Energy-Efficient Scheduling of Throughput-Constrained Streaming Applications by Periodic Mode Switching

Sobhan Niknam, Todor Stefanov
Leiden Institute of Advanced Computer Science
Leiden University, Leiden, The Netherlands
Email: {s.niknam, t.p.stefanov}@liacs.leidenuniv.nl

***Abstract*—In this paper, we address the problem of energy reduction when scheduling streaming applications with throughput constraints on homogeneous multiprocessor systems with voltage and frequency scaling capability. We propose a novel periodic scheduling framework which allows streaming applications to switch their execution periodically between a few energy-efficient schedules at run-time in order to meet a throughput constraint at long run. Using such periodic switching, we can benefit from adopting Dynamic Voltage and Frequency Scaling (DVFS) techniques to exploit available static slack time in the schedule of an application efficiently. The experimental results, on a set of real-life streaming applications, show that our novel scheduling approach can achieve up to 68% energy reduction depending on the application and the throughput constraint compared to related approaches.**

I. Introduction

Streaming applications have become prevalent in embedded systems in several application domains, such as image processing, video/audio processing, and digital signal processing. These applications usually have high computational demands and tight performance constraints, such as throughput constraints. Therefore, to handle the ever increasing computational demands and satisfy throughput constraints, Multi-Processor System-on-Chip (MPSoC) has become a standard platform that is widely adopted in embedded systems design. To exploit the available parallelism in a MPSoC and guarantee the application throughput, several Models of Computation (MoCs) have been proposed to parallelize streaming applications running on a MPSoC, e.g., Synchronous Dataflow (SDF) [1] and Cyclo-Static Dataflow (CSDF) [2]. Within a parallel MoC, a streaming application is represented as a task graph with concurrently executing and communicating tasks. As streaming MPSoC-based embedded systems operate very often using stand-alone power supply such as batteries, energy efficiency has become one of the primary design criterion of such embedded systems in order to prolong the operational time without replacing/recharging the batteries. Furthermore, energy-efficient design decreases the heat dissipation of the system which, in return, improves the system reliability [3].

Given the above discussion, designing a streaming MPSoC-based embedded system brings two interrelated challenges. The first challenge is how to assign and schedule the tasks of an application to a MPSoC such that all the timing constraints are guaranteed. The second challenge is how to achieve energy efficiency of the MPSoC. To address the first challenge, self-timed scheduling has been widely considered as the scheduling policy (e.g., [4], [5]) for streaming applications modeled as data-flow graphs to guarantee the throughput constraints. The analysis techniques for self-timed scheduling, however, necessitate a complex design space exploration (DSE) to determine the minimum number of processors needed to schedule the applications and the assignment of tasks to processors. Moreover, self-timed scheduling is not able to guarantee temporal isolation between applications running on the same platform. In contrast, a technique has been recently presented in [6] that can convert an initial streaming application to a periodic real-time task set. As a result, this conversion enables the designer to employ many algorithms from the classical hard real-time multiprocessor scheduling theory [7] to guarantee throughput constraints and temporal isolation among different applications, using fast schedulability tests. In addition, these algorithms facilitate the computation of the number of processors required to schedule the applications using several fast approaches, instead of performing a complex and time-consuming design space exploration. Therefore, because of the advantages of [6] over the self-timed scheduling, we adopt [6] in this paper as a primary technique for scheduling the streaming applications.

To address the energy efficiency challenge, mentioned above, Voltage and Frequency Scaling (VFS) has been traditionally adopted as an efficient and commonly-used technique when the Processing Elements (PEs) in a MPSoC are capable of operating at different discrete supply voltage and operating frequency levels [3]–[5], [8]–[10]. The general idea behind these approaches is to exploit available static slack time in the schedule of an application in order to slow down the execution of running tasks by using the VFS technique to reduce the energy consumption while still meeting the throughput constraint. However, to the best of our knowledge, applying the VFS technique in the context of [6], considered in this paper, to achieve energy-efficiency has not been studied yet. Therefore, in this paper, we investigate how the scheduling framework presented in [6] can be combined efficiently with the VFS technique to achieve energy-efficiency. To do so, we first show in the motivational example in Section IV that a straightforward way of applying VFS similar to [3]–[5], [8]–[10] is not energy efficient in the context of [6]. Therefore, we introduce a novel energy-efficient periodic scheduling framework which combines VFS and [6] in a sophisticated way, thereby achieving energy efficiency. In this framework, the execution of an application is periodically switched at run-time between a few off-line determined energy-efficient schedules, called *operating modes*, to meet the throughput constraint at long run. As a result, this framework can reduce the energy consumption significantly by exploiting static slack time more efficiently using Dynamic VFS (DVFS), where

This research is supported by the Dutch Technology Foundation STW under the Robust Cyber Physical Systems program (Project 12695).

multiple operating frequencies are computed at design-time for the PEs to be used at run-time.

The specific novel contributions of this paper are the following:

- A simple scheme has been devised for determining a set of discrete operating modes of a system at different operating frequencies where each operating mode provides a unique pair of throughput and minimum power consumption to achieve this throughput.
- With such set of discrete operating modes and a given throughput constraint, we have devised an energy-efficient periodic scheduling framework which allows the streaming applications to switch their execution periodically between operating modes at run-time to meet the throughput constraint at long run. Using this specific switching scheme, we can benefit from adopting the Dynamic Voltage and Frequency Scaling (DVFS) technique to exploit the available static slack time in the scheduling of an application efficiently.
- The experimental results, on a set of 6 real-life streaming applications, show that our scheduling approach can achieve up to 68% energy reduction depending on the application and the throughput constraint compared to the straightforward way of applying VFS similar to related works.

The reminder of the paper is organized as follows: Section II gives an overview of the related work. Section III introduces the preliminary materials needed for understanding the contributions of this paper. Section IV gives a motivational example. Section V presents the proposed scheduling framework. Section VI presents the results of the evaluation of our proposed framework. Finally, Section VII ends the paper with conclusion.

II. Related Work

Several approaches aiming at reducing the energy consumption of streaming applications have been presented in the past decade. Among these approaches, [3]–[5], [8]–[10] are the closest to our work. These approaches have a common goal to reduce the energy consumption of a system by exploiting the static slack time in the schedule of throughput-constrained streaming applications using per-task [3], [8], per-core [4], [5], [9], [10] or global [9] VFS.

The approaches in [3], [8], [9], formulate the energy optimization problem as mixed integrated linear programming (MILP) problem to integrate the VFS capability of PEs with application scheduling. Compared to these approaches our approach mainly differs in two aspects. First, these approaches consider streaming applications modeled either as a Directed Acyclic Graph (DAG) [3], [9] or a Homogeneous SDF (HSDF) graph [8] derived by applying a certain transformation on an initial SDF graph. Therefore, these approaches can not be directly applied to streaming applications modeled with a more expressive MoCs, e.g. (C)SDF as considered in this paper. In addition, transforming a graph from SDF to HSDF is a crucial step in [8] where the number of tasks in the streaming application can exponentially grow. This growth of the application in terms of number of tasks can lead to time-consuming analysis and significant memory overhead for storing the tasks code. In contrast, our approach directly handles a more expressive MoC, such as (C)SDF. Second, the approach in [9] uses per-core VFS where the off-line computed operating frequencies of PEs are fixed at run-time and can not be changed. In contrast, our approach uses DVFS where a sequence of frequency changes that is computed off-line is used on the PEs during execution at run-time while guaranteeing the throughput constraint. As a result, the DVFS technique enables our approach to exploit the available static slack time in the application's schedule more efficiently for better energy reduction. The approaches in [3], [8] use a fine-grained DVFS, i.e., per-task VFS, where the operating frequency of PEs can be changed before executing each task. Fine-grained DVFS, like in [3], [8], can be beneficial only when the overhead of DVFS is negligible. In contrast to these approaches, we adopt a coarse-grained DVFS where the operating frequencies of PEs are changed at the granularity of graph iterations to avoid large overhead associated with the operating frequency changes.

The approaches in [4], [5], [10] perform energy minimization directly on an SDF graph. [5] and [10] perform design space exploration at design time to find an energy-efficient task mapping of an SDF graph scheduled in self-timed manner on a MPSoC platform with per-core VFS capability such that a throughput constraint is guaranteed. In [4], the authors proposed heuristics to find per-core VFS for a given task mapping and static order schedule such that the throughput constraint is met. However, as shown in the motivation example in Section IV, applying VFS in a similar way as in [4], [5], [10] for streaming applications scheduled using the framework presented in [6] and considered in this paper, is not energy-efficient. Compared to the approaches in [4], [5], [10], our approach is different in two aspects. First, these works use self-timed scheduling which analysis techniques suffer from a complex design space exploration (DSE). In contrast, we use the scheduling technique in [6] that can benefit from using fast analysis using existing real-time theories [7]. Second, these works used per-core VFS to exploit the static slack time in the application's schedule. In contrast, our approach uses a coarse-grained DVFS technique. As a result, the PEs are able to run periodically at lower operating frequencies by exploiting the available static slack time more efficiently which yields lower energy consumption.

III. Preliminaries

In this section, we first introduce the CSDF MoC (Section III-A) and the system model (Section III-B) considered in this paper. Then, we review the Strictly Periodic Scheduling (SPS) framework [6], which we use to schedule tasks in a CSDF graph (Section III-C) and present the energy model (Section III-D) used in this paper.

A. Cyclo-Static Dataflow (CSDF)

A CSDF graph is defined as a directed graph $G = (V, E)$, where V is a set of tasks and E is a set of edges. Task $\tau_i \in V$ represents computation and edges represent FIFO channels to transfer data tokens between tasks. Each task $\tau_i \in V$ may consume/produce a varied but predefined number of data tokens in its consequent executions, called consumption/production sequence. It has been proven in [2] that a valid static schedule of a CSDF graph can be generated at design-time if the graph is consistent and live. A CSDF graph is said to be consistent if a non-trivial solution exists for the repetition vector $\vec{q} = [q_1, q_2, \ldots, q_i]$. An entry q_i indicates the number of invocations of task τ_i in one iteration of the graph. For more details, we refer the reader to [2].

B. System Model

The considered MPSoC platforms in this work are homogeneous, i.e., a platform contains a set $\Psi = \{\Psi_1, \Psi_2, \cdots, \Psi_m\}$ of m identical PEs with distributed memories. The considered scheduling algorithms on each PE are dynamic scheduling algorithms, such as Partitioned Earliest Deadline First (EDF)

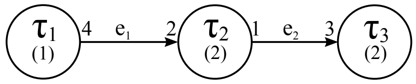


Fig. 1: A simple example of a (C)SDF graph G. The worst case execution times of the tasks at the maximum operating frequency are shown between parentheses.

[11]. We assume that each PE supports only a discrete set $\theta = \{f_{min} = f_1, f_2, \cdots, f_n = f_{max}\}$ of n operating frequencies and different PEs can operate at different frequencies at the same time. Without loss of generality, we assume that the operating frequencies in the set θ are in ascending order, in which f_1 is the lowest operating frequency and f_n is the highest operating frequency.

C. Strictly Periodic Scheduling (SPS) framework

In [6], the real-time strictly periodic scheduling (SPS) framework for acyclic CSDF graphs is proposed. In this framework, the tasks in an acyclic CSDF graph are converted to a set of real-time periodic tasks by deriving the *minimum period* (T_i) and *earliest start time* (S_i) of each task τ_i. As a result, this conversion enables the designer to apply the well-developed hard real-time scheduling theories [7].

In this framework, the *earliest start time* (S_i) of task τ_i is calculated such that τ_i is never blocked on reading data tokens from any FIFO channel connected to it during its periodic execution. The *minimum period* (T_i) of tasks is also derived by the following expression:

$$T_i = \frac{lcm(\vec{q})}{q_i} \, . \, s, \; \forall \tau_i \in V, \tag{1}$$

$$s = \left\lceil \frac{max_{\tau_j \in V}\{C_j . q_j\}}{lcm(\vec{q})} \right\rceil, \tag{2}$$

where $lcm(\vec{q})$ is the *least common multiple* of all repetition entries in $\vec{q}$ and C_j is the worst-case execution time of task τ_j. In general, the derived periods of tasks must satisfy the following condition:

$$q_1 T_1 = q_2 T_2 = \cdots = q_n T_n = \alpha \tag{3}$$

where α is the *graph iteration period*, also called *hyper period*, representing the duration needed by the graph to complete one iteration.

With the given and computed parameters, mentioned above, task τ_i is characterized by a tuple $\tau_i = \{C_i, S_i, T_i\}$, where C_i is the worst-case execution time of the task, T_i is the task's period and S_i is the start time of the task. The throughput of each task τ_i can be computed as $1/T_i$. The throughput R of a graph G when its tasks are scheduled as strictly periodic tasks is determined by the period of the output task (T_{out}) and is equal to

$$R = 1/T_{out} \; . \tag{4}$$

In this paper, we only consider implicit-deadline tasks, which have relative deadline D_i equal to their period T_i. Each task invocation releases a *job*. The kth job of task τ_i is denoted by $\tau_{i,k}$ which arrives to the system at time instant $r_{i,k} = S_i + kT_i$ for all $k \in \mathbb{N}_0$. Similarly, the absolute deadline of $\tau_{i,k}$ is $d_{i,k} = S_i + (k+1)T_i$, which coincides with the arrival of job $\tau_{i,k+1}$. The utilization of task τ_i is given by $u_i = \frac{C_i}{T_i}$. The total utilization of task set Π_j containing all assigned tasks to processing element Ψ_j is denoted by $U_j = \sum_{\tau_i \in \Pi_j} u_i$.

D. Energy Model

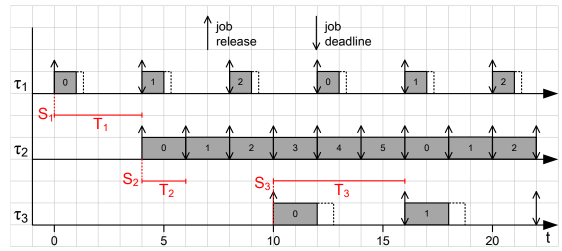
In this paper, we use the energy model described in [12]. We use this model because the parameters of this model are derived by performing real measurements on a real MPSoC platform and the model is shown to be accurate. According to [12], the power consumption of a CMOS circuit is modeled as follows:

$$P(f) = kf^b + s \tag{5}$$

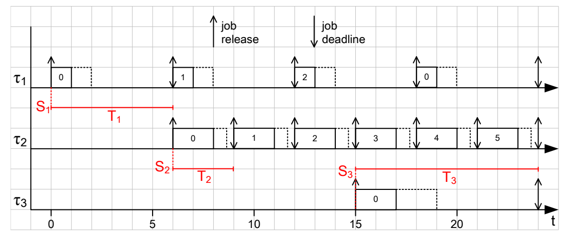
where the first term is the dynamic power consumption and includes all frequency-dependent components, the second term is the static power consumption and includes all frequency-independent components, and f is the operating frequency. Parameters k and b are dependent on the technology process and they are determined in [12]. After all tasks are assigned to PEs, the power consumption of the jth PE (Ψ_j), can be computed by the following equation:

$$P_j = k \cdot f_j^b \cdot \frac{f_{max}}{f_j} \sum_{\forall \tau_i \in \Pi_j} \frac{C_i}{T_i} + s \tag{6}$$

where f_j is the operating frequency of Ψ_j. Therefore, the energy consumption of Ψ_j within one hyper-period is $E_j = \alpha \cdot P_j$ and the energy consumption of the whole system within one hyper-period is $E = \sum_{\forall \Psi_j \in \Psi} \alpha \cdot P_j$.



(a)



(b)

Fig. 2: The (a) SPS and (b) scaled SPS of the (C)SDF graph G in Fig. 1. Up arrows represent job releases, down arrows represent job deadlines. Dotted rectangles show the increase of the tasks execution time when using the VFS technique. Different jobs of each task are shown with different indexes. Note that we adopt partitioned EDF to schedule the assigned tasks on each PE.

IV. Motivational Example

In this section, we motivate the necessity of devising a new energy-efficient scheduling approach using VFS in the context of the SPS scheduling framework [6]. To do so, this motivational example consists of two parts. In the first part, we show that a straightforward way of applying the VFS technique in the context of the SPS framework [6] is not energy efficient. Then, in the second part, we show how we can schedule an application more energy efficiently using our novel periodic scheduling framework.

A. Apply VFS similar to related works

Let us consider the simple streaming application modeled as a (C)SDF graph in Fig. 1. This graph has three tasks (τ_1, τ_2, τ_3) with worst-case execution times at the maximum operating frequency indicated between parentheses $(C_1 = 1, C_2 = 2, C_3 = 2)$

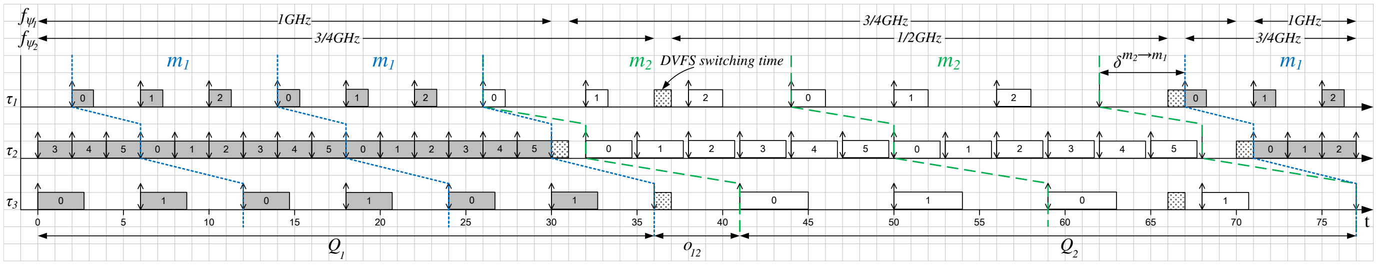


Fig. 3: Our proposed periodic schedule of graph G in Fig. 1. In this schedule, graph G periodically executes according to schedules of operating mode m_1 and operating mode m_2 in Fig. 2(a) and Fig. 2(b), respectively. Note that this schedule repeats periodically. $o_{12} = 5$ and $o_{21} = 0$.

and production/consumption rates indicated above the corresponding edges. The repetition vector of this graph, using the theory in [2], is $\vec{q} = [q_1 = 3, q_2 = 6, q_3 = 2]$, meaning that during each graph iteration, tasks τ_1, τ_2, and τ_3 execute 3, 6, and 2 times, respectively.

By applying the SPS framework [6], briefly discussed in Section III-C, for the example in Fig. 1, we can represent the tasks as strictly periodic tasks by the following tuples: $\tau_1 = \{C_1 = 1, S_1 = 0, T_1 = 4\}$, $\tau_2 = \{2,4,2\}$, and $\tau_3 = \{2,10,6\}$. Note that to derive the periods, $s = \left\lceil \frac{max_{\tau_j \in V}\{C_j \cdot q_j\}}{lcm(\vec{q})} \right\rceil = \left\lceil \frac{12}{6} \right\rceil = 2$ is used in Eq. (1). Based on these tuples, we can derive the strictly periodic schedule for this application shown in Fig. 2(a). In this schedule, for instance, task τ_3 starts at time instant 10, executes for 2 time units, and repeats its execution every 6 time units. Since task τ_3 is the output task in this graph, the throughput of this schedule can be computed as $R = \frac{1}{T_3} = \frac{1}{6}$.

So far, we have assumed that the tasks run at the maximum operating frequency of the PEs. Let us assume that each PE can only support a discrete set $\theta = \{1/4, 1/2, 3/4, 1\}(GHz)$ of 4 operation frequencies. The minimum number of processors needed to run the schedule in Fig. 2(a) is two. Therefore, our MPSoC consists of two PEs, $\Psi = \{\Psi_1, \Psi_2\}$, where we assign task τ_2 to Ψ_1 and tasks τ_1 and τ_3 to Ψ_2. In order to make this schedule more energy efficient, we can use the VFS technique and exploit the available static slack time in the schedule for the purpose of slowing down the execution of tasks by decreasing the operating frequency of PEs. For this example, we can decrease the operating frequency of Ψ_2 to $3/4$ GHz while still meeting all timing constraints, i.e., job deadlines shown as down arrows in Fig. 2(a). This slowing down of tasks is visualized by extending the gray boxes with the dotted boxes in Fig. 2(a). Using the energy model described in Section III-D, the power consumption of this schedule is 0.61 mW. The energy consumption of this schedule for a period of 36 time units, that is equivalent to 3 graph iterations of this schedule, is 21.96 mJ.

To further reduce the power consumption by decreasing the operating frequency of PEs, more static slack time is needed to be created by slowing down the application. Note that periods computed by Eq. (1) are the minimum periods for tasks scheduled by SPS. To slow down the application, we can derive larger valid periods for tasks by taking any integer $s > \left\lceil \frac{max_{\tau_j \in V}\{C_j \cdot q_j\}}{lcm(\vec{q})} \right\rceil$. We refer to this approach as *task period scaling* in this paper. In this way, if we take $s = 3 > \left\lceil \frac{12}{6} \right\rceil$, a new schedule can be derived using SPS, as shown in Fig. 2(b), with throughput $R = \frac{1}{T_3} = \frac{1}{9}$. As a result, there is more available static slack time in the application's schedule which enables the PEs to work at lower operating frequencies of $3/4$ GHz and $1/2$ GHz for Ψ_1 and Ψ_2, respectively. This is visualized by extending the white boxes with the dotted boxes in Fig. 2(b). Using the energy model in Section III-D, the power consumption of this schedule is 0.43 mW. The energy consumption of this schedule for a period of 36 time units, that is equivalent to 2 graph iterations of this schedule, is 15.48 mJ. As a result, the energy consumption is reduced by 29.5% using the schedule in Fig. 2(b) corresponding to $s = 3$ compared to the schedule in Fig. 2(a) corresponding to $s = 2$ for the same period of time at the expense of decreasing the throughput of the application from $1/6$ to $1/9$. By increasing the value of scaling parameter s and scaling the periods as much as possible such that the corresponding schedule still meets the throughput constraint, we can apply VFS in the straightforward way, described above, similar to the related works [4], [5], [10]. Therefore, the maximum created static slack time in the application's schedule can be exploited using the VFS technique to reduce the energy consumption as much as possible.

TABLE I: Operating modes for graph G

Mode	α	f_{Ψ_1}	f_{Ψ_2}	$(R\ [\frac{Token}{Time\ units}],\ P\ [_{mW}])$	*Energy* $[_{mJ}]$
m_1 $(s=2)$	12	1	$3/4$	$(1/6, 0.61)$	439.2
m_2 $(s=3)$	18	$3/4$	$1/2$	$(1/9, 0.43)$	309.6
m_3 $(s=4)$	24	$1/2$	$1/2$	$(1/12, 0.36)$	259.2
m_4 $(s=5)$	30	$1/2$	$1/4$	$(1/15, 0.34)$	244.8
m_5 $(s=8)$	48	$1/4$	$1/4$	$(1/24, 0.31)$	223.2

Now, assume that a throughput constraint of $1/8$ has to be met. Following the scaling approach, explained above, the schedule corresponding to $s = 2$ with the throughput of $1/6$ must be selected to meet the throughput constraint of $1/8$, as shown in Fig. 2(a). However, this schedule is not the most energy efficient one. This is because, although the throughput constraint of $1/8$ is met, more energy than needed is consumed as a result of delivering higher throughput, i.e, $1/6$, than needed.

B. Our proposed scheduling approach

In this section, we introduce our novel energy-efficient scheduling for the example in Fig. 1 that meets the same throughput constraint of 1/8 while consuming less energy compared to the scheduling explained in Section IV-A above. In our approach, among all possible application's schedules corresponding to different values of scaling parameter s to scale periods, we select only Pareto optimal schedules and form a set γ of schedules called *operating modes*. For instance, the set $\gamma = \{m_1, m_2, m_3, m_4, m_5\}$ of five operating modes for our example application in Fig. 1, is given in Table I. In this table, every row shows an operating mode with the graph iteration α, the operating frequencies of the two PEs

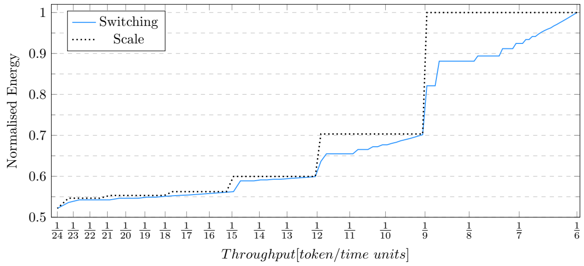


Fig. 4: Normalized energy consumption of scaling and proposed periodic switching schedulings of the (C)SDF graph in Fig. 1 for a wide range of throughput constraints.

(f_{Ψ_1}, f_{Ψ_2}), the pair of throughput and power consumption (R, P) corresponding to this scheduling mode. In the last column, the energy consumption of the operating modes is given for a period of 720 time units which is the *least common multiply* of their graph iterations α. As can be seen in this column, the energy consumption of operating modes is being reduced by slowing down the application during this common period of time. The value of parameter s corresponding to each operating mode is also given in the first column. For instance, operating mode m_4 is the application scheduling corresponding to $s = 5$ that delivers throughput of $1/15$. In this scheduling, Ψ_1 and Ψ_2 must operate at frequencies of $1/2$ GHz and $1/4$ GHz in order to meet all task's job deadlines. Therefore, the power consumption of this scheduling is 0.34 mW. Finally, the energy consumption of this scheduling mode for 720 time units is 244.8 mJ.

Looking at set γ of operating modes in Table I, the throughput constraint of $1/8$, we consider, is between the throughput of operating modes m_1 and m_2. Therefore, we propose the idea of periodically switching the system mode between operating modes m_1 and m_2 to meet the throughput constraint. Such periodic switching schedule is depicted for one period in Fig. 3, where the application executes for three graph iterations according to the schedule of operating mode m_1 and two graph iterations according to the schedule of operating mode m_2. Different graph iterations are separated by dotted and dashed lines for consecutive executions of the application in operating mode m_1 and m_2, respectively, in Fig. 3. Note that this schedule repeats periodically every 77 time units, as shown in Fig. 3 ($Q_1 + Q_2 + o_{12} = 77$). In one period, output task τ_3 executes 10 times in total during 77 time units, meaning that throughput of $10/77 = 1/7.7$ is delivered at long run that is more closer to the required throughput constraint of $1/8$ compared to the throughput of $1/6$ delivered as a result of the schedule in Fig. 2(a). More importantly, the energy consumption of our proposed novel scheduling in Fig. 3 for a period of 924 time units, which is the *least common multiply* of the period of our approach (77 time units) and the graph iteration of the schedule in Fig. 2(a) (12 time units), is 496.68 mJ. The energy consumption of the schedule in Fig. 2(a) in the same period of 924 time units is 563.64 mJ. Therefore, our novel scheduling approach can reduce the energy consumption by 11.87% when the throughput constraint of $1/8$ has to be met. The energy reduction of our proposed schedule, referred as Switching, compared to the scheduling approach explained in Section IV-A, referred as Scale, for a wide range of throughput constraints is given in Fig. 4. In this figure, the x-axis shows different throughput constraints for the example (C)SDF graph in Fig. 1 while the y-axis shows the normalized energy consumption. From Fig. 4, we can see that our proposed scheduling approach Switching can reduce the energy consumption significantly compared to Scale for a large set of throughput constraints.

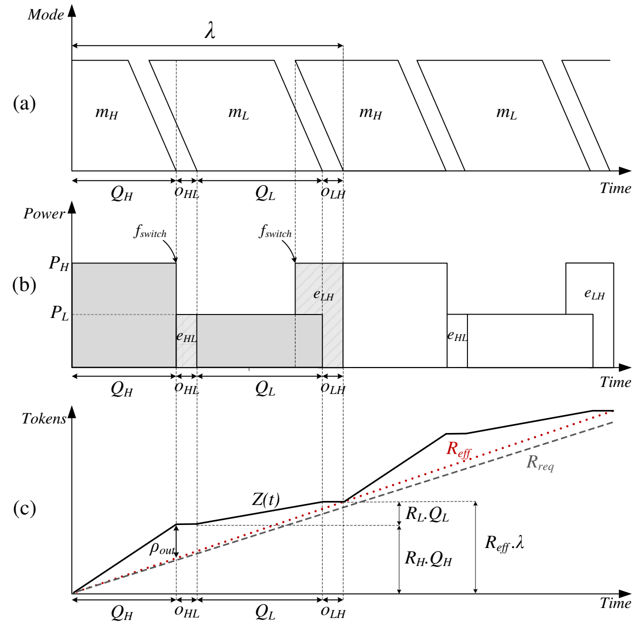


Fig. 5: (a) Switching scheme, (b) Associated energy consumption of the switching scheme and (c) Token production Function Z(t).

Note that our proposed scheduling approach uses DVFS. This is because, PEs run at different operating frequencies in each operating mode. Therefore, when the application switches to execute in a different operating mode, the operating frequencies of the PEs are changed accordingly. The way of changing the operating frequencies of the PEs, for our example, is shown by the horizontal arrows on top of Fig. 3. In this paper, we also consider the switching time cost of DVFS in our analysis that is shown by the boxes with dotted pattern in Fig. 3.

From the above example, we can see the necessity and usefulness of our novel scheduling approach, presented in detail in Section V, to obtain more energy-efficient application scheduling when VFS is used in the context of [6].

V. Proposed Scheduling Framework

In this section, we describe our proposed energy-efficient scheduling framework for throughput-constrained streaming applications. The basis of our approach is to determine a set of *operating modes* where each operating mode provides a unique pair of throughput and minimum power consumption to achieve this throughput. Then, for a given throughput constraint, there may exist an operating mode whose throughput matches the throughput constraint. In this unlikely case, we simply select this operating mode. Otherwise, we choose the two operating modes with the closest higher and lower throughput to the throughput constraint and we call them *Higher operating mode* (m_H) and *Lower operating mode* (m_L), respectively. Then, we meet the throughput constraint at long run by periodically switching the execution of the application between these two operating modes.

A general overview of our scheme for switching the system between the higher and lower operating modes is illustrated in Fig. 5. The periodic execution of an application between the higher and lower operating modes in our approach is shown in Fig. 5(a) and the period of switching is denoted by λ. The associated application's energy consumption and token production caused by our switching scheme corresponding to Fig. 5(a) are also shown in Fig. 5(b) and Fig. 5(c), respectively. According to Fig. 5(a), the execution of the application in each period λ consists of four parts. In the first part, the application executes in the higher operating mode for Q_H time units where the application has throughput R_H and power consumption P_H. Then, in the second part, the execution of the application switches to the lower operating mode m_L. However, this switching can not happen immediately and it takes some time, denoted as o_{HL}, before the application is able to produce tokens again in the lower operating mode. Therefore, during the switching, the application does not have any token production for o_{HL} time units while consuming the energy of e_{HL}, as shown in Fig. 5(b) and Fig. 5(c), respectively. After completing the switching, in the third part, the application executes in the lower operating mode for Q_L time units where the application has the throughput and power consumption of R_L and P_L, respectively. Finally, in the fourth part, the application switches again to the higher operating mode m_H for the next period of λ. However, this switching can not happen immediately and it takes some time that is denoted by o_{LH}. During the switching o_{LH}, no tokens are produced by the application while the energy of e_{LH} is consumed. As a result of the switching scheme in Fig. 5, the application generates a number of tokens in total, see the curve $Z(t)$ in Fig. 5(c), by executing in the higher and lower operating modes during every period of λ and in every λ the application effectively delivers the throughput of R_{eff} in the long run. The curves corresponding to the token production $Z(t)$ in our switching scheme and the effective throughput of R_{eff} are shown in Fig. 5(c) with a solid line and a dotted line, respectively. The throughput constraint R_{req} is also shown with a dashed line in this figure. Therefore, to satisfy the throughput constraint, we have to keep the effective throughput R_{eff} above the throughput constraint R_{req}. This ensures that the number of produced tokens in any time instant are more or equal than what is needed.

Considering Fig. 5(c), the effective throughput obtained by executing the application in operating mode m_H for Q_H time units and in operating mode m_L for Q_L time units is computed by the following expression:

$$R_{eff} = \frac{R_H Q_H + R_L Q_L}{Q_H + Q_L + o_{HL} + o_{LH}} = \frac{R_H Q_H + R_L Q_L}{\lambda} \tag{7}$$

where R_H and R_L are the throughput of the output task in the higher and lower operating modes, respectively, and $R_H Q_H$ and $R_L Q_L$ are the number of produced tokens in the higher and lower operating modes, respectively. Similarly, the effective power consumption for the same operating mode switching is computed as follows:

$$P_{eff} = \frac{P_H Q_H + P_L Q_L + e_{HL} + e_{LH}}{\lambda} = \frac{P_H Q_H + P_L Q_L}{\lambda} + \frac{e_{HL} + e_{LH}}{\lambda} \tag{8}$$

where P_H and P_L are the power consumption of the higher and lower operating modes, respectively, and $P_H Q_H$ and $P_L Q_L$ are the energy consumption in the higher and lower operating modes, respectively.

Using the periodic switching scheme, described above, we can benefit from adopting the Dynamic Voltage and Frequency Scaling (DVFS) technique to exploit available static slack time in the scheduling of the application efficiently. Moreover, PEs are now enabled to run periodically at lower operating frequencies in the lower operating mode, as shown in Fig. 5(b). Therefore, we can achieve significant energy reduction. The shaded area in Fig. 5(b) shows the energy consumption corresponding to one period λ in our scheduling approach. Although the throughput constraint of the application is met by our proposed approach, the mentioned energy reduction comes at the expense of increasing the memory requirement. This is because, the application samples the input data stream and produces output data tokens in the higher operating mode more frequently than in the lower operating mode. As a consequence, this results in irregularity of sampling the input data stream and producing the output data tokens over the time. Therefore, to solve this irregular sampling/production problem, we need extra memory buffers for the input and output of our system, as shown in Fig. 6. The reason to use an output buffer, is to gather the produced tokens and release them regularly over the time in order to deliver the throughput constraint in a long run. In the same manner, to regularly sample the input data stream coming to the application, regardless of which operating mode the application is running in, we need an extra buffer at the input of the system. This is needed to distribute the sampled data regularly over the input data stream to guarantee certain sampling accuracy instead of sampling the input data stream differently in each operating mode leading to different accuracy in every operating mode.

According to the discussion above and looking at Fig. 5, there are some parameters in our switching approach that have to be determined, namely, the time duration to stay in the higher and lower operating modes (Q_H, Q_L), as well as switching costs $(o_{HL}, o_{LH}, e_{HL}, e_{LH})$. Therefore, in the rest of this section, we explain how to compute these parameters. We first explain how the operating modes are determined in Section V-A. Then, we compute the switching costs, o_{HL}, e_{HL}, o_{LH} and, e_{LH} and the time duration of staying in the higher and lower operating modes, Q_H and Q_L, that are key elements in our approach, in Section V-B and Section V-C, respectively. Finally, we compute the memory overhead (the inpute and output buffers in Fig. 6) associated with our scheduling framework in Section V-D.

Algorithm 1: Operating modes determination.

Input: A CSDF graph $G = (V, E)$.
Input: A set $\Psi = \{\Psi_1, \Psi_2, \cdots, \Psi_m\}$ of m identical PEs.
Input: A set $\theta = \{f_{min} = f_1, f_2, \cdots, f_n = f_{max}\}$ of n discrete operating frequencies for the PEs.
Input: A set $\Pi = \{\Pi_1, \Pi_2, \cdots, \Pi_m\}$ of task assignments on the PEs.
Output: A set γ of operating modes.

1 $\gamma \leftarrow \emptyset$;
2 Compute s using Eq. (2);
3 **while** *true* **do**
4   **for** $\forall\ \tau_i \in V$ **do**
5         $T_i = \frac{lcm(\vec{q})}{q_i} \cdot s$;
6     $\Gamma \leftarrow \{\tau_1, \tau_2, \cdots, \tau_{|v|}\}$;
7   **for** $\forall\ \Psi_j \in \Psi$ **do**
8   Compute a minimum operating frequency f_j such that $U_j = \frac{f_{max}}{f_j} \sum_{\forall \tau_i \in \Pi_j} \frac{C_i}{T_i} \leq 1$;
9   R = Compute the throughput of new schedule using Eq. (4);
10   P = Compute the power consumption of new schedule corresponding to the operating frequency set $\overrightarrow{f}$ using Eq. (6);
11     $m \leftarrow (R, P, \Gamma, \overrightarrow{f})$;
12   **if** $\{\neg\exists\ m_i \in \gamma : \overrightarrow{f_i} = \overrightarrow{f}\}$ **then**
13         $\gamma \leftarrow \gamma + m$;
14   **if** *the operating frequency of all PEs reaches to* f_{min} **then**
15   **return** γ;
16     $s = s + 1$;

A. Determining operating modes

The procedure for determining the operating modes is given in Algorithm 1. The inputs of this algorithm are a CSDF graph, a homogeneous platform consists of m PEs, a set of n discrete operating frequencies for the PEs, and a set of task assignments on the PEs. The output of this algorithm is a set of determined operating modes. First, Line 2 in this algorithm initializes the scaling parameter s using Eq. (2). Then, we use this initial value of s in Lines 4 and 5 to compute the minimum period of each task in the CSDF graph G using Eq. (1). By computing the minimum period of the tasks and using the theory in [6], we can derive the set of strictly periodic

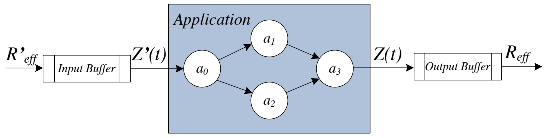


Fig. 6: Input and Output buffers.

tasks Γ in Line 6. Then, the minimum operating frequencies of the PEs are computed in Lines 7 and 8 in such a way that the schedulability of the assigned tasks on each PE is still preserved. To do so, a simple utilization check is performed where the total utilization of the assigned tasks on each PE has to be less than 1, for partitioned EDF, for the selected operating frequency. These operating frequencies are then stored in frequency set $\overrightarrow{f}$. In Lines 9 and 10, the throughput R and power consumption P of the periodic scheduling of task set Γ are computed using Eq. (4) and Eq. (6), respectively. Then, in Line 11 a new operating mode m that is characterized with the strictly periodic task set Γ corresponding to s, throughput R, power consumption P and the set of operating frequencies $\overrightarrow{f}$ for the PEs is created. Line 12 checks a condition whether to include the newly created mode to the set γ of operating modes. According to this condition, an operating mode is included to the set γ if there does not exist any operating mode in set γ with the same operating frequency set $\overrightarrow{f}$. This is because, if there exists such an operating mode in set γ, it corresponds to smaller s than the new operating mode. Therefore, the tasks in the existing operating mode have shorter periods where less unused slack time remains in the application schedule with the same operating frequency of the PEs. This selection strategy ensures that the static slack time in the application schedule is exploited more efficiently using the DVFS technique. Then, the explained procedure from Lines 4 to 13 repeats by incrementing s in Line 16 until the operating frequency of all PEs reaches to the minimum available operating frequency. Finally, the set γ of all determined operating modes is returned by this algorithm. As an example, following Algorithm 1, the operating modes for the (C)SDF graph in Fig. 1 are determined and listed in the Table I.

B. Switching costs $o_{HL}, o_{LH}, e_{HL}, e_{LH}$

In this section, we introduce the switching costs associated with our proposed switching scheduling scheme and explain the way we compute them.

(1) *Time Costs*: As shown in Fig. 5(a), we switch the operating mode in our approach between m_H and m_L. Mode switching has been investigated in [13] to determine when the tasks in the new operating mode after switching are allowed to start their execution assuming the tasks in each operating mode are scheduled using the framework in [6] as assumed in our paper as well. In [13], it has been shown that the tasks in the new operating mode can not be executed immediately. Therefore, their execution has to be offset by δ time units according to Eq. (7.18) in [13]. As a consequence, the system may not have any token production during the operating mode switching. In our case, the time cost of switching from the higher operating mode m_H to the lower operating mode m_L and vice versa using the offset δ from [13], can be computed as follows:

$$o_{HL} = S_{out}^{m_L} + \delta^{m_H \to m_L} - S_{out}^{m_H}, \; o_{LH} = S_{out}^{m_H} + \delta^{m_L \to m_H} - S_{out}^{m_L} \quad (9)$$

where $S_{out}^{m_L}$ and $S_{out}^{m_H}$ are the starting time of the output task in the lower and higher operating modes, respectively. This time cost is exactly the elapsed time between the finishing of the output task in one operating mode and the starting time of the output task in the other operating mode. However, since the operating frequencies of the PEs are changed during the switching, the computed δ offset in [13] may not be sufficient. This is because, the time that is needed for physically changing the operating frequencies in the PEs, denoted by Δ, is not considered in the computation in [13]. Apparently, the operating frequency must not be changed when the tasks in the higher operating mode are still executing in the system. Therefore, when the operating mode is switched from the higher operating mode to the lower operating mode, the operating frequency of the PEs must be changed after the end of the execution of the assigned tasks on the PEs in the higher operating mode. Similarly, when the operating mode is switched from the lower operating mode to the higher operating mode, the operating frequency of the PEs must be changed before the start of the execution of the assigned tasks on the PEs in the higher operating mode. This ensures that the tasks' job deadlines in both operating modes are met. For instance, for the proposed switching scheduling approach in Fig. 3, the time instants of changing the operating frequencies of Ψ_1 and Ψ_2 are show by the boxes with dotted pattern where the size of these boxes denotes the frequency switching delay Δ. The δ offset in [13] is a function of the tasks utilization. Therefore, to involve such switching delay Δ associated with DVFS into the δ offset, we have changed the utilization C_i/T_i to $(C_i+\Delta)/T_i$ of tasks τ_i in the lower operating mode that are executing when the operating frequency change happens. As a result, using Eq. (7.18) in [13], we can compute a sufficient δ with the new tasks utilization to make sure that the job deadlines of all tasks in both operating modes are still met during operating mode switching. Clearly, the last starting time instant of the new operating mode, using Eq. (7.18) in [13], can be when the execution of the previous operating mode is completely finished and the operating frequencies of the PEs are also changed. This is the safest starting time for the new operating mode while no extra schedulability test is needed as there are no overlapping execution between two operating modes. Using the method, explained above, for the proposed scheduling approach in Fig. 3, the starting offset of $\delta^{m_1 \to m_2} = 0$ can be computed for operating mode m_2 when the operating mode is switched from m_1 to m_2. Similarly, the starting offset of $\delta^{m_2 \to m_1} = 5$ can be computed for operating mode m_1 when the operating mode is switched from m_2 to m_1. Finally, the time cost of $o_{12} = 5$ and $o_{21} = 0$ can be computed using Eq. (9) for the operating mode switching from m_1 to m_2 and vice versa, respectively, as can be seen in Fig. 3.

(2) *Energy Costs*: By applying sufficient δ offset, as computed in Section V-B(1) above, tasks belonging to both the lower and higher operating modes may be concurrently executing on the PEs during mode switching. For instance, in Fig. 3 tasks in both operating modes m_1 and m_2 execute from time instant 26 to 36 and from time instant 67 to 77 when the operating mode is switched from m_1 and m_2 and vice versa, respectively. To meet the tasks' job deadlines in both operating modes, the PEs must run at the operating frequency corresponding to the higher operating mode during operating mode switching. Therefore, the total energy consumption of our proposed scheduling approach is more than the summation of energy consumption of operating modes m_H and m_L for the execution intervals of Q_H and Q_L time unit, respectively. As a result, we define e_{HL} and e_{LH} as extra energy consumption when the operating mode is switched from the high operating mode to the low operating mode and vice versa, respectively and we computed them using the following expressions:

$$e_{HL} = o_{HL} P_H \quad (10)$$

Algorithm 2: Finding the less power consuming pair of N_H and N_L.

```
Input: R_req, m_H, m_L.
Output: N_L, N_H.
1 Prev_Power = +∞;
2 N_L = 1;
3 while True do
4     Calculate N_H using Eq. (14) and R_req;
5     Power = Calculate power consumption by using Eq. (8);
6     if (Prev_Power − Power)/Prev_Power × 100 < 1 then
7         return N_L, N_H;
8     Prev_Power = Power;
9     N_L = N_L + 1;
```

$$e_{HL} = (S_{out}^{m_H} - o_{LH})(P_H - P_L) + o_{LH}P_H = S_{out}^{m_H}(P_H - P_L) + o_{LH}P_L \tag{11}$$

where the $S_{out}^{m_H}$ is the starting time of the output task in the higher operating mode. These energy costs are visualized by the hatched boxes in Fig. 5(b). These energy costs are overestimate using the above expressions because a single time instant is assumed for changing the operating frequency of all PEs in each operating mode switching. This time instant is referred by f_{switch} in Fig. 5(b). Note that we also include the energy overhead of DVFS into this energy costs.

C. Computing Q_H and Q_L

In our approach, we only allow the switching of operating modes at the graph iteration boundary. This means that the operating mode can be switched as soon as an application graph iteration is completed. Under this assumption, the time that an application is executed, in any operating mode, must be a multiple of the duration of one graph iteration. Therefore, the time that the application spends in the higher and lower operating modes can be defined as follows:

$$Q_H = N_H \cdot \alpha_H, \quad N_H \in \mathbb{N} \tag{12}$$
$$Q_L = N_L \cdot \alpha_L, \quad N_L \in \mathbb{N} \tag{13}$$

where N_H and N_L are the number of graph iterations in the higher and lower operating modes, respectively, and α_H and α_L are the graph iteration period (*hyper period*) in the higher and lower operating modes, respectively, as defined in Eq. (3). Finally, by substituting Eq. (12) and Eq. (13) in Eq. (7) and setting $R_{eff} = R_{req}$, the number of graph iterations to stay in the higher operating mode, N_H, can be derived as follows:

$$N_H = \left\lceil \frac{\alpha_L N_L (R_{req} - R_L) + R_{req}(o_{HL} + o_{LH})}{\alpha_H (R_H - R_{req})} \right\rceil \tag{14}$$

Note that, in the above equation, the ceiling function is used to derive an integer value for N_H such that the effective throughput R_{eff} can still meet the throughput constraint R_{req}. This fact is shown in Fig. 5(c) where our proposed effective throughput R_{eff} is higher than the throughput constraint R_{req}. Using Eq. (14), we have to derive the pair of N_H and N_L that satisfies the throughput constraint R_{req}. Clearly, Eq. (14) has more than one solution for the pair of N_H and N_L. Since all of these solutions have the same timing constraint, i.e., throughput constraint, the energy minimization is equivalent with the power minimization. Therefore, to find the less power consuming solution that consequently results in the less energy consumption, we can see from Eq. (8) that less power is consumed when we have an arbitrarily large period λ. This is because, the contribution of the switching power consumption $\frac{e_{HL}+e_{LH}}{\lambda}$ becomes negligible in the total power consumption P_{eff}. Moreover, as the period λ is enlarged, the delivered effective throughput R_{eff} using our switching scheme becomes closer to the throughput constraint R_{req}. This is because, as N_L increases in Eq. (14), the ceiling function becomes less contributing and the pair of N_L and N_H can produce the effective throughput R_{eff} more closely to the throughput constraint R_{req}. As a result, this leads to exploiting static slack times in the application scheduling more efficiently leading to further power reduction. Therefore, to find a valid solution for N_H and N_L that satisfies Eq. (14) and reduces the power consumption significantly, we search for the largest N_L where if it is further enlarged, the power reduction diminishes to less than one percent.

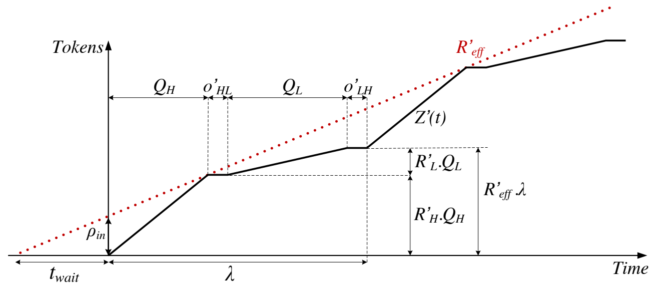


Fig. 7: Token consumption Function $Z'(t)$. Note that, $o_{HL} + o_{LH} = o'_{HL} + o'_{LH} = \delta^{H \to L} + \delta^{L \to H}$.

Algorithm 2 presents the pseudo-code of finding the less power consuming pair of N_H and N_L. The inputs of this algorithm are the throughput constraint and the higher and lower operating modes. The output of this algorithm is the pair of N_H and N_L. First, we initialize $N_L = 1$ in Line 2 and compute the corresponding N_H using Eq. (14) in Line 4. Then, we compute the power consumption corresponding to the derived pair of N_H and N_L using Eq. (8) in Line 5. We repeat this procedure by incrementing N_L in Line 9 until further power reduction compared to the previous iteration becomes less than one percent. This condition to terminate the procedure is given in Line 6. Then, the pair N_H and N_L is returned by the algorithm.

D. Memory Overhead

In this section, we compute the memory overhead that our approach incurs to the system, that is, the input and output buffers shown in Fig. 6. In order to compute the output buffer, we should consider Fig. 5(c) which shows the variable rate of token production $Z(t)$ delivered by our scheduling approach (the solid curve) and the needed constant rate of token production R_{eff} (the dotted line). When the application executes in the higher operating mode, it produces more token than needed while in the lower operating mode it produces less token than needed. Therefore, the purpose of using the output buffer is to accumulate the maximum difference between the number of produced and needed tokens over the time. This maximum difference is given by ρ_{out} in Fig. 5(c). Therefore, the size of the output buffer must be at least

$$B_{out} = \lceil \rho_{out} \rceil = \lceil Q_H (R_H - R_{eff}) \rceil \tag{15}$$

To compute the input buffer, the same method as for the output buffer can be used. To do so, we should consider Fig. 7 which shows the rate of sampling data tokens $Z'(t)$ in our scheduling approach given by the solid curve. As can be seen, the application samples the data tokens in the higher operating mode more often than in the lower operating mode. To solve such irregular sampling of the input data tokens over the time, we introduce a constant rate of sampling data tokens R'_{eff} give by the dotted line in Fig. 7 for the application and we compute it as follows:

$$R'_{eff} = \frac{R'_H Q_H + R'_L Q_L}{Q_H + Q_L + o'_{HL} + o'_{LH}} \tag{16}$$

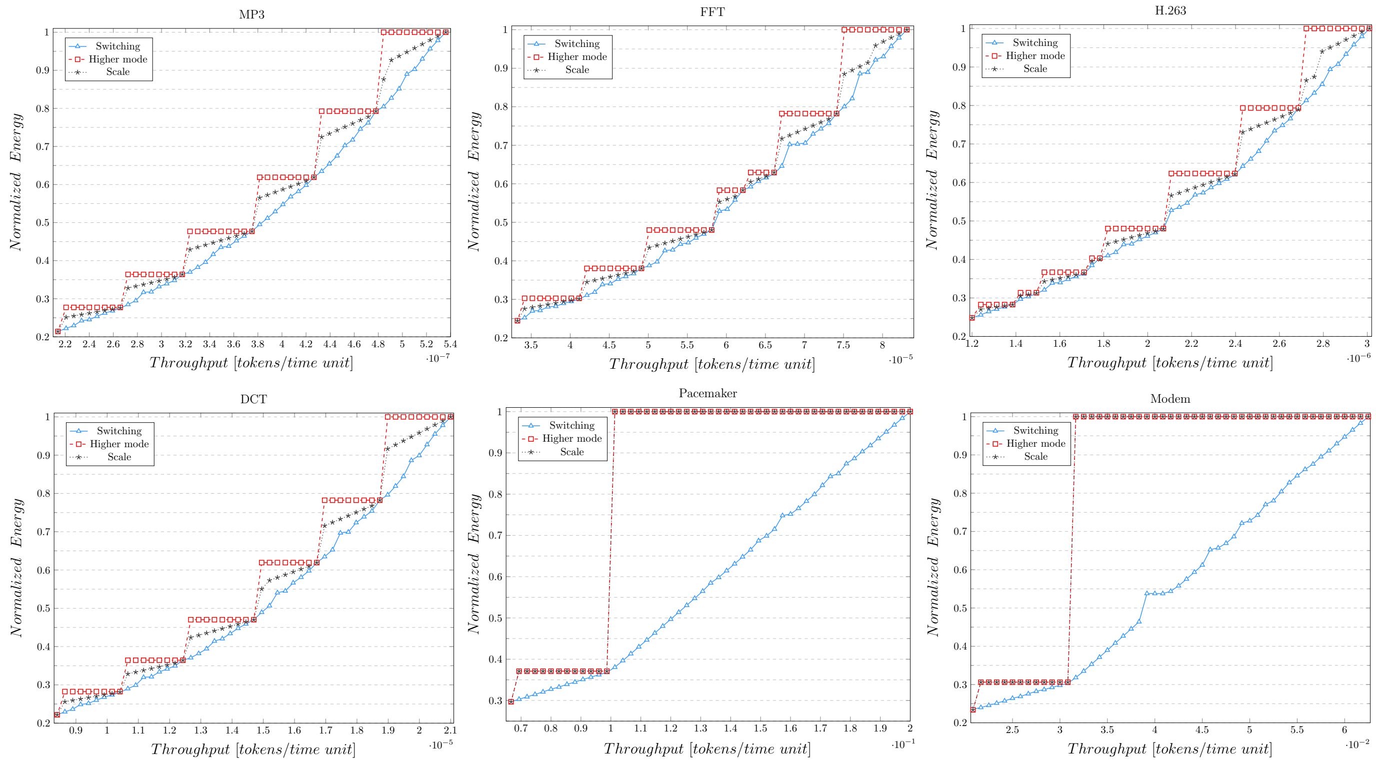


Fig. 8: Normalized energy consumption vs throughput constraints.

where R'_H and R'_L are the throughput of the input task in the higher and lower operating modes, $R'_H Q_H$ and $R'_L Q_L$ are the number of sampled data tokens from the input data stream in the higher and the lower operating modes, and o'_{HL} and o'_{LH} are the time overhead for the input task where no input data stream is sampled during switching from the higher to lower operating mode and vice versa, respectively. These time overheads are equal to the offset δ computed in [13]. Apparently, the constant sampling rate of R'_{eff} has to always provide sufficient sampled data tokens in both operating modes. Thus, to be able to guarantee this feature, the sampling of the input data stream with the rate of R'_{eff} must be started t_{wait} time units before the application starts executing, as shown in Fig. 7. This time can be computed as follows:

$$t_{wait} = \frac{(R'_H - R'_{eff})Q_H}{R'_{eff}} \tag{17}$$

Finally, the size of the input buffer must be at least

$$B_{in} = \left\lceil \rho_{in} \right\rceil = \left\lceil t_{wait} R'_{eff} \right\rceil = \left\lceil Q_H (R'_H - R'_{eff}) \right\rceil \tag{18}$$

where ρ_{in} is the maximum difference between the number of sampled and needed tokens, as shown in Fig. 7.

VI. Experimental Evaluation

In this section, we evaluate the effectiveness of our mode switching scheduling approach in terms of energy reduction. We compare our proposed switching scheduling approach referred as Switching in terms of energy reduction with two approaches: the straightforward approach of always selecting the operating mode whose throughput is the closest higher to the throughput constraint referred as Higher mode and the scaling approach, referred as Scale, explained in Section IV-A, which is the way of using the VFS technique similar to related works [4], [5], [10] in the context of [6]. In the following, we first explain our experimental setup in Section VI-A. Then, we present the experimental result in the Section VI-B.

TABLE II: Benchmarks used for evaluation.

Application	$\lvert V \rvert$	$\lvert E \rvert$
Discrete cosine transform (DCT) [14]	8	7
Fast Fourier transform (FFT) [14]	17	16
Data modem [15]	6	5
MP3 audio decoder [15]	14	18
H.263 video decoder [15]	4	3
Heart pacemaker [16]	4	3

A. Experimental setup

Benchmarks. We have performed experiments on 6 real-life streaming applications collected from the StreamIt benchmark suit [14], SDF^3 suit [15] and the individual research article [16], where all streaming applications are modeled as CSDF graphs. An overview of all streaming applications is given in Table II. In this table, $|V|$ denotes the number of tasks in a CSDF graph, while $|E|$ denotes the number of communication channels among tasks.

Architecture and Power Model. In the experiments, we use the power model presented in Section III-D. In this model, we adopt the power parameters of the Cortex A15 core given in [12], where these parameters have been obtained based on real measurements on the ODROID XU-3 platform [17]. The overhead of DVFS is set to values taken from [18], i.e., $10\mu s$ and $1\mu J$ are used for the delay and energy overhead associated with the physical change of the frequency in PEs, respectively. We evaluate the effectiveness of our scheduling approach on platforms with limited number of PEs. To this end, we compute the minimum number of PEs needed to schedule each benchmark application using a partitioned scheduling, e.g., First-Fit-Decreasing (FFD), when the maximum achievable throughput is required.

B. Experimental results

All experimental results are shown in Fig. 8 and Fig. 9, where the comparison is made for a set of selected application throughputs as throughput constraints. In Fig. 8, we show the different throughput constraints for the benchmarks on the x-axis and the normalized energy consumption of all three approaches is shown on the y-axis. As can be seen in Fig. 8, the energy reduction varies considerably among different applications and throughput constraints. When compared to the approach Higher mode, our approach Switching achieves significant energy reduction for all benchmarks. This energy reduction for the Modem, Pacemaker, DCT, MP3, FFT, and H.263 benchmarks can be up to 68.18%, 61.94%, 21.14%, 22.4%, 19.9%, and 19%, respectively. Compared to the approach Scale, our approach Switching can still reduce the energy consumption considerably. This energy reduction for the Modem, Pacemaker, DCT, MP3, FFT, and H.263 benchmarks can be up to 68.18%, 61.94%, 13.1%, 13.78%, 10.7%, and 12.07%, respectively. Among all these benchmarks, the Modem and Pacemaker are the two benchmarks for which our approach can obtain the largest energy reduction when compared to the approach Scale. This is mainly because the period of the tasks in Pacemaker and Modem benchmarks are quickly increased by applying the *task period scaling* approach, explained in Section IV-A. Therefore, a fewer number of operating modes can be determined for these benchmarks and no other application scheduling remains between the operating modes. As a consequence, the same application scheduling as the approach Higher mode is selected in the approach Scale to meet the throughput constraint in these benchmarks. This fact can be seen in Fig. 8 for Pacemaker and Modem benchmarks in which the result of the approach Scale and the approach Higher mode are overlapped on each other.

As can be seen in Fig. 8, for some throughput constraints no energy reduction is achieved by our approach Switching compared to approach Higher mode and approach Scale. This happens when the throughput constraints match with the throughput of one of the operating modes. In such cases, we simply select the operating mode whose throughput matches with the throughput constraint because mode switching is not needed.

Finally, the memory overhead, discussed in Section V-D, introduced by our scheduling approach, is given in Fig. 9. In this figure, the x-axis shows the different benchmarks while the y-axis shows the maximum total buffer size for each benchmark which is the sum of the maximum size of input and output buffers shown Fig. 6. In this figure, we only show the maximum total buffer size collected among all throughput constraints for each benchmark. The memory overhead for the H.263 benchmark is 1.7 MB whereas for the other benchmarks it is less than 83 KB. Given such maximum memory overhead and given the size of memory available in modern embedded systems, we can conclude that the memory overhead introduced by our scheduling approach is acceptable.

VII. Conclusion

In this paper, we propose a novel periodic scheduling approach for streaming applications. This approach can meet a system throughput requirement at long run by periodically switching between two selected operating modes. Contrary to related approach, our scheduling approach benefits from using multiple voltage and frequency levels at run-time leading to more efficient static slack time utilization while the throughput requirement is still satisfied. The experimental results, on

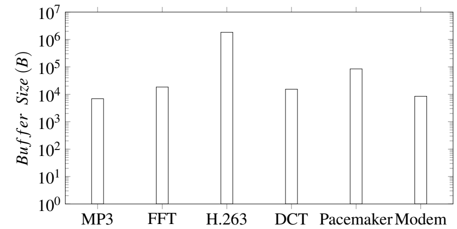


Fig. 9: Maximum total buffer sizes needed in our scheduling approach for different benchmarks. Note that the y axis has a logarithmic scale.

a set of 6 real-life streaming applications, show that our approach reduces the energy consumption up to 68% while meeting the same throughput requirement when compared to related energy minimization scheduling approaches. However, for some throughput constraints that match with the throughput of one of the operating modes, no energy reduction can be achieved by our approach compared to the related approaches. This is because, in such cases, we can simply select the operating mode whose the throughput matches with the throughput constraint instead of adopting mode switching technique. Finally, although the throughput constraint of the applications is met by the proposed approach, the mentioned energy reductions come at the expense of increased memory requirements.

References

[1] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[2] G. Bilsen et al. Cycle-static dataflow. *Signal Processing, IEEE Transactions on*, 44(2):397–408, 1996.

[3] G. Chen, K. Huang, and A. Knoll. Energy optimization for real-time multiprocessor system-on-chip with optimal dvfs and dpm combination. *ACM Transactions on Embedded Computing Systems (TECS)*, 2014.

[4] M. Damavandpeyma, S. Stuijk, T. Basten, M. Geilen, and H. Corporaal. Throughput-constrained dvfs for scenario-aware dataflow graphs. In *RTAS*, 2013.

[5] A. K. Singh et al. Mapping on multi/many-core systems: survey of current and emerging trends. In *DAC*, 2013.

[6] M. Bamakhrama and T. Stefanov. On the hard-real-time scheduling of embedded streaming applications. *DAES*, 17(2):221–249, 2013.

[7] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys (CSUR)*, 43(4):35, 2011.

[8] A. Nelson, O. Moreira, A. Molnos, S. Stuijk, B. T. Nguyen, and K. Goossens. Power minimisation for real-time dataflow applications. In *DSD*, 2011.

[9] P. Huang, O. Moreira, K. Goossens, and A. Molnos. Throughput-constrained voltage and frequency scaling for real-time heterogeneous multiprocessors. In *SAC*, 2013.

[10] J. Zhu, I. Sander, and A. Jantsch. Energy efficient streaming applications with guaranteed throughput on mpsocs. In *EMSOFT*, 2008.

[11] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 1973.

[12] D. Liu, J. Spasic, G. Chen, and T. Stefanov. Energy-efficient mapping of real-time streaming applications on cluster heterogeneous mpsocs. In *ESTIMedia*, 2015.

[13] J. T. Zhai. *Adaptive streaming applications: analysis and implementation models*. PhD thesis, Leiden Institute of Advanced Computer Science (LIACS), Leiden University, 2015.

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

[15] S. Stuijk et al. Sdf3: Sdf for free. In *ACSD*, 2006.

[16] R. Pellizzoni, P. Meredith, M. Nam, M. Sun, M. Caccamo, and L. Sha. Handling mixed-criticality in soc-based real-time embedded systems. In *EMSOFT*, 2009.

[17] ODROID. “http://www.hardkernel.com/”.

[18] S. Park et al. Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(5):695–708, 2013.