Chapter 3

Resource Optimization for Real-Time Streaming Application

Di Liu, Jelena Spasic, Jiali Teddy Zhai, Gang Chen, and Todor Stefanov,
"Resource optimization for CSDF-modeled streaming applications with latency constraints,"
2014 *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, 2014, pp. 1-6.

STREAMING applications, such as video/audio processing and digital signal processing, 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) [LM87] and its generalization Cyclo-Static Dataflow (CSDF) [BELP96].

Traditionally, CSDF graphs are scheduled by self-timed scheduling [MB07], where an actor starts to execute as soon as it receives enough tokens from its predecessors. However, since the self-timed scheduling normally provides best-effort services, it is difficult to provide a hard-real-time timing guarantee for every actor in a CSDF graph. Bamakhrama and Stefanov in [BS11] proposed the hard-real-time (HRT) scheduling framework to schedule acyclic CSDF graphs, explained in detail in Section 2.3. This HRT scheduling framework provides *HRT timing guarantee* and *fast admission control* for an application modeled as a CSDF graph, at the expense of increasing the graph latency [BS12]. The same authors in [BS12] identified this issue of the HRT scheduling framework and proposed to reduce the graph latency by scaling down actors' relative deadlines. In [BS12], Bamakhrama and Stefanov proposed to use a scaling factor to uniformly reduce the deadlines of all actors/tasks. However, since the deadlines of periodic tasks play a crucial role in determining the minimum number of processors required to schedule the task set, this uniform scaling factor may unnecessarily increase the required number of processors.

In this chapter, we address this above-mentioned problem of minimizing the number of processors required to schedule a latency-constrained streaming application modeled as a CSDF graph, which is scheduled by using the HRT scheduling framework. We formalize this problem and prove that it is an integer convex programming problem such that it can be solved effectivelly by an off-the-shelf convex programming solver, e.g., CVX [GB14]. 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 scheduled under the HRT scheduling.
- Based on the above contribution, we formalize the problem of minimizing the number of processors for a latency-constrained CSDF graph under the *HRT* scheduling 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 [BS12] 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 48% of the conducted experiments.

Spasic *et al.* in [SLCS16] proposed a new approach to improve the HRT scheduling framework. The proposed approach in this chapter is also applicable to the improved HRT scheduling and more details can be found in [SLCS16].

3.1 Background

We have introduced the CSDF application model, real-time theories, and the HRT scheduling framework in Chapter 2. Here, we present the system model considered in this chapter. The platform, we target, is a homogeneous multiprocessor platform which consists of identical processors. Since the global scheduling, explained in Section 2.2.3, has been proven to be a theoretically optimal scheduling algorithm on multiprocessor systems [DB11], we adopt a global optimal scheduling, such as PFair [BCPV93b], to schedule periodic tasks generated from an acyclic CSDF graph.

To reduce the graph latency, we scale down the relative deadline of each actor and thus a *constrained deadline* task set (i.e., deadline is smaller than or equal to period) is derived. For the *constrained deadline* task model (i.e., $D_i \leq T_i$), Baker and Baruah in

[BB07] gave the following sufficient schedulability for the global optimal scheduling:

$$\delta_\Gamma \leq M \tag{3.1}$$

δ_Γ is the total density of a real-time task set Γ, explained in Section 2.2.1. M denotes the number of processors. This sufficient test can be converted to compute the minimum required number of processors for the global optimal scheduling as follows:

$$M = \lceil \delta_\Gamma \rceil \tag{3.2}$$

For global optimal scheduling, we see from Equation (3.2) that the total density δ_Γ plays a crucial role in computing the minimum number of processors needed to schedule a task set. Therefore, *we are able to minimize the number of processors required to schedule a task set* Γ *by minimizing the value of the total density* δ_Γ.

3.2 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 [DZDN$^+$07] optimize periods for dependent tasks on hard real-time distributed automotive systems in order to meet a latency constraint. In [HCH11], Hong et al. propose a distributed approach to assign local deadlines for each task on distributed systems to meet a latency constraint. In contrast to [DZDN$^+$07] and [HCH11], our work selects deadlines for data dependent tasks in order to meet a latency constraint while minimizing the number of processors required for scheduling the application. Such minimization is not considered in [DZDN$^+$07] and [HCH11]. Balbastre et al. [BRC06] propose an analysis to select deadlines for periodic tasks on a uniprocessor to reduce the output jitters. Comparing to [BRC06], 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. [CWLH08] 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. [JHIP10] 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 objective function cannot be written in a linear form and thus not amenable to an ILP formulation. The authors in [CGHJ09] 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 [CGHJ09] is not applicable to our problem. It is worth noting that the throughput constraint in [JHIP10] and [CGHJ09] can be trivially added into our approach.

3.3 Motivational Example

In this section, we take the CSDF graph in Figure 2.1 as our motivational example to demonstrate the deficiency of the approach in [BS12], where deadlines of actors are computed by Equation (3.3) below with the global uniform deadline scaling factor *df*.

$$\forall \tau_j \qquad D_j = C_j + df \times (T_j - C_j) \qquad 0 \leq df \leq 1 \tag{3.3}$$

We use Equation (3.2) to compute the required number of processors. Given a latency constraint of 20 clock cycles, if we use the approach in [BS12], it finds that the global deadline scaling factor df should be set to 0 in order to meet the latency constraint. By using Equation (3.3), we compute that $D_j = C_j$, and the parameters of the tasks are given in Table 3.1. Using these parameters we obtain that the total density $\delta_A = \frac{2}{2} + \frac{3}{3} + \frac{3}{3} + \frac{6}{6} = 4$. 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_A, which in turn 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 2.3.1 in Section 2.3. We see that in this specific case shown in Table 3.2, although we have changed two deadlines, the start times S_j have not changed. By using Equation (2.11) in Section 2.3 to compute the latency, we see that the latency of 20 clock cycles can be met with the new parameters, but the total density $\delta_A = \frac{2}{2} + \frac{3}{9} + \frac{3}{12} + \frac{6}{6} = 2.58$ decreases. This means that 3 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 [BS12] is not optimal in terms of the required number of processors.

3.4 Proposed Approach

As we show in Section 3.3, although the deadline selection approach in [BS12] 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

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 3.1: Tasks Parameters 1

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

Table 3.2: Tasks Parameters 2

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 Equation (2.11), 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 time instant $t = S_j + kT_j$ for each $k \in \mathbb{N}^+$. The first condition is imposed by the firing rule [BELP96] 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 2.3.1 in Section 2.3 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 3.4.1. *For an acyclic CSDF graph G, the earliest start time of an actor $\tau_j \in A$, denoted S_j, under HRT schedule is given by*

$$S_j = \begin{cases} 0 & \textit{if}\, \Omega(\tau_j) = \emptyset \\ \max_{\tau_i \in \Omega(\tau_j)} \{S_i + (S^{min}_{i \to j} - S^{min}_i - C_i) + D_i\} & \textit{if}\, \Omega(\tau_j) \neq \emptyset \end{cases} \tag{3.4}$$

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^{min}_i is the earliest start time of τ_i given by Equation (2.9) when $D_n = C_n, \forall \tau_n \in A$, and $S^{min}_{i \to j}$ is given by Equation (2.10) when $D_n = C_n, \forall \tau_n \in A$.

Proof. Consider an arbitrary edge $e_u = (\tau_i, \tau_j) \in \mathcal{E}$. 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:

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

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

where $x_i^u(k)$ is the k^{th} element in the production sequence of actor τ_i, $y_j^u(k)$ is the k^{th} element in the consumption sequence of actor τ_j, $\mathcal{N}_i$ and $\mathcal{N}_j$ are the execution lengths of τ_i and τ_j, respectively, as defined in [BELP96]. $u(t)$ is the unit step function. Suppose that $D_n = C_n, \forall \tau_n \in A$. The curves that depict the production and consumption functions of τ_i and τ_j are plotted in Figure 3.1. Interval Δ in Figure 3.1 depends only on the production and consumption rates of τ_i and τ_j on e_u and can be calculated as:

$$\Delta = S_{i\to j}^{\text{min}} - S_i^{\text{min}} - C_i \tag{3.5}$$

Now, suppose that $D_n > C_n, \forall \tau_n \in A$. 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 (2.10) 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:

$$\Delta = S_{i\to j} - S_i - D_i \tag{3.6}$$

We can re-write Equation (3.5) and Equation (3.6) as:

$$S_{i\to j} = S_i + (S_{i\to j}^{\text{min}} - S_i^{\text{min}} - C_i) + D_i \tag{3.7}$$

□

Now, we can derive from Equation (3.4) 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:

$$S_i + (S_{i\to j}^{\text{min}} - S_i^{\text{min}} - C_i) + D_i \leq S_j \quad \forall e_u \in \mathcal{E} \tag{3.8}$$

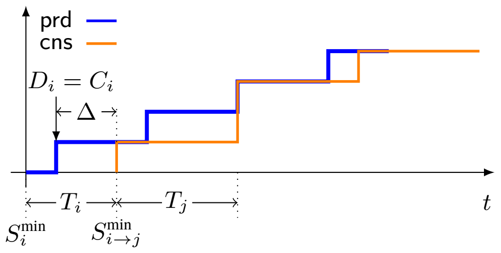

Figure 3.1: Production and consumption curves on edge $e_u = (\tau_i, \tau_j)$

As we explain Since computing the required number of processors depends on the total density δ_A of the task set (see Equation 3.2 Section 3.1), for a CSDF graph $G = (A, \mathcal{E})$, our objective is to minimize δ_A in order to minimize the number of processors. Therefore, we formulate our density minimization (DM) problem as follows:

$$\text{Minimize} \qquad \delta_A = \sum_{\tau_n \in A} \frac{C_n}{D_n} \tag{3.9a}$$

$$\text{subject to:} \quad S_{\text{out}} + D_{\text{out}} - S_{\text{in}} \leq L + g_{\text{in}}^{P} T_{\text{in}} - g_{\text{out}}^{C} T_{\text{out}} \quad \forall w_{\text{in}\to\text{out}} \in W \tag{3.9b}$$

$$S_i + D_i - S_j \leq -(S_{i\to j}^{\min} - S_i^{\min} - C_i) \quad \forall e_u \in E \tag{3.9c}$$

$$- D_n \leq -C_n, D_n \leq T_n \quad \forall \tau_n \in A \tag{3.9d}$$

where Equation (3.9a) is the objective function and D_n is an optimization variable. We want the objective function (3.9a) with $|A|$ optimization variables to be subject to a latency constraint L. Therefore, Inequality (3.9b) comes from Equation (2.11). In addition, inequality constraints (3.9c) are the constraints given by (3.8), and inequality (3.9d) bounds all optimization variables in the objective function by the worst-case execution time and period as explained in Section 2.3. S_i and S_j (including S_{in}, S_{out}) are implicit variables which are not in the objective function (3.9a), 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 3.4.1. *The DM problem (3.9) 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[BV04]. We first prove the convexity of the objective function.

$$f(x) = \frac{a}{x} \tag{3.10}$$

Function (3.10) has been proven to be convex for $x \in (0, \infty)$ and $a > 0$ [BV04]. 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, $\delta_A = \sum_{\tau_n \in A} \frac{C_n}{D_n}$ is a convex function.

A closed halfspace which is convex is a set of the form $\{\mathbf{x} | \mathbf{a}^T\mathbf{x} \leq \mathbf{b}\}$[BV04], where $\mathbf{a} \neq \mathbf{0}$ and all entries in $\mathbf{x}$ are continuous. Since all constraints (3.9b), (3.9c), and (3.9d) are in the form of the closed halfspace, all constraints are convex. Hence, the DM problem (3.9) is a convex programming problem.

Given that all D and S in the DM problem (3.9) have to take only integer values in practice, the DM problem is an ICP problem. $\square$

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 [BV04] and some algorithms for solving mixed integer linear programming can be used to find a global optimal solution for ICP. In Section 3.5.2, the evaluation shows the efficiency of the existing CVX solver [GB14] to solve our DM problem.

3.5 Evaluation

In this section, we evaluate our DM approach and compare it with the Baseline Approach (BA) proposed in [BS12]. This baseline approach uses Binary Search to find the maximum *df* (in Equation 3.3) which makes the latency constraint met. Finding this maximum *df* reduces the required number of processors to schedule the CSDF actors. Our DM problem is solved by using mixed integer disciplined convex programming (MIDCP) in CVX [GB14]. All experiments are performed on an Intel i7 dual-core processor running at 2.7GHz with 4 GB RAM.

We have selected 13 real-life streaming applications modeled as CSDF graphs from the StreamIt [TA10] benchmark suit. The WCET of each actor in the application benchmarks which we use in this evaluation is the same as specified in [BS11]. The characteristics of these benchmarks are given in Table 3.3, including the number of actors ($|A|$), the number of edges ($|\mathcal{E}|$), the maximum latency (L_{max}) and the minimum latency (L_{min}) in clock cycles. The maximum latency is the latency obtained by using the implicit deadline periodic task 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

Realistic Applications	$\lvert A \rvert$	$\lvert \mathcal{E} \rvert$	$L_{\max}$	$L_{\min}$
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 3.3: Characteristics of application benchmarks

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

Table 3.4: Latency Constraints

benchmark and compare it to the BA for the three latency constraints per benchmark, shown in Table 3.4, while the achieved throughput of each benchmark is the same in both BA and DM approaches.

3.5.1 The effectiveness of our DM approach

First, we evaluate the effectiveness of our DM approach in terms of the number of required processors. Figure 3.2 to 3.4 show the results under global scheduling. The number of processors needed to schedule a task set is computed using Equation (3.2).

Figure 3.2 shows the results with latency constraint $L_0(L_{\min})$. Under such stringent latency constraint, the intervals in which deadlines of actors may vary are limited. Our DM approach is still capable 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

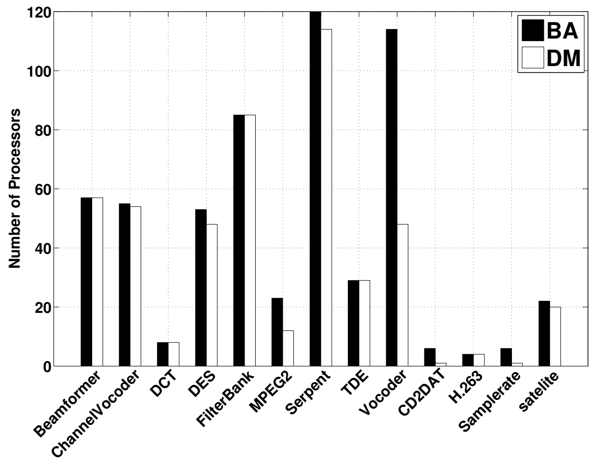

Figure 3.2: Global scheduling with L_0 constraint

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.

Figure 3.3 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. In this case, the Beamformer benchmark with symmetric structure, also benefits 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/actors. For DCT, TDE, and H.263, although we can see a reduction in the total density δ_A of the task set, the reduction is smaller and insufficient to decrease 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_A. Figure 3.4 shows the results for a very relaxed latency constraint, where only 5 benchmarks get reduction on the number of processors by using our DM approach.

In summary, our DM approach obtains reduction in the number of processor in more than 48% of the conducted experiments.

3.5.2 The time complexity of solving our DM problem

In this section, we evaluate the efficiency of our DM approach in terms of the execution time of CVX [GB14] for solving our DM problem. We set a maximum runtime of 4 hours for the solver, and all results are summarized in Table 3.5 where the time

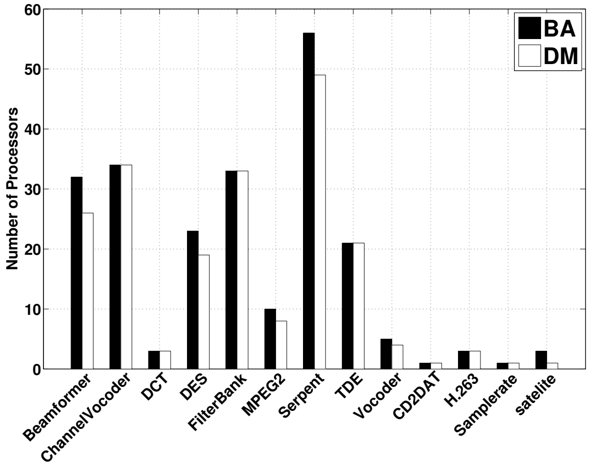

Figure 3.3: Global scheduling with L_1 constraint

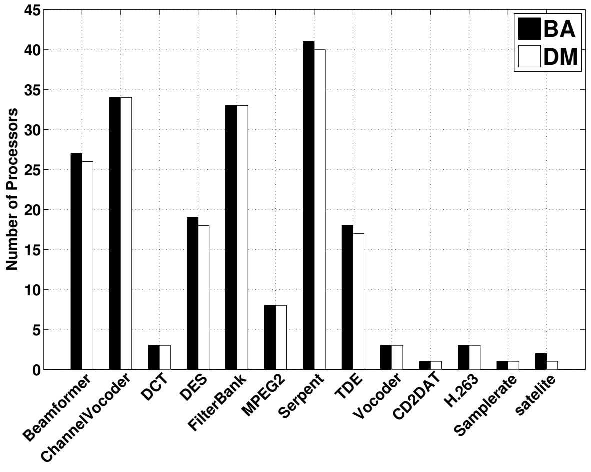

Figure 3.4: Global scheduling with L_2 constraint

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

Table 3.5: The execution time of our DM approach (in second)

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 3.5 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.

3.6 Discussion

In this work, we address the resource minimization problem of CSDF-model streaming applications under the global optimal scheduling when considering the *HRT* framework [BS11, BS12]. The proposed DM approach is also applicable to partitioned scheduling. The approach given in [SLCS16] shows how our DM approach can be extended to partitioned scheduling. In summary, since there exists no optimal partitioned scheduling, we cannot directly build a convex objective function for it. Therefore, in [SLCS16], our DM approach is first used to select optimal deadlines for actors using the same objective function as in Equation (3.9a) and then a partitioned algorithm, e.g., FFD, WFD, etc (explained in Section 2.2.3), is deployed to assign task to processor. If the current number of processors cannot suffice the schedulability of the new input task, the number of processors increments by one. The procedure repeats until all tasks are successfully mapped to the system. *It is worth to notice that when our DM approach is applied in partitioned scheduling, as in [SLCS16], it only derives a suboptimal result.*