v_i) + \varrho(\pi(w_i)) - \beta'(w_i) \quad (6)$$

$$\forall e_{ij} \in E_2 : s(v_j) \geq s(v_i) + \varrho(\pi(w_i)) \lambda(w_i) \chi(w_i) - \delta'(e_{ij}) \mu(e_{ij}) \quad (7)$$

$$\forall w_i \in W_Q : \lambda(w_i) \beta'(w_i) \geq 1 \quad (8)$$

$$\forall p \in P : \varrho(p) \geq o(p) + \sum_{w_i \in \tau(p)} (\beta'(w_i) + g) \quad (9)$$

$$\forall m \in M : \varsigma(m) \geq \sum_{e_i \in \psi(m)} (\delta'(e_i) + 1) \zeta(e_i) \quad (10)$$

From $\beta'(w)$, the budget is obtained with $\beta(w) = g \lceil \beta'(w) / g \rceil$. This is a correct conservative approximation, because we know from Constraint (1) that if a PAS exists with period $\varphi(G)$ for firing duration $\rho(v)$, then this PAS with period $\varphi(G)$ also exists for firing duration $\rho'(v)$, with $\rho'(v) \leq \rho(v)$. With $\beta'(w) \leq \beta(w)$, the firing durations of the actors modelling task w are reduced after the rounding of the budget, and therefore rounding up of the budgets still allows for satisfaction of the throughput constraint. We have that $\beta(w) - \beta'(w) = g \lceil \beta'(w) / g \rceil - \beta'(w) \leq g$. Therefore, by adding g to each budget $\beta'(w)$ in the sum of budgets in Constraint (4) we conservatively take this rounding into account. This is done in Constraint (9) that also includes the worst-case scheduling overhead $o(p)$ as an a-priori allocated budget.

From $\delta'(e)$, the number of tokens is obtained with $\delta(e) = \lceil \delta'(e) \rceil$. This is a correct conservative approximation, because we know from Constraint (1) that if the throughput constraint is satisfied with $\delta'(e)$ then the throughput constraint is also satisfied with $\lceil \delta'(e) \rceil$. We have that $\delta(e) - \delta'(e) = \lceil \delta'(e) \rceil -$

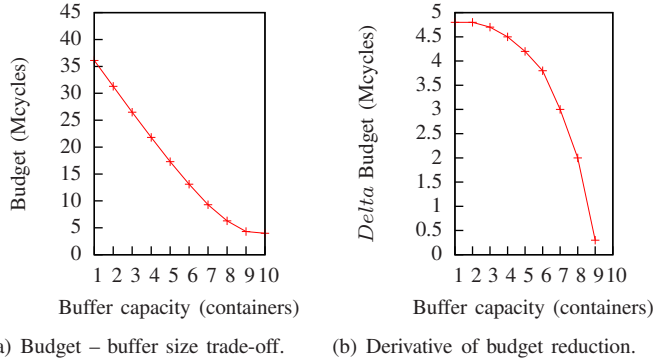


Fig. 2. Illustration of the non-linear trade-off between the budget and buffer sizes that are required to satisfy the throughput requirement.

$\delta'(e) \leq 1$. Therefore, we add one additional token to each number of tokens $\delta'(e)$ in Constraint (10).

Because the rounding to the next larger integer value is not part of the SOCP formulation these non-integral approximations come at the cost of potential sub-optimality.

V. EXPERIMENTAL RESULTS

This section presents two experiments. The first experiment shows that the trade-off between budgets and buffer sizes is non-linear. The second experiment shows that the topology of the graph is important when determining minimal budgets for given maximum buffer sizes.

The first experiment considers a producer-consumer task graph T_1 as shown in Figure 1, i.e. two tasks w_a, w_b and a buffer b_{ab} . The two tasks are each bound to different processors: $\pi(w_{w_a}) = p_1$ and $\pi(w_{w_b}) = p_2$. Both processors have a replenishment interval of 40 Mcycles: $\varrho(p_1) = \varrho(p_2) = 40$ Mcycles. The worst-case execution time of both tasks is 1 Mcycle: $\chi(w_a) = \chi(w_b) = 1$ Mcycle. The throughput requirement is a period of 10 Mcycles: $\mu(T_1) = 10$ Mcycles. Containers have unit size. Given this set-up, we configure the weights in the objective function of the SOCP formulation to prefer minimisation of the budgets over minimisation of the buffer sizes. The trade-off between the budgets $\beta(w_a)$ and $\beta(w_b)$, which are equal, and the buffer size is shown in Figure 2(a). This trade-off is explored by constraining the maximum buffer size. Figure 2(b) shows the reduction in required budget compared to the budget that is required with one fewer container. A buffer capacity of 10 containers minimises the budgets.

The second experiment considers a task graph T_2 , which extends the previous task graph T_1 with task w_c and buffer b_{bc} . Task w_c is bound to a different processor: $\pi(w_c) = p_3$, with $\varrho(p_3) = 40$ Mcycles. Task w_c also has $\chi(w_c) = 1$ Mcycle. The throughput requirement of this task graph is again 10 Mcycles: $\mu(T_2) = 10$ Mcycles. Given this set-up, the trade-off between the buffer sizes and the budgets is shown in Figure 3. The interesting aspect is that because the budget of task w_b interacts with two buffer sizes in this trade-off, the budgets of tasks w_a and w_c are reduced first before the budget of w_b is reduced. Both experiments were performed using the commercial solver CPLEX [3]. The run-time is milliseconds.

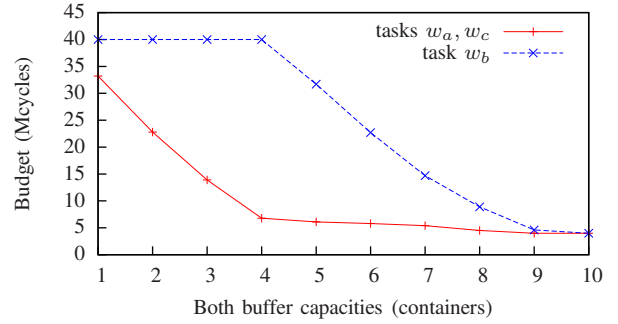


Fig. 3. Illustration of topology-dependence of optimisation of sum of budgets for given maximum buffer sizes.

VI. CONCLUSION

In contrast to existing work that computes budget and buffer sizes in two separate phases of a multiprocessor mapping flow, this work simultaneously computes budget and buffer sizes that satisfy a throughput constraint in polynomial time. Different trade-offs between budget and buffer sizes can be made by changing the coefficients of the optimised cost function. The trade-off between budget and buffer sizes is non-linear. We address this non-linearity by formulating the problem as a second-order cone program, which is a generalisation of linear programming. Our experiments show that the trade-off between budget and buffer sizes is non-linear and dependent on the topology of the task graph. The run-time of our analysis is milliseconds for these experiments.

This paper is focused on applications that can be modelled with a relatively static dataflow model. An essential next step is to extend the presented approach to include more dynamic applications. As explained, we believe that this is very well possible. Another important next step will be to extend the current formulation and also compute the binding of tasks to processors and of buffers to memories.

This paper combines two essential steps of a multiprocessor mapping flow, budget and buffer size computation, in a single problem formulation. The generality of the approach combined with its polynomial complexity make this work an important enabler for an automated multiprocessor mapping flow for stream processing applications.

REFERENCES

- [1] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge U. Press, 2004.
- [2] F. Commoner, et al. Marked directed graphs. *Journal of Computer and System Sciences*, 5, 1971.
- [3] ILOG. CPLEX. <http://www-01.ibm.com/software/integration/optimization/cplex>.
- [4] E. A. Lee and D. G. Messerschmitt. Synchronous Dataflow. *Proceedings of the IEEE*, 75(9):1235–1245, September 1987.
- [5] O. Moreira, et al. Scheduling Multiple Independent Hard-Real-Time Jobs on a Heterogeneous Multiprocessor. In *Proc. EMSOFT*, 2007.
- [6] R. Reiter. Scheduling Parallel Computations. *Journal of the ACM*, 15(4):590–599, October 1968.
- [7] S. Sriram and S.S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker Inc., 2000.
- [8] S. Stuijk et al. Multiprocessor Resource Allocation for Throughput-Constrained Synchronous Dataflow Graphs. In *Proc. DAC*, 2007.
- [9] M. H. Wiggers. *Aperiodic Multiprocessor Scheduling for Real-Time Stream Processing Applications*. PhD thesis, University of Twente, 2009.
- [10] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit. Monotonicity and Run-Time Scheduling. In *Proc. EMSOFT*, 2009.