Generalized strictly periodic scheduling analysis, resource optimization, and implementation of adaptive streaming applications

Niknam, S.

Chapter 5

Energy-Efficient Scheduling of Streaming Applications

Sobhan Niknam, Todor Stefanov. "Energy-Efficient Scheduling of Throughput-Constrained Streaming Applications by Periodic Mode Switching". *In Proceedings of the 17th IEEE International Conference on Embedded Computer Systems: Architectures, MOdeling, and Simulation (SAMOS)*, Samos, Greece, July 17 - 20, 2017.

IN this chapter, we present our energy-efficient periodic scheduling approach, which corresponds to the third research contribution, briefly introduced in Section 1.5.3, to address the research question **RQ2(B)**, described in Section 1.4.2. The remainder of this chapter is organized as follows. Section 5.1 introduces, in more details, the problem statement and the addressed research question. It is followed by Section 5.2, which gives a summary of the contributions presented in this chapter. Section 5.3 gives an overview of the related work. Section 5.4 introduces the extra background material needed for understanding the contributions of this chapter. Section 5.5 gives a motivational example. Section 5.6 presents the proposed scheduling approach. Section 5.7 presents the experimental evaluation of the proposed scheduling approach. Finally, Section 5.8 ends the chapter with conclusions.

5.1 Problem Statement

As mentioned in Section 1.1, energy efficiency has become a critical challenge for the design of modern embedded systems, especially for those which are battery-powered. To address the energy efficiency challenge, many approaches

have been proposed in the past decades by several research communities [11]. These approaches mostly exploit the Voltage and Frequency Scaling (VFS) mechanism that is widely adopted in modern processors. The general idea behind these approaches is to exploit available idle, i.e., slack, time in the schedule of an application in order to slow down the execution of tasks of the application, by running processors at a lower voltage and operating clock frequency, using the VFS mechanism and to reduce the energy consumption while satisfying a given throughput requirement for the application.

Concerning the SPS framework, briefly described in Section 2.3, some heuristic approaches have been proposed in [25, 55, 80] to find an energy-efficient task mapping and scheduling using the VFS mechanism. Recall from Equation (2.12) that under the SPS framework, briefly described in Section 2.3, the period of real-time periodic tasks corresponding to the actors of a CSDF graph can be enlarged by taking any $s \geq \check{s} \in \mathbb{N}$ as long as a given application throughput requirement is satisfied. This period enlargement under the SPS framework, however, results in a set of application schedules that can only satisfy a discreet set of application throughputs, as the timing requirement. Therefore, given a required application throughput that is not in this set of guaranteed throughputs by the SPS framework, the schedule that provides the closest higher throughput to the required one must be selected from the set. As a consequence, this reduces the amount of available slack time in the application schedule, that can be potentially exploited using the VFS mechanism to reduce the energy consumption, and limits the energy-efficiency of the approaches in [25, 55, 80]. Thus, in this chapter, we investigate the possibility to exploit more slack time in the schedule of an application, modeled as a CSDF graph, under the SPS framework with a given throughput requirement using the VFS mechanism to achieve more energy efficiency.

5.2 Contributions

In order to address the problem described in Section 5.1, in this chapter, we propose a novel energy-efficient scheduling approach that combines the VFS mechanism [71] and the SPS framework [8] in a sophisticated way. In this novel approach, the execution of an application is periodically switched at run-time between a few off-line determined energy-efficient schedules, called *operating modes*, to satisfy a given throughput requirement at a long run. As a result, this approach can reduce the energy consumption significantly by exploiting the slack time in the application schedule more efficiently using the Dynamic Voltage and Frequency Scaling (DVFS) mechanism [50], where

multiple operating frequencies are computed at design-time for the processors to be used at run-time. More specifically, the main contributions of this chapter are as follows:

- 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 a set of discrete operating modes and a given throughput requirement, we have devised an energy-efficient periodic scheduling approach which allows streaming applications to switch their execution periodically between operating modes at run-time to satisfy the throughput requirement at a long run. Using this specific switching scheme, we can benefit from adopting the DVFS mechanism to exploit the available static slack time in an application schedule efficiently.
- The experimental results, on a set of real-life streaming applications, show that our scheduling approach can achieve energy reduction by up to 68% depending on the application and the throughput requirement compared to the straightforward way of applying VFS as done in related works.

5.3 Related Work

Several approaches aiming at reducing the energy consumption of streaming applications have been presented in the past decades. Among these approaches, [26, 42, 61, 74, 96] 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 [26, 61], per-core [42, 74, 96] or global [42] VFS.

The approaches in [26, 42, 61], formulate the energy optimization problem as a mixed integrated linear programming (MILP) problem to integrate the VFS capability of processors 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) [26, 42] or a Homogeneous SDF (HSDF) graph [61] derived by applying a certain transformation on an initial SDF graph. Therefore, these approaches cannot be directly applied to streaming applications modeled with more expressive MoCs, e.g., (C)SDF as considered in our work. In addition, transforming a graph from SDF to HSDF is a crucial step in [61] where the number of tasks in the streaming application can exponentially grow. This

growth of the application in terms of the 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 [42] uses per-core VFS where the off-line computed operating frequencies of processors are fixed at run-time and cannot be changed. In contrast, our approach uses DVFS where a sequence of frequency changes which is computed off-line is used on the processors during execution at run-time while satisfying the throughput requirement. As a result, the DVFS mechanism enables our approach to exploit the available static slack time in the application schedule more efficiently for better energy reduction. The approaches in [26, 61] use a fine-grained DVFS, i.e., per-task VFS, where the operating frequency of processors can be changed before executing each task. Fine-grained DVFS, like in [26, 61], 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 processors are changed at the granularity of graph iterations to avoid the large overhead associated with the operating frequency changes.

The approaches in [74, 96] perform energy reduction directly on an SDF graph. To this end, the approaches in [74, 96] perform design space exploration (DSE) at design time to find an energy-efficient schedule (in a self-timed manner) of an SDF graph mapped on an MPSoC platform with per-core VFS capability such that a given throughput requirement is satisfied. However, as shown in the motivation example in Section 5.5, applying VFS in a similar way as in [74, 96] for streaming applications scheduled using the SPS framework [8] is not energy-efficient. Compared to the approaches in [74, 96], our approach is different in two aspects. First, these approaches use self-timed scheduling for which analysis techniques suffer from a complex DSE. In contrast, we use the SPS framework that enables the utilization of many scheduling algorithms with fast analysis techniques from the classical hard real-time scheduling theory [29]. Second, these approaches use per-core VFS to exploit static slack time in the application schedule. In contrast, our approach uses a coarse-grained DVFS. As a result, the processors are able to run periodically at lower operating frequencies by exploiting available static slack time more efficiently which can result in lower energy consumption.

5.4 Background

In this section, we define the system model and present the power model considered throughout this chapter.

5.4.1 System Model

In this section, we define the system model used in this chapter. The considered MPSoC platforms in this chapter are homogeneous, i.e., a platform contains a set $\Pi = \{\pi_1, \pi_2, \cdots, \pi_m\}$ of m identical processors with distributed memories. We assume that processors are endowed with the VFS capability. In this regard, we assume that each processor supports only a discrete set $\theta = \{f_{min} = f_1, f_2, \cdots, f_n = f_{max}\}$ of n operating frequencies and different processors 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.

5.4.2 Power Model

This section defines the power model used in this chapter. According to [55], the power consumption of a (fully utilized) processor can be computed by the following equation:

$$P(f) = \alpha f^b + \beta$$

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 α, b, and β are dependent on the platform and they are determined in [55] by performing real measurements on a real MPSoC platform. When all tasks are allocated on processors of platform Π, the power consumption of processor π_j can be computed by the following equation:

$$P_j = \alpha \cdot f_{\pi_j}^b \cdot \frac{f_{max}}{f_{\pi_j}} \sum_{\forall \tau_i \in {}^m\Gamma_j} \frac{C_i}{T_i} + \beta \tag{5.1}$$

where $f_{\pi_j} \in \theta$ is the operating frequency of π_j and ${}^m\Gamma_j \in {}^m\Gamma$ represent the set of tasks allocated on processor π_j. Therefore, the energy consumption of π_j within one graph iteration period (hyper period) is $E_j = H \cdot P_j$ and the energy consumption of the platform within one iteration period is $E = \sum_{\forall \pi_j \in \Pi} H \cdot P_j$.

5.5 Motivational Example

In this section, we motivate the necessity of devising a new energy-efficient scheduling approach using the VFS mechanism in the context of the SPS

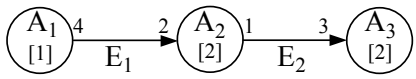


Figure 5.1: *An SDF graph G.*

framework [8]. To do so, this motivational example consists of two parts. In the first part, we show that a straightforward way of applying the VFS mechanism in the context of the SPS framework is not energy efficient. Then, in the second part, we show how we can schedule an application more energy efficient using our novel periodic scheduling approach.

5.5.1 Applying VFS Similar to Related Works

Let us consider a simple streaming application modeled as the SDF graph G shown in Figure 5.1. This graph has three actors $\mathcal{A} = \{A_1, A_2, A_3\}$ with worst-case execution times $C_1 = 1$, $C_2 = 2$, and $C_3 = 2$ at the maximum processor operating clock frequency. The repetition vector of this graph, according to Theorem 2.1.1, is $\vec{q} = [3, 6, 2]^T$. By applying the SPS framework for graph G, the task set $\Gamma = \{\tau_1 = (C_1 = 1, T_1 = 4, S_1 = 0, D_1 = 4), \tau_2 = (2, 2, 4, 2), \tau_3 = (2, 6, 10, 6)\}$ of three IDP tasks can be derived. Note that the derived periods of the tasks are the minimum periods by using the scaling factor $s = \check{s} = \lceil \frac{12}{6} \rceil = 2$ in Equation (2.12). Based on these tuples, a strictly periodic schedule, as shown in Figure 5.2(a), can be obtained for this graph. Using Equation (2.15), the throughput of this schedule can be computed as $\mathcal{R} = \frac{1}{T_3} = \frac{1}{6}$. The minimum number of processors needed for this schedule under partitioned First-Fit Decreasing (Utilization) EDF (FFD-EDF) is two. Therefore, we consider a homogeneous MPSoC platform $\Pi = \{\pi_1, \pi_2\}$ containing two processors, where we allocate task τ_2 on processor π_1 and tasks τ_1 and τ_3 on processor π_2, i.e., ${}^2\Gamma = \{{}^2\Gamma_1 = \{\tau_2\}, {}^2\Gamma_2 = \{\tau_1, \tau_3\}\}$.

So far, we have assumed that the tasks run at the maximum operating frequency of the processors. Let us assume that each processor can only support a discrete set $\theta = \{1/4, 1/2, 3/4, 1\}$(GHz) of four operating frequencies. In order to make this schedule more energy efficient, we use the VFS mechanism to 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 the processors. For this example, we can only decrease the operating frequency of processor π_2 to 3/4 GHz while still satisfying all timing requirements, i.e., job deadlines shown as down arrows in Figure 5.2(a). This slowing down of the execution of tasks is visualized by extending the gray boxes with the

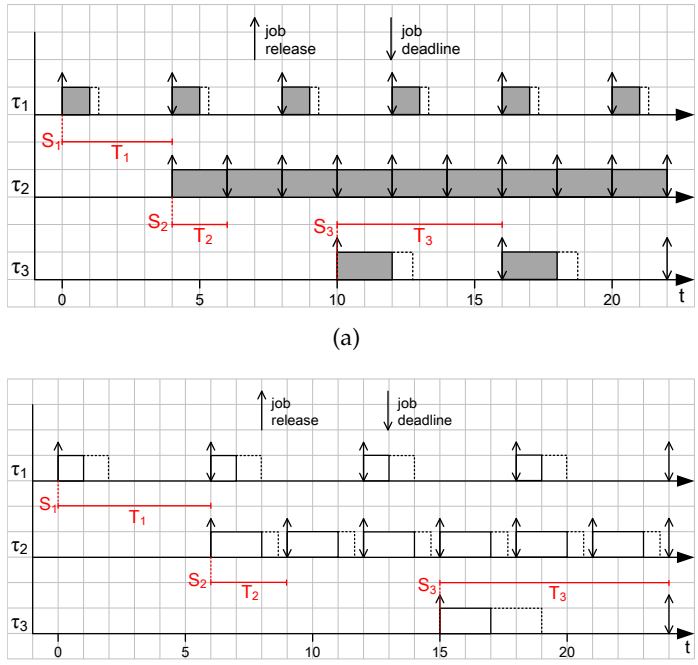


Figure 5.2: *The (a) SPS and (b) scaled SPS of the (C)SDF graph G in Figure 5.1. Up arrows represent job releases, down arrows represent job deadlines. Dotted rectangles show the increase of the tasks execution time when using the VFS mechanism.*

dotted boxes in Figure 5.2(a). Using Equation (5.1), the power consumption of this schedule is 0.61 mW. The energy consumption of this schedule for a period of 36 time units, which is equivalent to 3 graph iterations, is 21.96 mJ.

To further reduce the power consumption by decreasing the operating frequency of processors, more static slack time is needed to be created in the application schedule. To do so, we can derive larger periods for tasks by using any integer scaling factor $s > \check{s} = 2$ in Equation (2.12). We refer to this approach as *period scaling* in this chapter. In this way, if we take $s = 3$, a new schedule can be derived using the SPS framework, as shown in Figure 5.2(b), with throughput $\mathcal{R} = \frac{1}{T_3} = \frac{1}{9}$. As a result, there is more static slack time available in the application schedule which enables the processors π_1 and π_2 to run at lower operating frequencies of $3/4$ GHz and $1/2$ GHz, respectively. This is visualized by extending the white boxes with the dotted boxes in Figure 5.2(b). Using Equation (5.1), the power consumption of this schedule is 0.43 mW. The energy consumption of this schedule for a period

of 36 time units, which is equivalent to 2 graph iterations, is 15.48 mJ. As a result, the energy consumption is reduced by 29.5% using the schedule in Figure 5.2(b) corresponding to $s = 3$ compared to the schedule in Figure 5.2(a) corresponding to $s = 2$ for the same time period at the expense of decreasing the application throughput from $1/6$ to $1/9$. By increasing the value of scaling factor s and enlarging the periods of tasks as much as possible such that the corresponding schedule still satisfies a given throughput requirement, we can apply the VFS mechanism in the straightforward way, described above, similar to the related works [74, 96]. Therefore, the maximum created static slack time in the application schedule can be exploited using the VFS mechanism to reduce the energy consumption as much as possible.

Now, assume that a throughput requirement of $1/8$ has to be satisfied. Following the period scaling approach, described above, the schedule corresponding to $s = 2$ with the throughput of $1/6$, shown in Figure 5.2(a), must be selected to satisfy the throughput requirement of $1/8$. However, this schedule is not the most energy-efficient one. This is because, although the throughput requirement of $1/8$ is satisfied, more energy is consumed as a result of delivering higher throughput than needed.

5.5.2 Our Proposed Scheduling Approach

In this section, we introduce our novel energy-efficient scheduling approach for graph G in Figure 5.1 that satisfies the same throughput requirement of $1/8$ while consuming less energy compared to the scheduling approach explained in Section 5.5.1. In our approach, among all possible application schedules corresponding to different values of scaling factor s to enlarge periods, we select only Pareto optimal schedules and form a set γ of schedules called *operating modes*. For instance, the set $\gamma = \{\mathrm{SI}^1, \mathrm{SI}^2, \mathrm{SI}^3, \mathrm{SI}^4, \mathrm{SI}^5\}$ of five operating modes for graph G is given in Table 5.1. In this table, every row shows an operating mode with the iteration period H, the operating frequencies of the two processors (f_{π_1}, f_{π_2}), the pair of throughput and power consumption $(\mathcal{R}, P)$, and the energy consumption corresponding to the operating 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 the iteration periods H of all operating modes. As can be seen in this column, the energy consumption of the operating modes is being reduced by slowing down the application execution during this common period of time. The value of scaling factor s corresponding to each operating mode is also given in the first column. For instance, operating mode SI^4 is the application schedule corresponding to $s = 5$ that delivers throughput of $1/15$. In this schedule, processors π_1 and

Table 5.1: *Operating modes for graph G*

Mode	H	f_{π_1}	f_{π_2}	$(\mathcal{R}\ [\frac{Token}{Time\ units}],\ P\ [mW])$	$E\ [mJ]$
SI^1 $(s=2)$	12	1	3/4	$(1/6, 0.61)$	439.2
SI^2 $(s=3)$	18	3/4	1/2	$(1/9, 0.43)$	309.6
SI^3 $(s=4)$	24	1/2	1/2	$(1/12, 0.36)$	259.2
SI^4 $(s=5)$	30	1/2	1/4	$(1/15, 0.34)$	244.8
SI^5 $(s=8)$	48	1/4	1/4	$(1/24, 0.31)$	223.2

π_2 must operate at frequencies of 1/2 GHz and 1/4 GHz in order to meet all task's job deadlines. The power consumption of this schedule is 0.34 mW and the energy consumption of this schedule for 720 time units is 244.8 mJ.

Looking at set γ of operating modes in Table 5.1, the throughput requirement of 1/8, we consider in this example, is between the throughput of operating modes SI^1 and SI^2. Therefore, we propose the idea of periodically switching the application execution between operating modes SI^1 and SI^2 to satisfy the throughput requirement. Such a periodic switching schedule is depicted for one period in Figure 5.3, where the application executes for three graph iterations according to the schedule of operating mode SI^1 and two graph iterations according to the schedule of operating mode SI^2. Different graph iterations are separated by dotted and dashed lines for consecutive executions of the application in operating mode SI^1 and SI^2, respectively, in Figure 5.3. Note that this schedule repeats periodically every 77 time units, as shown in Figure 5.3 ($Q_1 + Q_2 + o_{12} = 77$). In one period, task τ_3 executes 10 times in total during 77 time units, meaning that throughput of 10/77=1/7.7 is delivered at a long run that is more closer to the throughput requirement of 1/8 compared to the throughput of 1/6 delivered as a result of the schedule in Figure 5.2(a). More importantly, the energy consumption of our proposed novel schedule in Figure 5.3 for a period of 924 time units, which is the *least common multiply* of the period of our schedule (77 time units) and the iteration period of the schedule in Figure 5.2(a) (12 time units), is 496.68 mJ. The energy consumption of the schedule in Figure 5.2(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 requirement of 1/8 has to be satisfied. The energy reduction of our proposed schedule, referred as Switching, compared to the scheduling approach explained in Section 5.5.1, referred as Scale, for a wide range of throughput requirements is given in Figure 5.4. In this figure, the x-axis shows different throughput requirements for graph G in Figure 5.1 while the y-axis shows the normalized energy consumption. From Figure 5.4, we can see that our proposed scheduling approach

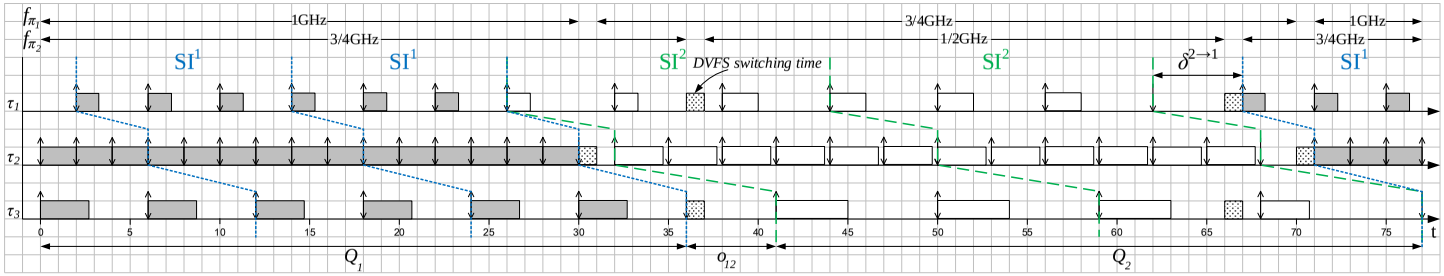


Figure 5.3: *Our proposed periodic schedule of graph G in Figure 5.1. In this schedule, graph G periodically executes according to schedules of operating mode SI^1 and operating mode SI^2 in Figure 5.2(a) and Figure 5.2(b), respectively. Note that this schedule repeats periodically.* $o_{12} = 5$ *and* $o_{21} = 0$.

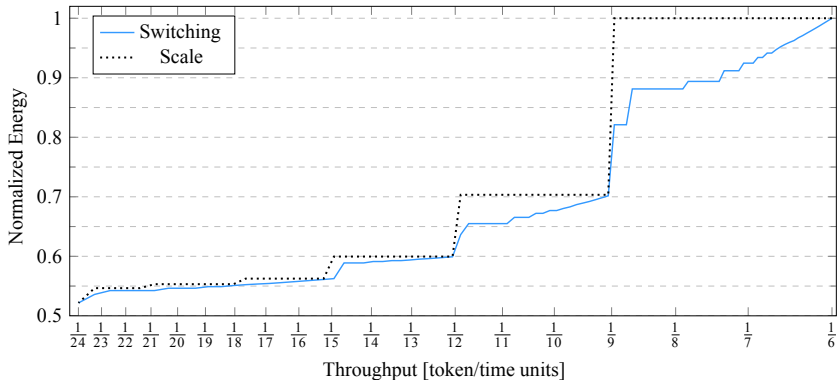


Figure 5.4: *Normalized energy consumption of the scaled scheduling and our proposed scheduling of the graph G in Figure 5.1 for a wide range of throughput requirements.*

Switching can reduce the energy consumption significantly compared to Scale for a large set of throughput requirements.

Note that our proposed scheduling approach uses the DVFS mechanism. This is because, processors 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 processors are changed accordingly. The way of changing the operating frequencies of the processors, for our example, is shown by the horizontal arrows on top of Figure 5.3. Note that we also consider the switching time cost of the DVFS mechanism in our analysis that is shown by the boxes with dotted pattern in Figure 5.3.

From the above example, we can see the necessity and usefulness of our novel scheduling approach, presented in detail in Section 5.6, to obtain more energy-efficient application schedule when the VFS mechanism is used in the context of the SPS framework.

5.6 Proposed Scheduling Approach

In this section, we describe our proposed energy-efficient periodic scheduling approach 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 requirement, there may exist an operating mode whose throughput matches the throughput

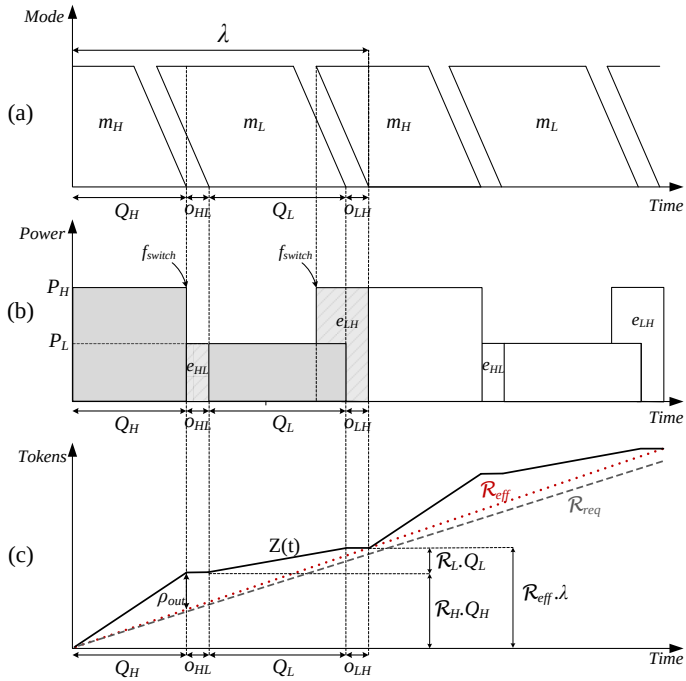


Figure 5.5: *(a) Switching scheme, (b) Associated energy consumption of the switching scheme and (c) Token production function Z(t).*

requirement. 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 requirement, referred as *higher operating mode* (SI^H) and *lower operating mode* (SI^L), respectively. Then, we satisfy the throughput requirement at a long run by periodically switching the execution of the application between these two operating modes.

A general overview of our proposed switching scheme for the execution of an application between the higher and lower operating modes is illustrated in Figure 5.5. The periodic execution of the application between the higher and lower operating modes in our approach is shown in Figure 5.5(a) and the period of switching is denoted by λ. The associated energy consumption and token production of the application caused by our switching scheme corresponding to Figure 5.5(a) are also shown in Figure 5.5(b) and Figure 5.5(c), respectively. According to Figure 5.5(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 $\mathcal{R}_H$ and power consumption P_H. Then, in the second part,

the execution of the application switches to the lower operating mode SI^L. However, this switching cannot happen immediately and it takes some time, denoted as o_{HL}, before the application can 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 Figure 5.5(b) and Figure 5.5(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 $\mathcal{R}_L$ and P_L, respectively. Finally, in the fourth part, the application switches again to the higher operating mode SI^H for the next period of λ. However, this switching cannot happen immediately and it takes some time that is denoted by o_{LH}. During the switching time 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 Figure 5.5(a), the application generates a number of tokens in total, see the curve $Z(t)$ in Figure 5.5(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 $\mathcal{R}_{eff}$ in a long run. The curves corresponding to the token production $Z(t)$ in our switching scheme and the effective throughput of $\mathcal{R}_{eff}$ are shown in Figure 5.5(c) with a solid line and a dotted line, respectively. The throughput requirement $\mathcal{R}_{req}$ is also shown with a dashed line in this figure. Therefore, to satisfy the throughput requirement, we have to always keep the effective throughput $\mathcal{R}_{eff}$ above the throughput requirement $\mathcal{R}_{req}$. This ensures that the number of produced tokens at any time instant is greater than or equal to what is needed.

Considering Figure 5.5(c), the effective throughput obtained by executing the application in operating mode SI^H for Q_H time units and operating mode SI^L for Q_L time units is computed by the following expression:

$$\mathcal{R}_{eff} = \frac{\mathcal{R}_H Q_H + \mathcal{R}_L Q_L}{Q_H + Q_L + o_{HL} + o_{LH}} = \frac{\mathcal{R}_H Q_H + \mathcal{R}_L Q_L}{\lambda} \tag{5.2}$$

where $\mathcal{R}_H$ and $\mathcal{R}_L$ are the throughputs of the application in the higher and lower operating modes, respectively, and $\mathcal{R}_H Q_H$ and $\mathcal{R}_L Q_L$ 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:

$$P_{eff} = \frac{P_H Q_H + P_L Q_L + e_{HL} + e_{LH}}{\lambda} = \frac{P_H Q_H + P_L Q_L}{\lambda} + \frac{e_{HL} + e_{LH}}{\lambda} \tag{5.3}$$

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.

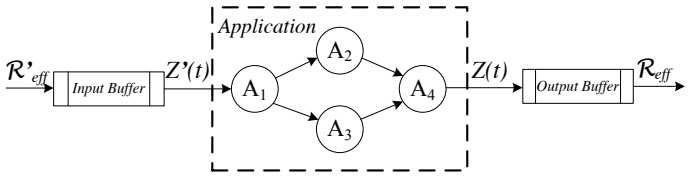


Figure 5.6: *Input and Output buffers.*

Using the periodic switching scheme, described above, we can benefit from adopting the DVFS mechanism to exploit the available static slack time in the application schedule more efficiently that can reduce the energy consumption considerably. The shaded area in Figure 5.5(b) shows the energy consumption corresponding to one period λ in our scheduling approach. Although the throughput requirement of the application is satisfied 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 time. Therefore, to solve this irregular sampling/production problem, we need extra memory buffers for the input and output of the application, as shown in Figure 5.6. The reason to use an output buffer is to gather the produced tokens and release them regularly over time in order to deliver the throughput requirement 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 application. This buffer 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 Figure 5.5, there are some parameters in our scheduling 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 5.6.1. 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 5.6.2 and Section 5.6.3, respectively. Finally, we compute the memory overhead (the input and output buffers in Figure 5.6)

Algorithm 2: Operating modes determination.

Input: A CSDF graph $G = (\mathcal{A}, \mathcal{E})$.
Input: A set $\Pi = \{\pi_1, \pi_2, \cdots, \pi_m\}$ of m identical processors.
Input: A set $\theta = \{f_{min} = f_1, f_2, \cdots, f_n = f_{max}\}$ of n discrete operating frequencies for the processors.
Input: A set ${}^m\Gamma = \{{}^m\Gamma_1, {}^m\Gamma_2, \cdots, {}^m\Gamma_m\}$ of task allocation on the processors.
Output: A set γ of operating modes.

1 $\gamma \leftarrow \emptyset$;
2 Compute $s = \check{s}$ using Equation (2.13);
3 **while** *true* **do**
4 **for** $\forall\ \tau_i \in \Gamma$ **do**
5 $T_i = \frac{\text{lcm}(\vec{q})}{q_i} \cdot s$;
6 **for** $\forall\ \pi_j \in \Pi$ **do**
7 Compute a minimum operating frequency f_{π_j} such that $u_{\pi_j} = \frac{f_{max}}{f_{\pi_j}} \sum_{\forall \tau_i \in {}^x\Gamma_j} \frac{C_i}{T_i} \leq 1$;
8 $\mathcal{R}$ = Compute the throughput of new schedule using Equation (2.15);
9 P = Compute the power consumption of new schedule corresponding to the operating frequency set $\vec{f}$ using Equation (5.1);
10 SI $\leftarrow (\mathcal{R}, P, \Gamma, \vec{f})$;
11 **if** $\{\neg\exists\ SI^i \in \gamma : \vec{f}_i = \vec{f}\}$ **then**
12 $\gamma \leftarrow \gamma +$ SI;
13 **if** *the operating frequency of all processors reaches to* f_{min} **then**
14 **return** γ;
15 $s = s + 1$;

associated with our scheduling approach in Section 5.6.4.

5.6.1 Determining Operating Modes

The procedure for determining the operating modes is given in Algorithm 2. The inputs of this algorithm are a CSDF graph G, a homogeneous platform Π containing m processors, a set θ of n discrete operating frequencies for the processors, and a set ${}^m\Gamma$ of task allocations on the processors. The output of this algorithm is a set γ of determined operating modes. First, Line 2 in this

algorithm initializes the scaling factor $s = \check{s}$ using Equation (2.13). Then, we use this initial value of s in Lines 4 and 5 to compute the minimum period of tasks corresponding to the actors in the CSDF graph G using Equation (2.12). Then, the minimum operating frequencies of the processors are computed in Lines 6 and 7 in such a way that the schedulability of the allocated tasks on each processor is still preserved. To do so, a simple utilization check is performed where the total utilization of the allocated tasks on each processor has to be less than or equal to 1, for partitioned EDF, for the selected operating frequency. These operating frequencies are then stored in the frequency set $\vec{f}$. In Lines 8 and 9, the throughput $\mathcal{R}$ and power consumption P of the periodic scheduling of task set Γ are computed using Equation (2.15) and Equation (5.1), respectively. Then, in Line 10 a new operating mode SI that is characterized with the strictly periodic task set Γ corresponding to s, throughput $\mathcal{R}$, power consumption P, and the set of operating frequencies $\vec{f}$ for the processors is created. Line 11 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, in Line 12, if there does not exist any operating mode in set γ with the same operating frequency set $\vec{f}$. 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 processors. This selection strategy ensures that the static slack time in the application schedule is exploited more efficiently using the DVFS mechanism. Then, the explained procedure from Lines 4 to 12 repeats by incrementing s in Line 15 until the operating frequency of all processors reaches to the minimum available operating frequency. Finally, the set γ of all determined operating modes is returned by this algorithm in Line 14. As an example, following Algorithm 2, the operating modes for the graph G shown in Figure 5.1 are determined and listed in the Table 5.1.

5.6.2 Switching Costs $o_{HL}, o_{LH}, e_{HL}, e_{LH}$

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

(1) *Time Costs*: As shown in Figure 5.5(a), we switch the operating mode in our approach between SI^H and SI^L. In Section 2.4, mode switching has been investigated for an MADF graph to determine the earliest time that tasks in the new operating mode can start their execution during mode switching instants.

In Section 2.4, it has been shown that the tasks in the new operating mode cannot be executed immediately. Therefore, their execution has to be offset by δ time units according to Equation (2.20). 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 SI^H to the lower operating mode SI^L and vice versa using the offset δ, according to Equation (2.20), can be computed as follows:

$$o_{HL} = S^L_{out} + \delta^{H\to L} - S^H_{out},\ o_{LH} = S^H_{out} + \delta^{L\to H} - S^L_{out} \tag{5.4}$$

where S^L_{out} and S^H_{out} 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 processors are changed during the switching, the computed δ offset in Equation (2.20) may not be sufficient. This is because, the time that is needed for physically changing the operating frequencies in the processors, denoted by ζ, is not considered in Equation (2.20). 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 processors must be changed after the end of the execution of the allocated tasks on the processors 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 processors must be changed before the start of the execution of the allocated tasks on the processors 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 Figure 5.3, the time instants of changing the operating frequencies of π_1 and π_2 are shown by the boxes with a dotted pattern where the size of these boxes denotes the frequency switching delay ζ. The δ offset in Equation (2.20) is a function of the tasks utilization. Therefore, to involve such switching delay ζ associated with the DVFS mechanism into the δ offset, we have changed the utilization of each task τ_i in the lower operating mode SI^L, i.e., τ^L_i, from C^L_i / T^L_i to $(C^L_i + \zeta) / T^L_i$ that is executing when the operating frequency change happens. As a result, using Equation (2.20), we can compute a sufficient δ with the new utilization of tasks 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 Equation (2.20), can be when

the execution of the previous operating mode is completely finished and the operating frequencies of the processors are also changed. This is the safest starting time for the new operating mode while no extra schedulability test is needed as there is no overlapping execution between two operating modes. Using the method, explained above, for the proposed schedule in Figure 5.3, the starting offset of $\delta^{1\rightarrow 2} = 0$ can be computed for operating mode SI^2 when the operating mode is switched from SI^1 to SI^2. Similarly, the starting offset of $\delta^{2\rightarrow 1} = 5$ can be computed for operating mode SI^1 when the operating mode is switched from SI^2 to SI^1. Finally, the time cost of $o_{12} = 5$ and $o_{21} = 0$ can be computed using Equation (5.4) for the operating mode switching from SI^1 to SI^2 and vice versa, respectively, as can be seen in Figure 5.3.

(2) *Energy Costs*: By applying sufficient δ offset, as computed in Section 5.6.2(1) above, tasks belonging to both the lower and higher operating modes may be concurrently executing on the processors during mode switching instants. For instance, in Figure 5.3 tasks in both operating modes SI^1 and SI^2 execute from time instant 26 to 36 and from time instant 67 to 77 when the operating mode is switched from SI^1 and SI^2 and vice versa, respectively. To meet the tasks' job deadlines in both operating modes, the processors must run at the operating frequency corresponding to the higher operating mode during operating mode switching instants. Therefore, the total energy consumption of our proposed scheduling approach is more than the summation of the energy consumption of operating modes SI^H and SI^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 compute them using the following expressions:

$$e_{HL} = o_{HL} P_L \tag{5.5}$$

$$e_{LH} = (S^H_{out} - o_{LH})(P_H - P_L) + o_{LH} P_H = S^H_{out}(P_H - P_L) + o_{LH} P_L \tag{5.6}$$

where the S^H_{out} is the start time of the task corresponding to output actor A_{out} in the graph in the higher operating mode. These energy costs are visualized by the hatched boxes in Figure 5.5(b). These energy costs are overestimated using the above expressions because a single time instant is assumed for changing the operating frequency of all processors in each operating mode switching. This time instant is referred by f_{switch} in Figure 5.5(b). Note that we also include the energy overhead of DVFS into this energy costs.

5.6.3 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:

$$Q_H = N_H \cdot H_H,\ N_H \in \mathbb{N} \tag{5.7}$$

$$Q_L = N_L \cdot H_L,\ N_L \in \mathbb{N} \tag{5.8}$$

where N_H and N_L are the number of graph iterations in the higher and lower operating modes, respectively, and H_H and H_L are the graph iteration period in the higher and lower operating modes, respectively, as defined in Equation (2.14). Finally, by substituting Equation (5.7) and Equation (5.8) in Equation (5.2) and setting $\mathcal{R}_{eff} = \mathcal{R}_{req}$, the number of graph iterations to stay in the higher operating mode, N_H, can be derived as follows:

$$N_H = \left\lceil \frac{H_L N_L(\mathcal{R}_{req} - \mathcal{R}_L) + \mathcal{R}_{req}(o_{HL} + o_{LH})}{H_H(\mathcal{R}_H - \mathcal{R}_{req})} \right\rceil. \tag{5.9}$$

Note that, in the above equation, the ceiling function is used to derive an integer value for N_H such that the effective throughput $\mathcal{R}_{eff}$ can still satisfy the throughput requirement $\mathcal{R}_{req}$. This fact is shown in Figure 5.5(c) where our proposed effective throughput $\mathcal{R}_{eff}$ is higher than the throughput requirement $\mathcal{R}_{req}$. Using Equation (5.9), we have to derive the pair of N_H and N_L that satisfies the throughput requirement $\mathcal{R}_{req}$. Clearly, Equation (5.9) has more than one solution for the pair of N_H and N_L. Since all of these solutions have the same timing requirement, i.e., throughput requirement, the energy reduction is equivalent with the power reduction. Therefore, to find the less power consuming solution that consequently results in the less energy consumption, we can see from Equation (5.3) 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 $\mathcal{R}_{eff}$ using our switching scheme becomes closer to the throughput requirement $\mathcal{R}_{req}$. This is because, as N_L increases in Equation (5.9), the ceiling function becomes less contributing and the pair of N_L and N_H can produce the effective throughput $\mathcal{R}_{eff}$ more closely to the throughput requirement $\mathcal{R}_{req}$. As a result, this leads to exploiting static slack

Algorithm 3: Finding the least power consuming pair of N_H and N_L.

Input: $\mathcal{R}_{req}$, SI^H, SI^L.
Output: N_L, N_H.

```
1 Prev_Power = +∞;
2 N_L = 1;
3 while True do
4     Calculate N_H using Equation (5.9) and R_req;
5     Power = Calculate power consumption by using Equation (5.3);
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;
```

time in the application scheduling more efficiently leading to further power reduction. Therefore, to find a valid solution for N_H and N_L which satisfies Equation (5.9) 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.

Algorithm 3 presents the pseudo-code of finding the least power consuming pair of N_H and N_L. The inputs of this algorithm are the throughput requirement 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 Equation (5.9) in Line 4. Then, we compute the power consumption corresponding to the derived pair of N_H and N_L using Equation (5.3) 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.

5.6.4 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 Figure 5.6. In order to compute the output buffer, we should consider Figure 5.5(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 $\mathcal{R}_{eff}$ (the dotted line). When the application executes in the higher operating

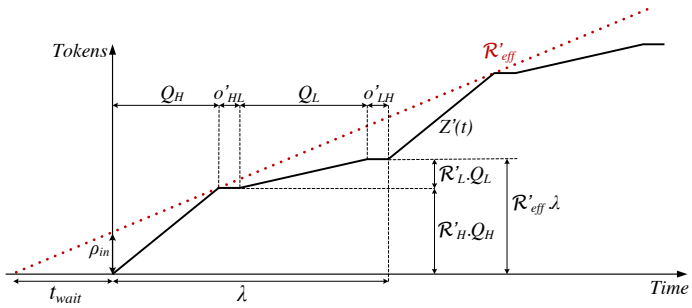


Figure 5.7: *Token consumption function $Z'(t)$. Note that, $o_{HL} + o_{LH} = o'_{HL} + o'_{LH} = \delta^{H\to L} + \delta^{L\to H}$.*

mode, it produces more tokens than needed while in the lower operating mode it produces less tokens 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 time. This maximum difference is given by ρ_{out} in Figure 5.5(c). Therefore, the size of the output buffer must be at least

$$B_{out} = \left\lceil \rho_{out} \right\rceil = \left\lceil Q_H(\mathcal{R}_H - \mathcal{R}_{eff}) \right\rceil \tag{5.10}$$

To compute the input buffer, the same method as for the output buffer can be used. To do so, we should consider Figure 5.7 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 $\mathcal{R}'_{eff}$ give by the dotted line in Figure 5.7 for the application and we compute it as follows:

$$\mathcal{R}'_{eff} = \frac{\mathcal{R}'_H Q_H + \mathcal{R}'_L Q_L}{Q_H + Q_L + o'_{HL} + o'_{LH}} \tag{5.11}$$

where $\mathcal{R}'_H$ and $\mathcal{R}'_L$ are the throughput of the input task in the higher and lower operating modes, $\mathcal{R}'_H Q_H$ and $\mathcal{R}'_L Q_L$ 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 using Equation (2.20). Apparently, the constant sampling rate of $\mathcal{R}'_{eff}$ 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 $\mathcal{R}'_{eff}$ must be started t_{wait} time units before the application starts executing, as shown in Figure 5.7. This time can be computed as follows:

$$t_{wait} = \frac{(\mathcal{R}'_H - \mathcal{R}'_{eff})Q_H}{\mathcal{R}'_{eff}} \tag{5.12}$$

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

$$B_{in} = \left\lceil \rho_{in} \right\rceil = \left\lceil t_{wait}\mathcal{R}'_{eff} \right\rceil = \left\lceil Q_H(\mathcal{R}'_H - \mathcal{R}'_{eff}) \right\rceil \tag{5.13}$$

where ρ_{in} is the maximum difference between the number of sampled and needed tokens, as shown in Figure 5.7.

5.7 Experimental Evaluation

In this section, we evaluate the effectiveness of our scheduling approach in terms of energy reduction. We compare our proposed scheduling approach, referred as Switching, in terms of energy reduction with two related approaches: the straightforward approach of always selecting the operating mode whose throughput is the closest higher to the throughput requirement, referred as Higher mode, and the period scaling approach, referred as Scale, explained in Section 5.5.1, which is the way of using the VFS mechanism similar to the related works [74,96] in the context of the SPS framework [8]. In the following, we first explain our experimental setup in Section 5.7.1. Then, we present the experimental results in the Section 5.7.2.

5.7.1 Experimental Setup

Applications

We have performed experiments on a set of six real-life streaming applications collected from the StreamIt benchmark suit [88], the SDF^3 suit [84] and the individual research article [69], where all streaming applications are modeled as CSDF graphs. An overview of all streaming applications is given in Table 5.2. In this table, $|\mathcal{A}|$ denotes the number of actors in a CSDF graph, while $|\mathcal{E}|$ denotes the number of FIFO communication channels among actors.

Table 5.2: *Benchmarks used for evaluation.*

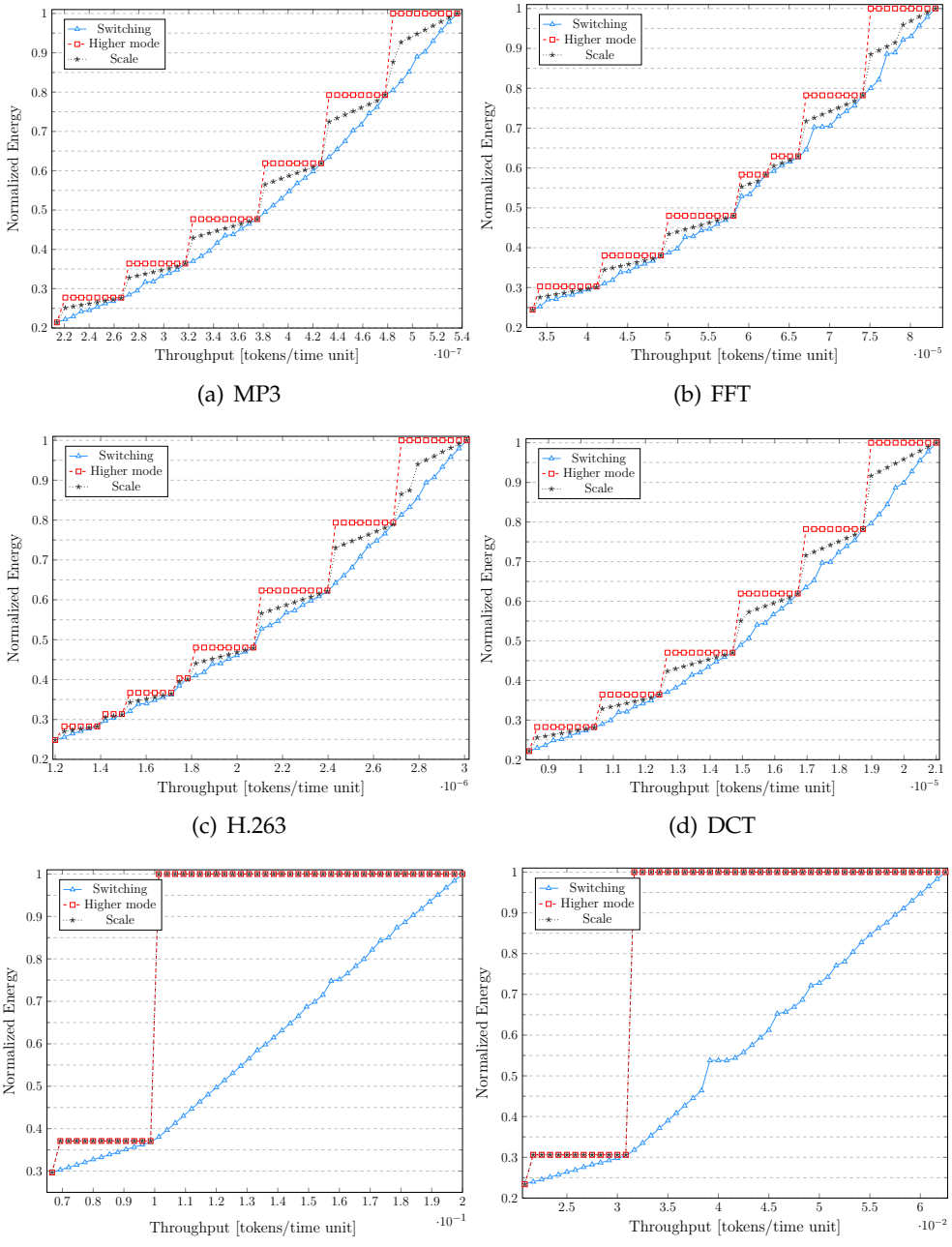
Application	$\|\mathcal{A}\|$	$\|\mathcal{E}\|$	Source
Discrete cosine transform (DCT)	8	7	[88]
Fast Fourier transform (FFT)	17	16	[88]
Data modem	6	5	[84]
MP3 audio decoder	14	18	[84]
H.263 video decoder	4	3	[84]
Heart pacemaker	4	3	[69]

Architecture and Power Model

In the experiments, we use the power model presented in Section 5.4.2. In this model, we adopt the power parameters of the Cortex A15 core given in [55], where these parameters have been obtained based on real measurements on the ODROID XU-3 platform [66]. The overhead of the DVFS mechanism is set to values taken from [67], i.e., $10\mu s$ and $1\mu J$ are used for the delay and energy overhead associated with the physical change of the operating frequency in processors, respectively. We evaluate the effectiveness of our scheduling approach on platforms with limited number of processors. To this end, we compute the minimum number of processors needed to schedule each application using FFD-EDF when the maximum achievable throughput under the SPS framework is required.

5.7.2 Experimental Results

All experimental results are shown in Figure 5.8 and Figure 5.9, where the comparison is made for a set $\mathcal{R}_{app}$ of selected application throughputs as throughput requirements. In Figure 5.8, we show the different throughput requirements for the applications on the x-axis and the normalized energy consumption of all three approaches is shown on the y-axis. As can be seen in Figure 5.8, the energy reduction varies considerably among different applications and throughput requirements. When compared to the approach Higher mode, our proposed approach Switching achieves significant energy reduction for all applications. This energy reduction for the Modem, Pacemaker, DCT, MP3, FFT, and H.263 applications 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 applications can be up to 68.18%, 61.94%, 13.1%, 13.78%, 10.7%, and 12.07%, respectively. Among all these applications, the Modem and Pacemaker are the two applica-



(a) MP3 (b) FFT

(c) H.263 (d) DCT

(e) Pacemaker (f) Modem

Figure 5.8: *Normalized energy consumption vs. throughput requirements.*

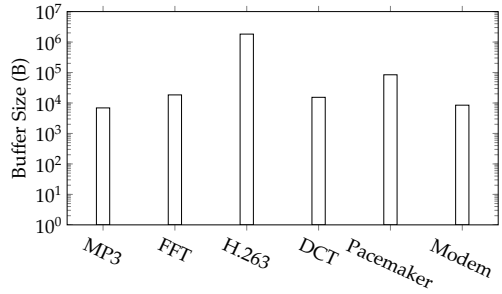


Figure 5.9: *Total buffer sizes needed in our scheduling approach for different applications. Note that the y axis has a logarithmic scale.*

tions 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 applications are quickly increased by applying the *period scaling* approach, explained in Section 5.5.1. Therefore, a fewer number of operating modes can be determined for these applications 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 satisfy the throughput requirement in these applications. This fact can be seen in Figure 5.8 for Pacemaker and Modem applications in which the result of the approach Scale and the approach Higher mode are overlapped on each other.

As can be seen in Figure 5.8, for some throughput requirements no energy reduction is achieved by our approach Switching compared to approach Higher mode and approach Scale. This happens when the throughput requirements 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 requirement because mode switching is not needed.

Finally, the memory overhead, discussed in Section 5.6.4, introduced by our scheduling approach, is given in Figure 5.9. In this figure, the x-axis shows the different applications while the y-axis shows the buffer size for each application which is calculated as follows:

$$B_{app} = \max_{\mathcal{R}_i \in \mathcal{R}_{app}} (B^i_{in} + B^i_{out})$$

where B^i_{in} and B^i_{out} are the size of the input and output buffers shown in Figure 5.6, computed by using Equation (5.13) and Equation (5.10), respectively, for a required application throughput $\mathcal{R}_i$. In this regard, the memory overhead for the H.263 application is 1.7 MB whereas for the other applications it is

less than 83 KB. Given such 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.

5.8 Conclusions

In this chapter, we have proposed a novel energy-efficient periodic scheduling approach for streaming applications. This approach can satisfy a system throughput requirement at a long run by periodically switching the application schedule between two selected schedules, referred as operating modes. Contrary to related approaches, 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 a set of six real-life streaming applications, show that our approach can reduce the energy consumption by up to 68% while satisfying the same throughput requirement when compared to related approaches. However, for some throughput requirements 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 which throughput matches with the throughput requirement instead of adopting the mode switching scheme. Finally, although the throughput requirement of the applications is satisfied by our proposed approach, the mentioned energy reductions come at the expense of increased memory requirements.