Green Computing: Power Optimisation of VFI-based Real-time Multiprocessor Dataflow Applications (extended version)*

Waheed Ahmad[1], Philip K.F. Hölzenspies[2], Mariëlle Stoelinga[1], and Jaco van de Pol[1]

[1] University of Twente, The Netherlands,
{w.ahmad, m.i.a.stoelinga, j.c.vandepol}@utwente.nl
[2] Facebook Inc., United Kingdom,
drphil@fb.com

Abstract. Execution time is no longer the only performance metric for computer systems. In fact, a trend is emerging to trade raw performance for energy savings. Techniques like Dynamic Power Management (DPM, switching to low power state) and Dynamic Voltage and Frequency Scaling (DVFS, throttling processor frequency) help modern systems to reduce their power consumption while adhering to performance requirements. To balance flexibility and design complexity, the concept of Voltage and Frequency Islands (VFIs) was recently introduced for power optimisation. It achieves fine-grained system-level power management, by operating all processors in the same VFI at a common frequency/-voltage.

This paper presents a novel approach to compute a power management strategy combining DPM and DVFS. In our approach, applications (modelled in full synchronous dataflow, SDF) are mapped on heterogeneous multiprocessor platforms (partitioned in voltage and frequency islands). We compute an energy-optimal schedule, meeting minimal throughput requirements. We demonstrate that the combination of DPM and DVFS provides an energy reduction beyond considering DVFS or DMP separately. Moreover, we show that by clustering processors in VFIs, DPM can be combined with any granularity of DVFS. Our approach uses model checking, by encoding the optimisation problem as a query over priced timed automata. The model-checker Uppaal Cora extracts a cost minimal trace, representing a power minimal schedule. We illustrate our approach with several case studies on commercially available hardware.

1 Introduction

The power consumption of computing systems has increased exponentially [18]. Therefore, minimising power consumption has become one of the most critical, challenging and essential criteria for these systems. Therefore, over the past

* This research is supported by the EU FP7 project SENSATION (318490).

years, system-level power management based on the properties such as execution time of the tasks, frequency etc. has gained significant value and success [6][18][38].

Power Reduction Techniques. The total power consumption of a processor is the sum of static (leakage) and dynamic power (in terms of switching activity). Two well-known techniques for power optimisation in modern processors are Dynamic Power Management (DPM) [6] and Dynamic Voltage and Frequency Scaling (DVFS) [35]. DPM reduces the static power consumption, whereas DVFS is used to lower the dynamic power consumption.

Dynamic Power Management. DPM works on the principle of switching a processor to a low power state when it is not used, thus resulting in reduced power utilisation. For example, let us consider a processor of a typical mobile phone, having three power states, i.e., ON, DIM and OFF. If the processor runs in ON state, LCD and backlight of the phone is turned on. If the phone remains idle for some time, the processor enters the DIM state in which the backlight turns off but the LCD stays turned on. If the phone stays idle for some more time, the LCD is turned off too (the OFF state). It is very commonly assumed by power optimisation methods in the literature that the transition overhead of switching to another power state is negligible [27]. However, this may not be the case in real-life applications, where there is always a non-negligible overhead [30]. This paper considers transition overheads while moving to a different power state. DPM is widely used; many processor manufacturers, such as Intel and AMD, have implemented an open standard for power management named *Advanced Configuration and Power Interface* [12].

Dynamic Voltage and Frequency Scaling. On the other hand, DVFS [35] lowers the voltage and clock frequency at the expense of the execution time of a task. Power consumption of a processor scales linearly in frequency and quadratically in voltage. But, frequency and voltage also have a linear relation, therefore, when the clock frequency decreases, the voltage can also reduce, so that the power is reduced cubically. DVFS comes in two flavours, viz. local and global [24]. Local DVFS works on the principle that each processor has its own individual clock frequency/voltage, whereas all processors operate on the same clock frequency/voltage in the case of global DVFS. Local DVFS gives more freedom in choosing clock frequencies and is therefore more energy-efficient. However, local DVFS is expensive and complex to implement because it requires more than one clock domain. In contrast, global DVFS requires a simpler hardware design, but may lead to less reduction in power consumption [24]. Several modern processors such as Intel Core i7 and NVIDIA Tegra 2 employ global DVFS [12].

Voltage and Frequency Islands. To balance the energy efficiency and design complexity, the concept of *voltage and frequency islands* (VFIs) [16] was put forward. A VFI consist of a group of processors clustered together, and each VFI runs on a common clock frequency/voltage [29]. The clock frequencies/voltage supplies of a VFI may differ from other VFIs. Furthermore, different VFI partitions represent DVFS *policies* of different granularity. Recently, some modern multicore processors, such as IBM Power 7 series, have adopted the option of VFIs [15].

Shortcomings in Literature. While DPM and DVFS are popular power minimisation techniques, most of the earlier work [17, 26, 31, 37] focusses on DVFS only, neglecting static power completely. On the contrary, modern processors have significant static power, which must be addressed. Hence, optimal power minimisation cannot be guaranteed without considering both DVFS and DPM. This paper is the first to compute energy schedules for combined DVFS and DPM. Furthermore, with the help of VFIs, we combine DPM with a DVFS policy with *any* granularity, generalising local and global DVFS. This achieves fine-grained system-level power management.

The second shortcoming in existing literature is addressing the applications where inter-task dependencies are modelled by directed acyclic graphs (DAGs), without analysing periodicity [9, 22]; or frame-based periodic applications with no data dependencies between periods [10, 13]. In real-time streaming applications, there are three challenges in implementing power management [17]. First, the schedules of these applications are typically infinite, making the problem scope infinite. Second, the iterations overlap in time and we have to deal with data dependencies within and across iterations. Last, performance constraints such as throughput are critical, and must be met. Hence, we cannot capture all semantics of real-time streaming applications using DAGs or frame-based models.

Our Approach. Alternatively, we use Synchronous Dataflow (SDF) [21] as a computational model. SDF allows natural representation of real-time streaming and digital signal processing applications. SDF graphs are increasingly utilised for performance analysis and design optimisation of multimedia applications, implemented on multiprocessor Systems-On-Chip [22]. In this paper, SDF graphs are used to represent software applications which are partitioned into tasks, with inter-task dependencies and their synchronisation properties.

Contemporary SDF analysis tools e.g. SDF3 [32] lack support for cost optimisation. Therefore, we propose an alternative, novel approach based on Uppaal Cora [5], the tool for Cost Optimal Reachability Analysis, using priced timed automata (PTA) as a modelling language. PTA extend timed automata [4] (for the modelling of time-critical systems and time constraints) with costs, which we use to model energy consumption. Furthermore, power reduction techniques based on mathematical optimisation [17, 26, 34, 7] do not support quantitative model-checking and evaluating user-defined properties. PTA also bridges this gap to achieve benefits over the range of analysable properties such as the absence of deadlocks and unboundedness, safety, liveness and reachability. Finally, PTA provide straightforward compositional and extendible modelling capabilities to system engineers, as opposed to mathematical optimisation approaches.

Methodology. Our approach takes three inputs: an SDF graph that models the application tasks; a platform model that describes the specifics of the hardware such as VFI partitions, frequency levels and power usage per processor; and a throughput constraint. We compute a power optimal schedule that meets the constraint, utilising the dynamic and static slack in the application. The method can also be used to determine optimal VFI partitions in terms of design complexity and energy efficiency, facilitating system designers to build durable systems.

Contributions. The main contribution of this paper is a fully automated technique to compute power-optimal schedules. In particular, we demonstrate the following:

- We apply a combination of DPM and DVFS, confirming earlier results [10][13] that DPM and DVFS together result in lower energy consumption than considering only DVFS;
- Our method considers processors partitioned into VFIs; thus allowing to combine DPM, and DVFS policy with any granularity.
- We analyse SDF graphs as input which are more versatile and allow more realistic data-dependencies than acyclic applications considered in earlier work;
- We consider the transition overhead of transitions between different frequencies.
- Our approach is able to handle heterogeneous platforms, in which only specific processors can run a particular task.

Moreover, our technique is based on the solid semantic framework of Priced Timed Automata, enabling the verification of functional system correctness.

We only consider discrete frequency and voltage levels in this paper as real-life platforms can support only a limited set of discrete frequency and voltage levels [17].

Paper organisation. Section 2 reviews related work. Section 3 formalises the problem, and different power reduction techniques are illustrated in Section 4. Section 5 covers PTA and UPPAAL Cora, and Section 6 covers the translation of SDF graphs to PTA. The methodology of power optimisation of SDF graphs using UPPAAL Cora is explained in Section 7. Section 8 experimentally compares the results of different power optimisation techniques explained earlier, and Section 9 validates our approach via case studies. Finally, Section 10 draws conclusions and outlines possible future research.

2 Related Work

Considerable work has been done on power management. An extensive survey paper [18] outlines the research work in the field of algorithmic power management, but without reviewing any work done on SDF graphs. Another survey paper [38] discusses several energy-cognizant scheduling techniques. All of the presented techniques do not evaluate effectiveness of optimal combination of global DVFS with scheduling. The novel methods for VFI-aware power optimisation are discussed in [19] and [28]. It is assumed in these papers that task scheduling is finished beforehand, and therefore, task precedence is not considered. Whereas in practice, there are always precedence constraints due to inter-task data dependencies.

A state-of-the-art methods of applying DVFS only on SDF graphs is addressed in [26, 31, 37]. These papers, in comparison to ours, consider dynamic power only, and ignore static power which is non-negligible in modern processors. Moreover, work in [26] also requires to transform an SDF graphs to equivalent Homogeneous SDF (HSDF) graphs and model them with additional static ordering edges, which is not needed in our approach. Similarly, work in [37] uses self-timed execution and static order firing, which means we need as many processors as actors, unlike real-life applications where there is always a constraint on available number of processors. Therefore, this work is not scalable on any other hardware platform, where there are fewer processors than actors. Another method of throughput-constrained DVFS of SDF graphs on a heterogeneous multi-processor platform is proposed in [17]. A difference with our approach is that work in [17] ignores transition overheads. Therefore, optimality cannot be guaranteed. Moreover, this paper also suffers from the limitation of static ordering.

More advanced approaches that combine DPM and DVFS are presented in [34, 7]. Unlike our method, these approaches discuss a specific power optimisation policy where DPM and DVFS can be applied to each processor independently only. In contrast, we consider VFI-based hardware platforms where DPM and DVFS can be applied to any DVFS policy ranging from each processor having independent voltage/frequency level to all processors running at same voltage/frequency level. Furthermore, these papers are restricted to acyclic applications only, which makes the problem scope simpler. The work in [10, 13] describes an another novel algorithm for the optimal combination of DPM and DVFS. In comparison to our work, this technique is confined to global DVFS only, as it does not consider VFIs.

To the best of our knowledge, there are no papers that apply an optimal *combination* of DVFS and DPM on SDF graphs, mapped on a given number of *VFI-partitioned* processors, both in homogeneous and heterogeneous platforms.

3 Problem Formalisation

This section first introduces the power model used in this paper. Later, we present formal definitions of SDF graphs and heterogeneous *platform application model* capable of having VFIs and multiple discrete frequencies.

3.1 Power Model

The clock frequency of a processor represents its speed. We assume that the speed of the processors scale linearly with the clock frequency. However, in practice, the relation between speed and clock frequency is not perfectly linear. The reason is that the computer memory is a separate device and it is often running on a different clock frequency. Therefore, the speed of the computer memory typically does not scale with the clock frequency of the processor [10].

Nevertheless, if measured at the maximum frequency, the round-trip time for a memory access in terms of processor clock cycles is at its highest. Memory access for the same task running at a lower frequency level is cheaper in terms of processor clock cycles. Therefore, our assumption made in this paper that speed of the processors is linearly related to the clock frequencies, does not violate the deadline constraints of an application [12].

The total *power consumption* by a processor is given by [9]:

$$P_{Tot} = P_D + P_S + P_{tr} \tag{1}$$

where P_D and P_S is the dynamic and static power usage of a processor respectively. The dynamic power is consumed due to the activity of the processor and is given by:

$$P_D = aCv_{dd}^2 f \tag{2}$$

where a is the circuit switching activity, C is the switched capacitance, v_{dd} is the supply voltage, and f is the operating frequency. Here, a and C are technology dependent. The static power is consumed independently of the processor activity and clock frequency. The static power is given by:

$$P_S = V_{dd}I_{subn} + |V_{bs}|I_j \tag{3}$$

where V_{dd} is the supply voltage, and rest of the parameters are fixed technology dependent. The transition overhead of transition from a certain frequency level to another is denoted by P_{tr}.

3.2 SDF Graphs

Typically, real-time streaming applications execute a set of periodic tasks which consume and produce fixed amounts of data. Such applications are naturally modelled by a directed, connected graph named SDF graph in which these tasks are represented by a set of *actors* A. Actors communicate with each other by sending streams of data elements represented by *tokens*. Each *edge* (a, b, p, q) connected to a producer a and a consumer b, transports tokens between actors. The execution of an actor is known as an (*actor*) *firing*, and the number of tokens consumed or produced onto an edge (a, b, p, q) as a result of a firing is referred to as *consumption* q and *production* p *rates* respectively. An SDF graph is timed if each actor is assigned an execution time. As stated, we assume that speed of tasks scale linearly with the clock frequency of the processor, the execution times of the actors are characterised by the operating frequency of the processor.

Definition 1. *An* SDF graph *is a tuple* $G = (A, D, \mathsf{Tok_0}, \tau)$ *where:*

– *A is a finite set of actors,*

- *D is a finite set of dependency edges $D \subseteq A^2 \times \mathbb{N}^2$,*
- *$\mathsf{Tok}_0 : D \to \mathbb{N}$ denotes distribution of initial tokens in each edge, and*
- *the execution time of each actor is given by $\tau : A \to \mathbb{N}_{\geq 1}$.*

The sets of input edges *$In(a)$ and* output edges *$Out(a)$ of an actor $a \in A$ are defined as: $In(a) = \{(a', a, p, q) \in D | a' \in A \wedge p, q \in \mathbb{N}\}$ and $Out(a) = \{(a, b, p, q) \in D | b \in A \wedge p, q \in \mathbb{N}\}$. The* consumption rate *$CR(e)$ and* production rate *$PR(e)$ of an edge $e = (a, b, p, q) \in D$ are defined as: $CR(e) = q$ and $PR(e) = p$.*

Informally, for all actors $a \in A$, if the number of tokens on every input edge $(a'_i, a, p_i, q_i) \in In(a)$ is greater than or equal to q_i, actor a fires and removes q_i tokens from every $In(a)$. The firing takes place for $\tau(a)$ time units and it ends by producing p_i tokens on all $(a, b_i, p_i, q_i) \in Out(a)$.

Example 1. Figure 1 shows an SDF graph of an MPEG-4 decoder [33]. MPEG-4 is a method of defining compression of audio and visual digital data. The processing unit in video compression is termed *macroblock*. A macroblock typically consists of 16×16 array of pixels. The two major picture types used in the different video algorithms are I and P. An I-frame is an “Intra-coded picture”, representing a conventional static image file. On the other hand, an P-frame is an “Predicted picture”, and it carries only the changes in the image from the previous frame.

The MPEG-4 decoder shown in Figure 1 has five actors A={FD, VLD, IDC, RC, MC}. These actors represent individual tasks performed in MPEG-4 decoding. For example, the frame detector (FD) models the part of the application that determines the frame type and the number of macro blocks to decode. In our case, MPEG-4 can process only P-frames. To decode a single P-frame, FD must process between 0 and 99 macroblocks, i.e., $x \in \{0, 1, \ldots, 99\}$ in Figure 1. The rest of the steps in MPEG-4 decoding are Variable Length Decoding (VLD), Inverse Discrete Cosine (IDC) Transformation, Motion Compensation (MC), and Reconstruction (RC) of the final video picture.

Arrows between the actors depict the edges which hold tokens (dots) representing macroblocks. The execution time (ms) of the actors is represented by a number inside the actor nodes. The numbers near the source and destination of each edge are the rates. The initial token on the edge from RC to MC models the exchange of the previously decoded frame, while the initial token on the edge from RC to FD models that the decoder is capable of processing macroblocks of 1 frame. A schedule of an SDF graph is a firing sequence of actors to meet certain design objectives.

To avoid unbounded accumulation of tokens in a certain edge, we require SDF graph to be *consistent* [20]. Consistency is defined as follows.

Definition 2. *A repetition vector of an SDF graph $G = (A, D, \mathsf{Tok}_0, \tau)$ is a function $\gamma : A \to \mathbb{N}_0$ such that for every edge $(a, b, p, q) \in D$ from $a \in A$ to $b \in A$, the following relation exists.*

$$p.\gamma(a) = q.\gamma(b)$$

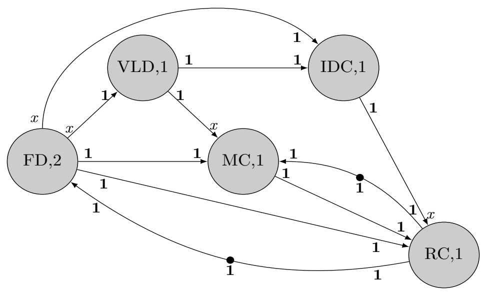


Fig. 1: MPEG-4 Decoder

Repetition vector γ is termed non-trivial if and only if $\forall a \in A, \gamma(a) > 0$. An SDF graph is consistent if it has a non-trivial repetition vector.

A repetition vector determines how often each actor must fire with respect to the other actors without a change in the token distribution. If each actor of an SDF graph is invoked according to its repetition vector in a schedule, then the number of tokens on each edge is the same after the schedule is executed as before.

Definition 3. *Let us assume that an SDF graph $G = (A, D, \mathsf{Tok}_0, \tau)$ has a repetition vector γ. An iteration is a set of actor firings such that for each $a \in A$, the set contains $\gamma(a)$ firings of a. Thus, each actor fires according the repetition vector in an iteration [14].*

3.3 Platform Application Model

The Platform Application Model (PAM) models the multi-processor platform where the application, modelled as SDF graph, is mapped on. Our PAM models supports several features, including

- heterogeneity, i.e., actors can run on certain type of processors only,
- a partitioning of the processors in voltage and frequency islands,
- different frequency levels each processor can run on
- power consumed by a processor in a certain frequency, both when in use and when idle,
- transition overhead required to switch between frequency levels.

Definition 4. *A* platform application model *is a tuple $\mathcal{P} = (\Pi, \zeta, F, P_{idle}, P_{occ}, P_{tr}, \tau_{act})$ consisting of,*

- *a finite set of processors Π. We assume that $\Pi = \{\pi_1, \ldots, \pi_n\}$ is partitioned into disjoint blocks of voltage/frequency islands (VFIs) such that $\bigcup \Pi_i = \Pi$, and $\Pi_i \cap \Pi_j = \emptyset$ if $i \neq j$,*
- *a function $\zeta : \Pi \to 2^A$ indicating which processors can handle which actors,*

Level	Voltage	Frequency	Level	Voltage	Frequency
1	1.2	1400	4	1.05	1128.7
2	1.15	1312.2	5	1.00	1032.7
3	1.10	1221.8			

Table 1: DVFS levels of Samsung Exynos 4210

- *a finite set of discrete frequency levels available to all processors denoted by* $F = \{f_1, \ldots, f_m\}$ *such that* $f_1 < f_2 < \ldots < f_m$*,*
- *a function* $P_{occ} : \Pi \times F \rightarrow \mathbb{N}$ *denoting the power consumption (static plus dynamic) of a processor operating at a certain frequency level* $f \in F$ *in the operating state,*
- *a function* $P_{idle} : \Pi \times F \rightarrow \mathbb{N}$ *assigning the power consumption (static) of a processor operating at a certain frequency level* $f \in F$ *in the idle state,*
- *a function* $P_{tr} : \Pi \times F^2 \rightarrow \mathbb{N}$ *expressing the transition overhead from one frequency level* $f \in F$ *to next frequency level* $f \in F$ *for each processor* $\pi \in \Pi$*, and*
- *the valuation* $\tau_{act} : A \times F \rightarrow \mathbb{N}_{\geq 1}$ *defining the actual execution time* τ_{act} *of each actor* $a \in A$ *mapped on a processor at a certain frequency level* $f \in F$*.*

The notations f_i and Π_j represent i^{th} frequency level and j^{th} VFI respectively. We also use the notation $[\pi]$ to denote the VFI of a processor $\pi \in \Pi$.

Example 2. Exynos 4210 [2] is a state-of-the-art processor used in high-end mobile platforms such as Samsung Galaxy Note, Galaxy SII etc. Table 1 shows different DVFS levels, and corresponding CPU voltage (V) and clock frequency (MHz), of Samsung Exynos 4210 based on ARM Cortex-A9 [30].

3.4 Example

In this subsection, we explain the aforementioned semantics of an SDF graph mapped on a processor application model by means of an example. Let us consider the SDF graph of an MPEG-4 decoder shown in Figure 1 mapped on four Samsung Exynos 4210 processors. For easy understanding, let us consider that the MPEG-4 decoder is capable of processing 5 macroblocks, i.e, $x = 5$ in Figure 1. The processors $\Pi = \{\pi_1, \pi_2, \pi_3, \pi_4\}$ are partitioned in three VFIs such that $\Pi_1 = \{\pi_1\}$, $\Pi_2 = \{\pi_2, \pi_3\}$ and $\Pi_3 = \{\pi_4\}$. Two DVFS levels (MHz) $\{f_1, f_2\} \in F$ taken from Table 1 i.e. $f_2 = 1400$ and $f_1 = 1032.7$, are available to all processors. The transition overhead (W) of all Exynos 4210 processors is, $P_{tr}(\pi, f_2, f_1) = 0.2$ and $P_{tr}(\pi, f_1, f_2) = 0.1$ [30]. Let us assume that all processors start at highest frequency level, i.e., $f_2 \in F$. Table 2 shows the formation of VFIs and experimental power consumption against each frequency level. We also assume that

Processor	**VFI**	**Voltage(V)**	**Frequency(MHz)**	$\mathbf{P_{idle}}$**(W)**	$\mathbf{P_{occ}}$**(W)**
π_1	$\mathit{\Pi}_1$	1.2	1400	0.1	4.6
		1.00	1032.7	0.4	1.8
π_2	$\mathit{\Pi}_2$	1.2	1400	0.1	4.6
		1.00	1032.7	0.4	1.8
π_3	$\mathit{\Pi}_2$	1.2	1400	0.1	4.6
		1.00	1032.7	0.4	1.8
π_4	$\mathit{\Pi}_3$	1.2	1400	0.1	4.6
		1.00	1032.7	0.4	1.8

Table 2: Description of Samsung Exynos 4210 based Platform

the execution times (ms) of all actors $a \in A$ at frequency level f_1 are rounded to the next integer. As $f_1 = 0.738 \times f_2$, $\tau_{act}(a, f_1) = \lceil \frac{\tau_{act}(a,f_2)}{0.738} \rceil$.

Figure 2 shows a schedule of our running example for a constraint of 125 frames per second (fps). To achieve 125 fps, MPEG-4 decoder completes the iteration in $\frac{1}{125}$= 8 ms. In this figure, grey and white coloured boxes denote, if a processor is running at frequency f_2 or f_1 respectively.

As we can see in Figure 2, processor $\pi_1 \in \mathit{\Pi}$ changes its frequency level from $f_2 \in F$ to $f_1 \in F$ at t=0 ms, thus incurring he transition overhead $P_{tr}(\pi_1, f_2, f_1)$=0.2 W. From thereon, it operates in frequency level $f_1 \in F$ for 5 ms. During this time interval, actors FD $\in A$ and VLD $\in A$ are fired once on $\pi_1 \in \mathit{\Pi}$. At t=5 ms, processor $\pi_1 \in \mathit{\Pi}$ switches frequency level from $f_1 \in F$ back to $f_2 \in F$ after spending $P_{tr}(\pi_1, f_1, f_2)$=0.1 W, and stays in frequency level $f_2 \in F$ for the rest of the iteration. During this time interval, actors IDC $\in A$ and RC $\in A$ claim $\pi_1 \in \mathit{\Pi}$ twice and once respectively. Thus per iteration, it consumes dynamic energy for 8 ms. As processor $\pi_1 \in \mathit{\Pi}$ does not remain idle during the iteration, it does not consume any static energy. The total energy consumption (mWs) per iteration of processor $\pi_1 \in \mathit{\Pi}$ is

$$E_{Tot} = E_S + E_D + E_{tr}$$

$$E_{Tot} = P_{idle} \times 0 + P_{occ}(\pi_1, f_2) \times 3 + P_{occ}(\pi_1, f_1) \times 5 + P_{tr}(\pi_1, f_2, f_1) + P_{tr}(\pi_1, f_1, f_2)$$

$$E_{Tot} = 0 + 4.6 \times 3 + 1.8 \times 5 + 0.2 + 0.1 = 23.1$$

In the same fashion, we can calculate energy consumption per iteration for each processor, which gives us total energy consumption equal to 57.7 mWs per iteration.

Definition 5. *The* throughput *of an SDF graph mapped on a processor application model is the average number of graph iterations that are executed per unit time, measured over a sufficiently long period.*

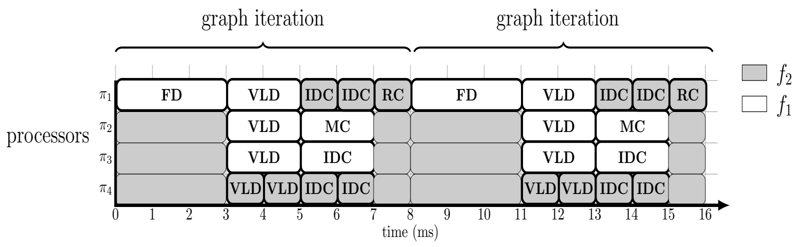


Fig. 2: Schedule of our running example

A specific schedule termed *self-timed* [14] determines the maximal throughput of an SDF graph, in which an actor fires as soon as it is enabled. However, it is assumed that we have sufficiently many processors to accommodate all the enabled firings simultaneously.

3.5 Semantics

The dynamic behaviour of an SDF graph mapped on a PAM can naturally be understood in terms of a labelled transition system (LTS). Below, we define the LTS of an SDF graph $G = (A, D, \mathsf{Tok}_0, \tau)$ and a PAM $(\Pi, \zeta, F, P_{idle}, P_{occ}, P_{tr}, \tau_{act})$ by giving its states and transitions.

Definition 6. *A* state *is a tuple* $(Tok, status, freq, TuC, TotPow)$ *with the following components.*

- *edge quantity* $Tok : D \to \mathbb{N}$ *associates with each edge the number of tokens currently present in that edge, and*
- $status : \Pi \to \{idle, occup\}$ *and* $freq : \Pi \to F$ *associates with each processor* $\pi \in \Pi$*, iwhether it is idle or occupied, and its current frequency level* $f \in F$*.*
- *To observe the progress of time,* $TuC : \Pi \to \mathbb{N}$ *records for each processor the remaining execution time required to complete its current task.*
- $TotPow : \Pi \to \mathbb{N}$ *records the total accumulated power consumption for each processor.*

The initial state $(Tok_0, status_0, freq_0, TuC_0, TotPow_0)$ *is given by* $status_0(\pi) = idle$*,* $freq_0(\pi) = f_m$*,* $TuC(\pi) = 0$*,* $TotPow_0(\pi) = 0$ *for all* $\pi \in \Pi$*.*

Example 3. The initial state of the example explain in Section 3.4 is given by $(Tok_0, status_0, freq_0, TuC_0, TotPow_0) = ((0,0,0,0,0,0,0,0,1,1), (idle, idle, idle, idle), (f_2, f_2, f_2, f_2), (0,0,0,0), (0,0,0,0))$. Here, initial tokens in all edges are represented by Tok_0. The initial availability of the processors and their active frequency level is given by $status_0$ and $freq_0$ respectively. Similarly, TuC_0 and $TotPow_0$ represents the initial, remaining execution times and power consumption, of the processors respectively.

Definition 7. *Let us consider an SDF graph $G = (A, D, \mathsf{Tok_0}, \tau)$ and a platform application model $(\Pi, \zeta, F, P_{idle}, P_{occ}, P_{tr}, \tau_{act})$. A* transition *from state $(Tok_1, status_1, freq_1, TuC_1, TotPow_1)$ to $(Tok_2, status_2, freq_2, TuC_2, TotPow_2)$ is denoted as,*

$$(Tok_1, status_1, freq_1, TuC_1, TotPow_1) \xrightarrow{\kappa} (Tok_2, status_2, freq_2, TuC_2, TotPow_2)$$

The label κ is defined as $\kappa \in (A \times \Pi \times F \times \{start, end\}) \cup \{tick\} \cup (\Pi \times F \times F \times jump)$ and corresponds to the type of transition.

- *Label $\kappa = (a, \pi, f, start)$ denotes mapping and starting of an firing of an actor $a \in A$ on a processor $\pi \in \Pi$ at a frequency level $f \in F$. This transition may occur if*
  - *$\forall d \in In(a)$, $Tok_1(d) \geq CR(d)$ i.e. all input edges $d \in D$ have sufficiently many tokens,*
  - *$status_1(\pi) = idle$ i.e. processor π is currently unoccupied,*
  - *$\forall \pi' \in [\pi]$, $freq_1(\pi') = f$ i.e. the active frequency level of all processors in VFI $[\pi]$ is $f \in F$, and*
  - *$a \in \zeta(\pi)$ i.e. if actor a can be mapped on the processor π.*

  This transition results in,
  - *$\forall d \in In(a)$, $Tok_2(d) = Tok_1(d) - CR(d)$ i.e CR(d) tokens are removed from each incoming edge,*
  - *$\forall \pi' \neq \pi$, $status_2(\pi') = status_1(\pi')$, and $status_2(\pi) = occup$ i.e. processor $\pi \in \Pi$ is claimed,*
  - *$freq_2 = freq_1$ i.e. the active frequency level of all processors does not change,*
  - *$TuC_2(\pi) = TuC_1(\pi) \cup \tau_{act}(a, f)$, i.e., $\tau_{act}(a, f)$ is attached to the processor π, and*
  - *$TotPow_2 = TotPow_1$, i.e., this transition doest not cost any power.*

  Example 4. The actor FD in the example given in Section 3.4 takes the transition (FD,f_2,π_1,start) at time t=0 ms. As a result, one token is subtracted from the edge RC-FD.

- *Label $\kappa = (a, \pi, f, end)$ denotes ending of an firing by an actor $a \in A$ and releasing a processor $\pi \in \Pi$ operating at a frequency level $f \in F$. This transition may occur if,*
  - *$TuC_1(\pi) = 0$, i.e., actor $a \in A$ has finished its execution.*

  This transition results in,
  - *$\forall d \in Out(a)$, $Tok_2(d) = Tok_1(d) + PR(d)$, i.e., PR(d) tokens are produced on all output edges,*
  - *$TuC_2 = TuC_1$,*
  - *$\forall \pi' \neq \pi$, $status_2(\pi') = status_1(\pi')$, and $status_2(\pi) = idle$ i.e. processor $\pi \in \Pi$ is released,*
  - *$freq_2 = freq_1$, i.e., active frequency level of all processors does not change, and*
  - *$TotPow_2 = TotPow_1$, i.e., this transition doest not cost any power.*

Example 5. In the example given in Section 3.4, the actor FD takes the transition (FD,f_2,π_1,end) at time t=2 ms. As a result, five tokens are produced on the edges FD-IDC and FD-VLD, and one token is produced on the edges FD-MC and FD-RC.

– *Label κ = tick denotes a clock tick transition. This transition is enabled if,*
  - $\forall \pi' \in \Pi$, $TuC_1(\pi') \neq 0$, *i.e., no end transition is enabled,*

  For all $d' \in D$ and $\pi' \in \Pi$, this transition results in
  - $Tok_2(d') = Tok_1(d')$,
  - $status_2(\pi') = status_1(\pi')$,
  - $freq_2(\pi') = freq_1(\pi')$,
  - $TuC_2(\pi') = TuC_1(\pi') - 1$, *i.e., the remaining execution time assigned to the processors is decreased by 1,*
  - *if* $status_1(\pi') = occup$, *then* $TotPow_2(\pi') = TotPow_1(\pi') + P_{occ}(\pi', freq_1(\pi'))$, *and*
  - *if* $status_1(\pi') = idle$, *then* $TotPow_2(\pi') = TotPow_1(\pi') + P_{idle}(\pi', freq_1(\pi'))$.

Example 6. In our running example, there are two tick transitions between t=0 ms and t=2 ms, because no end transition is no enabled in that period. At t=0 ms, the execution time of the actor FD, i.e., $\tau_{act}(FD, f_2)$ is attached to the processor π_1. After two tick transitions, the remaining execution time assigned to the processor π_1 equals 0, and therefore end transitions is taken at t=2 ms.

– *Label $\kappa = (\Pi_j, f_i, f', jump)$ denotes a transition of all processors $\pi' \in \Pi_j$ running at a frequency level $f_i \in F$, to another frequency level $f' \in F$ such that $f' = f_{i+1}$ or $f' = f_{i-1}$. This transition is enabled if,*
  - *for all $\pi' \in \Pi_j$, $freq_1(\pi') = f_i$ and $status_1(\pi') = idle$, i.e., processors in the same VFI can change to another frequency level only if they all are in the idle state at same frequency level.*

  This transition results in,
  - $\forall d \in D$, $\rho_2(d) = \rho_1(d)$, *i.e., token distribution does not change,*
  - $\forall \pi' \in \Pi_j$, $status_2(\pi') = idle$ *and* $freq_2(\pi') = f'$, *i.e., active frequency level of all processors in the same VFI changes to $f' \in F$,*
  - $TuC_2 = TuC_1$, *and*
  - $\forall \pi' \in \Pi_j$, $TotPow_2(\pi') = TotPow_1(\pi') + P_{tr}(\pi')$, *i.e., transition overhead of all processors belonging to the same VFI.*

Example 7. In Figure 3.4, there are two jump transitions at time t=0 ms, i.e., $(\Pi_2, f_2, f_1, jump)$ and $(\Pi_3, f_2, f_1, jump)$.

Definition 8. *Let us consider an SDF graph $G = (A, D, \mathsf{Tok}_0, \tau)$ and a platform application model $(\Pi, \zeta, F, P_{idle}, P_{occ}, P_{tr}, \tau_{act})$. An* execution σ *is defined as an finite or infinite sequence of states and transitions;* $\sigma = s_0 \xrightarrow{\kappa_0} s_1 \xrightarrow{\kappa_1} \ldots$.

SDF graphs may end up in a deadlock due to inappropriate consumption and production rates.

Definition 9. *Let us consider an SDF graph $G = (A, D, \mathsf{Tok_0}, \tau)$ and a platform application model $(\Pi, \zeta, F, P_{idle}, P_{occ}, P_{tr}, \tau_{act})$. An SDF graph experiences a deadlock if and only if its execution has a state $(Tok, status, freq, TuC, TotPow)$ in which $\forall a \in A$, $\exists d \in In(a)$ and $\forall \pi \in \Pi$, we have $Tok(d) \ngeq CR(d)$ and $TuC(\pi) = 0$.*

4 Power Optimisation

This section illustrates the importance of considering DPM along with DVFS, with the help of a non-trivial observation. Furthermore, we explain how VFIs allow us to achieve fine-grain power optimisation, by combining DPM with any granularity of DVFS. Let us consider a real-time periodic application mapped on a single processor. Figure 3 shows the behaviour of static (E_S) and dynamic (E_D) energy consumption of the processor as a function of processor frequency for the execution of an entire iteration. Note that E_S also includes transition overheads. The minimum frequency at which the task can meet its deadline is denoted by f_a. Similarly, f^* denotes the minimum frequency at which there is enough slack for the processor to move to the low power state. Thus, the processor can only move to the low power state, if its frequency is no less than f^*. Otherwise, it will not be able to meet the deadline.

As explained earlier, E_D increases cubically with the increase of frequency. However, E_S shows varying patterns. In Region A where $f_a \leq f < f^*$, idle period of the processor is too short to allow it to move to the low power state where static energy consumption is lower. Therefore, E_S is higher and constant in Region A. However, as frequency reaches f^*, slack, i.e., idle period of the processor increases, allowing the transition to less static power consuming states. Thus, E_S drops down at $f = f^*$. As frequency increases beyond f^* in Region B, idle period of the processor increases further in linear fashion, leading to switching to deeper sleep states by the processor. Without loss of generality, if we assume that transition overhead of switching to deeper low power states also increases linearly, we get linear decrease of E_S with the increase of frequency in Region B, as shown in Figure 3.

Figure 4 shows $E_{Tot} = E_S + E_D$, as a function of processor frequency. In Region A where E_{Tot} grows with the increasing frequency, local minimum f_{opt1} of E_{Tot} is $f_{opt1} = f_a$. Whereas, in Region B, E_{Tot} decreases with the increasing frequency. The local minimum f_{opt2} of E_{Tot} in Region B is $f_{opt2} \in [f^*, f_{max}]$. Depending on power consumption of low power states, and transition overheads, steepness of E_S can increase or decrease in Region B. As a result, the minimum value of E_{Tot} can have different values in Region B, as shown by dashed lines.

As we have seen that local optimal frequencies to minimise E_{Tot} in both regions are well defined. However, *there is no a priori reason that global minimum of E_{Tot} should lie in Region A or Region B*. Depending on the power consumption values of the processor and deadline of the application, the global optimal frequency can be either in Region A or B.

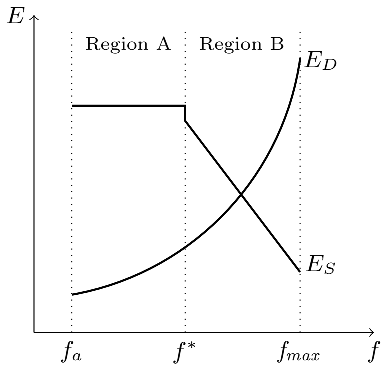


Fig. 3: Static (E_S) and Dynamic (E_D) Energy

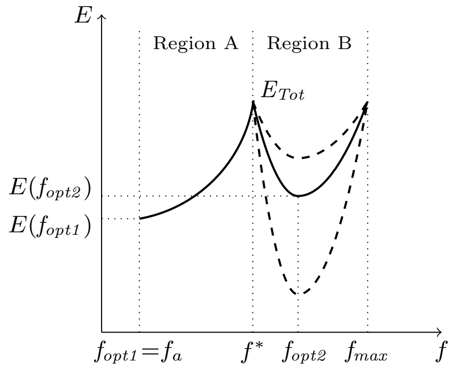


Fig. 4: Total Energy (E_{Tot})

Voltage(V)	**Frequency(MHz)**	**GDVFS**		**GDVFS+DPM**	
		P_{idle}(W)	P_{occ}(W)	P_{idle}(W)	P_{occ}(W)
1.2	1400	0.4	4.6	0.1	4.6
1.00	1032.7	0.4	1.8	0.4	1.8

Table 3: Platform description

Alternatively, if we do not consider DPM, E_S in Region B remains same as A, and consequently, E_{Tot} increases in Region B as well. Therefore, we can safely conclude that we must consider *both* DPM and DVFS to determine optimal power consumption. We can generalise this result for multiprocessors also. Moreover, partitioning processors into VFIs enable us to assign frequency per partition, rather than running all processors at the same frequency.

To illustrate earlier arguments, let us consider an example of an MPEG-4 decoder shown in Figure in 1, capable of processing 5 macroblocks, mapped on the platform containing four Samsung Exynos 4210 processors, i.e., $\Pi = \{\pi_1, \pi_2, \pi_3, \pi_4\}$. For the deadline of completing 3 graph iterations within 23 ms, we consider following scenarios.

– **Case 0**: Without Power Optimisation ($\texttt{No} - \texttt{PowerOpt}$)
Let's assume that the processors do not utilise any power management technique. The only frequency $f \in F$ available to the processors is $f = 1400$ MHz. The idle (static) and operating (dynamic) power consumption at $f = 1400$ MHz is $P_{idle}(\pi, f)$=0.4 W and $P_{occ}(\pi, f)$=4.8 W respectively. Figure 5 shows the optimal execution of this case, where we can see that the constraint of finishing 3 graph iterations is met well before the deadline, and the processors remain idle for the rest of the time resulting in dynamic slack. Hence, DVFS is needed to minimise dynamic slack. The total power consumption of this case is 204.2 mWs.

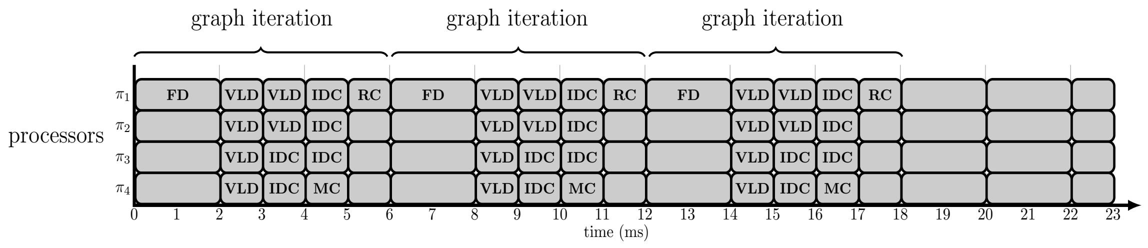


Fig. 5: Optimal Execution showing $\texttt{No} - \texttt{PowerOpt}$

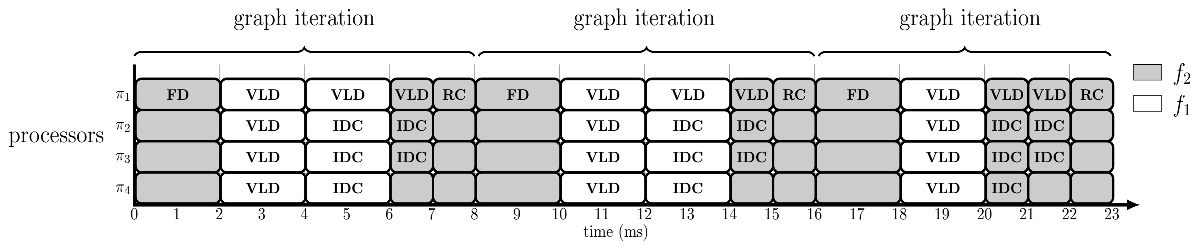


Fig. 6: Optimal Execution showing $\texttt{GDVFS}$

- **Case 1**: Global DVFS only ($\texttt{GDVFS}$) Now, to introduce DVFS in processors, we add an extra frequency level (MHz), i.e., $\{f_1, f_2\} \in F$ such that $f_2 = 1400$ and $f_1 = 1032.7$. In this case, the processors employ DVFS only, without considering DPM and VFIs. Table 3 shows the idle (static) and operating (dynamic) power consumption at both frequencies. Note that, idle power consumption of all processors $\pi \in \Pi$ is constant at both frequencies, i.e., $P_{idle}(\pi, f_2)$=$P_{idle}(\pi, f_1)$=0.4 W. Recall that $\texttt{GDVFS} + \texttt{DPM}$ assumes one VFI $\Pi_1 = \{\pi_1, \pi_2, \pi_3, \pi_4\}$. The optimal execution of this scenario is shown in Figure 6. As we can see in Figure 6, the constraint of finishing 3 graph iterations is fulfilled exactly at the deadline, as opposed to $\texttt{No} - \texttt{PowerOpt}$. Thus, DVFS helps to reduce dynamic slack. As a result, the total energy consumption drops to 185.2 mWs from 204.2 mWs. However, in the case of $\texttt{GDVFS}$, the processors consume high static power while being idle, leading to static slack. Therefore, we must utilise DPM to reduce static slack.
- **Case 2**: Global DVFS + DPM ($\texttt{GDVFS} + \texttt{DPM}$)
  In order to allow processors benefit from both DPM and DVFS, we introduce a low power state, i.e., idle power consumption of all processors $\pi \in \Pi$ at frequency level $f_2 = 1400$ MHz is changed to $P_{idle}(\pi, f_2)$=0.1 W because more idle time allows DPM. However, the operating power consumption of all processors $\pi \in \Pi$ at both frequencies, i.e., $P_{occ}(\pi, f_2)$ and $P_{occ}(\pi, f_1)$ remains same as $\texttt{GDVFS}$, as given in Table 3. The transition overhead (W) of all processors $\pi \in \Pi$ is, $P_{tr}(\pi, f_2, f_1) = 0.2$ and $P_{tr}(\pi, f_1, f_2) = 0.1$. In this case, the schedule remains the same. However, the total energy consumption drops significantly to 179.8 mWs. Hence, it shows that optimality of

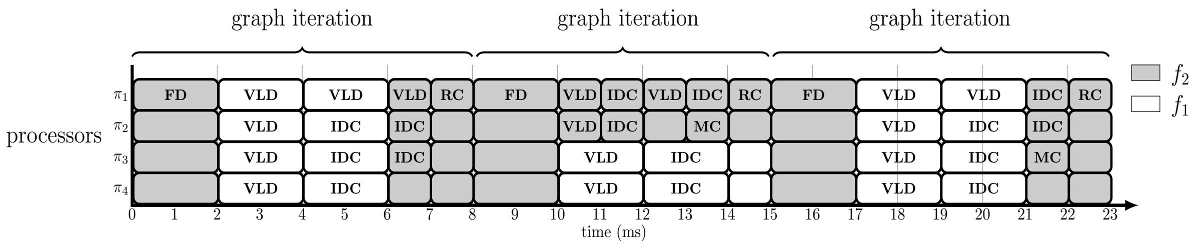


Fig. 7: Optimal Execution showing $\texttt{DVFS} + \texttt{DPM} - 2$

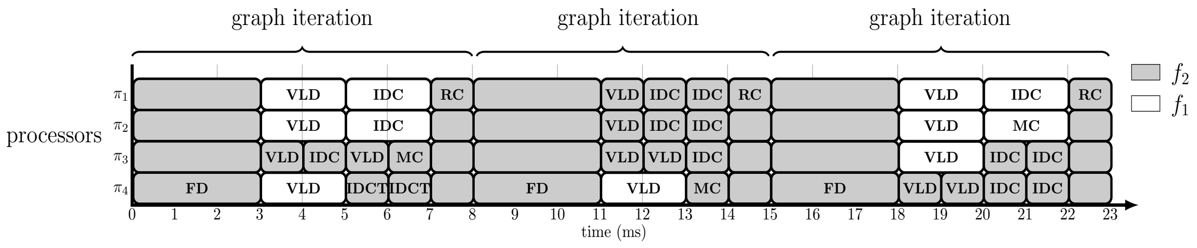


Fig. 8: Optimal Execution showing $\texttt{DVFS} + \texttt{DPM} - 3$

power minimisation can only be guaranteed by considering both DPM and DVFS. However, as we may observe, all processors run at the same frequency in $\texttt{GDVFS} + \texttt{DPM}$, which might be unnecessary. Instead, we may partition processors into VFIs so that only required processors run at the same frequency, and others may run at the different frequency.

- **Case 3**: DVFS + DPM with 2 VFIs ($\texttt{DVFS} + \texttt{DPM} - 2$)
  In this scenario, we partition processors into two VFIs such that $\Pi_1 = \{\pi_1, \pi_2\}$ and $\Pi_2 = \{\pi_3, \pi_4\}$, while utilising both DVFS and DPM. The power consumption values of processors at both frequencies remain the same as in Case 2. As a result, total energy consumption reduces to 179.2 mWs, demonstrating the effectiveness of VFIs to achieve fine-grain power management. The optimal execution of this case is shown in Figure 8.
- **Case 4**: DVFS + DPM with 3 VFIs ($\texttt{DVFS} + \texttt{DPM} - 3$)
  The total energy consumption drops further to 176.5 mWs, if we partition processors into three VFIs such that $\Pi_1 = \{\pi_1\}$, $\Pi_2 = \{\pi_2\}$ and $\Pi_3 = \{\pi_3, \pi_4\}$. Figure 7 shows the optimal execution of Case 4, i.e., $\texttt{DVFS} + \texttt{DPM} - 3$.

5 Priced Timed Automata

Timed automata are a popular and powerful formalism to model and analyse real-time systems [4]. TA are state-transition diagrams augmented with real-valued clocks, which can be used in enabling conditions for transitions and in state invariants that enforce deadlines. Networks of timed automata can be build from TA components and communicate via signals, i.e., action labels on transitions.

Price timed automata (PTA) extend TA with costs. Costs can either be accumulated in states, proportionally to the residence time, or by taking a transition. Moreover, TA and PTA can be analysed for a wide number of properties, including absence of deadlocks, safety, and liveness.

We use *B(C)* to denote the set of clock constraints for a finite set of clocks C. That is, *B(C)* contains all of conjunctions over simple conditions of the form $x \bowtie c$ or $x - y \bowtie c$, where $x, y \in C$, $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$.

Definition 10. *A* priced timed automaton $\mathcal{A}$ *over clocks* C *and actions* Act *is a tuple* $(L, E, Inv, \mathcal{P}, l^0)$*, where* L *is a set of locations;* $E \subseteq L \times Act \times B(C) \times 2^C \times L$ *is a set of edges;* $Inv : L \to B(C)$ *assigns an invariant to each location;* $\mathcal{P} : (L \cup E) \to \mathbb{N}$ *assigns costs to edges and locations, and* $l^0 \in L$ *is the initial location.*

In particular, Uppaal Cora has a support for finding cost-optimal schedules. Optimality is defined in terms of a variable named `cost`. The optimal trace can be found by using the `Best trace` option. With this option, Uppaal Cora keeps searching until a trace to a goal state with the smallest value for the cost variable has been found. The rate of growth of *cost* is specified as $cost'$.

6 Translation of SDF Graphs to Priced Timed Automata

Our framework consists of separate models of an SDF graph and the processor application model. In this way, we split the the problem of optimal power management in terms of tasks and resources. In this section, we describe the translation of an SDF graphs along with a processor application model to PTA using Uppaal Cora.

Given an an SDF graph $G = (A, D, \mathsf{Tok}_0, \tau)$ mapped on a processor application model $(\Pi, \zeta, F, P_{idle}, P_{occ}, P_{tr}, \tau_{act})$, we generate a parallel composition of PTA:

$$A_G \| Processor_1 \|, \ldots, \| Processor_n \| Scheduler.$$

PTA models of the example given in subsection 3.4 is shown in Figure 1. Here, the automaton A_G models the actors and edges of an SDF graph as shown in Figure 9a. The PTA $Processor_1, \ldots, Processor_n$ model the processors $\Pi = \{\pi_1, \ldots, \pi_n\}$, as shown in Figure 9b. Figure 9c presents the automaton of *Scheduler*, that decides when to switch the frequency level of all processors in the same VFI. Note that the resulting timed automaton is trivially extensible in the number of processors. Thus, the translation is, at least, composable with regards to the processor application model. We assume that the underlying LTS of G is given by $(S, Lab, \to_G)$ where $S = (Tok, status, freq, TuC, TotPow)$ denotes the states, $Lab = \kappa$ denotes the labels and $\to_G \subseteq S \times Lab \times S$ depicts the edges.

Priced Timed Automaton A_G. The automaton A_G is defined as, $A_G = (L, Act, P, E, Inv, l^0)$ where $L = l_0 = \{\mathsf{Initial}\}$ is the only location in our SDF graph model. The action set $Act = \{\mathsf{fire!}, \mathsf{end?}\}$ contains two parametrised actions, i.e., fire! (exclamation mark signifies a sending operation) and end? (question mark signifies a receiving operation) to synchronise with the PTA $Processor_1, \ldots, Processor_n$.

For each processor $\pi \in \Pi$ and $a \in A$, fire[π][a] represents the start of the execution of actor a on a processor π, and end[π][a] represents its ending. The action fire[π][a] is enabled if the incoming buffers of $a \in A$ have sufficient tokens. We do not have any clocks and invariants in A_G. Therefore, *Inv:* $L \rightarrow B(C)$ and $Inv(l^0) = true$. For each $a \in A$ and $d \in D$, E contains two edges such that:

– $\mathsf{Initial} \xrightarrow{Tok(\mathsf{d}) \geq \mathsf{CR(d)}, \mathsf{fire}[\pi][\mathsf{a}]!, Tok(\mathsf{d}) := Tok(\mathsf{d}) - \mathsf{CR(d)}} \mathsf{Initial}$
– $\mathsf{Initial} \xrightarrow{\varnothing, \mathsf{end}[\pi][\mathsf{a}]?, Tok(\mathsf{d}) := Tok(\mathsf{d}) + \mathsf{PR(d)}} \mathsf{Initial}.$

Here, $Tok(d) \geq CR(d)$ refers to a *guard* and it signifies that tokens on all input edges $d \in In(a)$ of an actor $a \in A$ must be greater than or equal to their consumption rate in order to take the action fire!. As a result of taking the action fire!, tokens on all input edges $d \in In(a)$ of an actor $a \in A$ are subtracted, i.e., $Tok(d) = Tok(d) - CR(d)$. Similarly, by taking the action end?, actor firing is completed and tokens are produced on all output edges $d \in Out(a)$ of an actor $a \in A$, i.e., $Tok(d) = Tok(d) + PR(d)$.

A_G contains a number of variables: for each edge from actors $a \in A$ to $b \in B$, an integer variable buff_a2b $= Tok(\mathsf{a}, \mathsf{b}, \mathsf{p}, \mathsf{q})$ containing the number of tokens in the buffer from a to b. The variable counter_a counts how many times actor $a \in A$ has been fired. counter_a is used to keep an account of number of firings of each actor $a \in A$ in an execution. Initially, counter_a $= 0$ and buff_a2b $= Tok_0(\mathsf{a}, \mathsf{b}, \mathsf{p}, \mathsf{q})$ contains the number of tokens in the initial distribution of the edge (a, b, p, q). The function *estimate*() provides a estimate of lower bound on the remaining cost, which is used to improve the performance of Uppaal Cora.

The action fire[π][a] consumes, from each input edge $(b, a, p, q) \in In(a)$ in G, the q tokens from the buffer buff_b2a, and is carried out by the function consume(buff_b2a, q).

The action end[i][a] adds, for each actor $a \in A$ and output edge $(a, b, p, q) \in Out(a)$ in G, the p tokens on the buffer buff_a2b by carrying out the function produce(buff_a2b, p).

Finally, we note that the edges are parametrised in processor id's but not in actors. This is because each edge in Uppaal Cora can contain only one parameter. As stated earlier, this paper uses SDF graphs to represent software applications. Since the translation is defined by induction on the structure of SDF graphs, it is also composable in the (software) applications.

Priced Timed Automata $Processor_j$. Likewise, for each $\pi_j \in \Pi$, we define PTA $Processor_j = (L_j, Act_j, P_j, E_j, Inv_j, l_j^0)$. For each frequency level $f_i \in F$, we include both an idle state and an active state running on that frequency level. Thus, for each $a \in \zeta(\pi_j)$ and $F = \{f_1, \ldots, f_m\}$ such that $f_1 < f_2 < \ldots < f_m$, let $L_j = \{\mathsf{Idle\_f1}, \ldots, \mathsf{Idle\_fm}, \mathsf{InUse\_a\_f1}, \ldots, \mathsf{InUse\_a\_fm}\}$ indicating that processor

$\pi_j \in \mathit{\Pi}$ is currently used by the actor $a \in A$ in the frequency level $f_i \in F$, either in idle or running state. For $F = \{f_1, \ldots, f_m\}$ such that $f_1 < f_2 < \ldots < f_m$, $l_j^0 = \mathsf{Idle\_fm}$. This explains that a processor $\pi \in \mathit{\Pi}$ always start at the highest frequency level $f_m \in F$. Furthermore, for each actor $a \in \zeta(\pi)$ and frequency level $f_i \in F$, $P_j(\mathsf{Idle\_fi}) = P_{idle}(\pi, \mathsf{f_i})$, $P_j(\mathsf{InUse\_a\_fi}) = P_{occ}(\pi, \mathsf{f_i})$, $Inv_j(\mathsf{Idle\_fi}) = true$, and $Inv_j(\mathsf{InUse\_a\_fi}) \le \tau_{act}(a, f_i)$ enforcing the system to stay in InUse_a_fi for at most the execution time $\tau_{act}(a, f_i)$. As we only can have integer costs, all values of power consumption is multiplied by 10 and rounded to the nearest integer. Please note that $Processor_j$ contains exactly one clock x_j; since clocks in Uppaal Cora are local, we can abbreviate x_j by x. A separate clock variable *global* observes the overall time progress.

The action set $Act_j = \{\mathsf{fire?}, \mathsf{end!}, \mathsf{jump\_ik?}\}$ contains three actions fire?, end! and jump_ik?. The actions fire? and end! in Act_j are parametrised with the processor and actor ids, and synchronise with A_G. The action jump_ik? in Act_j is parametrised with the VFI id. For all $f_i, f_k \in F$, and $\pi_j \in \mathit{\Pi}_y$, the broadcast action $\mathsf{jump\_ik}[y]$ synchronises the automata $Processor_1, \ldots, Processor_n$ with the automaton *Scheduler*, to switch all processors in the VFI $[\pi_j]$ from the frequency level f_i to f_k.

For each $\pi \in \mathit{\Pi}$, $a \in \zeta(\pi)$ and $f_i \in F$, E contains two transitions such that:

- $\mathsf{Idle\_fi} \xrightarrow{\varnothing, \mathsf{fire}[\pi][\mathsf{a}]?, \{\mathsf{x:=0}\}} \mathsf{InUse\_a\_fi}$, and
- $\mathsf{InUse\_a\_fi} \xrightarrow{\mathsf{x}=\tau_{act}(\mathsf{a},\mathsf{f_i}), \mathsf{end}[\pi][\mathsf{a}]!, \varnothing} \mathsf{Idle\_fi}$.

The action $\mathsf{fire}[\pi][\mathsf{a}]$ is enabled in the idle state Idle_fi and leads to the location InUse_a_fi. Thus, $\mathsf{fire}[\pi][\mathsf{a}]$ "claims" the processor $\pi \in \mathit{\Pi}$ at frequency level $f_i \in F$, so that any other firing cannot run on $\pi \in \mathit{\Pi}$ before the current firing of $a \in A$ is finished. As each location InUse_a_fi has an invariant $Inv_j(\mathsf{InUse\_a\_fi}) \le \tau_{act}(\mathsf{a}, \mathsf{f_i})$, the automaton can stay in InUse_a_fi for at most the execution time of actor $a \in A$ at frequency level $f_i \in F$, i.e., $\tau_{act}(a, f_i)$. If $x = \tau_{act}(a, f_i)$, the system has to leave InUse_a_fi at exactly the execution time of actor $a \in A$ at frequency level $f_i \in F$, by taking the $\mathsf{end}[\pi][\mathsf{a}]$ action. In this way, A_G is notified that the execution of $a \in A$ has ended, so that A_G updates the buffers and other variables.

For $F = \{f_1, \ldots, f_k, f_i\}$ such that $f_1 < f_2 < \ldots < f_k < f_i$, and $\pi_j \in \mathit{\Pi}_y$, E has following transitions $E_{broad} \in E$ for handling broadcast such that:

- $\mathsf{Idle\_fi} \xrightarrow{\varnothing, \mathsf{jump\_ik}[\mathsf{y}]?, \varnothing} \mathsf{Idle\_fk}$,
- $\mathsf{Idle\_fk} \xrightarrow{\varnothing, \mathsf{jump\_ki}[\mathsf{y}]?, \varnothing} \mathsf{Idle\_fi}$,

  $\vdots$
- $\mathsf{Idle\_f1} \xrightarrow{\varnothing, \mathsf{jump\_12}[\mathsf{y}]?, \varnothing} \mathsf{Idle\_f2}$

Furthermore, for all $f_i, f_k \in F$, $\pi_j \in \mathit{\Pi}_y$, and $E_{broad} \in E$, $P_j(\mathsf{Idle\_fi} \xrightarrow{\varnothing, \mathsf{jump\_ik}[\mathsf{y}]?, \varnothing} \mathsf{Idle\_fk}) = P_{tr}(\pi_\mathsf{j}, \mathsf{f_i}, \mathsf{f_k})$. For all $\pi_j \in \mathit{\Pi}_y$, $Processor_j$ has a variable $\mathsf{freq\_lev}[\mathsf{y}]$ to count the processors in the running state. Initially, $\mathsf{freq\_lev}[\mathsf{y}] = 0$ for all $\pi_j \in \mathit{\Pi}_y$. If a processor $\pi_j \in \mathit{\Pi}_y$ is claimed by an actor $a \in A$, the counter $\mathsf{freq\_lev}[\mathsf{y}]$ is incremented by one. Similarly, if a processor $\pi_j \in \Pi_j$ is released, the value of the counter freq_lev[y] is reduced by one.

For all $\pi_j \in \Pi_y$, $Processor_j$ has another variable curr_freq[y] that determines the current frequency level of all $\pi_j \in \Pi_y$. Initially, for $F = \{f_1, f_2, \ldots, f_m\}$ such that $f_1 < f_2 < \ldots < f_m$, and for all $\pi_j \in \Pi_y$, curr_freq[y] $= m$. In Figure 9b, for all $\pi_j \in \Pi_y$, the initial value of curr_freq[y] $= 2$ denoting that the highest frequency level is $f_2 \in F$. For all $\pi_j \in \Pi_y$, when action jump_21[y] is taken, the value of curr_freq[y] changes to 1.

Priced Timed Automaton Scheduler. The automaton *Scheduler* is defined as, (L, Act, P, E, Inv, l^0) where $L = l_0 = \{\text{Initial}\}$ is the only location in the scheduler model. For $F = \{f_1, \ldots, f_k, f_i\}$ such that $f_1 < f_2 < \ldots < f_k < f_i$, $Act = \{\text{jump\_12}, \ldots, \text{jump\_ik}\}$ parametrised with the VFI ids, synchronises with the PTA $Processor_1, \ldots, Processor_n$. We do not have any clocks and invariants in *Scheduler*. Therefore, *Inv: L → B(C)* and $Inv(l^0) = true$. For $F = \{f_1, \ldots, f_k, f_i\}$ such that $f_1 < f_2 < \ldots < f_k < f_i$, and $\pi_j \in \Pi_y$, E has following transitions for broadcast such that:

- $\text{Initial} \xrightarrow{\text{freq\_lev[y]==0} \wedge \text{curr\_freq[y]==i,jump\_ik[y]!,} \varnothing} \text{Initial}$,
- $\text{Initial} \xrightarrow{\text{freq\_lev[y]==0} \wedge \text{curr\_freq[y]==k,jump\_ki[y]!,} \varnothing} \text{Initial}$,

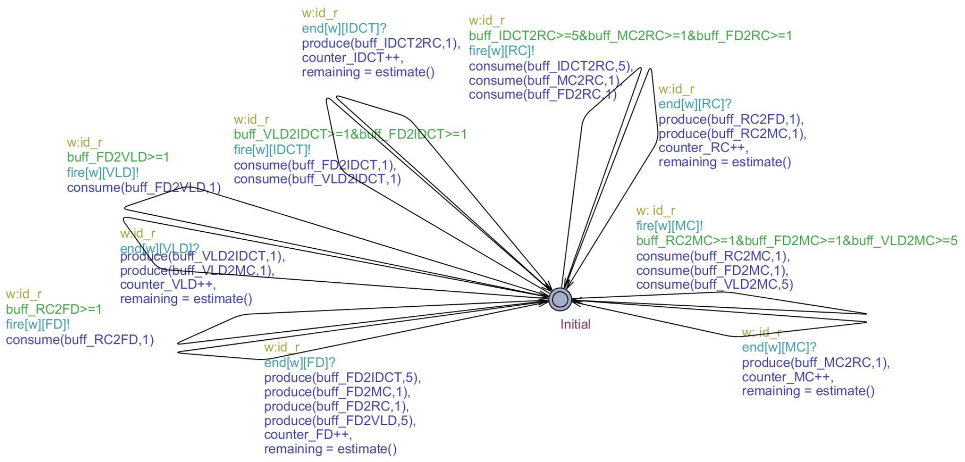
  $\vdots$
- $\text{Initial} \xrightarrow{\text{freq\_lev[y]==0} \wedge \text{curr\_freq[y]==1,jump\_12[y]!,} \varnothing} \text{Initial}$

For example, in Figure 9c, the action jump_21[y] is enabled when all processors $\pi_j \in \Pi_y$ are in the idle state and current frequency level is $f_2 \in F$, i.e., freq_lev[y] == 0&&curr_freq[y] == 2. When this action is taken, all processors $\pi_j \in \Pi_y$ synchronise with the automaton *Scheduler*, and change the frequency level to $f_1 \in F$. Same is the case with the other action jump_12[y].

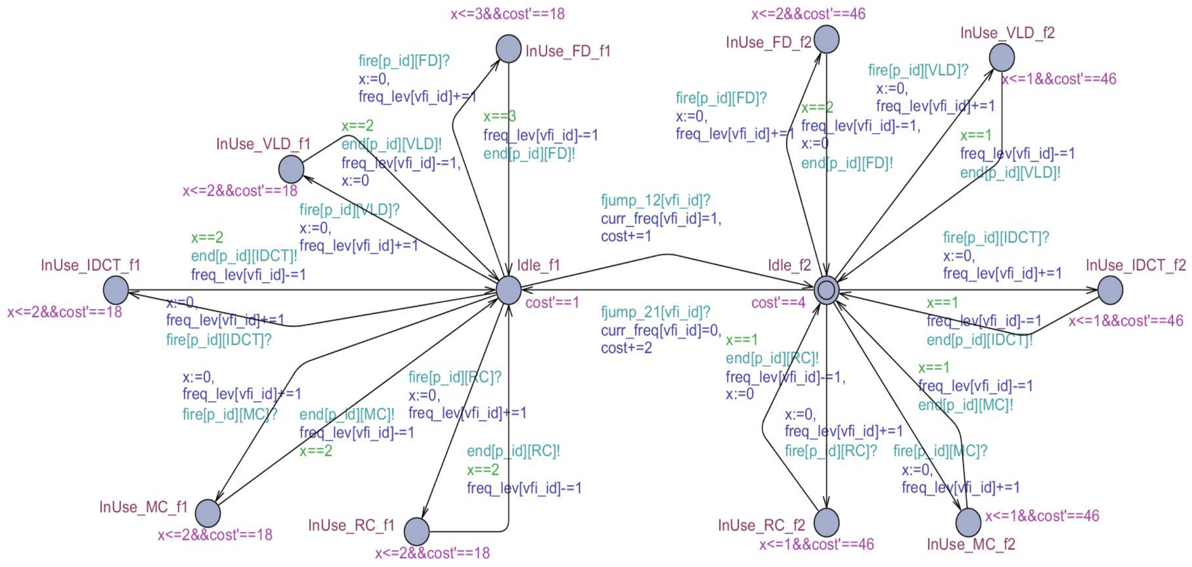
7 Power Optimisation using Uppaal Cora

This section illustrates how we use Uppaal Cora to obtain power optimal schedules. As explained earlier, each actor fires according the repetition vector γ in an iteration. For each actor $a \in A$ in the SDF graph, we define its corresponding entry in the repetition vector as $\gamma(a)$. We also define the number of iterations per period as m.

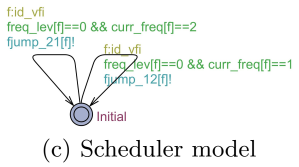
A technique of calculating the maximum throughput of an SDF graph mapped on a given number of processors via timed automata (TA), using the model-checker Uppaal is proposed in [3]. This work demonstrates that the *fastest execution* of every consistent and strongly connected SDF graph, mapped on a platform application model, repeats the periodic phase n times if each actor $a \in A$ fires equal to $(nm + k)\gamma(a)$ for some constants n and k. The maximal throughput of the SDF graph is determined from the periodic phase. For example, we know that repetition vector γ of the example SDF graph given in Section



(a) UPPAAL Cora model A_G



(b) UPPAAL Cora model $Processor_j$


f:id_vfi
freq_lev[f]==0 && curr_freq[f]==2
fjump_21[f]!
f:id_vfi
freq_lev[f]==0 && curr_freq[f]==1
fjump_12[f]!
Initial

(c) Scheduler model

Fig. 9: UPPAAL Cora editor showing SDF graph, Processor and Scheduler

6 is $\langle \mathrm{FD}, \mathrm{VLD}, \mathrm{IDC}, \mathrm{RC}, \mathrm{MC} \rangle = \langle 1, 5, 5, 1, 1 \rangle$. We can find out the throughput using the 3rd multiple of the repetition vector, i.e., $(nm + k) = 3\gamma(a)$ for all actors $a \in A$, by selecting the `Fastest` trace option in Uppaal and verifying the query: E⟨⟩`(counter_FD==3&counter_VLD==15&counter_IDC==15&counter_RC==3&counter_MC==3)`. We find the periodic phase from the generated trace, representing the maximum throughput. Thus, to compute a power minimal schedule from an SDF graph G and a PAM $\mathcal{P}$, we perform the following steps.

1. TA models are extracted such that $A_G \| Processor_1 \|, \ldots, \| Processor_n$.
2. We obtain the time T needed to complete the *fastest execution* of A_G, by running the query $Q_1 = E\langle\rangle \; (\bigwedge_{a \in A} counter\_a = (nm + k).\gamma(a))$ in Uppaal.
3. PTA models are extracted such that $A_G \| Processor_1 \|, \ldots, \| Processor_n \| Scheduler$.
4. We obtain the cheapest trace finished within time T, by running the query $E\langle\rangle \; (Q_1 \wedge time \leq T)$ in Uppaal Cora.
5. The trace is translated into a power optimal schedule. That is, by considering the action labels on the transitions, we know which actor is executed on which processor at which frequency.

For example, the *fastest execution* of the query mentioned above completes in 18 time units, if the corresponding SDF graph is mapped on 4 processors. If we add the constraint `global` = `18` to our earlier query in Uppaal Cora, we get the optimal schedule in terms of power utilisation at the maximum throughput on a given number of processors. The clock variable *global* is used to observe the overall time progress, and is never reset.

8 Experimental Evaluation via MPEG-4 Decoder

We analyse results of power optimisation by means of an example of the MPEG-4 decoder example in Figure 1 capable of 5 macroblocks. We evaluate energy consumption with respect to (1) fixed number of processors (2) varying number of processors. Finally, the method of verifying various user-defined properties using model-checking is explained in subsection 8.3.

8.1 Fixed Number of Processors

We consider an MPEG-4 decoder mapped on the platform containing four Samsung Exynos 4210 processors, i.e., $\Pi = \{\pi_1, \pi_2, \pi_3, \pi_4\}$. For the constraint of finishing 3 graph iterations with respect to varying deadlines, Figure 10 shows the energy consumption calculated for each scenario. The first two scenarios are compared as follows.

GDVFS vs GDVFS+DPM

– In almost all cases, considering DVFS only (`GDVFS`) results in higher energy consumption, as compared to considering the combination of DVFS and

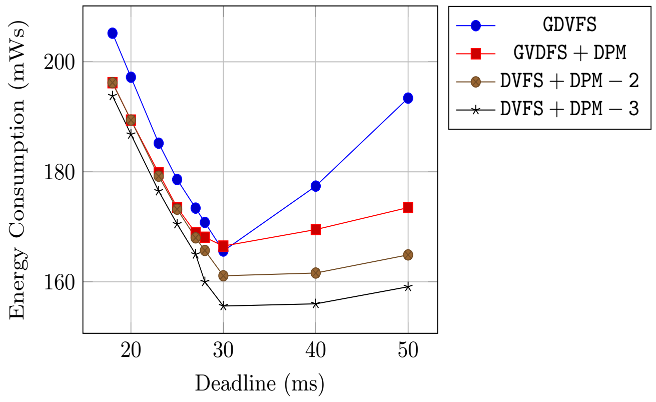


Fig. 10: Comparison of power optimisation techniques

DPM (GDVFS + DPM). However, at the deadline of 30ms, energy consumption in GDVFS + DPM surpasses GDVFS. If we compare the schedule of GDVFS and GDVFS + DPM at the deadline of 30 ms, we notice that it remains same. However, considering GDVFS + DPM includes transition overheads incursion to move to idle states, makes it less energy optimal than GDVFS.

- At tighter deadlines when idle time of processors is not sufficient to move to low power state, the difference between GDVFS and GDVFS + DPM is not significant. Thus, E_{Tot} lies in Region A. However, as deadline is relaxed, processors spend more time in low power state and E_{Tot} moves to Region B. Consequently, GDVFS + DPM gets more promising, implying the benefits of DPM. For example, at the deadline of 50 ms, GDVFS + DPM saves significant energy consumption equal to 10.3%, as compared to GDVFS.

Therefore, the results explained above prove our earlier claim that static power is non-negligible in order to guarantee optimality, and both Region A and B must be analysed to determine minimum energy consumption.

Now we have seen the benefits of DPM, the effect of varying the number of VFI partitions, i.e., GDVFS + DPM, DVFS + DPM − 2 and DVFS + DPM − 3 is described below.

DVFS+DPM with VFIs

- At tighter deadlines, for the reason that system is at maximum capacity all the time, having higher number of VFIs does not result in major energy reduction.
- But, as deadline is relaxed, we see that increasing the number of VFIs prove to be more effective, and produce considerable reduction in energy consumption. For example, for the deadline of 50 ms, DVFS + DPM − 2 and DVFS + DPM − 3 save 4.9% and 8.3% energy consumption respectively, as compared to GDVFS + DPM. The reason is that in GDVFS + DPM where we have

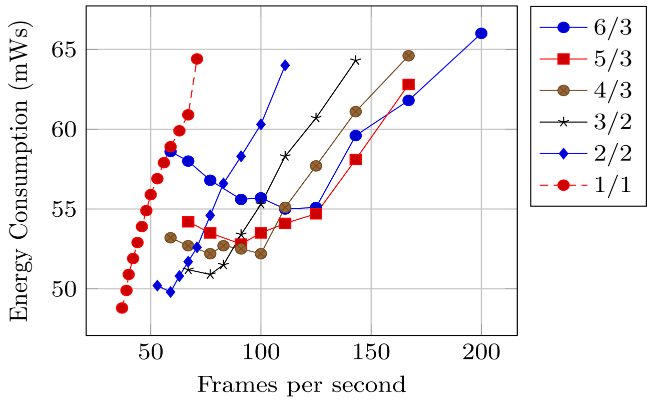


Fig. 11: Energy usage per frame against Frames per second. The legend shows the number of processors/VFIs.

one VFI only, all processors have to run at the same frequency, even though fewer might be required. By partitioning into more VFIs, we can cluster processors in such a way that only required processors run at the specific frequency, and others may run at the different frequency; thus, trading system's complexity for energy minimisation.

Hence, VFIs provide better control over energy optimisation and design complexity. Without VFIs, system designers are left with two options only, i.e. either local or global DVFS. However, with the help of VFIs, it is possible to achieve fine-grain power reduction by employing any DVFS policy, ranging from local to global. Therefore, the use of VFIs enables system designers with the larger range of design choices.

8.2 Varying Number of Processors

We also evaluated the performance of the MPEG-4 decoder on a varying number of processors. The maximum number of processors required for a self-timed execution [14] of this example is 6, calculated by SDF3. We obtain a Pareto front by sweeping the throughput constraint, as shown in Figure 11. We get three majors results from Figure 11, as explained below.

– Achieving higher frames per second at fewer processors increases the energy consumption. The reason is the smaller slack at the tighter frames per second constraint. Therefore, more work is done on fewer processors to attain same frames per second.
– As we relax the frames per second constraint, slack increases, and same frames per second can be achieved by consuming less energy on fewer processors. For instance, in Figure 11, we can reach 100 frames per second on 4 processors with 2.4% less energy consumption, as compared to 5 processors.

For higher slack in the application, this difference gets bigger. Thus, we may not require more processors in our platform, and reach a certain throughput at a considerably lower energy consumption, contributing to prolonged battery life.

- Relaxing the frames per second beyond a certain limit increases the energy consumption, as static energy surpasses the dynamic energy. For instance, the energy consumption of 3 processors increases by 1.9%, when moving from 77 to 59 frames per second.

8.3 Quantitative Analysis

We can analyse several functional and temporal properties of an MPEG-4 decoder using model-checking. This includes simple reachability properties such as, "does RC eventually fire?" and "after five consecutive VLD firings, MC must fire at least once". We can also check safety properties such as, "all processors belonging to the same VFI should never run at the different frequency". Similarly, liveness properties such as, "after a processor is occupied, it is eventually released" can also be verified.

9 Other case studies

Apart from the MPEG-4 decoder example, we present other real-life case studies, namely a bipartite graph [11] in Figure 12, an MP3 playback application [36] in Figure 13, an MP3 decoder [8] in Figure 14 and an audio echo canceller [36] in Figure 15. The execution times of these case studies are given in ms. We assume that these case studies are mapped on a multiprocessor platform containing Samsung Exynos 4210 processors $\Pi = \{\pi_1, \ldots, \pi_n\}$. Table 2 shows the considered frequency levels, and assumed power consumption of Exynos 4210 processors. For easier understanding, we only consider deadline constraint equal to minimum achievable time (ms) per iteration on a given number of processors. We also assume that, for all actors $a \in A$, $\tau_{act}(a, f_1) = \lceil \frac{\tau_{act}(a,f_2)}{0.738} \rceil$.

Table 4 shows the results of the experiments to find out the least power consumption. The first column displays the given number of processors, and the second column represents division of processors into VFIs. Columns 3-4 depict per iteration, minimum achievable time (ms) and minimum energy consumption (mWs) respectively, on the given processors.

We could determine the exact number of processors required for a self-timed execution, using SDF3. Then, we apply our approach to derive an optimal schedule on a smaller number of processors to determine least power usage. As we can see in case of a Bipartite Graph in Table 4, reducing the number of processors to 3 does not deteriorate minimum time per iteration considerably, and decreases slightly to 44 ms. Nonetheless, the decrease in energy consumption per iteration is significant, equal to 6.8 mWs, due to presence of higher slack in the application. It clearly shows, that similar performance with substantially less power dissipation can be achieved, even with fewer processors than required for a self-timed

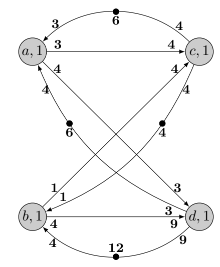


Fig. 12: Bipartite Graph [11]

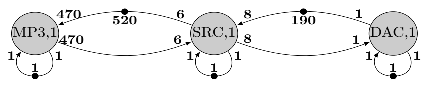


Fig. 13: MP3 Playback Application [36]

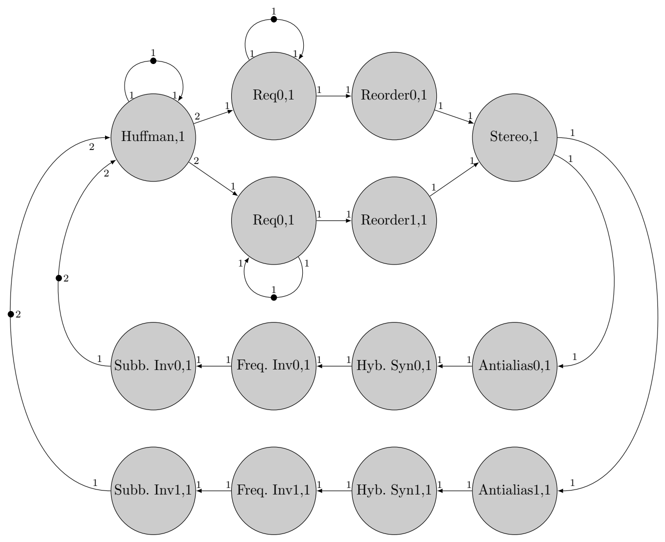


Fig. 14: MP3 Decoder [8]

execution. Thus, using model-checking, we generate a power optimal schedule automatically in a simple manner, on a given number of processors partitioned into VFIs, once the target state is specified in a query. We also check deadlock freedom effectively if a certain SDF graph is mapped on fewer processors than required for a self-timed execution.

So far, we have assumed a homogeneous system in which an actor can be mapped on any processor. A homogeneous system gives more freedom to decide which actor to assign to a particular processor. However, this freedom is limited

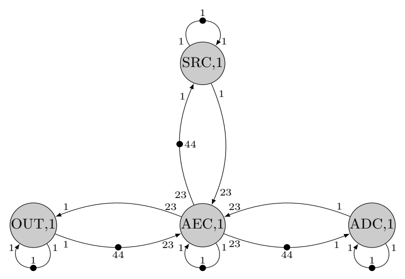


Fig. 15: Audio Echo Canceller [36]

Table 4: Experimental Results

Processor	**VFIs**	**Time per Iteration**	**Energy Consumption**
	Bipartite Graph in Figure 12		
4	$\Pi_1 = \{\pi_1, \pi_2\}, \Pi_2 = \{\pi_3, \pi_4\}$	42	345·3
3	$\Pi_1 = \{\pi_1\}, \Pi_2 = \{\pi_2, \pi_3\}$	44	338·5
2	$\Pi_1 = \{\pi_1\}, \Pi_2 = \{\pi_2\}$	51	333·1
1	$\Pi_1 = \{\pi_1\}$	73	335·8
	MP3 Playback Application in Figure 13		
2	$\Pi_1 = \{\pi_1, \pi_2\}$	1880	9907
1	$\Pi_1 = \{\pi_1\}$	2118	9742·8
	MP3 Decoder in Figure 14		
2	$\Pi_1 = \{\pi_1, \pi_2\}$	8	64·6
1	$\Pi_1 = \{\pi_1\}$	14	64·4
	Audio Echo Canceller in Figure 15		
4	$\Pi_1 = \{\pi_1, \pi_2\}, \Pi_2 = \{\pi_3, \pi_4\}$	23	324·2
3	$\Pi_1 = \{\pi_1\}, \Pi_2 = \{\pi_2, \pi_3\}$	24	322·3
2	$\Pi_1 = \{\pi_1, \pi_2\}$	35	322
1	$\Pi_1 = \{\pi_1\}$	73	335·8

in a heterogeneous system by which processors could be utilised to execute a particular actor.

In Uppaal Cora, we can utilise the same models described earlier in a heterogeneous system. Let us consider the SDF graph of the running MPEG-4 Decoder example mapped on a heterogeneous system containing two Samsung Exynos 4210 processors $\Pi' = \{\pi'_1, \pi'_2\}$ and two Samsung Exynos 4212 [1] processors $\Pi'' = \{\pi''_1, \pi''_2\}$. We assume that both Exynos 4210 and 4212 processors are available with same frequency level (MHz) $\{f_1, f_2\} \in F$ such that $f_2 = 1400$ and $f_1 = 1032.7$. Furthermore, let us consider following assumptions also.

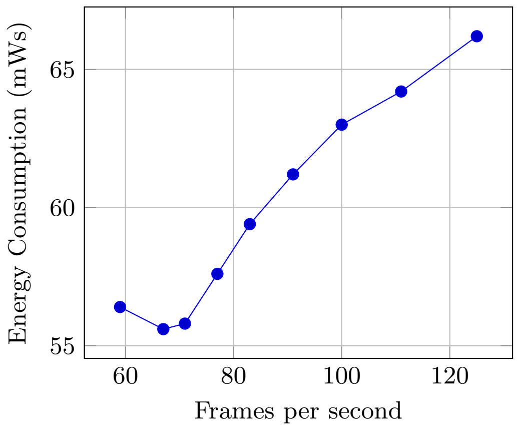


Fig. 16: Power consumption in a heterogeneous system

$$P_{tr}(\pi', f_2, f_1) = P_{tr}(\pi'', f_2, f_1)$$
$$P_{tr}(\pi', f_1, f_2) = P_{tr}(\pi'', f_1, f_2)$$

For all $\pi' \in \Pi', \pi'' \in \Pi''$ and $f \in F$,

$$P_{idle}(\pi', f) = P_{idle}(\pi'', f)$$

$$P_{occ}(\pi', f) = P_{occ}(\pi'', f)$$

Let us consider that the platform is implemented in such a way that actor $\{\text{FD}\} \subseteq A$ can be mapped only on the processor $\{\pi'_1\} \subseteq \Pi'$, actors $\{\text{VLD, IDC}\} \subseteq A$ can be executed only on the processors $\{\pi'_2, \pi''_2\} \subseteq \Pi' \cup \Pi''$, and the processor $\{\pi''_1\} \subseteq \Pi''$ is assigned to execute actors $\{\text{RC, MC}\} \subseteq A$ only. The processors are partitioned into VFIs in such a way that, $\Pi_1 = \{\pi'_1, \pi''_1\}$ and $\Pi_2 = \{\pi'_2, \pi''_2\}$. Figure 16 shows the Pareto front of total energy consumption for varying throughput constraint.

10 Conclusions

Despite the remarkable progress in power optimisation of deadline-constrained applications, compact methods for optimal power management of SDF graphs are still needed. In addition, with the growth of processing power in battery-constrained devices, efforts must be made to keep power utilisation to minimum. We demonstrate a novel power reduction technique for SDF-modelled streaming applications, which combines the benefits of DVFS and DPM using model-checking. This technique can be applied to any multiprocessor heterogeneous platform, having transition overheads and partitions of VFIs. By translating SDF graphs to PTA, we have also combined the flexibility of automata with the efficiency of SDF to obtain optimal schedules. Furthermore, with the help of contemporary model-checkers, benefits of analysable properties such as the absence of deadlocks, reachability etc. are also obtained.

Future research directions are to carry on from the results achieved in this paper and explore the possibilities of battery-aware scheduling of SDF graphs after including the kinetic battery model [23]. Future work also includes power optimal reachability analysis using weighted makespan [12], optimal VFI partitioning, and extending processor models with stochastics to accomplish advantages of probabilistic model checking. This will allow us to design self energy-supporting systems where energy generation, storage and consumption are kept in balance over the lifetime of a system. Another exciting prospect is to add a third dimension, taking also reliability relaxations and constraints into account.

References

1. Samsung Exynos 4 Dual 32nm (Exynos 4212). http://www.samsung.com/global/business/semiconductor/file/product/Exynos_-4_Dual_32nm_User_Manaul_Public_REV100-0.pdf.
2. Samsung Exynos 4 Dual 45nm (Exynos 4210). http://www.samsung.com/global/business/semiconductor/file/product/Exynos_-4_Dual_45nm_User_Manaul_Public_REV1.00-0.pdf.
3. W. Ahmad, P. K. F. Hölzenspies, M. Stoelinga, and J. van de Pol. Resource-constrained optimal scheduling of synchronous dataflow graphs via timed automata. ACSD, 2014.
4. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
5. G. Behrmann, K. G. Larsen, and J. I. Rasmussen. Optimal scheduling using priced timed automata. *SIGMETRICS Perform. Eval. Rev.*, 32(4):34–40, Mar. 2005.
6. L. Benini, A. Bogliolo, and G. De Micheli. A survey of design techniques for system-level dynamic power management. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 8(3):299–316, June 2000.
7. 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.
8. M. Damavandpeyma, S. Stuijk, T. Basten, M. Geilen, and H. Corporaal. Hybrid code-data prefetch-aware multiprocessor task graph scheduling. In *Digital System Design (DSD), 2011 14th Euromicro Conference on*, pages 583–590, Aug 2011.
9. P. de Langen and B. Juurlink. Leakage-aware multiprocessor scheduling for low power. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 8 pp.–, April 2006.
10. V. Devadas and H. Aydin. On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications. *Computers, IEEE Transactions on*, 61(1):31–44, Jan 2012.
11. M. Geilen, T. Basten, and E. Stuijk. Minimising buffer requirements of synchronous dataflow graphs with model checking. In *DAC '05*, pages 819–824. ACM, 2005.
12. M. Gerards, J. Hurink, and J. Kuper. On the interplay between global dvfs and scheduling tasks with precedence constraints, 2014.

13. M. E. T. Gerards and J. Kuper. Optimal dpm and dvfs for frame-based real-time systems. *ACM Trans. Archit. Code Optim.*, 9(4):41:1–41:23, Jan. 2013.
14. A.-H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, A. J. M. Moonen, M. Bekooij, B. Theelen, and M. Mousavi. Throughput analysis of synchronous data flow graphs. In *Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on*, pages 25–36, June 2006.
15. J.-J. Han, X. Wu, D. Zhu, H. Jin, L. T. Yang, and J.-L. Gaudiot. Synchronization-aware energy management for vfi-based multicore real-time systems. *IEEE Trans. Comput.*, 61(12):1682–1696, Dec. 2012.
16. S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pages 38–43, Aug 2007.
17. P. Huang, O. Moreira, K. Goossens, and A. Molnos. Throughput-constrained voltage and frequency scaling for real-time heterogeneous multiprocessors. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1517–1524, New York, NY, USA, 2013. ACM.
18. S. Irani and K. R. Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, June 2005.
19. W. Jang, D. Ding, and D. Pan. A voltage-frequency island aware energy optimization framework for networks-on-chip. In *Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on*, pages 264–269, Nov 2008.
20. E. Lee. Consistency in dataflow graphs. *IEEE Transactions on Parallel and Distributed Systems*, 2(2):223–235, 1991.
21. E. A. Lee and D. G. Messerschmitt. Synchronous data flow: Describing signal processing algorithm for parallel computation. In *COMPCON '87*, pages 310–315, 1987.
22. H. Liu, Z. Shao, M. Wang, and P. Chen. Overhead-aware system-level joint energy and performance optimization for streaming applications on multiprocessor systems-on-chip. In *Real-Time Systems, 2008. ECRTS '08. Euromicro Conference on*, pages 92–101, July 2008.
23. J. F. Manwell and J. G. McGowan. Lead acid battery storage model for hybrid energy systems. *Solar Energy*, 50(5):399 – 405, 1993.
24. J. L. March, J. Sahuquillo, H. Hassan, S. Petit, and J. Duato. A new energy-aware dynamic task set partitioning algorithm for soft and hard embedded real-time systems. *Comput. J.*, 54(8):1282–1294, Aug. 2011.
25. N. Navet and S. Merz. *Modeling and Verification of Real-time Systems*. Wiley, 2010.
26. A. Nelson, O. Moreira, A. Molnos, S. Stuijk, B. Nguyen, and K. Goossens. Power minimisation for real-time dataflow applications. In *Digital System Design (DSD), 2011 14th Euromicro Conference on*, pages 117–124, Aug 2011.
27. G. Norman, D. Parker, M. Kwiatkowska, S. Shukla, and R. Gupta. Using probabilistic model checking for dynamic power management. *Formal Aspects of Computing*, 17(2):160–176, 2005.
28. U. Ogras, R. Marculescu, P. Choudhary, and D. Marculescu. Voltage-frequency island partitioning for gals-based networks-on-chip. In *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, pages 110–115, June 2007.
29. S. Pagani, J.-J. Chen, and M. Li. Energy efficiency on multi-core architectures with multiple voltage islands, 2014.
30. S. Park, J. Park, D. Shin, Y. Wang, Q. Xie, M. Pedram, and N. Chang. Accurate modeling of the delay and energy overhead of dynamic voltage and frequency

scaling in modern microprocessors. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(5):695–708, May 2013.

31. A. K. Singh, A. Das, and A. Kumar. Energy optimization by exploiting execution slacks in streaming applications on multiprocessor systems. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 115:1–115:7, New York, NY, USA, 2013. ACM.
32. S. Stuijk, M. Geilen, and T. Basten. SDF^3: SDF For Free. In *Application of Concurrency to System Design, 6th International Conference, ACSD 2006, Proceedings*, pages 276–278. IEEE Computer Society Press, Los Alamitos, CA, USA, June 2006.
33. B. Theelen, M. C. W. Geilen, T. Basten, J. P. M. Voeten, S. V. Gheorghita, and S. Stuijk. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *Formal Methods and Models for Co-Design, 2006. MEMOCODE '06. Proceedings. Fourth ACM and IEEE International Conference on*, pages 185–194, July 2006.
34. Y. Wang, H. Liu, D. Liu, Z. Qin, Z. Shao, and E. H.-M. Sha. Overhead-aware energy optimization for real-time streaming applications on multiprocessor system-on-chip. *ACM Trans. Des. Autom. Electron. Syst.*, 16(2):14:1–14:32, Apr. 2011.
35. M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, 1994. USENIX Association.
36. M. Wiggers, M. Bekooij, and G. J. M. Smit. Efficient computation of buffer capacities for cyclo-static dataflow graphs. In *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, pages 658–663, June 2007.
37. J. Zhu, I. Sander, and A. Jantsch. Energy efficient streaming applications with guaranteed throughput on mpsocs. In *Proceedings of the 8th ACM & IEEE International conference on Embedded software, EMSOFT 2008, Atlanta, GA, USA, October 19-24, 2008*, pages 119–128, 2008.
38. S. Zhuravlev, J. Saez, S. Blagodurov, A. Fedorova, and M. Prieto. Survey of energy-cognizant scheduling techniques. *Parallel and Distributed Systems, IEEE Transactions on*, 24(7):1447–1464, July 2013.