Managing Latency in Embedded Streaming Applications under Hard-Real-Time Scheduling

Mohamed A. Bamakhrama
mohamed@liacs.nl

Todor Stefanov
stefanov@liacs.nl

Leiden Institute of Advanced Computer Science
Leiden University, Leiden, The Netherlands

ABSTRACT

In this paper, we consider the problem of hard-real-time scheduling of embedded streaming applications, modeled using dataflow graphs, while minimizing the application latency. Recently, it has been shown that the actors in an acyclic Cyclo-Static Dataflow (CSDF) graph can be scheduled as a set of implicit-deadline periodic tasks. Such scheduling approach has been shown to yield the maximum achievable throughput for a large set of graphs, called matched I/O rates graphs. We show that scheduling the graph actors as implicit-deadline periodic tasks increases the latency significantly for a class of graphs called unbalanced graphs. To alleviate this problem, we propose a new task-set representation for the actors in which the actors are scheduled as a set of constrained-deadline periodic tasks. We prove that scheduling the actors as constrained-deadline periodic tasks delivers optimal throughput (i.e., rate) and latency for graphs with repetition vector equal to $\vec{1}$. Furthermore, we evaluate the constrained-deadline representation using a set of 19 real-life applications and show that it is capable of achieving the minimum achievable latency for more than 70% of the applications, and even if the application has a repetition vector not equal to $\vec{1}$. We show that choosing the task deadline involves a trade-off between the latency and the resources requirements. Finally, we propose a decision tree to assist the designer in choosing the appropriate real-time periodic task model for scheduling acyclic CSDF graphs.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: [Real-time and embedded systems]; D.4.7 [**Operating Systems**]: Organization and Design—*Real-time systems and embedded systems*

General Terms

Algorithms, Design, Performance

CODES+ISSS'12, October 7–12, 2012, Tampere, Finland.

Keywords

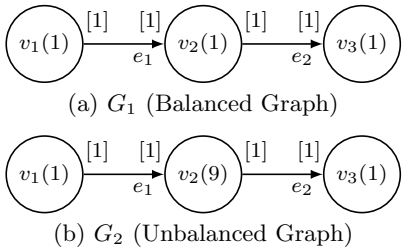
Real-time multiprocessor scheduling, streaming applications

1. INTRODUCTION

Embedded streaming systems have grown in complexity to the point that many of them require hard-real-time (HRT) execution of applications on multiprocessor platforms [11]. Additionally, some of these systems require support for dynamic (i.e., run-time) addition/removal of applications. To cope with such requirements, the system must use fast resource management and scheduling solutions which provide guaranteed services. To this end, algorithms from the hard-real-time multiprocessor scheduling theory represent an attractive solution approach. These algorithms provide many nice properties such as timing guarantees for each application, temporal isolation, and fast admission control of new incoming applications.

Recently, it has been shown that acyclic CSDF graphs with periodic input streams can be scheduled as asynchronous sets of implicit-deadline periodic tasks [3]. An implicit deadline periodic (IDP) task τ_i is defined by a 3-tuple $\tau_i = (S_i, C_i, T_i)$. The interpretation is as follows: τ_i is invoked (i.e., released) at time instants $t = S_i + kT_i$ and it has to execute for C_i time-units before time $t = S_i + (k+1)T_i$ for all $k \in \mathbb{N}_0$, where S_i is the start time of τ_i and T_i is the task period. Such scheduling approach is called strictly periodic scheduling (SPS). In [3], the authors considered the impact of strictly periodic scheduling (SPS) on the throughput and processor requirements of the applications. They showed that scheduling the graph actors as implicit-deadline periodic tasks yields the maximum achievable throughput (i.e., rate) for a large set of graphs, called matched I/O rates graphs. One advantage of using the IDP model is the low complexity of the schedulability test. However, the authors of [3] did not investigate the latency of the applications when scheduled using the IDP model. Generally speaking, latency is defined as the elapsed time between the arrival of a sample to an application and the output of the processed sample by the application. For many embedded streaming applications, latency is the primary performance metric rather than throughput.

In this paper, we show that the IDP model 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 (see Definition 6 in Section 4.1). In contrast, an unbalanced graph is the one where such product differs between actors.



(a) G_1 (Balanced Graph)

(b) G_2 (Unbalanced Graph)

Figure 1: Example of balanced and unbalanced graphs

Table 1: Throughput and latency metrics for G_1 and G_2 under different scheduling schemes

	STS			IDP			CDP		
	R	L	M	R	L	M	R	L	M
G_1	1/1	3	3	1/1	3	3	1/1	3	3
G_2	1/9	11	2	1/9	27	2	1/9	11	2

To illustrate the impact of the IDP model on the latency, we present the following motivational example.

1.1 Motivational Example

In Figure 1, we show two graphs: G_1 and G_2. A schedule consists of an infinitely repeated finite sequence of actor firings. A repetition vector represents the number of actor firings in such finite sequence. Both graphs in Figure 1 have a repetition vector equal to $\vec{1}$. The execution time of each actor is shown next to its name between round brackets (e.g., 9 for v_2 in G_2). G_1 (Figure 1(a)) is an example of a balanced graph since the product of actor execution time and repetition is the same for all actors – 1 in this case. On the other hand, G_2 (Figure 1(b)) is an example of unbalanced graphs.

Let R, L, and M denote the throughput (i.e., rate), latency, and minimum processor requirements of a graph G, respectively. It has been shown that the maximum achievable throughput and minimum achievable latency of a dataflow graph are the ones achieved under self-timed scheduling (STS) [9,17]. Thus, the maximum throughput and minimum latency for both graphs is the one shown under the STS column in Table 1. The throughput and latency resulting from scheduling the actors of both graphs as IDP tasks is shown under the IDP column in Table 1. We see that the IDP model yields the maximum throughput for both graphs, however, it pays a high price in terms of increased latency for the unbalanced graph (up to 2.5x for G_2). Instead, if the actors are to be scheduled as constrained-deadline periodic (CDP) tasks, then it is possible to achieve the optimal throughput and latency (i.e., maximum throughput and minimum latency) as shown under the CDP column in Table 1. In terms of resource usage, we see that the CDP model achieves optimal throughput and latency for G_2 using the same number of processors needed by IDP and STS – 2 processors. Thus, in this example, the CDP model is equally good to the STS in terms of throughput, latency, and resources usage.

1.2 Problem Statement

Given an application modeled as an acyclic CSDF graph:

1. derive its latency when its actors are scheduled as implicit-deadline periodic tasks, and
2. derive a task-set representation for the actors, based on the constrained-deadline model, that minimizes the latency compared to the implicit-deadline model.

1.3 Paper Contributions

We identify two classes of CSDF graphs called balanced and unbalanced graphs. For balanced graphs, we show that the latency of the graph when its actors are scheduled as IDP tasks is equal to the minimum achievable latency. For unbalanced graphs, we devise a task-set representation based on the constrained-deadline model to reduce the latency. We prove that the proposed CDP representation achieves optimal throughput and latency for graphs with repetition vector equal to $\vec{1}$. Furthermore, we evaluate the proposed CDP representation using a set of 19 real-life applications and show that it is capable of achieving the minimum achievable latency for 70% of the applications, and even if the application has a repetition vector not equal to $\vec{1}$. Finally, we propose a decision tree to assist the designer in choosing the appropriate real-time periodic task model for scheduling acyclic CSDF graphs.

The remainder of this paper is organized as follows: Section 2 gives an overview of the related work. Section 3 introduces the background material needed for understanding the contributions of this paper. Section 4 presents the proposed task-set representation. Section 5 presents the evaluation of the proposed task-set representation. Finally, Section 6 ends the paper with conclusions.

2. RELATED WORK

In [17], it is shown that the minimum achievable latency of a Homogeneous Synchronous Dataflow (HSDF, [12]) graph is the one achieved under self-timed scheduling. This minimum latency is equal to the sum of execution times along the critical path in the graph. In [9], Ghamarian et al. extended this result to general Synchronous Dataflow (SDF, [12]) graphs and provided an algorithm to determine the minimum achievable latency. In contrast, our work analyzes the latency of CSDF graphs under strictly periodic scheduling considering different models of periodic tasks, i.e., IDP and CDP. Moreover, our analysis considers CSDF graphs which are more expressive model than SDF/HSDF.

In [10], Goddard studied applying real-time scheduling to dataflow programs modeled using the processing graphs method (PGM). He used a task model called rate-based execution (RBE) in which a real-time task τ_i is characterized by a 4-tuple $\tau_i = (x_i, y_i, d_i, c_i)$. The interpretation is as follows: τ_i executes x_i times in time period y_i with a relative deadline d_i per job release and c_i execution time per job release. Goddard analyzed the latency of PGM graphs scheduled using the RBE model and identified two sources of latency: 1) *inherit latency* which is the latency due to different production and consumption rates in the PGM graph assuming zero execution time for the actors, and 2) *imposed latency* which is the latency when non-zero execution times are imposed on the actors. In contrast, our approach uses CSDF graphs which are more expressive model than PGM graphs in that PGM supports only a *constant* production/consumption rate on edges (same as SDF), whereas CSDF supports *varying* (but predefined) production/consumption rates. As a result, the analysis technique in [10] is not applicable to CSDF graphs.

In [14], Moreira and Bekooij analyzed the latency of SDF graphs under self-timed scheduling and provided bounds on

the maximum latency for applications with periodic, sporadic, and bursty sources, together with a technique to check latency requirements. In [15], they proposed algorithms to find combined Time-Division-Multiplexing (TDM)/static order schedules that guarantee a requested minimum throughput and maximum latency, while minimizing the usage of processing resources. Our approach differs from [14, 15] in: 1) we consider the periodic task models which allow applying a variety of proven hard-real-time scheduling algorithms for multiprocessors, and 2) we use the CSDF model which is more expressive than SDF graphs.

3. BACKGROUND

In this section, we provide an overview of the considered dataflow and real-time models. This overview is necessary for understanding the contributions in Section 4.

3.1 Cyclo-Static Dataflow (CSDF)

In [6], the CSDF model is defined as a directed graph $G = \langle V, E \rangle$, where V is a set of actors and $E \subseteq V \times V$ is a set of communication channels. Actors represent functions that transform incoming data streams into outgoing data streams. The communication channels carry streams of data, and an atomic data object is called a *token*. A channel $e_u \in E$ is a first-in, first-out (FIFO) queue with unbounded capacity defined by a tuple $e_u = (v_i, v_j)$. The tuple means that e_u is directed from v_i (called source) to v_j (called destination). The number of actors in a graph G is denoted by $N = |V|$. An actor receiving an input stream of the application is called *input actor*, and an actor producing an output stream of the application is called *output actor*. A path $w_{a \rightsquigarrow z}$ between actors v_a and v_z is an ordered sequence of channels defined as $w_{a \rightsquigarrow z} = \{(v_a, v_b), (v_b, v_c), \cdots, (v_y, v_z)\}$. A path $w_{i \rightsquigarrow j}$ is called *output path* if v_i is an input actor and v_j is an output actor. $\mathcal{W}$ denotes the set of all output paths in G. In this work, we consider only acyclic CSDF graphs. An acyclic graph G has a number of levels, denoted by $\mathcal{L}$, which is given by Algorithm 1. An actor v_i that belongs to set A_j in Algorithm 1 has a level index $\sigma_i = j$. Each actor $v_i \in V$ is associated with two sets of channels and two sets of actors. The sets of channels are the input channels set, denoted by $\mathsf{inp}(v_i)$, which consists of all the input channels to v_i, and the output channels set, denoted by $\mathsf{out}(v_i)$, which consists of all the output channels from v_i. The sets of actors are the successors set, denoted by $\mathsf{succ}(v_i)$, and the predecessors set, denoted by $\mathsf{prec}(v_i)$. They are given by:

$$\mathsf{succ}(v_i) = \{v_j \in V : \exists e_u = (v_i, v_j) \in E\} \quad (1)$$

$$\mathsf{prec}(v_i) = \{v_j \in V : \exists e_u = (v_j, v_i) \in E\} \quad (2)$$

Algorithm 1 LEVELS(G)

Require: Acyclic CSDF graph $G = \langle V, E \rangle$
1: $i \leftarrow 1$
2: **while** $V \neq \emptyset$ **do**
3: $A_i \leftarrow \{v_j \in V : \mathsf{prec}(v_j) = \emptyset\}$
4: $Z_i \leftarrow \{e_u \in E : \exists v_k \in A_i$ that is the source of $e_u\}$
5: $V \leftarrow V \setminus A_i$
6: $E \leftarrow E \setminus Z_i$
7: $i \leftarrow i + 1$
8: **end while**
9: $\mathcal{L} \leftarrow i - 1$
10: **return** $\mathcal{L}$ disjoint sets $A_1, A_2, ..., A_{\mathcal{L}}$, where $\bigcup_{i=1}^{\mathcal{L}} A_i = V$

Every actor $v_j \in V$ has an execution sequence $[f_j(1), f_j(2), \cdots, f_j(P_j)]$ of length P_j. The interpretation of this sequence is: The nth time that actor v_j is fired, it executes the code of function $f_j(((n-1) \bmod P_j) + 1)$. Similarly, production and consumption of tokens are also sequences of length P_j in CSDF. The token production of actor v_j on channel e_u is represented as a sequence of constant integers $[x_j^u(1), x_j^u(2), \cdots, x_j^u(P_j)]$. The nth time that actor v_j is fired, it produces $x_j^u(((n-1) \bmod P_j) + 1)$ tokens on channel e_u. The consumption of actor v_k is completely analogous; the token consumption of actor v_k from a channel e_u is represented as a sequence $[y_k^u(1), y_k^u(2), \cdots, y_k^u(P_j)]$. The firing rule of a CSDF actor v_k is evaluated as "true" for its nth firing iff all its input channels contain at least $y_k^u(((n-1) \bmod P_j)+1)$ tokens. The total number of tokens produced by actor v_j on channel e_u during the first n invocations, denoted by $X_j^u(n)$, is given by $X_j^u(n) = \sum_{l=1}^{n} x_j^u(l)$. Similarly, the total number of tokens consumed by actor v_k from channel e_u during the first n invocations, denoted by $Y_k^u(n)$, is given by $Y_k^u(n) = \sum_{l=1}^{n} y_k^u(l)$

An important property of the CSDF model is its decidability, which is the ability to derive at compile-time a schedule for the actors. This is formulated in the following definitions and results from [6].

Definition 1. Given a connected CSDF graph G, a *valid static schedule* for G is a *finite sequence of actors invocations* that can be repeated infinitely on the incoming sample stream while the amount of data in the buffers remains bounded. A vector $\vec{q} = [q_1, q_2, \cdots, q_N]^T$, where $q_j > 0$, is a *repetition vector* of G if each q_j represents the number of invocations of an actor v_j in a valid static schedule for G. The repetition vector of G in which all the elements are relatively prime[1] is called the *basic repetition vector* of G, denoted by $\vec{q}$. G is *consistent* if there exists a repetition vector. If a deadlock-free schedule can be found, G is said to be *live*. Both consistency and liveness are required for the existence of a valid static schedule.

THEOREM 1 (FROM [6]). *In a CSDF graph G, a repetition vector $\vec{q} = [q_1, q_2, \cdots, q_N]^T$ is given by*

$$\vec{q} = \mathbf{P} \cdot \vec{r}, \qquad with \qquad P_{jk} = \begin{cases} P_j & \text{if } j = k \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

where $\vec{r} = [r_1, r_2, \cdots, r_N]^T$ is a positive integer solution of the balance equation

$$\mathbf{\Gamma} \cdot \vec{r} = \vec{0} \quad (4)$$

and where the topology matrix $\mathbf{\Gamma} \in \mathbb{Z}^{|E| \times |V|}$ *is defined by*

$$\Gamma_{uj} = \begin{cases} X_j^u(P_j) & \text{if actor } v_j \text{ produces on ch. } e_u \\ -Y_j^u(P_j) & \text{if actor } v_j \text{ consumes from ch. } e_u \\ 0 & \text{Otherwise.} \end{cases} \quad (5)$$

Definition 2. For a consistent and live CSDF graph G, an *actor iteration* is the invocation of an actor $v_i \in V$ for q_i times, and a *graph iteration* is the invocation of *every* actor $v_i \in V$ for q_i times, where $q_i \in \vec{q}$.

Example 1. Figure 2 shows an example of a CSDF graph. The graph has a number of levels $\mathcal{L} = 3$. It also has three

[1] i.e., $\gcd\{q_1, q_2, \cdots, q_N\} = 1$.

output paths given by $\mathcal{W}$= $\{w_1 = \{(v_1, v_2)\ ,\ (v_2, v_4)\}\ ,\ w_2 = \{(v_1, v_3)\ ,\ (v_3, v_4)\}$,$w_3 = \{(v_1, v_4)\}\}$. The repetition vector is $\vec{q} = [3, 2, 1, 3]^T$.

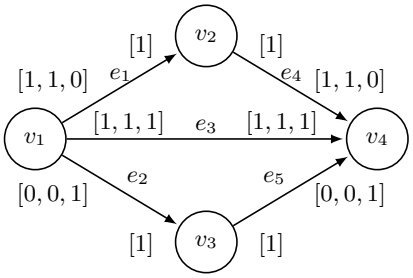


Figure 2: Example of a CSDF graph

3.2 System Model and Scheduling Algorithms

In this section, we introduce the system model and the related schedulability results.

3.2.1 System Model

A system Π consists of a set $\pi = \{\pi_1, \pi_2, \cdots, \pi_m\}$ of m homogeneous processors. The processors execute a task set $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$ of n periodic tasks, and a task may be preempted at any time. A periodic task $\tau_i \in \tau$ is defined by a 4-tuple $\tau_i = (S_i, C_i, T_i, D_i)$, where $S_i \geq 0$ is the start time of τ_i, $C_i > 0$ is the worst-case execution time of τ_i, $T_i \geq C_i$ is the task period, and D_i, where $C_i \leq D_i \leq T_i$, is the relative deadline of τ_i. A periodic task τ_i is invoked (i.e., *released*) at time instants $t = S_i + kT_i$ for all $k \in \mathbb{N}_0$. Upon invocation, τ_i executes for C_i time-units. The relative deadline D_i is interpreted as follows: τ_i has to finish executing its kth invocation before time $t = S_i + kT_i + D_i$ for all $k \in \mathbb{N}_0$. If $D_i = T_i$, then τ_i is said to have *implicit-deadline*. If $D_i < T_i$, then τ_i is said to have *constrained-deadline*. If all the tasks in a task-set τ have the same start time, then τ is said to be *synchronous*. Otherwise, τ is said to be *asynchronous*.

The utilization of a task τ_i is $U_i = C_i/T_i$. For a task set τ, the total utilization of τ is $U_{\text{sum}} = \sum_{\tau_i \in \tau} U_i$ and the maximum utilization factor of τ is $U_{\max} = \max_{\tau_i \in \tau} U_i$. Similarly, the density of a task τ_i is $\delta_i = C_i / \min(D_i, T_i)$, the total density of τ is $\delta_{\text{sum}} = \sum_{\tau_i \in \tau} \delta_i$, and the maximum density of τ is $\delta_{\max} = \max_{\tau_i \in \tau} \delta_i$. Note that the density is equivalent to the utilization for implicit-deadline tasks.

Given a system Π and a task set τ, a *valid schedule* is one that allocates a processor to a task $\tau_i \in \tau$ for exactly C_i time-units in the interval $[S_i + kT_i, S_i + kT_i + D_i)$ for all $k \in \mathbb{N}_0$ with the restriction that a task may not execute on more than one processor at the same time.

Based on whether a task can migrate between processors upon preemption, scheduling algorithms are classified into:

- *Partitioned*: Each task is allocated to a processor and no migration is permitted
- *Global*: Migration is permitted for all tasks
- *Hybrid*: Hybrid algorithms mix partitioned and global approaches and they can be further classified to:
  1. Semi-partitioned: Most tasks are allocated to processors and few tasks are allowed to migrate
  2. Clustered: Processors are grouped into clusters and the tasks that are allocated to one cluster are scheduled by a global scheduler

Since we will use both of the implicit and constrained deadline periodic task models, we present below the schedulability analysis results for both models on multiprocessor systems.

3.2.2 Schedulability Analysis

A necessary and sufficient (i.e., exact) condition for an asynchronous set of implicit-deadline periodic tasks τ to be scheduled on m processors to meet all the deadlines (i.e., τ is *feasible*) is:

$$U_{\text{sum}} \leq m \tag{6}$$

A scheduling algorithm $\mathcal{A}$ is said to be *optimal* iff it can schedule any feasible task set τ on Π. Several global and hybrid algorithms were proven optimal for scheduling asynchronous sets of implicit-deadline periodic tasks [7]. In contrast, partitioned scheduling is known to be non-optimal.

For constrained-deadline periodic tasks, we need first to introduce some concepts that are essential for its schedulability analysis.

Definition 3. The *processor demand* of a task τ_i over the time interval $[t_1, t_2]$, denoted by $\text{DBF}(\tau_i, t_1, t_2)$, is the total computation time of all the instances of τ_i having activation time and deadline within $[t_1, t_2]$.

According to [5], $\text{DBF}(\tau_i, t_1, t_2)$ is given by:

$$\text{DBF}(\tau_i, t_1, t_2) = \zeta(\tau_i, t_1, t_2) C_i \tag{7}$$

where $\zeta(\tau_i, t_1, t_2)$ is the total number of τ_i instances that are activated in the interval $[t_1, t_2]$ and have a deadline within the interval $[t_1, t_2]$. The authors in [5] showed that $\zeta(\tau_i, t_1, t_2)$ is given by:

$$\zeta(\tau_i, t_1, t_2) = \max\{0, \left\lfloor \tfrac{t_2 - S_i - D_i}{T_i} \right\rfloor - \max\{0, \left\lceil \tfrac{t_1 - S_i}{T_i} \right\rceil\} + 1\} \tag{8}$$

For the whole task-set τ, the processor demand of τ in the interval $[t_1, t_2]$, denoted by $\text{DBF}(\tau, t_1, t_2)$, is given by:

$$\text{DBF}(\tau, t_1, t_2) = \sum_{\tau_i \in \tau} \text{DBF}(\tau_i, t_1, t_2) \tag{9}$$

For uniprocessor systems, an exact schedulability test for constrained-deadline periodic task-sets is stated in the following lemma:

LEMMA 1 (FROM [5]). *A periodic task-set τ is feasible on one processor iff $U_{sum} \leq 1$ and $\text{DBF}(\tau, t_1, t_2) \leq (t_2 - t_1)$ for all $0 \leq t_1 < t_2 < s + 2p$, where $s = \max\{S_1, \cdots, S_n\}$ and $p = \mathsf{lcm}\{T_1, \cdots, T_n\}$ (lcm denotes the least-common-multiple operator).*

For multiprocessor systems, a sufficient schedulability test is obtained by replacing U_{sum} in Equation 6 with δ_{sum} [2]. Hence, a task-set τ is feasible on m processors if:

$$\delta_{\text{sum}} \leq m \tag{10}$$

The exact test in Lemma 1 is known to be co-NP-hard in the strong sense [5], while the sufficient test in Equation 10 is known to be pessimistic. As a result, several tests with less complexity than the exact test and better accuracy than the test in Equation 10 have been proposed in the literature (e.g., [1, 4, 8, 13, 16]).

4. MANAGING LATENCY

In this section, we first introduce the assumptions and definitions needed for our analysis of the latency. Then, we analyze the latency of applications, modeled as CSDF graphs, when their actors are scheduled as implicit-deadline periodic (IDP) tasks. Finally, we propose a new latency minimized task-set representation for the applications based on the constrained-deadline periodic (CDP) model.

4.1 Assumptions and Definitions

In the remainder of this paper, a graph G refers to an acyclic consistent CSDF graph. We base our analysis on the following assumptions:

A1 A graph G has a set $I = \{I_1, I_2, \cdots, I_\mathcal{K}\}$ of $\mathcal{K}$ periodic input streams connected to the input actors of G. These streams satisfy the following:

1. The set of input streams connected to an actor v_i is disjoint from all the other sets of input streams connected to other actors. That is, there are no two actors sharing the same input stream.
2. The first samples of all the streams arrive prior to or at the same time when the actors of G start executing.
3. Each input stream I_j is characterized by a constant inter-arrival time of the samples (also called period). This period is assumed to be equal to the period of the input actor which receives I_j. This assumption indicates that the period for input streams can be controlled by the designer to match the periods of the actors.

A2 An actor v_i consumes its input data immediately when it starts its firing and produces its output data just before it finishes its firing.

Now, we introduce the following definitions and results.

Definition 4. For a graph G, the *execution time vector* of G, denoted by $\vec{\mu} \in \mathbb{N}^N$, is a vector such that $\mu_i \in \vec{\mu}$ is the worst-case execution time (WCET) of actor $v_i \in V$.

We assume that the worst-case execution time of an actor includes the worst-case times needed for reading and writing the tokens. That is, the communication cost is included in μ_i.

Let $\eta = \max_{v_i \in V}(\mu_i q_i)$ and $Q = \mathsf{lcm}\{q_1, q_2, \cdots, q_N\}$. Now, we give the following definition:

Definition 5. A graph G is said to be *matched input and output (I/O) rates graph* iff:

$$\eta \bmod Q = 0 \tag{11}$$

If Equation 11 does not hold, then G is said to be *mismatched input/output (I/O) rates graph.*

If $\eta \bmod Q = 0$, then there exists at least a single actor in the graph which is fully utilizing the processor on which it runs. This allows the graph to achieve optimal throughput. On the other hand, if $\eta \bmod Q \neq 0$, then there exist idle durations in the period of each actor which results in sub-optimal throughput.

Definition 6. A graph G is called *balanced* iff:

$$q_1\mu_1 = q_2\mu_2 = \cdots = q_N\mu_N \tag{12}$$

where $q_i \in \vec{q}$ is the repetition of actor $v_i \in V$. If Equation 12 does not hold, then the graph is called *unbalanced.*

Definition 7. Let $w_{a \rightsquigarrow z} = \{(v_a, v_b), \cdots, (v_y, v_z)\}$ be an output path in a graph G. The *latency* of $w_{a \rightsquigarrow z}$ under periodic input streams, denoted by $L(w_{a \rightsquigarrow z})$, is the elapsed time between the start of the first firing of v_a which produces data to (v_a, v_b) and the finish of the first firing of v_z which consumes data from (v_y, v_z).

Consequently, we define the maximum latency of G as follows:

Definition 8. For a graph G, the *maximum latency* of G under periodic input streams, denoted by $L(G)$, is given by:

$$L(G) = \max_{w_{i \rightsquigarrow j} \in \mathcal{W}} L(w_{i \rightsquigarrow j}) \tag{13}$$

Definition 9. A *self-timed schedule* (STS) is one where all the actors are fired as soon as their input data are available.

A self-timed schedule does not impose any extra latency (e.g., by the scheduler) on the actors. This leads us to the following result proven in [17] for HSDF graphs and extended in [9] to SDF graphs.

THEOREM 2 (FROM [9, 17]). *For a graph G, the minimum achievable latency and the maximum achievable throughput are obtained when the actors of G are scheduled using self-timed scheduling policy.*

Theorem 2 applies to CSDF graphs since any CSDF graph can be converted to an equivalent SDF graph.

The authors of [3] proved that it is possible to schedule a graph G actors as implicit-deadline periodic tasks using periods given by the following definition:

Definition 10. For a graph G, a *period vector* $\vec{\lambda}$, where $\vec{\lambda} \in \mathbb{N}^N$, represents the periods, measured in time-units, of the actors in G. $\lambda_j \in \vec{\lambda}$ is the period of actor $v_j \in V$. $\vec{\lambda}$ is given by the solution to both

$$q_1\lambda_1 = q_2\lambda_2 = \cdots = q_{N-1}\lambda_{N-1} = q_N\lambda_N \tag{14}$$

and

$$\vec{\lambda} - \vec{\mu} \geq \vec{0}, \tag{15}$$

where $q_j \in \vec{q}$ (The basic repetition vector of G).

The minimum solution to both Equations 14 and 15 is given by the following lemma:

LEMMA 2 (FROM [3]). *The minimum period vector of G, denoted by $\vec{\lambda}^{min} \in \mathbb{N}^N$, is given by*

$$\lambda_i^{min} = \frac{Q}{q_i}\left\lceil \frac{\eta}{Q} \right\rceil \text{ for } v_i \in V, \tag{16}$$

where $q_i \in \vec{q}$ is the repetition of actor v_i.

Definition 10 implies an equal iteration period for all the actors. This common iteration period is formulated in the following definition:

Definition 11. For G, the *iteration period* under strictly periodic scheduling, denoted by α, is given by:

$$\alpha = q_i\lambda_i \text{ for any } v_i \in V \tag{17}$$

where λ_i is the period of v_i given by Definition 10.

In order to facilitate computing the latency, we introduce the following definitions:

Definition 12. The *cumulative production function* of an actor v_i producing into a channel e_u during the interval $[t_s, t_e)$, denoted by $\mathsf{prd}_{[t_s,t_e)}(v_i, e_u)$, is the sum of the number of tokens produced by v_i into e_u during the interval $[t_s, t_e)$.

Similarly, we define the cumulative consumption function as follows:

Definition 13. The *cumulative consumption function* of an actor v_i consuming from a channel e_u over the interval $[t_s, t_e]$, denoted by $\mathsf{cns}_{[t_s,t_e]}(v_i, e_u)$, is the sum of the number of tokens consumed by v_i from e_u during the interval $[t_s, t_e]$.

Now, we proceed to the latency analysis of CSDF when it is scheduled using the IDP model.

4.2 Latency Analysis under the IDP Model

Let ϕ_i be the earliest start time of an actor $v_i \in V$. Then, according to Definitions 7 and 8, the graph latency $L(G)$ is given by:

$$L(G) = \max_{w_{i\rightsquigarrow j} \in \mathcal{W}} (\phi_j + \hat{y}_j^u \lambda_j + D_j - (\phi_i + \hat{x}_i^r \lambda_i)) \tag{18}$$

where ϕ_j and ϕ_i are the earliest start times of the output actor v_j and the input actor v_i, respectively, λ_j and λ_i are the periods of v_j and v_i, D_j is the deadline of v_j, and $\hat{y}_j^u$ and $\hat{x}_i^r$ are two constants, such that for an output path $w_{i\rightsquigarrow j}$ in which e_r is the first channel and e_u is the last channel, $\hat{x}_i^r$ and $\hat{y}_j^u$ are given by:

$$\hat{x}_i^r = \min\{k \in \mathbb{N} : x_i^r(k) > 0\} - 1 \tag{19}$$

$$\hat{y}_j^u = \min\{k \in \mathbb{N} : y_j^u(k) > 0\} - 1 \tag{20}$$

Under the IDP model, Equation 18 becomes:

$$L(G) = \max_{w_{i\rightsquigarrow j} \in \mathcal{W}} (\phi_j + (\hat{y}_j^u + 1)\lambda_j - (\phi_i + \hat{x}_i^r \lambda_i)) \tag{21}$$

The earliest start time of actors under the IDP model is given by the following lemma.

LEMMA 3. *For a graph G, the earliest start time of an actor $v_j \in V$, denoted by ϕ_j, under a strictly periodic schedule is given by*

$$\phi_j = \begin{cases} 0 & \text{if } \mathsf{prec}(v_j) = \emptyset \\ \max\limits_{v_i \in \mathsf{prec}(v_j)} (\phi_{i\to j}) & \text{if } \mathsf{prec}(v_j) \neq \emptyset \end{cases} \tag{22}$$

where

$$\phi_{i\to j} = \min_{t \in [0, \phi_i + \alpha]} \{t : \mathop{\mathsf{prd}}_{[\phi_i, \max(\phi_i, t)+k)} (v_i, e_u) \geq \mathop{\mathsf{cns}}_{[t, \max(\phi_i, t)+k]} (v_j, e_u) \quad \forall k = 0, 1, \cdots, \alpha\} \tag{23}$$

In case of implicit-deadline periodic tasks, $\mathsf{prd}_{[t_s,t_e)}(v_i, e_u)$ is given by:

$$\mathop{\mathsf{prd}}_{[t_s,t_e)} (v_i, e_u) = \begin{cases} X_i^u\left(\left\lfloor \frac{t_e - t_s}{\lambda_i} \right\rfloor\right) & \text{if } (t_e - t_s) \geq \lambda_i \\ 0 & \text{if } (t_e - t_s) < \lambda_i \end{cases} \tag{24}$$

Similar to Equation 24, $\mathsf{cns}_{[t_s,t_e]}(v_i, e_u)$ is given by:

$$\mathop{\mathsf{cns}}_{[t_s,t_e]} (v_i, e_u) = \begin{cases} 0 & \text{if } t_e < t_s \\ Y_i^u(\lceil \frac{t_e - t_s}{\lambda_i} \rceil + 1) & \text{if } (t_e - t_s) \bmod \lambda_i = 0 \\ Y_i^u(\lceil \frac{t_e - t_s}{\lambda_i} \rceil) & \text{if } (t_e - t_s) \bmod \lambda_i \neq 0 \end{cases} \tag{25}$$

Using Equation 21, it is possible to compute the latency under the IDP model for any acyclic CSDF graph.

Example 2. For the graph shown in Figure 2, recall that the repetition vector is $\vec{q} = [3, 2, 1, 3]^T$. Now, assume that the execution time vector is $\vec{\mu} = [5, 8, 24, 4]^T$. It follows that $\eta = \max_{v_i \in V}(\mu_i q_i) = 24$, $Q = \mathsf{lcm}\{q_1, q_2, q_3, q_4\} = 6$, and $\alpha = 24$. Based on that, we compute the minimum period vector $\vec{\lambda}^{\min} = [8, 12, 24, 8]^T$. To compute the earliest start time, we first find the cumulative production/consumption functions. For example, for v_1 on e_1, $\mathsf{prd}_{[t_s,t_e)}(v_1, e_1)$ is:

$$\mathop{\mathsf{prd}}_{[t_s,t_e)} (v_1, e_1) = \begin{cases} X_1^1\left(\left\lfloor \frac{t_e - t_s}{8} \right\rfloor\right) & \text{if } (t_e - t_s) \geq 8 \\ 0 & \text{if } (t_e - t_s) < 8 \end{cases}$$

If we set $[t_s, t_e)$, for example, to $[0, 16)$, then $\mathsf{prd}_{[0,16)}(v_1, e_1) = 2$. Computing ϕ_2 results in:

$$\phi_2 = \max_{v_i \in \mathsf{prec}(v_2)} (\phi_{i\to 2}) = \phi_{1\to 2}$$

Using (23) to compute $\phi_{1\to 2}$ gives:

$$\phi_{1\to 2} = \min_{t \in [0,24]} \{t : \mathop{\mathsf{prd}}_{[0, \max(0,t)+k)} (v_1, e_1) \geq \mathop{\mathsf{cns}}_{[t, \max(0,t)+k]} (v_2, e_1) \quad \forall k = 0, ..., 24\} = 8$$

Similarly, we compute the earliest start time under the IDP model for each actor which results in $\vec{\phi} = [0, 8, 24, 32]^T$. Using (21), we can compute the latency of the individual output paths as follows:

$$L(w_1) = 32 + (0 + 1) \times 8 - (0 + 0 \times 8) = 40$$
$$L(w_2) = 32 + (2 + 1) \times 8 - (0 + 2 \times 8) = 40$$
$$L(w_3) = 32 + (0 + 1) \times 8 - (0 + 0 \times 8) = 40$$

Therefore, the graph maximum latency is given by

$$L_{\mathsf{IDP}}(G) = \max_{w_{i\rightsquigarrow j} \in \mathcal{W}} L(w_{i\rightsquigarrow j}) = 40$$

Now, we show that for two sub-classes of CSDF graphs, we can derive a simpler expression for the latency. These two classes are: 1) balanced graphs, and 2) graphs where $\vec{q} = \vec{1}$.

THEOREM 3. *The minimum achievable latency of a balanced graph G when its actors are scheduled as implicit-deadline periodic tasks is equal to its minimum achievable latency under self-timed scheduling.*

PROOF. By Definitions 6 and 10, a balanced graph has a period vector in which each actor has a period equal to its execution time (i.e., $\lambda_i = \mu_i$). Therefore, each actor requires execution on a dedicated processor (since $U_i = \mu_i/\lambda_i = 1$). In this case, the actor utilizes fully the processor on which it executes. Hence, there is no idle time or latency imposed by the scheduler and the actors execute in a way that emulates self-timed execution. As a result, the graph has the same latency as if it is executed in a self-timed way. □

Theorem 3 implies that a balanced graph scheduled to achieve the minimum achievable latency requires a number of processors $M = N$, where N is the number of actors in the graph. In such a case, the system has "one-to-one" mapping (i.e., one task per processor). Hence, the type of scheduler is irrelevant since each task requires a dedicated processor, which is fully utilized, in order to achieve the minimum achievable latency.

THEOREM 4. *The minimum achievable latency of a graph G with basic repetition vector $\vec{q} = \vec{1}$, when its actors are scheduled as implicit-deadline periodic tasks, is $L_{IDP}(G) = \mathcal{L} \max_{v_i \in V} \mu_i$.*

PROOF. Based on Lemma 2, a CSDF graph with basic repetition vector $\vec{q} = \vec{1}$ will have a period vector in which all the periods are the same and equal to $\max_{v_i \in V} \mu_i$. Thus, the latency of the graph is the sum of the periods along the longest output path. By Algorithm 1, the longest output path has $\mathcal{L}$ actors in it. Thus, the graph latency is $L(G) = \mathcal{L} \max_{v_i \in V} \mu_i$. □

Theorem 4 is illustrated in the motivational example in Section 1.1 for G_2 when scheduled using the IDP model. $\mathcal{L}$ for G_2 is equal to 3 and $\max_{v_i \in V}(\mu_i q_i) = 9$. Thus, $L_{\mathsf{IDP}}(G_2) = 3 \times 9 = 27$.

4.3 Latency Minimized Scheduling using the CDP model

For an actor v_j, $\phi_{i \to j}$ depends on the time at which the predecessors have generated enough data to ensure periodic execution of v_j once it starts. Under the IDP model, such time is always constrained to be at the end of a predecessor actor period. Now, let v_m be the predecessor actor that has the maximum $\phi_{i \to j}$. The idea is to set the deadline of v_m to be less than λ_m. The exact value is determined by the designer using a deadline factor $d_m \in [0, 1]$, such that $D_m = \mu_m + d_m(\lambda_m - \mu_m)$. Consequently, the definition of $\mathsf{prd}_{[t_s,t_e)}(v_i, e_u)$ becomes:

$$\underset{[t_s,t_e)}{\mathsf{prd}}\ (v_i, e_u) = \begin{cases} 0 & \text{if } (t_e - t_s) < \lambda_i \\ X_i^u(\lfloor \frac{t_e - t_s}{\lambda_i} \rfloor + 1) & \text{if } \frac{(t_e - t_s) \bmod \lambda_i}{D_i} \geq 1 \\ X_i^u(\lfloor \frac{t_e - t_s}{\lambda_i} \rfloor) & \text{if } \frac{(t_e - t_s) \bmod \lambda_i}{D_i} < 1 \end{cases} \tag{26}$$

Based on the new definition of $\mathsf{prd}_{[t_s,t_e)}(v_i, e_u)$ in Equation 26, the predecessor v_m generates enough tokens to guarantee periodic execution of v_j earlier. Thus, we reduce $\phi_{m \to j}$. However, reducing $\phi_{m \to j}$ might cause another actor (e.g., v_k) to have a larger $\phi_{k \to j}$ than $\phi_{m \to j}$. Thus, the deadline reduction is repeated until no other actor v_k with larger $\phi_{k \to j}$ is found. The whole procedure is described in Algorithm 2. We assume that the input actors are the ones with no incoming edges (i.e., level-1 actors), and the output actors are the ones with no outgoing edges. Note that Algorithm 2 assumes that $\mathsf{prd}_{[t_s,t_e)}(v_i, e_u)$ is given by Equation 26 rather than Equation 24.

Now, we define the task-set representation of G as follows:

Definition 14. For a graph G, $\tau_{\mathsf{CDP}}(G)$ is the constrained deadline periodic task-set representation of the actors in G such that $\tau_i \in \tau_{\mathsf{CDP}}(G)$ corresponds to actor $v_i \in V$. τ_i is given by

$$\tau_i = (S_i, C_i, T_i, D_i) \tag{27}$$

where:

Algorithm 2 SET-START-TIME-AND-DEADLINE$(G, \vec{d})$

Require: Acyclic CSDF graph $G = \langle V, E \rangle$
Require: Deadline factors $\vec{d}$
1: **for all** $v_j \in V$ **do**
2: **if** $\mathsf{prec}(v_j) = \emptyset$ **then**
3: $\phi_j \leftarrow 0$
4: **else**
5: Initialize the deadline of each actor $v_i \in \mathsf{prec}(v_j)$ to its period (i.e., $D_i = \lambda_i$)
6: Find $v_m \in \mathsf{prec}(v_j)$ such that $\phi_{m \to j} = \max_{v_i \in \mathsf{prec}(v_j)} \phi_{i \to j}$
7: Set the deadline of v_m to $D_m = \mu_m + d_m(\lambda_m - \mu_m)$
8: Find $\phi_j = \max_{v_i \in \mathsf{prec}(v_j)} \phi_{i \to j}$ with the new deadline of v_m
9: Find $v_k \in \mathsf{prec}(v_j)$ such that $\phi_{k \to j} = \max_{v_i \in \mathsf{prec}(v_j)} \phi_{i \to j}$
10: **if** $v_k \neq v_m$ **then**
11: Repeat lines 6 to 9 {A new actor v_k is the bottleneck}
12: **else**
13: $\phi_j \leftarrow \max_{v_i \in \mathsf{prec}(v_j)} \phi_{i \to j}$ {v_m remains the bottleneck even after reducing its deadline}
14: **end if**
15: **end if**
16: **end for**
17: Set the deadline of each output actor v_o to $D_o = \mu_o + d_o(\lambda_o - \mu_o)$
18: **return** $\vec{\phi}$, where $\phi_i \in \vec{\phi}$ is the start time of actor v_i, and $\vec{D}$, where $D_i \in \vec{D}$ is the deadline of actor v_i

- $S_i = \phi_i$ obtained by Algorithm 2
- $C_i = \mu_i$,
- $T_i = \lambda_i$ given by Lemma 2,
- $D_i = D_i$ obtained by Algorithm 2

THEOREM 5. *$\tau_{CDP}(G)$ is schedulable.*

PROOF. Theorems 2 and 3 in [3] prove the existence of a strictly periodic schedule for the actors of G when: 1) the start time of a level-k actor v_i is equal to ϕ_i given by Equation 22 assuming $\mathsf{prd}_{[t_s,t_e)}(v_i, e_u)$ is given by Equation 24, and 2) the deadline D_i is equal to λ_i. Thus, we need to show that changing ϕ_i and D_i according to Algorithm 2 does not affect the strict periodicity of the actors. According to the CDP model (see Section 3.2.1), changing the deadline does not affect the periodicity. Similarly, the changes to the start time by Algorithm 2 do not affect the periodicity. This is due to the fact that $\phi_{i \to j}$ given by Equation 23 guarantees that an actor starts only if it has enough tokens to continue executing in a strictly periodic way. Thus, any reduction in the start time is guaranteed to not break the strictly periodic execution requirement. □

Example 3. Going back to the graph shown in Figure 2, we compute now the cumulative production function for v_1 on e_1 under the CDP model assuming deadline factors equal to zero. This results in:

$$\underset{[t_s,t_e)}{\mathsf{prd}}\ (v_1, e_1) = \begin{cases} 0 & \text{if } (t_e - t_s) < 8 \\ X_1^1(\lfloor \frac{t_e - t_s}{8} \rfloor + 1) & \text{if } \frac{(t_e - t_s) \bmod 8}{5} \geq 1 \\ X_1^1(\lfloor \frac{t_e - t_s}{8} \rfloor) & \text{if } \frac{(t_e - t_s) \bmod 8}{5} < 1 \end{cases} \tag{28}$$

Now, applying Algorithm 2 assuming that $\vec{d} = \vec{0}$ results in $\vec{\phi} = [0, 5, 21, 29]^T$, $\vec{D} = [5, 12, 24, 4]^T$, and $L_{\mathsf{CDP}}(G) = 33$.

To compute the buffer sizes under the CDP model, we provide the following lemma.

LEMMA 4. *For a graph G, the minimum bounded buffer size b_u of a communication channel $e_u \in E$ connecting a source actor v_i with start time ϕ_i, and a destination actor v_j with start time ϕ_j, where $v_i, v_j \in V$, under a strictly periodic schedule is given by*

$$b_u = \max_{k \in [0,1,\cdots,\alpha]} \{ \underset{[\phi_i, \max(\phi_i,\phi_j)+k)}{\mathsf{prd}} (v_i, e_u) - \underset{[\phi_j, \max(\phi_i,\phi_j)+k)}{\mathsf{cns}} (v_j, e_u) \} \quad (29)$$

Note that under the CDP model, Equation 29 should be computed using $\mathsf{prd}_{[t_s,t_e)}(v_i, e_u)$ given by Equation 26.

Now, we prove the following theorem regarding the latency of graphs with $\vec{q} = \vec{1}$ under the CDP model.

THEOREM 6. *The minimum achievable latency of a graph G with basic repetition vector $\vec{q} = \vec{1}$, when the actors of G are scheduled as constrained-deadline periodic (CDP) tasks, denoted by $L^{min}_{CDP}(G)$, is equal to its minimum achievable latency under self-timed scheduling.*

PROOF. When the actors of G are scheduled as CDP tasks with $\vec{d} = \vec{0}$, an actor $v_i \in V$ is started immediately after all its predecessors have finished one firing. Hence, the latency encountered by the first sample is equal to the sum of actors' execution times along the output path with the largest sum of actors' execution times. This is equivalent to the latency under self-timed scheduling. □

Theorem 6 states that the CDP model achieves optimal latency and rate (i.e., minimum achievable latency and maximum achievable throughput) for graphs with $\vec{q} = \vec{1}$.

5. EVALUATION

We evaluate the constrained-deadline periodic representation proposed in Section 4 by performing an experiment on a set of 19 real-life streaming applications. The objective of the experiment is to compare the latency of streaming applications under the constrained-deadline periodic scheduling model to their minimum achievable latency obtained via self-timed scheduling.

5.1 Benchmarks

The streaming applications used in the experiment are the same as the ones used in [3]. In total, 19 applications are considered as shown in Table 2. The graphs are a mixture of CSDF and SDF graphs. The third column (N) shows the number of actors in each application , and the fourth column (Q) shows the least-common-multiple of the repetition vector elements (i.e., $Q = \mathsf{lcm}\{q_1, \cdots, q_N\}$). The fifth column ($R^{\max}_{\mathsf{SPS}}/R_{\mathsf{STS}}$) shows the ratio of the maximum throughput under strictly periodic scheduling (i.e., $R^{\max}_{\mathsf{SPS}}$) to the maximum achievable throughput of the application (i.e., R_{STS}).

We use the SDF^3 tool-set [18] for computing the minimum latency of SDF graphs. SDF^3 is a powerful analysis tool-set for analyzing CSDF/SDF graphs and is capable of computing several metrics like repetition vector, maximum achievable throughput, etc. For SDF graphs, SDF^3 is capable as well of computing the minimum achievable latency. Thus, we use this capability to compute the minimum achievable latency of a graph and use it as a reference point for comparing the latency under the IDP and CDP models. For CSDF graphs, we construct the self-timed schedule manually and use it to compute the latency.

Table 2: Benchmarks used for evaluation

Domain	Application	N	Q	$R^{\max}_{\mathsf{SPS}}/R_{\mathsf{STS}}$
Signal Processing	Multi-channel beamformer	57	1	1.0
	Discrete cosine transform (DCT)	8	1	1.0
	Fast Fourier transform (FFT) kernel	17	1	1.0
	Filterbank for multirate signal processing	85	1	1.0
	Time delay equalization (TDE)	29	1	1.0
Cryptography	Data Encryption Standard (DES)	53	1	1.0
	Serpent	120	1	1.0
Sorting	Bitonic Parallel Sorting	40	1	1.0
Video processing	MPEG2 video	23	1	1.0
	H.263 video decoder	4	594	1.0
Audio processing	MP3 audio decoder	14	2	1.0
	CD-to-DAT rate converter (SDF)	6	23520	0.04
	CD-to-DAT converter (CSDF)	6	658560	0.05
	Vocoder	114	1	1.0
Communication	Software FM radio with equalizer	43	1	1.0
	Data modem	6	16	1.0
	Satellite receiver	22	5280	0.2
	Digital Radio Mondiale receiver	4	288000	1.0
Medical	Heart pacemaker	4	320	1.0

Table 3: Configurations used for latency comparison

Configuration	Deadline Factor (d_i)	Deadline (D_i)
O_1	1	T_i
O_2	0.75	$C_i + 3(T_i - C_i)/4$
O_3	0.5	$C_i + (T_i - C_i)/2$
O_4	0.25	$C_i + (T_i - C_i)/4$
O_5	0	C_i

5.2 Experiment: Latency Comparison

In this experiment, we compare the latency resulting from the constrained-deadline periodic scheduling to the minimum achievable latency of a streaming application. The minimum achievable latency of a streaming application modeled as a CSDF graph is its latency under self-timed scheduling as mentioned in Section 4. In Algorithm 2, we do not specify an explicit value for the deadline factors vector $\vec{d}$. Thus, we evaluate the effect of the deadline factors values on the latency and resource requirements. This is accomplished by using five configurations with equal periods and different deadline factors. The used periods are the minimum ones given by Lemma 2, while the deadline factors are the ones outlined in Table 3. The first configuration, called O_1, is the one obtained by setting the deadline factor equal to 1 (i.e., $D_i = T_i$). Thus, O_1 results in the implicit-deadline representation. The last configuration O_5 is the one obtained by setting the deadline factor equal to 0 (i.e., $D_i = C_i$). O_2 till O_4 represent intermediate values of the deadline factor. The STS latency is computed using the `sdf3analysis` tool from SDF^3 using the `latency(min_st)` algorithm with auto-concurrency disabled and unbounded FIFO channels sizes.

Figure 3 shows the ratio of the latency of three configurations (O_1, O_3, and O_5) to the minimum achievable latency (i.e., STS latency). A ratio equal to 1.0 means that the CDP latency is equal to the STS latency. We see that the CDP model with $d_i = 0$ (i.e., configuration O_5) achieves the minimum achievable latency for 14 graphs, and even if the basic repetition vector of the graph is not equal to $\vec{1}$. We also see that the mis-matched I/O rates applications (i.e., CD2DAT-(S,C) and Satellite) have higher latency under strictly periodic scheduling. The only matched I/O rates applications with higher latency under strictly periodic scheduling are

Receiver and Pacemaker. They have higher latency because their execution times have large variations between firings. Such variation is captured under self-timed scheduling, while our strictly periodic scheduling assumes always the WCET. It is evident that the IDP model (i.e., configuration O_1) increases the latency on average by 5x compared to the STS latency. O_3 shows approximately a 36% average reduction in latency compared to O_1 and an average 3.2x increase in latency compared to the STS latency.

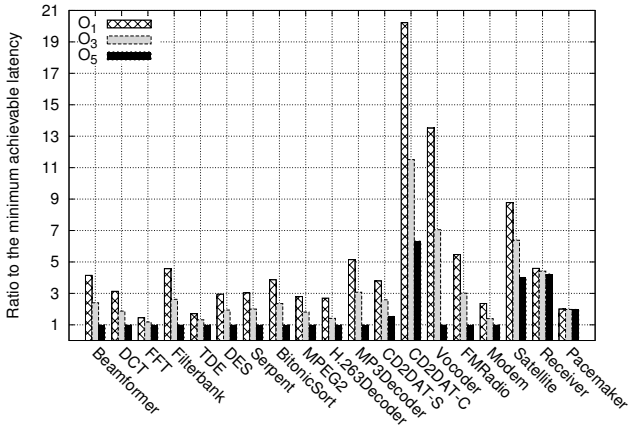


Figure 3: Ratios of the latency under configurations O_1, O_3, and O_5 from Table 3 to the STS latency

Now, we evaluate the same three configurations (i.e., O_1, O_3, and O_5) in terms of resource usage as shown in Figure 4. We do that by computing the minimum number of processors needed to schedule an application. For O_1, this is accomplished by using Equation 6. For O_3 and O_5, we use the sufficient test in Equation 10. We see that the minimum latency achieved by O_5 in most cases comes at a price of significant increase in resource requirements. In O_5, the total density δ_{sum} is equal to the number of actors (i.e., N in Table 2) for 16 applications. If achieving the minimum achievable latency is a requirement, then the designer can use a more accurate schedulablility test to reduce the number of processors (see Section 3.2.2). We also see that increasing the deadline increases the latency but reduces the resource requirements and vice versa. Thus, when the designer has flexibility over the desired latency, he/she might make a trade-off between latency and resources. Such a trade-off is illustrated in Figure 5. In Figure 5, the average ratios of the latency and the number of processors are shown as functions of the configuration. O_1-O_5 are the configurations shown in Table 3, while O_{STS} represents the self-timed schedule constructed using one-to-one mapping (i.e., one task per processor). The latency ratio for the jth configuration is given by

$$\Omega_j(L) = \frac{\sum_{G_i \in \mathcal{G}} (L_{O_j}(G_i)/L_{O_{STS}}(G_i))}{|\mathcal{G}|} \tag{30}$$

where $\mathcal{G}$ is the set of applications in Table 2 and G_i is the ith application. Similarly, the resource usage ratio for the jth configuration is given by

$$\Omega_j(M) = \frac{\sum_{G_i \in \mathcal{G}} (M_{O_j}(G_i)/M_{O_1}(G_i))}{|\mathcal{G}|} \tag{31}$$

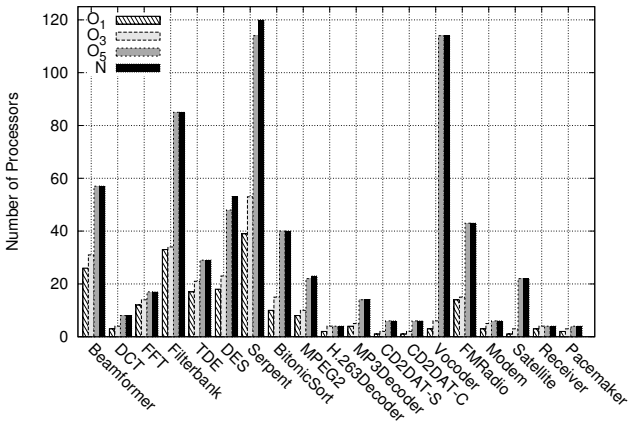


Figure 4: Resource usage for configurations O_1, O_3, and O_5 from Table 3

We see clearly that the latency decreases almost linearly as we decrease the deadline. However, we see that the number of processors increases non-linearly as we decrease the deadline. A "compromise" value for the deadline would be around O_4 where we see a latency of 2.4x compared to the STS latency achieved using around 1.7x the number of processors needed by O_1.

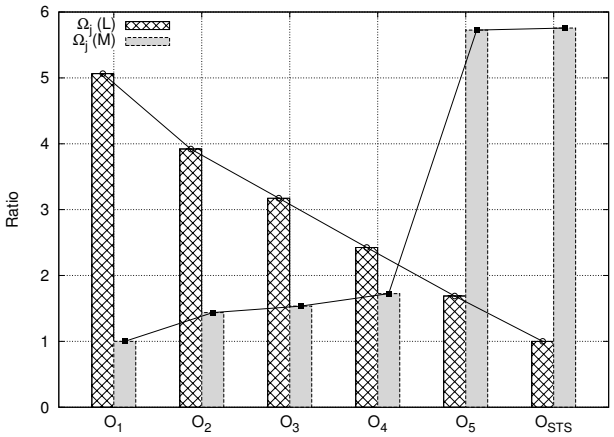


Figure 5: Average ratios of latency and number of processors as functions of the configuration

5.3 Decision Tree for Real-Time Scheduling of CSDF

Based on the results proven in Section 4 and the evaluation results in Section 5.2, we present a decision tree for selecting the real-time periodic task model for scheduling CSDF graphs. The decision tree is illustrated in Figure 6. The first decision is to determine whether the graph is matched I/O rates or not. It is proven in [3] that the periodic task model achieves the maximum achievable throughput for matched I/O rates graphs. If the graph is matched I/O rates and balanced, then the IDP model is *latency and rate optimal (LRO)*. If the graph is matched I/O but unbalanced, then the CDP model helps in reducing the latency. For certain

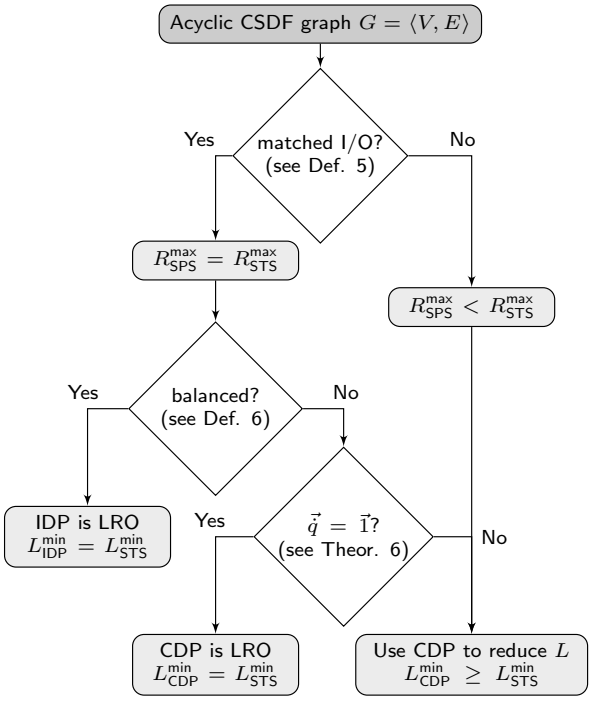


Figure 6: Decision tree for scheduling CSDF actors as real-time periodic tasks. $L^{\min}_{\mathsf{IDP}}$ and $L^{\min}_{\mathsf{CDP}}$ refer to the latency when $\vec{\lambda} = \vec{\lambda}^{\min}$ and $\vec{d} = \vec{0}$ (for the CDP case)

class of graphs (e.g., graphs with $\vec{q} = \vec{1}$), the CDP model is also latency and rate optimal. For mis-matched I/O graphs, the CDP helps in reducing the latency if the throughput achieved under strictly periodic scheduling is acceptable.

6. CONCLUSIONS

We analyze the latency of acyclic CSDF graphs when scheduled using the implicit-deadline periodic (IDP) model. We show that the IDP model achieves the minimum achievable latency for a class of graphs called balanced graphs, however, it increases the latency significantly for unbalanced graphs. To alleviate this problem, we propose an alternative task representation for unbalanced graphs actors based on the constrained-deadline periodic (CDP) model. We prove that the CDP representation achieves the minimum achievable latency for graphs with basic repetition vector equal to $\vec{1}$. Based on empirical evaluations, we show that the CDP representation delivers the minimum achievable latency even for graphs with basic repetition vector different from $\vec{1}$. We summarize our results in the form of a decision tree to assist the designer in choosing the appropriate real-time periodic task model for scheduling acyclic CSDF graphs.

7. ACKNOWLEDGMENTS

This work has been supported by CATRENE/MEDEA+ project number 2A718 (TSAR: Tera-scale multicore processor architecture).

8. REFERENCES

[1] K. Albers and F. Slomka. Efficient feasibility analysis for real-time systems with EDF scheduling. In *Proc. DATE*, pages 492–497, 2005.

[2] T. Baker and S. Baruah. Schedulability Analysis of Multiprocessor Sporadic Task Systems. In I. Lee et al., editors, *Handbook of Real-Time and Embedded Systems*. Chapman & Hall/CRC, Boca Raton, FL, 1st edition, 2007.

[3] M. Bamakhrama and T. Stefanov. Hard-real-time scheduling of data-dependent tasks in embedded streaming applications. In *Proc. EMSOFT*, pages 195–204, 2011.

[4] S. Baruah and N. Fisher. The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems. *IEEE Trans. Comput.*, 55(7):918–923, 2006.

[5] S. Baruah, L. Rosier, and R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Syst.*, 2:301–324, 1990.

[6] G. Bilsen et al. Cyclo-static dataflow. *IEEE Trans. Signal Process.*, 44(2):397–408, 1996.

[7] R. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, 2011.

[8] U. Devi. An improved schedulability test for uniprocessor periodic task systems. In *Proc. ECRTS*, pages 23–30, 2003.

[9] A. Ghamarian et al. Latency Minimization for Synchronous Data Flow Graphs. In *Proc. DSD*, pages 189–196, 2007.

[10] S. Goddard. *On the Management of Latency in the Synthesis of Real-Time Signal Processing Systems from Processing Graphs*. PhD thesis, University of North Carolina at Chapel Hill, 1998.

[11] L. Karam et al. Trends in multicore DSP platforms. *IEEE Signal Process. Mag.*, 26(6):38–49, 2009.

[12] E. Lee and D. Messerschmitt. Synchronous data flow. *Proc. IEEE*, 75(9):1235–1245, 1987.

[13] A. Masrur, S. Chakraborty, and G. Färber. Constant-Time Admission Control for Partitioned EDF. In *Proc. ECRTS*, pages 34–43, 2010.

[14] O. Moreira and M. Bekooij. Self-Timed Scheduling Analysis for Real-Time Applications. *EURASIP J. Adv. Signal Process.*, 2007:1–15, 2007.

[15] O. Moreira, F. Valente, and M. Bekooij. Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor. In *Proc. EMSOFT*, pages 57–66, 2007.

[16] R. Pellizzoni and G. Lipari. A new sufficient feasibility test for asynchronous real-time periodic task sets. In *Proc. ECRTS*, pages 204–211, 2004.

[17] S. Sriram and S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. CRC Press, Boca Raton, FL, 2nd edition, 2009.

[18] S. Stuijk, M. Geilen, and T. Basten. SDF^3: SDF For Free. In *Proc. ACSD*, pages 276–278, 2006.