A Hybrid Scheduling Algorithm Based on Self-Timed and Periodic Scheduling for Embedded Streaming Applications

Amira Dkhil, Xuankhanh Do, Stéphane Louise, Christine Rochange

A Hybrid Scheduling Algorithm based on Self-Timed and Periodic Scheduling for Embedded Streaming Applications

Amira Dkhil, Xuan Khanh Do, Stéphane Louise
CEA, LIST
PC172, 91191 Gif-sur-Yvette, France
Email: first.last@cea.fr

Christine Rochange
IRIT, Université de Toulouse
118 route de Narbonne, Toulouse, France
Email: rochange@irit.fr

***Abstract*—In this paper, we consider the problem of multiprocessor scheduling for safety-critical streaming applications modeled as acyclic data-flow graphs. To the best of our knowledge, most existing works have proposed periodic scheduling that ignore latency or can even have a negative impact on it: the results are quite far from those obtained under Self-Timed scheduling (STS). In this paper, we introduce a new scheduling policy noted Self-Timed Periodic (STP), which is an execution model combining self-timed scheduling with periodic scheduling. The proposed framework shows that the use of both strategies is possible and that they complement each other; STS improves the performance metrics of the programs, while the periodic model captures the timing aspects. We evaluate the performance of our scheduling policy for a set of 10 real-life streaming applications. We find that in most of the cases, our approach gives a significant improvement in latency compared to the Static Periodic Schedule (SPS), and results which are close to the best case latency of STS.**

I. Introduction

There is an increasing interest in developing applications on multiprocessor platforms due to their broad availability and the looming horizon of many-core chip, such as the MPPA chip from Kalray (256 cores) [1] or the SThorm chip from STMicroelectronics (64 cores). Given the scale of these new massively parallel systems, programming languages based on the data-flow model of computation have strong assets in the race for productivity and scalability. Nonetheless, as streaming applications must ensure *data-dependency constraints*, scheduling has serious impact on performance. Hence, multiprocessor scheduling for data-flow languages has been an active area and therefore many scheduling and resource management solutions was suggested.

The Self Timed Scheduling (STS) strategy (a.k.a. *as-soon-as-possible*) of a streaming application is a schedule where actors are fired as soon as data-dependency is satisfied. For a long time, this scheduling policy is considered as the most appropriate for streaming applications modeled as data-flow graphs [2] because it delivers the maximum achievable throughput and the minimum achievable latency if computing resources are sufficient [3]. However, this result can only be true if we ignore synchronization times [4]. Furthermore, STS does not provide real-time guarantees on the availability of a given result in conformance with time constraints. Therefore, analysis and optimization of self-timed systems under real-time constraints remains challenging. To cope with this challenge, periodic scheduling is receiving more attention for streaming applications [3], [5] with its good properties (*i.e.*, timing guarantees, temporal isolation and low complexity of the schedulability test). It was shown that interprocessor communication (IPC) overhead can be defined as a monotonically increasing function of the number of conflicting memory accesses in a given period of the schedule [5]. Moreover, periodic scheduling increases the latency significantly for a class of graphs called unbalanced graphs. A balanced graph is the one where the product of actor execution time and repetition is the same for all actors [6] but the most common case is unbalanced graph.

In this paper, we show that the Static Periodic Schedule (SPS) model, firstly presented in [3], increases significantly the latency of unbalanced graphs and that it is possible to resolve this problem by using a new scheduling policy noted Self-Timed Periodic (STP) schedule. STP is a hybrid execution model based on mixing Self-Timed schedule and periodic schedule while considering variable IPC times. To illustrate the impact of the STP model on the performance, we present the following motivational example.

A. Motivational Example

In Figure 1, we show a Cyclo-Static Dataflow (CSDF) [7] graph of an MP3 application. CSDF graphs are directed graphs where a set of nodes referred to as computation actors are connected by a set of edges which are communication FIFO channels. Each actor in the graph is executed through a periodically repeated sequence of sub-tasks. Any CSDF graph is characterized by two repetition vectors $\vec{q}$ and $\vec{r}$. $\vec{q}$ is the minimal set of sub-tasks firings returning the data-flow graph to its initial state (all inputs are consumed) (see Section III-A). For the example depicted in Figure 1, $\vec{r} = [1, 2, 2, 4, 4]$ and $\vec{q} = [3, 4, 4, 8, 8]$, respectively in the order $mp3, src1, src2, app, dac$. To get q_i, we multiply r_i by the length of the consumption and production rates of a_i. For example, if $r_1 = 1$ then $q_1 = 3$ because actor *mp3* contains 3 sub-tasks. The worst-case computation and

DOI 10.1109/PDP.2015.109

communication time of each actor is shown next to its name after a comma (*e.g.*, 4 for actor *mp3*). The graph is an example of an unbalanced graph since the product of actor execution time and repetition is not the same for all actors (*e.g.*, $3 \times 4 \neq 4 \times 9$). The latency resulting from scheduling the actors of this graph as static periodic tasks is $L_{SPS} = 192$ while this result for $STP^I_{q_i}$ and $STP^I_{r_i}$ is 144 and 108, respectively. We see that the SPS model pays a high price in terms of increased latency for the unbalanced graph. Instead, if the actors are to be scheduled as self-timed periodic tasks as introduced in this paper, then it is possible to achieve 25% to 60% improvement in latency compared to the SPS schedule.

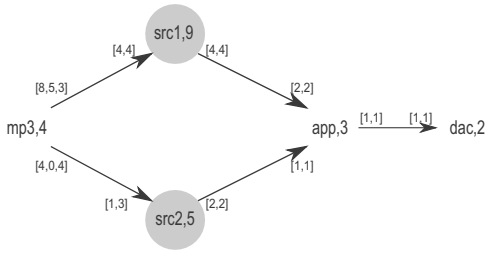

Figure 1. CSDF graph of the MP3 application

B. Paper Contributions

The contribution of this paper is two classes of STP schedules based on two different granularities. The first schedule, denoted $STP^I_{q_i}$, are based on the repetition vector q_i without including the sub-tasks of actors. A remaining schedule, denoted $STP^I_{r_i}$, have a finer granularity by including the sub-tasks of actors. It is based on the repetition vector r_i. For unbalanced graphs, we show that it is possible to significantly decrease the latency under the STP model for both granularities. We evaluate the proposed STP representation using a set of 10 real-life applications and show that it is capable of achieving significant improvements in term of latency (with a maximum of 96.6%) compared to the SPS schedule.

The remainder of this paper is organized as follows. In Section II, we represent a state of the art methods relative to the scheduling policies of CSDF graphs on multiprocessor systems. Section III introduces the background material needed for understanding the contributions of this paper. Section IV present our main contribution: the STP schedule. Section V present our evaluation of the proposed scheduling policy. Finally, Section VI ends the paper with conclusions.

II. Related Work

Latency is an useful performance indicator for concurrent real-time applications. Minimizing or analyzing latency of a stream program requires to find an efficient scheduling policy which should be achieved by hiding communication latencies whenever possible. Ghamarian *et al.* propose a heuristic for optimizing latency under a throughput constraint [8]. It gives optimal latency and throughput results under a constraint of maximal throughput for all DSP and multimedia models. However, his approach uses Synchronous Data-flow (SDF) graphs which are less expressive than CSDF graphs in that SDF supports only a constant production/consumption rate on edges, whereas CSDF supports varying (but predefined) production/consumption rates. As a result, the analysis result in [8] is not applicable to CSDF graphs. In [3], Bamakhrama and Stefanov present a complete framework for computing the periodic task parameters using an estimation of worst-case execution time. They assume that each write or read has constant execution time which is often not true. Our approach is somewhat similar to [3] in using the periodic task model which allows to apply a variety of proven hard-real-time scheduling algorithms for multiprocessors. However, it is different from [3] in: 1) in our model, actors will no longer be strictly periodic but self-timed assigned to periodic levels, and 2) we treat the case variable execution time of actors due to synchronization and contention in shared resources.

III. Background

We introduce in this section the timed graph, system model and schedulability of a CSDF graph which are important points for understanding our contribution.

A. Timed Graph

The timed graph is a more accurate representation of the CSDF graph [7], that associates to each sub-task or instance of an actor a computation time and a communication overhead. We consider the Timed graph $G = \langle A, E, \omega, \varphi \rangle$, where A is a set of actors, $E \subseteq A \times A$ is a set of communication channels, ω gives the worst-case computation time of each actor and φ is its communication time according to a scheduling policy. The set of actors is denoted by $A = \{a_1, a_2, ..., a_n\}$, where each actor represent one function that transform the input data streams into output data streams. The communication channels carry streams of data and work as a FIFO queue with unbounded capacity. An atomic piece of data carried out by a channel is called a $token$. A timed graph G is characterized by a *repetition vector* $\overrightarrow{q} = [q_1, q_2, ..., q_n]^T$, where $q_j > 0$ represents the number of invocations of an actor a_j in a *valid static schedule* for G. This repetition vector is given by [7]:

$$\overrightarrow{q} = P \cdot \overrightarrow{r} \ , with \quad P = P_{jk} = \begin{cases} \tau_j & , if \ \ j = k \\ 0 & , otherwise \end{cases} \tag{1}$$

And, $\overrightarrow{r} = [r_1, r_2, ..., r_n]^T$, where $r_i \in \mathbb{N}^\star$, is a solution of the balance equation:

$$\Gamma \cdot \overrightarrow{r} = 0, \tag{2}$$

where Γ is the *topology matrix* of G.

B. System's model and Schedulability

A system Π consists of a set $\pi = \{\pi_1, \pi_2, ..., \pi_m\}$ of m homogeneous processors. The processors execute a level set $V = \{V_1, V_2, ..., V_\alpha\}$ of α periodic levels. A periodic level $V_i \in V$ is defined by a 4-tuple $V_i = (S_i, \hat{\omega}_i, \hat{\varphi}_i, D_i)$, where $S_i \geqslant 0$ is the start time of V_i, $\hat{\omega}_i$ is the worst-case computation time (where $\hat{\omega}_i = \max_{k=1\rightarrow\beta_i} \omega_k$ with β_i representing the number of actors in level V_i), $\hat{\varphi}_i \geqslant 0$ is the worst-case communication time of V_i under STP schedule and D_i is the relative deadline of V_i where $D_i = \max_{k=1\rightarrow\beta_i} D_k$. A periodic level V_i is invoked at time $t = S_i + k\phi$, where $\phi \geqslant \hat{\omega}_i + \hat{\varphi}_i$ is the level period, and has to finish execution before time $t = S_i + k\phi + \hat{\omega}_i + \hat{\varphi}_i$. If $D_i = \phi$, then V_i is said to have *implicit-deadline*. If $D_i < \phi$, then V_i is said to have *constrained-deadline*. In this article, only the first case will be discussed.

Authors in [2] introduced a theorem that states the sufficient and necessary conditions for a *valid schedule*. However, this result was established for Synchronous Data-flow graphs where actors have constant execution times. The test in [4] is a novel contribution which allows the timing of firing to respect the firing rules of actors in a CSDF graph.

IV. Self-Timed Periodic Model

The effect of Self-timed Periodic (STP) scheduling can be modeled by replacing the period of the actor in each level by its worst-case execution time under periodic scheduling. The worst-case execution time is the total time of computation and communication parts of each actor. The period of each level *i* is the maximum time it needs to fire each actor $a_j \in V_i$, when resource arbitration and synchronization effects are taken into account. This is counted from the moment the actor meets its enabling conditions to the moment the firing is completed. There are 2 types of STP scheduling that we are interested in this article: **coarse-grained schedule** $STP^I_{q_i}$ by using $\overrightarrow{q}$ as the repetition vector and **fine-grained schedule** $STP^I_{r_i}$ by using $\overrightarrow{r}$ as the repetition vector.

A. Definitions

Definition 1: An *actor workload* is defined as:

$$W_i = v_i \times \omega_i, \tag{3}$$

where v_i is the ith component of the repetition vector used for STP schedule. For STP_{q_i}, $v_i = q_i$ and for STP_{r_i}, $v_i = r_i$. The maximum workload of level V_j is $\hat{W}_j = \max_{a_i \in V_j}\{W_i\}$.

Definition 2: Let $p_{a\rightsquigarrow z} = \{(a_a, a_b), \ldots, (a_y, a_z)\}$ be an output path in a graph G. The latency of $p_{a\rightsquigarrow z}$ under periodic input streams, denoted by $L(p_{a\rightsquigarrow z})$, is the elapsed time between the start of the first firing of a_a which produces data to (a_a, a_b) and the finish of the first firing of a_z which consumes data from (a_y, a_z).

Algorithm 1 GRAPH-LEVELS-STP-Ri(S)

Require: Timed graph $G = \langle A, E, \omega, \varphi \rangle$
1: $count_i \leftarrow 0$
2: $j \leftarrow 1$
3: $S \leftarrow \{a_1\}$
4: **if** $count_i = q_i \ \forall a_i \in A$ **then**
5: **break**
6: **else**
7: $V_j \leftarrow \{a_k \in S : there\ are\ enough\ tokens\ in\ all\ input\ edges\ of\ a_k\}$
8: $j \leftarrow j + 1$
9: **for all** $a_i \in S$ **do**
10: $count_i \leftarrow count_i + r_i$
11: **if** $count_i < q_i$ **then**
12: $S = S \bigcup$ **succ**(a_i)
13: GRAPH-LEVELS-STP-Ri(S)
14: **else**
15: **if** $count_i = q_i$ **then**
16: $S \leftarrow S \backslash \{a_i\}$
17: **end if**
18: **end if**
19: **end for**
20: **end if**
21: $\alpha' \leftarrow j$ - 1
22: **return** α' disjoint sets $V_1, V_2, \ldots, V_{\alpha'}$

B. Latency Analysis under STP Schedule

A self-timed schedule does not impose any extra latency on the actors. This leads us to the following result:

Definition 3: (Periods of Levels in STP_{q_i}) For a graph G, a period ϕ, where $\phi \in \mathbb{Z}^+$, represents the period, measured in time-units, of the levels in G. If we consider $\overrightarrow{q}$ as the basic repetition vector of G in Definition 1, then ϕ is given by the solution to:

$$\phi \geqslant \max_{j=1\rightarrow\alpha} (\hat{W}_j + \hat{\varphi}_j) \tag{4}$$

Definition 3 defines the level period ϕ as the maximum execution time of all levels. Similarly, we define the schedule function for the finer granularity of CSDF characterized by the repetition vector $\overrightarrow{r}$ if we consider $\overrightarrow{r}$ as the basic repetition vector of G in Equation 4.

For STP_{q_i}, we use the algorithm proposed in [3] to find the levels of G. For STP_{r_i}, Algorithm 1 is used because this scheduling policy has a finer granularity and requires an algorithm which depends also on the precedence constraints of actors. In this case, each actor could only be fired if there are enough tokens in all of their input edges.

An actor $a_i \in V_j$ is said to be a level-j actor. For STP_{q_i}, let ϕ denote the level period as defined in Definition 3, and let a_1 denote the level-1 actor. a_1 will complete one iteration when it fires q_1 times. Assume that a_1 starts executing at time $t = 0$. Then, by time $t = \phi \geqslant q_1\omega_1$ as defined in Definition 3, a_1 is guaranteed to finish one iteration in a

self-timed mode and generate enough data such that every actor $a_k \in V_2$ can execute q_k times (*i.e.* one iteration). By repeating this over all the α levels, a schedule S_α (shown in Figure 2) is constructed in which all actors $a_i \in V_j$ are started at *start time*, denoted $s_{i,j}$, given by:

$$s_{i,j} = (j-1)\phi \tag{5}$$

time	$[0,\phi)$	$[\phi,2\phi)$	$[2\phi,3\phi)$	...	$[(\alpha-1)\phi,\alpha\phi)$
level	$V_1(1)$	$V_2(1)$	$V_3(1)$	...	$V_\alpha(1)$
		$V_1(2)$	$V_2(2)$	...	$V_{\alpha-1}(2)$
			$V_1(3)$	...	$V_{\alpha-2}(3)$
				...	$V_{\alpha-3}(4)$
				...	...
					$V_1(\alpha)$

Figure 2. Schedule S_α

According to Definition 2, latency is defined as the maximum time elapsed between the first firing of source actor src in level V_1 and the finish of the first firing of sink actor snk in level V_α. Then, the graph latency $L(G)$ is given by:

$$L_{STP^I_{q_i/r_i}} = (s_{snk,\alpha} + \phi) - s_{src,1} = \alpha \times \phi \tag{6}$$

Example 1: We illustrate in Figure 3 different scheduling policies applied for the MP3 application shown in Figure 1.

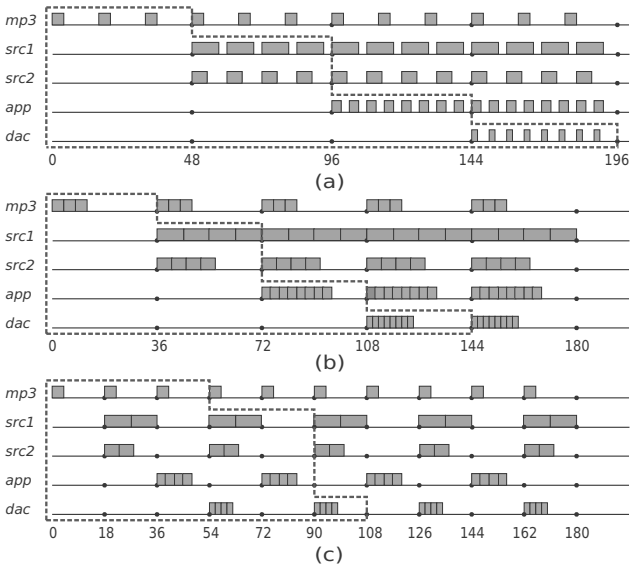

Figure 3. Illustration of latency path for the the MP3 application: (a) SPS (b) $STP^I_{q_i}$ (c) $STP^I_{r_i}$. The number of levels for $STP^I_{q_i}$ is $\alpha = 4$ and the number of levels for $STP^I_{r_i}$ is $\alpha = 6$. We use Equation 4 to find the period of levels $\phi = 36$ for $STP^I_{q_i}$ and $\phi = 18$ for $STP^I_{r_i}$. Using Equation 6, we have $L_{STP^I_{q_i}} = 144$ and $L_{STP^I_{r_i}} = 108$. An improvement of 25% to 60% in latency could be achieved by the $STP^I_{q_i}$ and $STP^I_{r_i}$ schedules compared to the SPS schedule

V. Evaluation Results

We evaluate our proposed scheduling policy in Section IV by performing an experiment on a set of 10 real-life streaming applications.

A. Benchmarks

We used benchmarks from different domains (*e.g.*, signal processing, video processing, mathematics, *etc.*) and different sources to check the efficiency of this scheduling in different architectures. The first source is the ΣC benchmark (CSDF based extension of C language [11]) which contributes 4 streaming applications. The second source is the SDF^3 benchmark which contributes 5 streaming applications [10]. The last source is the StreamIt benchmark [9]. In total, 10 applications are considered as shown in Table I. The graphs are a mixture of CSDF (ΣC's applications) and SDF (StreamIt and SDF^3 benchmark) graphs. The use of synchronous data-flow (SDF) models does not affect our scheduling policy because SDF, with static firing rules of actors, is a special case of CSDF model [7], [12]. The second column (N) shows the number of actors in each application, the third column (Q) shows the least-common-multiple of the repetition vector elements (*i.e.*, $Q = lcm(q_1, q_2, \ldots, q_n)$) and the fourth column is the maximum of the product $q_i\omega_i$ used to calculate the end-to-end latency by Formula (3), (4) and (6). The actors execution times of the ΣC benchmark are measured in clock cycles on the MPPA 256 cores [1], while the actors execution times of the SDF^3 benchmark are specified by its authors for ARM architecture. For the StreamIt benchmark, the actors execution times are specified in clock cycles measured on MIT RAW architecture.

B. Experiment: Latency comparison

In this experiment, we compare the end-to-end latency resulting from our STP approach to the minimum achievable latency of a streaming application obtained via self-timed scheduling and the one achieved under static periodic scheduling. Table I shows the latency obtained under STS, SPS, $STP^I_{q_i}$, $STP^I_{r_i}$ schedules as well as the improvement of these policies compared to the SPS model. We report the graph maximum latency according to Formula (6). For SPS schedule, we used the minimum period given in [3]. For STP chedule, we used the level period given by Definition 3. We see that the calculation of the STP schedule is not complicated because the graph is consistent and an automatic tool could be implemented to find this schedule.

For the $STP^I_{q_i}$, we see that it delivers an average improvement of 39.4% (with a maximum of 96.6%) compared to the SPS model for all the applications. In addition, we clearly see that our $STP^I_{q_i}$ provides at least 25% of improvement for 7 out of 10 applications. Only three applications (Filterbank, Beamformer and H.263 Encoder) have lower performance under our $STP^I_{q_i}$. To understand the impact of the results, we use the concept of *balanced* graph. According to [6], periodic models increase the latency significantly for *unbalanced*

Table I
BENCHMARKS USED FOR EVALUATION

Application	N	Q	max($q_i\omega_i$)	STS	SPS	$STP^I_{q_i}$	$Eff_{STP^I_{q_i}}(\%)$	$STP^I_{r_i}$	$Eff_{STP^I_{r_i}}(\%)$	Source
DCT	4	12	1800	2500	7200	5400	38.3	4500	57.5	CEA LIST
FFT	4	6	900	23000	36000	27000	69.2	32000	30.8	
Beamformer	4	12	7800	9500	25200	23400	11.5	30000	-30.6	
Filterbank	17	600	113430	124792	1254000	1247730	0.6	1247730	0.6	[9]
MP3	5	24	36	48	192	144	33.3	108	58.3	CEA LIST
Sample-rate	6	23520	960	1000	141120	5760	96.6	5760	96.6	[10]
H.263 Encoder	5	33	382000	664000	1584000	1528000	6.1	1528000	6.1	
H.263 Decoder	4	2376	10000	23506	47520	40000	31.3	40000	31.3	
Bipartite	4	144	252	293	576	504	25.4	504	25.4	
Satellite	22	5280	1056	1314	58080	11616	81.9	11616	81.9	

graphs. For our approach, Definition 3 and Formula (6) indicate that if the product $q_i\omega_i$ is too different between actors, so the period of levels ϕ and the latency L become higher. For actors where this product is much smaller, wasted time in each level increases the final value of latency. This main reason prompt us to reduce these bad effects by using the constrained-deadline self-timed periodic schedule $STP^C_{q_i}$ and $STP^C_{r_i}$. We also see that the $mis-matched$ I/O rates applications (*i.e.* with large Q such as Sample-rate, Satellite and Filterbank in Table I) have higher latency under static periodic scheduling. This result could be explained using an interesting finding reported in [9]: *Neighboring actors often have matched I/O rates. This reduces the opportunity and impact of advanced scheduling strategies proposed in the literature*. This issue can be resolved by using our approach. In fact, for nearly balanced graphs (*i.e.*, graphs where the product $q_i\omega_i$ is not too different between actors) such as Sample-rate and Satellite, we have an improvement of 96.6% and 81.9%, relatively, for the end-to-end latency of each benchmark. For the remaining applications, the SPS model increases the latency on average by 2.5× compared to the STS latency while this rate for $STP^I_{q_i}$ is 2×.

For the $STP^I_{r_i}$ approach, we have an average improvement of 35.8% compared to the SPS model for all the applications. For 8 out of 10 benchmarks, this scheduling policy give at least the result given by $STP^I_{q_i}$. Only two applications (Beamformer and FFT) have lower performance when using this scheduling policy. The main reason is that the $STP^I_{r_i}$ give a finer granularity based on the repetition vector r_i. This means that if r is too close to $\vec{1}$, the sum of wasted time in each level will significantly increases the end-to-end latency.

VI. CONCLUSIONS

We prove that the actors of a streaming application modeled as CSDF graph, can be scheduled as self-timed periodic tasks. As a result, we conserve the properties of a periodic scheduling and in the same time improve its performance. We also show how the different granularities offered by CSDF model can be explored to decrease latency. We present an analytical framework for computing the periodic task parameters while taking into account inter-processor communication and synchronization overhead. Based on empirical evaluations, we show that our STP approach gives a significant improvement in latency compared to the SPS model and a slight degradation compared to the maximum latency achieved under the STS model. As a future work, we want to compute the throughput delivered by the STP model and improve our scheduling policy by using the constrained deadline which requires different schedulability analysis.

REFERENCES

[1] B. D. de Dinechin, P. G. de Massas, G. Lager, C. Léger, B. Orgogozo, J. Reybert, and T. Strudel, "A distributed run-time environment for the kalray mppa-256 integrated manycore processor," *Procedia Computer Science*, 2013.

[2] O. Moreira, "Temporal analysis and scheduling of hard real-time radios running on a multi-processor," ser. PHD Thesis, Technische Universiteit Eindhoven, 2012.

[3] M. A. Bamakhrama and T. Stefanov, "On the hard-real-time scheduling of embedded streaming applications," *Design Automation for Embedded Systems*, 2012.

[4] A. Dkhil, X. Do, S. Louise, and C. Rochange, "Self-timed periodic scheduling for cyclo-static dataflow model," in *ICCS*, ser. Procedia Computer Science, vol. to be published, 2014.

[5] A. Dkhil, S. Louise, and C. Rochange, "Worst-Case Communication Overhead in a Many-Core based Shared-Memory Model," in *JRWRTC*, 2013.

[6] M. A. Bamakhrama and T. Stefanov, "Managing latency in embedded streaming applications under hard-real-time scheduling," in *CODES+ISSS*, 2012, pp. 83–92.

[7] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cyclo-static data flow," in *ICASSP*, May 1995, pp. 3255–3258 vol.5.

[8] A. H. Ghamarian, S. Stuijk, T. Basten, M. C. W. Geilen, and B. D. Theelen, "Latency minimization for synchronous data flow graphs," in *Proceedings of DSD*. IEEE Computer Society, 2007, pp. 189–196.

[9] W. Thies and S. Amarasinghe, "An empirical characterization of stream programs and its implications for language and compiler design," in *Proc. of PACT*, 2010, pp. 365–376.

[10] S. Stuijk, M. Geilen, and T. Basten, "Sdf^3: Sdf for free," in *Proceedings of ACSD*, 2006, pp. 276–278.

[11] T. Goubier, R. Sirdey, S. Louise, and V. David, "ΣC: A programming model and language for embedded manycores," in *Proceedings of ICA3PP*, 2011, pp. 385–394.

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