Resource Optimization for CSDF-modeled Streaming Applications with Latency Constraints

Di Liu*, Jelena Spasic*, Jiali Teddy Zhai*, Todor Stefanov*
*Leiden Institute of Advanced Computer Science
Leiden University, Leiden, The Netherlands
Email: {d.liu, j.spasic, j.t.zhai, t.p.stefanov}@liacs.leidenuniv.nl

Gang Chen†
†Institute of Robotics and Embedded Systems
Technical University of Munich, Munich, Germany
Email: cheng@in.tum.de

***Abstract*—In this paper, we study the problem of minimizing the number of processors required for scheduling latency-constrained streaming applications modeled as CSDF graphs, where the actors of a CSDF are executed as strictly periodic tasks. We formalize the problem and prove that due to the strict periodicity of actors the problem is an integer convex programming problem, that can be solved efficiently by using an existing convex programming solver. We evaluate our solution approach on a set of 13 real-life streaming applications modeled as CSDF graphs and demonstrate that it can reduce the number of processors in more than 52% of the conducted experiments in comparison to an existing approach.**

I. Introduction

Streaming applications, such as video/audio processing and digital signal processing, have become prevalent in embedded systems. These applications contain ample amount of parallelism which perfectly matches the processing power of Multi-Processor System-on-Chip (MPSoC) platforms. To efficiently program MPSoC platforms, Models-of-Computation (MoCs) are used to specify streaming applications. Prominent examples of MoCs include Synchronous Data Flow (SDF) [1] and its generalization Cyclo-Static Dataflow (CSDF) [2], in which actors representing computation are executed concurrently, thereby naturally exposing parallelism. Furthermore, the strong design-time analyzability of these MoCs makes them suitable for designing performance-constrained embedded systems.

Performance constraints of a streaming application are usually imposed on two principle metrics, throughput and latency. For many streaming applications, the latency is the main concern, where latency is the elapsed time between the arrival of a sample to an application and the output of the processed sample by the application. For instance, in video conferencing and automatic pattern recognition applications, a latency that exceeds a certain limit cannot be tolerated. At the same time, the required number of processors to execute the application should be minimized for better resource usage and energy efficiency.

A few efforts have been made to deal with latency of streaming applications specified as SDF/CSDF graphs. The authors in [3] studied minimizing latency for SDF graphs, where latency is computed by a state-space traversal which has exponential complexity. Moreover, they assumed that there is no constraint on the number of processors required to schedule an application. However, the number of processors is an important design concern for embedded systems with respect to power consumption and area. In another effort, the authors in [4] and [5] proposed a scheduling framework that schedules acyclic CSDF graphs in a strictly periodic fashion. In this scheduling framework, each CSDF actor executes strictly periodically and meets a given deadline. The periodic execution of actors guarantees a certain throughput and latency. In addition, the latency of the CSDF graph can be reduced by selecting proper deadlines of actors. Moreover, [4] and [5] showed that a class of CSDFs scheduled as a periodic task set can achieve the maximum throughput and minimum latency. Scheduling a CSDF graph as a periodic task set makes it possible to benefit from a large amount of proven theories developed by the real-time systems community, which provide several efficient and fast approaches to compute the number of processors instead of performing complex design space exploration. Due to the advantages, we mentioned above, we use the CSDF model to specify a streaming application and schedule the CSDF graph as a periodic task set.

In a real-time embedded system, computing the minimum number of processors needed to execute a periodic task set depends on the deadline of each task in the set. When CSDF actors are scheduled as periodic tasks, the deadline of each actor can be varied in a well-defined bounded interval (see Section III-B), thereby controlling the application latency and the number of processors needed to schedule the application. This means that selecting a proper deadline value for each actor is an important issue for reducing the latency and minimizing the number of processors. In [5], the authors give a method to select deadlines of the task set to reduce latency. However, their method is not optimal in terms of the required number of processors. Therefore, in this paper, we study the problem of minimizing the number of processors required to schedule a streaming application modeled as a CSDF, while satisfying a given latency constraint, when the actors of the CSDF are executed as strictly periodic tasks. We formalize this problem and prove that it is an integer convex programming problem such that it can be solved efficiently by an off-the-shelf convex programming solver, e.g., CVX [6]. The novel contributions of our work can be summarized as follows:

- We present a new method to compute the earliest starting time of actors in a CSDF graph when the actors are executed as strictly periodic tasks.
- Based on the above contribution, we formalize the problem of minimizing the number of processors for a latency constrained CSDF graph and prove that it is an integer convex programming (ICP) problem.
- We carry out experiments by solving the ICP problem on 13 real-life streaming applications and demonstrate the effectiveness and efficiency of our solution approach in comparison to the deadline selection approach [5] in terms of the minimum number of processors required to schedule an application. By applying our approach, we obtain reduction in the number of processors in more than 52% of the conducted experiments.

The rest of the paper is organized as follows: Section II gives an overview of the related work. Section III introduces

978-3-9815370-2-4/DATE14/©2014 EDAA

TABLE I
NOTATIONS

G	A CSDF graph
M	Number of processors
V	Set of actors (tasks) in G
τ_j	The j^{th} actor in G
C_j	Worst-case execution time of the j^{th} actor
S_j	Earliest start time of the j^{th} actor
T_j	Period of the j^{th} actor
D_j	Deadline of the j^{th} actor
U_{sum}	The total utilization $U_{\text{sum}} = \sum_{\tau_j \in V} C_j/T_j$
δ_j	Density of the j^{th} actor $\delta_j = C_j/\min(T_j, D_j)$
δ_{sum}	The total density $\delta_{\text{sum}} = \sum_{\tau_j \in V} \delta_j$
L	Latency Constraint

the necessary background to better understand the paper. Section IV shows a motivational example and Section V presents our proposed solution approach. Evaluation of the proposed approach is presented in Section VI. Finally, Section VII concludes the paper.

II. Related Work

In the context of real-time systems, several works deal with period and/or deadline selection for periodic tasks in order to achieve certain goals. The authors in [7] optimize periods for dependent tasks on hard real-time distributed automotive systems in order to meet a latency constraint. In [8], Hong et al. propose a distributed approach to assign local deadlines for each task on distributed systems to meet a latency constraint. In contrast to [7] and [8], our work selects deadlines for data dependent tasks in order to meet a latency constraint while minimizing the number of processors required for scheduling the application. Such minimization is not considered in [7] and [8]. Balbastre et al. [9] propose an analysis to select deadlines for periodic tasks on a uniprocessor to reduce the output jitters. Comparing to [9], our work differs in that we select deadlines in order to reduce the required number of processors in a multiprocessor system while guaranteeing the latency constraint. Chantem et al. [10] optimize the periods and deadlines simultaneously for an infeasible independent task set such that it can be scheduled on a uniprocessor. Their work concentrates on the schedulability of a system rather than optimizing the resources while meeting the latency constraint which is the main goal of our work.

In another aspect, only a few works deal with latency of streaming applications specified as dataflow/task graphs. Given latency or throughput constraints, Javaid et al. [11] optimized the area of MPSoCs which are comprised of Application Specific Instruction set Processors (ASIP). The problem is formulated as an integer linear programming (ILP) problem. In their work, the area is optimized by setting different configurations for each ASIP. In contrast to their work, we consider to minimize the number of processors and our problem is not linear and thus not amenable to an ILP formulation. The authors in [12] proposed a framework to synthesize homogeneous multiprocessor system for streaming applications with throughput constraints while optimizing latency and resources. However, their framework can not take the latency as a constraint. As a result, the framework in [12] is not applicable to our problem. It is worth noting that the throughput constraint in [11] and [12] can be trivially added into our approach.

III. Preliminaries

In this section, we provide an overview of the CSDF MoC, introduce the real-time scheduling of a CSDF and the system model we use, and briefly describe the deadline selection approach in [5] which we use as a baseline approach for comparison. For the sake of the discussion, we list all the notations used in this paper in Table I.

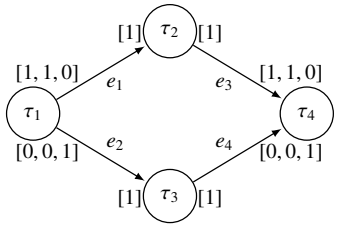


Fig. 1. A CSDF graph G

A. Cyclo-Static Dataflow (CSDF)

A CSDF [2] graph is defined as a directed graph $G = (V, E)$, where V is a set of actors and E is a set of edges. Actors $\tau_j \in V$ represent computation and edges represent the transfer of data tokens. An edge $e_u \in E$ is a FIFO defined as the pair $e_u = (\tau_i, \tau_j)$, denoting a connection from actor τ_i to τ_j. Each CSDF actor may produce/consume a variable but predefined number of data tokens in consecutive executions, called *production/consumption sequence*. Fig. 1 shows an example of a CSDF graph. For instance, the token production sequence on edge e_1 is $[1, 1, 0]$.

A valid static scheduling of a CSDF graph can be generated at design-time if the graph is consistent and live. Throughout this paper, all CSDF graphs are assumed to be consistent and live. A CSDF graph G is said to be consistent if a non-trivial solution exists [2] for a *repetition vector* $\vec{q} = [q_1, q_2, \cdots, q_N]$. An entry q_j represents the number of invocations of an actor τ_j in a graph iteration of G. For graph G shown in Fig. 1, the repetition vector is $\vec{q} = [3, 2, 1, 3]$.

B. Real-time Scheduling of CSDF

In [4], a real-time scheduling framework for CSDF graphs is proposed. In this framework, every actor in a CSDF graph is characterized by a 4-tuple $\tau_j = \{S_j, C_j, D_j, T_j\}$ in which S_j is the start time, C_j is the worst-case execution time (WCET), D_j is the relative deadline, and T_j is the period. D_j can be selected to take any value in the bounded interval $[C_j, T_j]$, thereby controlling the latency and changing the number of processors needed to schedule the actors. When $D_j = T_j$, the task set is said to be implicit deadline periodic (IDP) task set. If $D_j \leq T_j$, the task set is called constrained deadline periodic (CDP) task set. In [5], the value of D_j is determined by selecting a global deadline scaling factor df as follows:

$$\forall \tau_j \qquad D_j = C_j + df * (T_j - C_j) \qquad 0 \leq df \leq 1 \tag{1}$$

That is, the scaling factor df is the same for all actors and found/selected as explained in Section III-D.

To execute a CSDF graph as a peridic task set, a method to compute the earliest start time S_j of each actor is proposed in [4].

Lemma 1 (From [4])**.** *For an acyclic G, the earliest start time of an actor $\tau_j \in V$, denoted S_j, under a periodic schedule is given by*

$$S_j = \begin{cases} 0 & \text{if } \Omega(\tau_j) = \emptyset \\ \max_{\tau_i \in \Omega(\tau_j)} \left(S_{i \to j}\right) & \text{if } \Omega(\tau_j) \neq \emptyset \end{cases} \tag{2}$$

where $\Omega(\tau_j)$ is the set of predecessors of τ_j, and $S_{i\to j}$ is given by

$$S_{i\to j} = \min_{t \in [0, S_i + \alpha]} \{t : \underset{[S_i, \max(S_i, t)+k)}{\mathsf{prd}} (\tau_i) \geq \underset{[t, \max(S_i, t)+k]}{\mathsf{cns}} (\tau_j)\ \forall k \in [0, \alpha]\} \tag{3}$$

where S_i is the earliest start time of a predecessor actor τ_i, $\alpha = q_i T_i = q_j T_j$, $\mathsf{prd}_{[t_s, t_e)}(\tau_i)$ is the number of tokens produced by τ_i during the time interval $[t_s, t_e)$, and $\mathsf{cns}_{[t_s, t_e]}(\tau_j)$ is the number of tokens consumed by τ_j during the time interval $[t_s, t_e]$.

TABLE II
TASKS PARAMETERS 1

task	S_j	C_j	D_j	T_j
τ_1	0	2	2	6
τ_2	2	3	3	9
τ_3	14	3	3	18
τ_4	14	6	6	6

TABLE III
TASKS PARAMETERS 2

task	S_j	C_j	D_j	T_j
τ_1	0	2	2	6
τ_2	2	3	**9**	9
τ_3	14	3	**12**	18
τ_4	14	6	6	6

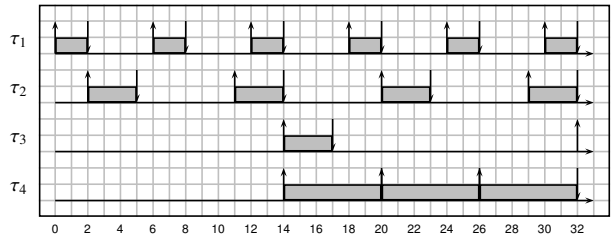


Fig. 2. The schedule of the CSDF graph

Table II gives an example of the parameters of the periodic task set corresponding to the CSDF graph in Fig. 1, computed using the theory in [4], when $df = 0$, i.e. $D_j = C_j$. The time unit is clock cycles. The strictly periodic schedule of the CSDF graph is shown in Fig. 2, where gray boxes represent executions of actors. For instance, according to Table II, actor τ_2 starts its execution at time $S_2 = 2$ and repeats its execution after $T_2 = 9$ clock cycles, and every firing of τ_2 finishes its execution by its relative deadline $D_2 = C_2 = 3$ clock cycles, shown as the downarrow.

To compute the latency of a CSDF graph G, we have to introduce the notions of input and output actors. An input actor τ_{in} is the actor receiving a stream of data from the external environment, while an actor producing an output stream to the external environment is called an output actor τ_{out}. [5] gives the following equation to compute the latency of G:

$$L(G) = \max_{w_{\text{in}\to\text{out}} \in \mathcal{W}} (S_{\text{out}} + g^C_{\text{out}} T_{\text{out}} + D_{\text{out}} - (S_{\text{in}} + g^P_{\text{in}} T_{\text{in}})) \quad (4)$$

where $\mathcal{W}$ is the set of all paths from input actor τ_{in} to output actor τ_{out}, and $w_{\text{in}\to\text{out}}$ is one path of the set. S_{out} and S_{in} are the earliest start times of τ_{out} and τ_{in}, respectively. T_{out} and T_{in} are the periods of τ_{out} and τ_{in}, respectively. D_{out} is the deadline of τ_{out}. g^C_{out} and g^P_{in} are two constants which denote the number of firings the actor waits for the non-zero consumption/production of tokens on a path $w_{\text{in}\to\text{out}} \in \mathcal{W}$. For example, in Fig. 1, for the path from input actor τ_1 to output actor τ_4 via τ_3, $g^P_{in} = 2$ because τ_1 produces to edge e_2 zero tokens in its first two firings. Similarly, output actor τ_4 starts consuming non-zero tokens from edge e_4 after its first two firings, so $g^C_{out} = 2$. Given the parameters in Table II and using (4) to compute the latency, we find that the latency of the CSDF graph in Fig. 1 is 20 clock cycles.

C. System Model

The platform, we target in this paper, is a homogeneous multiprocessor platform which consists of identical processors. As we mentioned in Section I, scheduling a CSDF graph as a periodic task set (see Section III-B) is able to benefit from a large amount of proven real-time systems theories to easily compute the number of processors needed to execute the tasks. Any real-time scheduling algorithm can be used to schedule the periodic task set. However, in this paper, we only consider the Earliest Deadline First (EDF) scheduling algorithm. For the CDP model, [13] gives a sufficient test for a global scheduling in which tasks may need to migrate between processors. Here, we use this sufficient test to compute the required number of processor for global scheduling as follows:

$$M = \lceil \delta_{\text{sum}} \rceil \quad (5)$$

For partitioned scheduling, task migration is not required, and [14] gives a sufficient test for the partitioned EDF scheduling with First-Fit Decreasing allocation algorithm (EDF-FFD). In our work, this test is used to compute the number of processors for partitioned scheduling:

$$M \geq \begin{cases} \lceil \frac{\delta_{\text{sum}} - \delta_{\max}}{1 - \delta_{\max}} \rceil & \delta_{\max} \leq \frac{1}{2} \\ \lceil 2(\delta_{\text{sum}} - \delta_{\max}) \rceil & \delta_{\max} \geq \frac{1}{2} \end{cases} \quad (6)$$

where $\delta_{\max} = \max_{\tau_j \in V}\{\delta_j\}$.

For both, global and partitioned scheduling, we see that the total density δ_{sum} plays a crucial role in computing the minimum number of processors needed to schedule a task set.

D. Baseline Approach (BA)

We consider the global deadline scaling factor approach proposed in [5] a baseline approach. This baseline approach uses Binary Search to find the maximum df which makes the latency constraint met. Finding this maximum df reduces the required number of processors to schedule the CSDF actors. Later in Section VI, we compare our approach to the baseline approach. We use this approach for comparison because it considers the same application model, task scheduling and optimization problem as we do.

IV. Motivational Example

In this section, we take the CSDF graph in Fig. 1 as our motivational example to demonstrate the deficiency of the approach in [5], where deadlines of actors are computed by (1) with the global deadline scaling factor df found as explained in Section III-D. For the sake of simplicity, we consider a global scheduling and use (5) to compute the required number of processors. Given a latency constraint of 20 clock cycles, if we use the approach in [5], it finds that the global deadline scaling factor df should be set to 0 in order to meet the latency constraint. By using (1), we compute that $D_j = C_j$, and the parameters of the tasks are given in Table II. Using these parameters we obtain that the total density $\delta_{\text{sum}} = \frac{2}{2} + \frac{3}{3} + \frac{3}{3} + \frac{6}{6} = 4$. This means that 4 processors are needed to schedule the task set.

However, larger deadlines can be selected for some actors without violating the latency constraint, thereby reducing the total density δ_{sum}, which can decrease the number of processors. We select new deadlines $D_2 = 9$ and $D_3 = 12$ for actors τ_2 and τ_3, respectively, and recompute the start time of the tasks using Lemma 1 in Section III-B. We see that in this specific case shown in Table III, although we have changed two deadlines, the start times have not changed. By using (4) to compute the latency, we see that the latency of 20 clock cycles can be met with the new parameters, but the total density $\delta_{\text{sum}} = \frac{2}{2} + \frac{3}{9} + \frac{3}{12} + \frac{6}{6} = 2.58$ decreases. This means that three processors are sufficient to schedule the task set without violating the latency constraint of 20 clock cycles. We can see from the motivational example that the approach from [5] is not optimal in terms of the required number of processors.

V. Our Approach

As we show in Section IV, although the deadline selection approach in [5] is able to meet the latency constraint, it is not optimal in terms of the number of processors. Hence, selecting deadlines in a proper way is a problem that should be solved in order to minimize the number of processors while meeting the latency constraint. To select deadlines properly, we devise the solution approach presented in this section that formalizes and formulates the problem as a mathematical programming problem.

According to the relationship given in (4), the latency depends on the earliest start time and deadline of the output actor, and

the earliest start time of the input actor. The earliest start time S_j of any actor depends on two conditions: 1) at the earliest start time, there should be sufficient number of tokens on all input edges to enable the actor's firing and 2) once an actor fires for the first firing, the consequent firings of the actor should be possible to happen at times $t = S_j + mT_j$ for each $m \in \mathbb{N}^+$. The first condition is imposed by the firing rule [2] of the CSDF model which is a data-driven model, where a sufficient number of tokens is the requirement to trigger an actor firing. The second condition makes sure that the CSDF graph is schedulable as a periodic task set. Although Lemma 1 in Section III-B is able to find the earliest start time of an actor, it is impossible to use its equations into any mathematical programming problem. Hence, we present a new computation method to calculate the start time of actors in a CSDF, in which the start times of actors can be represented as linear items and can be integrated into a mathematical programming problem.

Lemma 2. *For an acyclic CSDF graph G, the earliest start time of an actor $\tau_j \in V$, denoted S_j, under a periodic schedule is given by*

$$S_j = \begin{cases} 0 & \text{if } \Omega(\tau_j) = \emptyset \\ \max_{\tau_i \in \Omega(\tau_j)}\{S_i + (S_{i\to j}^{min} - S_i^{min} - C_i) + D_i\} & \text{if } \Omega(\tau_j) \neq \emptyset \end{cases} \tag{7}$$

where $\Omega(\tau_j)$ is the set of predecessors of τ_j, S_i, C_i, and D_i are the earliest start time, WCET, and deadline of the predecessor actor τ_i, respectively. S_i^{min} is the earliest start time of τ_i given by (2) when $D_n = C_n, \forall \tau_n \in V$, and $S_{i\to j}^{min}$ is given by (3) when $D_n = C_n, \forall \tau_n \in V$.

Proof: Consider an arbitrary edge $e_u = (\tau_i, \tau_j) \in E$. τ_j starts after τ_i has started and fired a "certain" number of times. This number of firings is independent from the execution speed of the actors and depends only on the production and consumption rates of τ_i and τ_j on e_u. The production and consumption functions are given by:

$$\underset{[t_s,t)}{\mathsf{prd}}(\tau_i) = \sum_{k=0}^{\lfloor (t-t_s)/T_i \rfloor} (x_i^u(((k-1) \bmod P_i) + 1) \cdot u(t - kT_i - D_i))$$

$$\underset{[t_s,t]}{\mathsf{cns}}(\tau_j) = \sum_{k=0}^{\lceil (t-t_s)/T_j \rceil} \left(y_i^u(((k-1) \bmod P_j) + 1) \cdot u(t - kT_j)\right)$$

where $x_i^u(k)$ is the k^{th} element in production sequence of actor τ_i, $y_j^u(k)$ is the k^{th} element in consumption sequence of actor τ_j, P_i and P_j are the execution lengths of τ_i and τ_j, respectively, as defined in [2]. $u(t)$ is the unit step function. Suppose that $D_n = C_n, \forall \tau_n \in V$. The production and consumption curves of τ_i and τ_j are shown in Fig. 3. Interval Δ in Fig. 3 depends only on the production and consumption rates of τ_i and τ_j on e_u and can be calculated as:

$$\Delta = S_{i\to j}^{\min} - S_i^{\min} - C_i \tag{8}$$

Now, suppose that $D_n > C_n, \forall \tau_n \in V$. The production curve will move to the right for certain time units, and the new start time of τ_i is S_i. If the consumption curve does not move, the relation between the production and consumption given by Equation (3) will be violated, i.e. it will happen in some point in time that the cumulative consumption is greater than the cumulative production. This means that we have to move the consumption curve to the right by the same number of time units such that the new start time $S_{i\to j}$ is minimum and the relation is preserved. Because the production and consumption rates are unchanged, interval Δ will stay the same, and we can calculate it as follows:

$$\Delta = S_{i\to j} - S_i - D_i \tag{9}$$

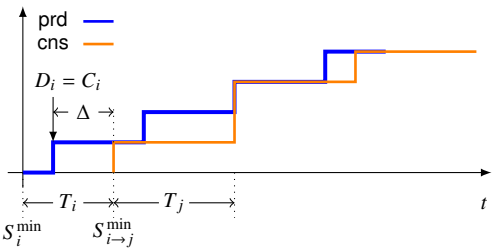


Fig. 3. Production and consumption curves on edge $e_u = (\tau_i, \tau_j)$

We can re-write (8) and (9) as:

$$S_{i\to j} = S_i + (S_{i\to j}^{\min} - S_i^{\min} - C_i) + D_i \tag{10}$$

■

Now, we can derive from Equation (7) the following set of linear inequality constraints, where the number of the linear inequality constraints is equal to the number of edges in the CSDF:

$$S_i + (S_{i\to j}^{\min} - S_i^{\min} - C_i) + D_i \leq S_j \quad \forall e_u \in E \tag{11}$$

Since computing the required number of processors depends on the total density δ_{sum} of the task set (see Section III-C), our objective is to minimize δ_{sum} in order to minimize the number of processors. Therefore, we formulate our density minimization (DM) problem as follows:

$$\text{Minimize} \quad \delta_{\text{sum}} = \sum_{\tau_n \in V} \frac{C_n}{D_n} \tag{12a}$$

$$\text{subject to:} \quad S_{\text{out}} + D_{\text{out}} - S_{\text{in}} \leq L + g_{\text{in}}^P T_{\text{in}} - g_{\text{out}}^C T_{\text{out}} \quad \forall w_{\text{in}\to\text{out}} \in W \tag{12b}$$

$$S_i + D_i - S_j \leq -(S_{i\to j}^{\min} - S_i^{\min} - C_i) \quad \forall e_u \in E \tag{12c}$$

$$-D_n \leq -C_n, D_n \leq T_n \quad \forall \tau_n \in V \tag{12d}$$

where (12a) is the objective function and D_n is an optimization variable. We want the objective function (12a) with $|V|$ optimization variables to be subject to a latency constraint L. Therefore, (12b) comes from (4). In addition, (12c) are the constraints given by (11), and (12d) bounds all optimization variables in the objective function by the worst-case execution time and period as explained in Section III-B. S_i and S_j (including S_{in}, S_{out}) are implicit variables which are not in the objective function (12a), but still need to be considered in the optimization procedure. L, $g_{\text{in}}^P T_{\text{in}}$, $g_{\text{out}}^C T_{\text{out}}$, $S_{i\to j}^{\min}$, $S_i^{\min}$, C_n, and T_n are constants.

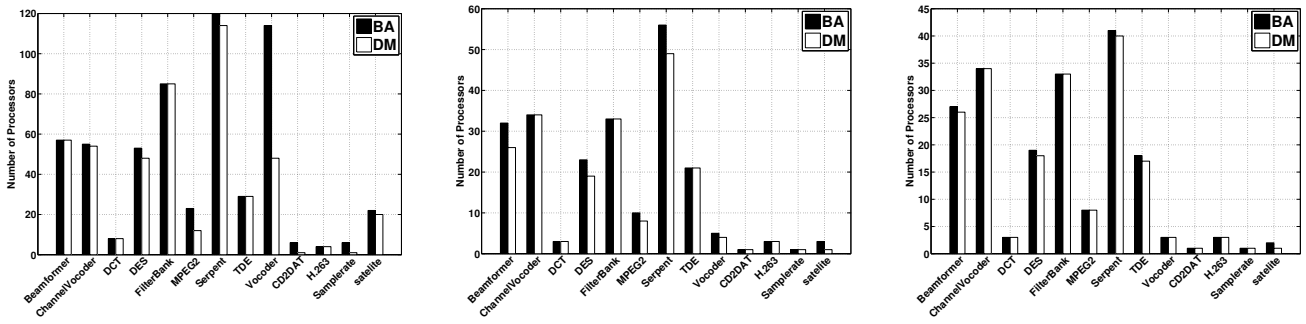
Theorem 1. *The DM problem (12) is an integer convex programming (ICP) problem.*

Proof: First, we prove that the DM problem is a convex programming problem if the values of D and S are continuous. In a convex programming problem, the objective function and the constraints both should be convex [15]. We first prove the convexity of the objective function.

$$f(x) = \frac{a}{x} \tag{13}$$

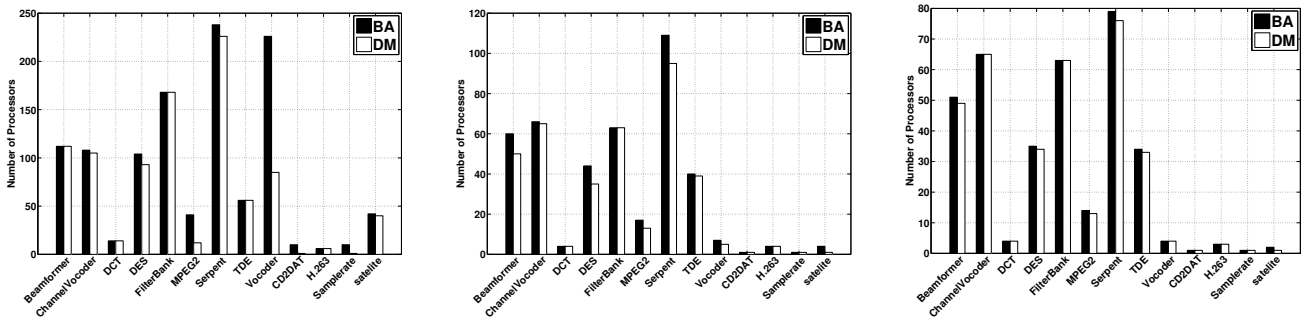
Function (13) has been proven to be convex for $x \in (0, \infty)$ and $a > 0$ [15]. Since D_n is always greater than 0, all $\delta_n(D_n) = \frac{C_n}{D_n}$ are convex functions, where D_n and C_n are the variable x and the constant a, respectively. Moreover, if f_1 and f_2 are both convex, so is their sum $f_1 + f_2$. Hence, $\delta_{\text{sum}} = \sum_{\tau_n \in V} \frac{C_n}{D_n}$ is a convex function.

A closed halfspace which is convex is a set of the form $\{\mathbf{x}|\mathbf{a}^T\mathbf{x} \leq \mathbf{b}\}$ [15], where $\mathbf{a} \neq \mathbf{0}$ and all entries in $\mathbf{x}$ are continuous. Since all constraints (12b), (12c), and (12d) are in the form of the closed halfspace, all constraints are convex. Hence, the DM



(a) $L = L_0$ (b) $L = L_1$ (c) $L = L_2$

Fig. 4. The number of processors for global scheduling with different latency constraints L



(a) $L = L_0$ (b) $L = L_1$ (c) $L = L_2$

Fig. 5. The number of processors for partitioned scheduling with different latency constraints L

problem (12) is a convex programming problem.

Given that all D and S in the DM problem (12) have to take only integer values in practice, the DM problem is an ICP problem. ■

In mathematical programming, a convex programming problem can be solved efficiently to find a global optimum. If the variables have to only take integer values, the problem becomes an integer convex programming problem. This integer problem is an NP-hard problem that can not be solved efficiently using only the conventional convex programming. Fortunately, the combination of the conventional convex programming [15] and some algorithms for solving mixed integer linear programming can be used to find a global optimal solution for ICP. In Section VI-B, the evaluation shows the efficiency of the existing CVX solver [6] to solve our DM problem.

VI. Evaluation

In this section, we evaluate our DM approach and compare it with the Baseline Approach (BA) proposed in [5] and briefly explained in Section III-D. The DM problem is solved by using mixed integer disciplined convex programming (MIDCP) in CVX [6]. All experiments are performed on an Intel i7 dual-core processor running at 2.7GHz with 4 GB RAM.

We selected 13 real-life streaming applications modeled as CSDF graphs from the StreamIt [16] benchmark suit. We use the application parameters as specified in [4]. The characteristics of these benchmarks are given in Table IV, including the number of actors ($|V|$), the number of edges ($|E|$), the maximum latency ($L_{\max}$) and the minimum latency ($L_{\min}$) in clock cycles. The maximum latency is the latency obtained by using the IDP model, i.e. $df = 1$, whereas the minimum latency is the latency obtained when $df = 0$. To demonstrate the effectiveness of our DM approach, the latency constraints of the graphs are varied during the experiments. We evaluate our DM approach in terms of the number of processors needed for each benchmark and compare it to BA for three latency constraints per benchmark, shown in Table V, while the achieved throughput of each benchmark is the same in both BA and DM approaches.

TABLE IV
Characteristics of benchmarks

Realistic Applications	$\|V\|$	$\|E\|$	$L_{\max}$	$L_{\min}$
Beamformer	57	70	60912	14692
ChannelVocoder	55	70	284000	106755
DCT	8	7	380928	121672
Data Encryption Standard (DES)	53	60	46080	15602
FilterBank	85	99	158368	34638
MPEG2	23	26	138240	49452
Serpent	120	128	370296	122108
Time Delay Equalization (TDE)	29	28	1071840	628151
Vocoder	114	147	291360	21554
CD2DAT	6	5	829	258
H.263	4	3	996697	369508
Samplerate	6	5	3792	1531
Satelite	22	26	11746	5484

TABLE V
Latency Constraints

Constraint	Latency
L_0	$L_{\min}$
L_1	$0.4(L_{\max} - L_{\min}) + L_{\min}$
L_2	$0.9(L_{\max} - L_{\min}) + L_{\min}$

A. The effectiveness of the DM approach

First, we evaluate the effectiveness of the DM approach in terms of the number of required processors. Fig. 4 shows the results under global scheduling. The number of processors needed to schedule a task set is computed using (5).

Fig. 4(a) shows the results with latency constraint $L_{\min}$. Under such stringent latency constraint, the intervals in which deadlines of actors may vary are limited. Our DM approach is still capable

TABLE VI
THE EXECUTION TIME OF THE DM APPROACH (IN SEC.)

Applications	L_0	L_1	L_2
Beamformer	0.05	0.18	0.11
ChannelVocoder	0.07	0.11	0.11
DCT	0.05	0.07	0.06
DES	5.9	14.7	0.23
FilterBank	0.1	0.32	0.17
MPEG2	12.9	0.13	0.07
Serpent	20.59	900.98	4751
TDE	0.13	0.1	0.24
Vocoder	1909	2256	0.31
CD2DAT	0.11	0.21	0.1
H.263	0.22	11.72	0.23
Samplerate	0.14	0.1	0.06
Satelite	0.14	0.08	0.24

of reducing the number of processors compared to the BA for 8 out of 13 benchmarks. The largest reduction is obtained for the Vocoder benchmark, with a reduction of 66 processors. The DCT, TDE and H.263 benchmarks have only a single data path in the corresponding CSDF graphs. The Beamformer and Filterbank benchmarks have symmetric graph structures, i.e., multiple paths consist of the same type of actors. Therefore, for all these benchmarks, small intervals in which deadlines of actors may vary restrict the possibility to reduce the total density, consequently the required number of processors is not reduced.

Fig. 4(b) presents the results for a relaxed latency constraint for each benchmark. There are 6 benchmarks for which our DM approach reduces the number of processors. The Beamformer and Filterbank benchmarks with symmetric structure, also benefit from the DM approach. That is because the redistribution of deadlines on a path makes it possible to decrease the densities δ_i of some tasks. For DCT, TDE, and H.263, although we can see a reduction in the total density δ_{sum} of the task set, the reduction is insufficient for decreasing the number of processors. The Samplerate and ChannelVocoder benchmarks keep unchanged on the number of processors because the total density is very close to the total utilization which is the lower bound of δ_{sum}. Fig. 4(c) shows the results for a very relaxed latency constraint, where only 5 benchmarks get reduction on the number of processors.

In Fig. 5, the number of processors required for partitioned scheduling to schedule a streaming application is shown. From (5) and (6) we can see that scheduling the task set in partitioned scheduling requires more processors than in global scheduling. For partitioned scheduling, the benchmarks can get larger reduction in the number of processors and the reduction can be obtained for more benchmarks compared to the global scheduling case in Fig. 4. As shown in Fig. 5(a), the Vocoder benchmark gets the larger reduction of 141 processors comparing to 66 processor reduction in global scheduling. Fig. 5(b) shows that the ChannelVocoder and TDE benchmarks which do not benefit from the DM approach for global scheduling get reduction in number of processors for partitioned scheduling. The rationale behind is that since in (6) the density is always multiplied with a number greater than 2, the reduction on the total density δ_{sum} is scaled up by at least 2 times. Hence, we can see more and larger reduction in the number of processors for the benchmarks when partitioned scheduling is used.

From the experimental results shown in Fig. 4 and Fig. 5, we can see that by applying our DM approach we obtain reduction in the number of processors in more than 52% of the conducted experiments. For the rest of the experiments, both approaches give the same number of processors.

B. The time complexity of solving the DM problem

Next, we evaluate the efficiency of our approach in terms of the execution time of CVX [6] for solving our DM problem. We set a maximum runtime of 4 hours for the solver, and all results are summarized in Table VI where the time unit is seconds. We can see that the runtime of CVX is not a function of the number of actors and edges in the corresponding application CSDF graph. For example, the FilterBank benchmark with more actors and edges than the DES benchmark just needs 0.32 second to find the optimal solution for L_1, while the DES benchmark needs 14.7 seconds. Additionally, even for the same benchmark with different latency constraints, the runtime for finding the optimal solution is fluctuating significantly. The execution times of our DM approach with the Vocoder benchmark for L_1 and L_2 is 2256 seconds and 0.31 second, respectively. According to Table VI most of the problems can be solved in a second. However, for the two very complex benchmarks, Serpent and Vocoder which have the largest number of constraints, the solver spent a long time to find the optimal solution. A method to speed up solving a very complex problem is to set an initial solution for optimization variables which can be obtained from BA, but unfortunately CVX does not support initialization of the optimization variables.

VII. CONCLUSION

In this paper, we optimize the number of processors needed to schedule a streaming application with a given latency constraint modeled as CSDF and scheduled as a periodic task set. First, a new computation method is proposed for computing the earliest start times of the task set in order to integrate them into a mathematical programming problem. Next, we formulate our DM problem as a mathematical programming problem and prove that it is an ICP problem. Our DM approach has been applied on 13 real-life streaming applications to show its effectiveness and efficiency. It reduces the number of processors in more than 52% of the conducted experiments, while in most cases the time needed for solving our DM problem is less than a second.

REFERENCES

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

[2] G. Bilsen *et al.*, "Cycle-static dataflow," *IEEE Trans. Signal Process.*, vol. 44, no. 2, pp. 397–408, 1996.

[3] A. H. Ghamarian *et al.*, "Latency minimization for synchronous data flow graphs," in *DSD*, 2007.

[4] M. Bamakhrama and T. Stefanov, "Hard-real-time scheduling of data-dependent tasks in embedded streaming applications," in *EMSOFT*, 2011.

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

[6] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 2.0 beta," http://cvxr.com/cvx, Sep. 2012.

[7] A. Davare *et al.*, "Period optimization for hard real-time distributed automotive systems," in *DAC*, 2007.

[8] S. Hong, T. Chantem, and X. S. Hu, "Meeting end-to-end deadlines through distributed local deadline assignments," in *RTSS*, 2011.

[9] P. Balbastre, I. Ripoll, and A. Crespo, "Optimal deadline assignment for periodic real-time tasks in dynamic priority systems," in *ECRTS*, 2006.

[10] T. Chantem *et al.*, "Period and deadline selection for schedulability in real-time systems," in *ECRTS*, 2008.

[11] H. Javaid *et al.*, "Optimal synthesis of latency and throughput constrained pipelined mpsocs targeting streaming applications," in *CODES+ISSS*, 2010.

[12] J. Cong *et al.*, "Synthesis algorithm for application-specific homogeneous processor networks," *IEEE Trans. VLSI Syst.*, vol. 17, no. 9, pp. 1318–1329, 2009.

[13] J. W. S. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[14] S. Baruah and N. Fisher, "The partitioned multiprocessor scheduling of sporadic task systems," in *RTSS*, 2005.

[15] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[16] W. Thies and S. Amarasinghe, "An empirical characterization of stream programs and its implications for language and compiler design," in *PACT*, 2010.