Synthesizing Energy-Optimal Controllers for Multiprocessor Dataflow Applications with Uppaal Stratego*

Waheed Ahmad and Jaco van de Pol

University of Twente, Enschede, The Netherlands
{w.ahmad, j.c.vandepol}@utwente.nl

Abstract. Streaming applications for mobile platforms impose high demands on a system's throughput and energy consumption. Dynamic system-level techniques have been introduced, to reduce power consumption at the expense of performance. We consider DPM (Dynamic Power Management) and DVFS (Dynamic Voltage and Frequency Scaling). The complex programming task now includes mapping and scheduling every task onto a heterogeneous multi-processor hardware platform. Moreover, DPM and DVFS parameters must be controlled, to meet all throughput constraints while minimizing the energy consumption.
Previous work proposed to automate this process, by modeling streaming applications in SDF (Synchronous Data Flow), modeling the processor platform, translating both models to PTA (Priced Timed Automata, where prices model energy), and using Uppaal Cora to compute energy-optimal schedules that adhere to the throughput constraints.
In this paper, we experiment with an alternative approach, based on stochastic hybrid games. We investigate the applicability of Uppaal Stratego to first synthesize a permissive controller satisfying a throughput constraint, and then select a near-optimal strategy that additionally minimizes the energy consumption. Our goal is to compare the Uppaal Cora and Uppaal Stratego approaches in terms of modeling effort, results and computation times, and to reveal potential limitations.

1 Introduction

Power management. The power consumption of computing systems has increased exponentially [19]. Minimizing power consumption has become one of the most critical challenges for these systems. Therefore, over the past years, dynamic system-level power management has gained significant value and success [8, 19, 32]. Two well-known techniques are DVFS (Dynamic Voltage and Frequency Scaling) and DPM (Dynamic Power Management). 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 is also reduced, so the power is reduced cubically. Switching off idle processors saves on static power consumption.

---

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

DVFS [28] lowers the dynamic power consumption of modern processors, by lowering the voltage and clock frequency, at the expense of the execution time of a task. DPM switches the processor to a low power state when it is not used, thus reducing static power consumption in idle mode. Besides the savings in dynamic and static power usage, one should also take into account the non-negligible costs of switching between power states [25]. DPM is widely used, for instance in modern processors by Intel and AMD. Global DVFS is employed in modern processors such as Intel Core i7 and NVIDIA Tegra 2 [15]. It has been shown [14, 2] that optimal energy savings require a combination of DPM and DVFS.

DVFS can be applied globally, or locally per processor [23]. Clearly, local DVFS provides more flexibility in choosing clock frequencies and voltage, so it is potentially more energy-efficient. However, it requires complex logic to implement many clock domains. To balance energy efficiency with design complexity, the concept of *Voltage and Frequency Islands* (VFIs) has been put forward [18]. One VFI consists of a clustered group of processors, running on a common clock frequency/voltage domain. Recently, some modern multicore processors, such as IBM Power 7 series, have adopted VFIs [16].

Programming streaming applications. We consider streaming applications for multi-processor mobile systems, like cell phones and PDAs. These applications consist of a series of encoding/decoding, signal processing and other computational tasks. We assume that the task graph has been modeled in Synchronous Data Flow (SDF [21]). The hardware consists of multiple processors, partly to increase the performance (streaming applications demand an ever higher throughput), partly since some tasks require specialised hardware capabilities.

Programming streaming applications on heterogeneous multi-processor hardware is difficult. Besides programming the basic functionality, the programmer must also design a mapping of the computation tasks to appropriate processors, and schedule them in such a way that all throughput constraints are met. With the advent of flexible energy management techniques, this becomes even more complicated: also the DVFS and DPM parameters must be adapted dynamically to save energy. Typically, a task should run at the lowest possible frequency, while still meeting its deadline.

Previous work. In previous work, we proposed to automate this mapping and scheduling process. First [1], we complemented the SDF application model with a separate hardware platform model of the heterogeneous multi-processors. This model specifies on which processors each task can run (processor capabilities), together with an upper bound on the running time. We also provided a translation of the SDF graph and processor models to Timed Automata. We used UPPAAL [6] to compute a mapping/schedule with maximal throughput on a limited set of processors. This provides a tradeoff between throughput and the used number of processors, potentially saving energy.

Subsequently [2], the hardware platform model was extended with the DPM, DVFS and VFI energy management techniques. In particular, it now also describes the VFI partitioning, the available frequency levels, power usage, switching costs, and task durations per processor/frequency. That paper provides a mapping of SDF graphs and the extended platform to *Priced Timed Automata* (PTA). We used prices to model the energy usage in various power modes and frequency levels, and to model the power costs of switching between those modes. We proposed Uppaal Cora [7] to synthesize safe energy schedules. In this way, we computed concrete mappings and schedules that always meet a given throughput constraint, while minimizing the energy consumption. The translation has been implemented as a model transformation between SDF metamodels and Uppaal metamodels [4], and was applied to an industrial Face Detection and Recognition application [26].

Recently [3], we extended our work in [2] to systems with batteries. Once the batteries are out of charge, the processors cannot run anymore. This signifies the end of *system life time*. In this work, we considered the concise *Kinetic Battery Model* (KiBaM) [22]. We modeled the system as a hybrid automaton [17], and applied statistical model checking to evaluate its Quality of Service in terms of, (1) the achievable application performance limited by a given battery capacity; and (2) the minimum required battery capacity to achieve a required application performance.

However, all these approaches are pessimistic, since they consider that the actors require worst-case execution time (WCET). On the contrary, practical systems have variability in execution time requirements. This variability can be modeled with stochastic systems, and analysed with statistical model checking, as in Uppaal Smc [10]. However, Uppaal Smc does not feature non-deterministic scheduling decisions. Hence, we model our system as *Stochastic Hybrid Games*, which distinguish controllable and non-controllable actions, which allows us to consider stochastic execution times and synthesize efficient strategies.

Contribution. In this paper, we derive energy-optimal solutions with a novel approach, based on stochastic hybrid games. These feature clock variables (used to model task duration and throughput constraints), continuous variables (used to model power consumption), controllable choices (used to model mapping and scheduling decisions), and uncontrollable choices (used for environment actions, e.g. the exact finishing time of a task). Remaining choices are implicitly resolved by uniform or exponential distributions. Stochastic hybrid games can be analysed by the tool Uppaal Stratego [12], which provides synthesis of safe and near-optimal strategies for stochastic hybrid games, using a combination of symbolic synthesis, statistical model checking, and reinforcement learning. Here, by near-optimal we mean strategies which are not optimal because they are computed using simulations (statistical model checking) instead of classical model checking. However, as these simulations are run multiple times, they are close to optimal strategies. Hence, we call them near-optimal.

The second purpose of this paper is to compare the approaches of Uppaal Cora and Uppaal Stratego. In particular, we are interested in the modeling effort for the transition. We also compared the computed results, since we only compute near-optimal strategies. We are interested in the potential performance gain provided by the use of statistical model checking. Finally, we wanted to reveal potential limitations. We try to answer these questions by repeating a case study, based on an MPEG-4 decoder, modeled in [27].

Paper organization. Section 2 recapitulates some important notions, in particular SDF graphs (2.1), our hardware platform model (2.2), and the analysis of stochastic hybrid games with UPPAAL STRATEGO (2.3). Section 3 offers a translation from an SDF graph plus hardware platform model to a stochastic hybrid game; we also describe the steps of our method to compute a safe and near-optimal scheduler using UPPAAL STRATEGO. Section 4 describes our experiment on an MPEG-4 decoder; we also compare the results with our previous approach using UPPAAL CORA. Finally, in Section 5 we discuss some related work and a research perspective.

2 Preliminaries

This section recapitulates *Synchronous Data Flow* (SDF) graphs to model task graphs of streaming applications, and *hardware platform models* including heterogeneous processors organized in VFIs, and featuring energy management techniques DVFS and DPM. We also recapitulate the analysis of *Stochastic Hybrid Games* with UPPAAL STRATEGO.

2.1 SDF Graphs

Typically, real-time streaming applications execute a set of periodic tasks, which consume and produce a fixed amount of data. Such applications are naturally modelled as SDF graph: directed graphs, in which nodes represent *actors* (tasks) and edges represent data buffers (communication streams). Individual data elements are represented by *tokens*, produced and consumed by actors, and stored in data buffers.

Definition 1. *An* SDF graph *is a tuple* $G = (A, D, \mathsf{Tok}_0)$ *where* A *is a finite set of actors,* $D \subseteq A^2 \times \mathbb{N}^2$ *is a finite set of dependency edges, and* $\mathsf{Tok}_0 : D \to \mathbb{N}$ *denotes the initial distribution of tokens per edge.*

Definition 2. *Given an SDF graph* $G = (A, D, \mathsf{Tok}_0)$*, the sets of* input and output edges *of an actor* $a \in A$ *are* $In(a) := \{(a', a, p, q) \in D \mid a' \in A, p, q \in \mathbb{N}\}$ *and* $Out(a) := \{(a, b, p, q) \in D \mid b \in A, p, q \in \mathbb{N}\}$*, respectively. The* consumption and production rate *of an edge* $e = (a, b, p, q) \in D$ *are defined as* $CR(e) := q$ *and* $PR(e) := p$*, respectively.*

The execution of an actor is known as an *actor firing*. Edges connect producers to consumers, and serve as token buffers. Actor *a can fire* if each input edge $(a', a, p, q) \in In(a)$ contains at least q tokens. If it fires, actor a removes q tokens from each input edge $(a', a, p, q) \in In(a)$ and produces p' tokens on each output edge $(a, b, p', q') \in Out(a)$.

Example 1. Figure 1 shows the SDF graph of an MPEG-4 decoder [27]. It contains five actors $A = \{FD, VLD, IDC, RC, MC\}$, representing tasks performed in MPEG-4 decoding. For example, the frame detector (FD) determines the number of macro blocks to decode. By decoding a single frame, FD produces 5 macroblocks (in reality this is an arbitary number between 0 and 99). The other modeled tasks are Variable Length Decoding (VLD), Inverse Discrete Cosine transformation (IDC), Motion Compensation (MC), and Reconstruction (RC) of the final video picture.

To avoid unbounded accumulation of tokens in a certain edge, we require SDF graphs to be *consistent*, i.e. an iteration can be defined that does not change the token distribution.

Definition 3. *A* repetition vector *of an SDF graph* $G = (A, D, \mathsf{Tok}_0)$ *is a function* $\gamma : A \rightarrow \mathbb{N}_{>0}$ *such that for every edge* $(a, b, p, q) \in D$*, the equation* $p.\gamma(a) = q.\gamma(b)$ *holds. An SDF graph is consistent if and only if it admits a repetition vector. In that case, an* iteration *of* G *is a multiset of actor firings, which contains exactly* $\gamma(a)$ *firings of each actor* $a \in A$*.*

2.2 Hardware Platform Model

The Hardware Platform Model (HPM) models the multi-processor platform on which the application (modelled as SDF graph) is mapped. Our HPM supports several features, including (1) heterogeneity: actors can only run on certain processors; (2) VFI: a partitioning of the processors in Voltage Frequency Islands; (3) DVFS: different frequency levels each processor can run on; (4) DPM: power consumption by a processor at a certain frequency, both when in use and when idle; (5) power-overhead required to switch between frequency levels; and (6) best- and worst-case computation times of tasks at a particular frequency level.

Definition 4. *A* Hardware Platform Model *(HPM) is a tuple* $\mathcal{P} = (\Pi, \zeta, F, P_{idle}, P_{occ}, P_{tr}, \tau_{worst}, \tau_{best})$*, 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);*

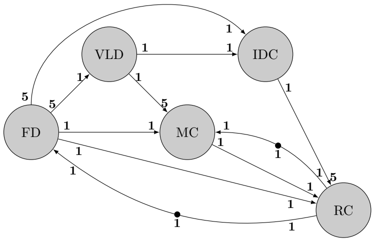


Fig. 1: MPEG-4 Decoder

- *a function* $\zeta : \Pi \to 2^A$ *indicating which processors can handle which actors;*
- *a finite set of discrete frequency levels available to all processors denoted by* $F = \{f_1, \ldots, f_m\}$ *such that* $f_1 < f_2 < \ldots < f_m$*;*
- *functions* $P_{idle}, P_{occ} : \Pi \times F \to \mathbb{N}$ *denoting the static power consumption (in idle state) and static plus dynamic power consumption (in operating state) of a processor* $\pi \in \Pi$ *at a certain frequency level* $f \in F$*,*
- *a partial function* $P_{tr} : \Pi \times F^2 \nrightarrow \mathbb{N}$ *expressing the transition overhead between frequency levels* $f_1, f_2 \in F$ *for each processor* $\pi \in \Pi$*, and*
- *functions* $\tau_{best}, \tau_{worst} : A \times F \to \mathbb{N}_{\geq 1}$ *defining the best- and worst-case execution times for each actor* $a \in A$ *operating at frequency level* $f \in F$*.*

Note that it is straightforward to refine this model further, by explicitly introducing processor types, distinguishing frequency levels per processor type, and let execution times depend on processor types. By incorporating memory elements and buses, it would also be possible to model communication costs. Currently, we assume that communication times are included in the actor execution times.

Example 2. Exynos 4210 is a state-of-the-art processor used in high-end platforms such as Samsung Galaxy Note, SII etc. Table 1 shows its different DVFS levels, and corresponding CPU voltage (V) and clock frequency (MHz) [25].

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

Table 1: DVFS levels of Samsung Exynos 4210

2.3 Stochastic Hybrid Games and UPPAAL STRATEGO

We review Stochastic Hybrid Games and their analysis in UPPAAL STRATEGO. *Timed Automata* [5] have locations, transitions and clock variables. Residence time in states is constrained by invariants on clocks. Transitions are guarded by clock constraints as well and can reset a subset of the clock variables. For convenience, UPPAAL adds discrete variables (that can be used in guards, invariants and updates) and allows networks of timed automata that synchronize by means of handshake or broadcast channels. *Hybrid Automata* extend Timed Automata with continuous variables, governed by differential equations. They generalize *Priced Timed Automata*, where prices are hybrid variables that cannot be used in guards. In *Stochastic Automata*, choices and time delays are governed by stochastic distributions, like uniform and exponential distributions. *Timed Games* distinguish controllable actions (like scheduling choices by the system) and uncontrollable actions (like inputs or time delays that are determined by the environment). Finally, *Stochastic Hybrid Games* combine all features.

UPPAAL STRATEGO [12] supports strategy synthesis for Stochastic Hybrid Games. It integrates the symbolic algorithms for model checking Priced Timed Automata (from UPPAAL CORA [7]) and for synthesizing optimal strategies of

Timed Games (from Uppaal Tiga [9]) with the statistical model checking algorithms for Stochastic Timed Automata (Uppaal SMC [13]). Moreover, it implements reinforcement learning to synthesize near-optimal strategies for Stochastic Hybrid Games [11]. Uppaal Stratego comes with an extended query language, where strategies are first class objects that may be synthesized, compared, further optimized or restricted, and analyzed for correctness and performance. New symbolic or statistical model checking and synthesis queries on Stochastic Timed Games can be performed under the constraints of previously synthesized stragies.

We illustrate the features and queries of Uppaal Stratego with a small example adapted from [29]. Figure 2 models a job with two phases. In the first phase, the scheduler must choose between two machines, indicated by the locations A and B. The slow machine A takes up to 100 time-units to finish (indicated by the invariant on clock variable x), but consumes less power (indicated by the differential equation $c' == 3$ (e.g. 3 kW/h). The alternative machine B is twice as fast, but consumes considerably more power. In the second phase, the scheduler must choose between the machines C and D. The choice of machine is in both phases controllable by the scheduler (indicated by the solid transitions), while the exact completion time within the specified upperbound is left to an uncontrollable environment (indicated by the dashed transitions). Implicitly, a uniform distribution of the actual computation time is assumed. Next, we are interested in synthesizing controllers for various objectives. We consider the following scenarios.

Scenario 1: Safe Strategy. The job in Figure 2 must be completed before 175 time-units. To this end, we generate the most permissive (non-deterministic) strategy `Safe`, and compute its expected cost (based on 1000 simulation runs), using the following two queries. The expected cost (when the choice for the exact finishing time is resolved uniformly) appears to be 437.317.

strategy `Safe` = **control** : `A <> Job.End and time <= 175` .

E[<= `175; 1000`](**max** : `c`) **under** `Safe` .

Scenario 2: Optimal Strategy. Now we are interested in completing the job with minimal energy consumption. We compute a near-optimal strategy and

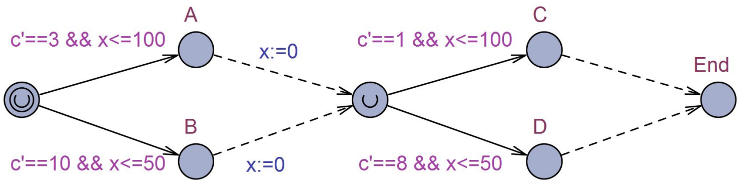


Fig. 2: A job with two phases

visualize it with 10 random simulation runs with the following two queries.

$$\textbf{strategy}\ \texttt{Opt} = \textbf{minE}(\texttt{c})\ [<= \texttt{175}]\ : <>\ \texttt{Job.End}\ .$$

$$\textbf{simulate}\ 10\ [<= \texttt{200}]\ \{\texttt{Job.A}, 2 + \texttt{Job.B}, 4 + \texttt{Job.C}, 6 + \texttt{Job.D}, 8 + \texttt{Job.End}\}\ \textbf{under}\ \texttt{Opt}\ .$$

UPPAAL STRATEGO employs random simulation and reinforcement learning to select a strategy that minimizes the expected completion cost, which is estimated to be 276.661. The 10 random simulation runs are shown in Figure 3(a). Clearly, in these runs only the cheaper machines (A and C) are chosen. However, strategy `Opt` does not always finish within 175 time-units.

Scenario 3: Optimal and Safe Strategy. To find a strategy that both finishes within 175 time units and minimizes the energy consumption, we query:

$$\textbf{strategy}\ \texttt{OptSafe} = \textbf{minE}(\texttt{c})[<= \texttt{200}] : <>\ \texttt{Job.End}\ \textbf{under}\ \texttt{Safe}\ .$$

This learns a new sub-strategy `OptSafe` under the constraints of the strategy `Safe` derived in Scenario 1. Figure 3(b) shows 10 random runs according to `OptSafe`. One sees that in this case sometimes machine D is used: If machine A finishes its job early, the slower but cheaper machine C can be utilized in the second phase. However, if the machine A takes longer, only the faster, more expensive machine D can be selected, or we would miss the deadline. The expected completion cost of `OptSafe` appears to be 316.738, which is higher than `Opt`, but better than `Safe`. Note that, opposed to `Opt`, the strategy `OptSafe` will always finishes within the deadline of 175 time-units, thus being guaranteed safe and near-optimal.

3 Energy-optimal schedules under throughput constraints

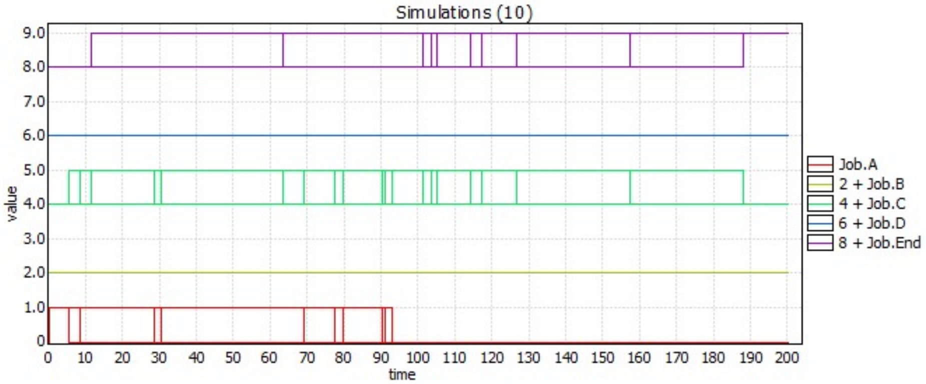
3.1 Translating SDF Graphs to Stochastic Hybrid Games

In our framework [1, 2], the input consists of separate models of an SDF task graph, the hardware platform model, and a throughput constraint. In this way, we split the problem statement of optimal energy management in terms of tasks and resources. In this section, we describe a systematic translation of an SDF graph along with a hardware platform model into a Stochastic Hybrid Game. Subsequently, we summarize our method of using UPPAAL STRATEGO for computing an energy-optimal strategy under the given throughput constraint.

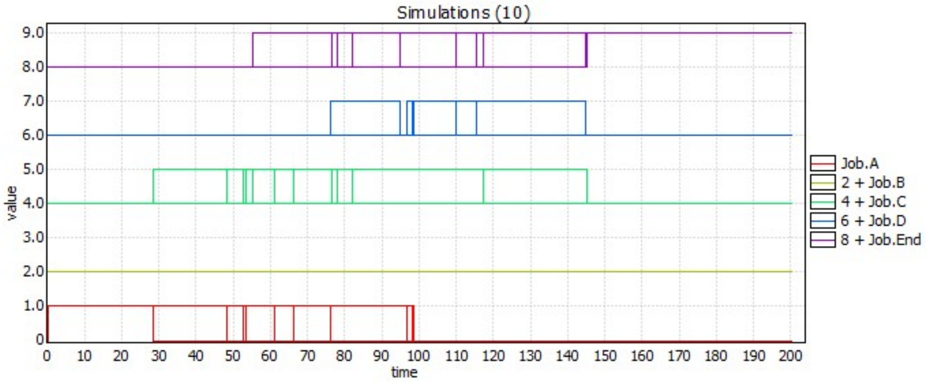
Given an SDF graph $G = (A, D, \mathsf{Tok}_0)$ mapped on a hardware platform model $(\Pi, \zeta, F, P_{idle}, P_{occ}, P_{tr}, \tau_{worst}, \tau_{best})$, we generate a parallel composition of stochastic hybrid games:

$$A_G \| \mathit{Processor}_1 \|, \ldots, \| \mathit{Processor}_n \| \mathit{Scheduler}\ .$$

Here A_G encodes the SDF task graph G, keeping track of the number of tokens in all buffers. $\mathit{Processor}_i$ models $\pi_i \in \Pi$, keeping track if it is idle or occupied,



(a) Runs under `Opt`



(b) Runs under `OptSafe`

Fig. 3: UPPAAL STRATEGO simulations for `Opt` and `OptSafe` Controllers

and of the current frequency level for DVFS. The task of the *Scheduler* is to synchronize frequency switches between all processors in the same voltage and frequency island (VFI).

Translating the SDF graph. The automaton A_G has a single location and no clocks. Figure 4 illustrates A_G for the MPEG decoder in Figure 1. For each edge (a, b, p, q), it contains an integer variable buff_a2b containing the number of buffered tokens in this edge. It also counts the number of times a has fired in a variable counter_a for later querying. Initially, counter_a $= 0$ and buff_a2b $= Tok_0(\mathsf{a}, \mathsf{b}, \mathsf{p}, \mathsf{q})$. Moreover, for each processor $\pi_i \in \Pi$, the automaton A_G reads a boolean variable Pbusy[i]. This variable ensures that the actors in the SDF graph G are only mapped on the processor $\pi_i \in \Pi$ when it is free. This extra administration is needed because UPPAAL STRATEGO only supports broadcast communication, but not blocking handshake communication. Furthermore, UPPAAL STRATEGO requires that each location must either be urgent, committed, have an exponential rate or carry an invariant. Therefore, we have 5 as a *Rate of Exponential* in location Initial.

There are two parametrized actions in A_G namely, fire![i][a] and end?[i][a] to communicate between A_G and $Processor_i$, representing the start and end of executing actor a on processor π_i. For each $a \in A$, we add two edges in A_G. The controllable transition labeled fire![i][a] consumes q tokens from buffer buff_b2a, for each input edge (b, a, p, q) in G (specified by an auxiliary function consume(buff_b2a, q) not detailed here). The uncontrollable transition labeled end?[i][a] produces p tokens into buffer buff_a2b, for each output edge (a, b, p, q) in G (specified by auxiliary function produce(buff_a2b, p)).

Translating the hardware platform model. For each $\pi_i \in \Pi_y \subseteq \Pi$, we introduce an automaton $Processor_i$. See Figure 5 for a part of this automaton. It maintains two variables, to properly synchronize with the *Scheduler* (only needed since UPPAAL STRATEGO doesn't support handshake synchronization). Variable freq_lev[y] counts the number of currently occupied processors in the VFI Π_y, and curr_freq[y] determines the current frequency level of all $\pi_i \in \Pi_y$. Initially, freq_lev[y] $= 0$ and curr_freq[y] $= m$, where $f_m = \max\{F\}$. The counter freq_lev[y] is incremented and decremented by one on each fire?[i][a] and end![i][a] transition.

$Processor_i$ has location Idle_f for each $f \in F$, and for each $a \in A$ an additional location InUse_a_f. The edges between these states both synchronize with A_G. The worst-case execution time $\tau_{worst}(a, f)$ is encoded in the invariant of InUse_a_f. The best-case execution time $\tau_{best}(a, f)$ is encoded as the guard. In this paper, we assume that the execution time of the actor is determined stochastically by the environment, uniformly in the interval $[\tau_{best}(a, f), \tau_{worst}(a, f)]$. The power consumption P_{idle} at each frequency $f \in F$ and P_{occ} for each actor $a \in A$ are encoded as a differential equation in the invariant in Idle_f and InUse_a_f, respectively, using the hybrid variable $cost_i'$. As required by UPPAAL STRATEGO, we have 5 as a *Rate of Exponential* in all locations Idle_f.

Encoding frequency switches. See Figure 6 for the *Scheduler*. It triggers fre-

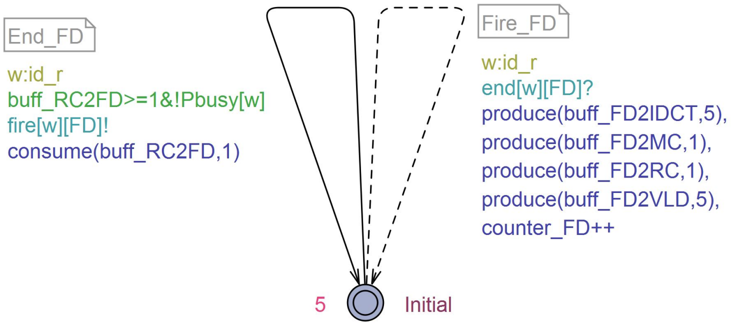


Fig. 4: UPPAAL STRATEGO model A_G for SDF graph G

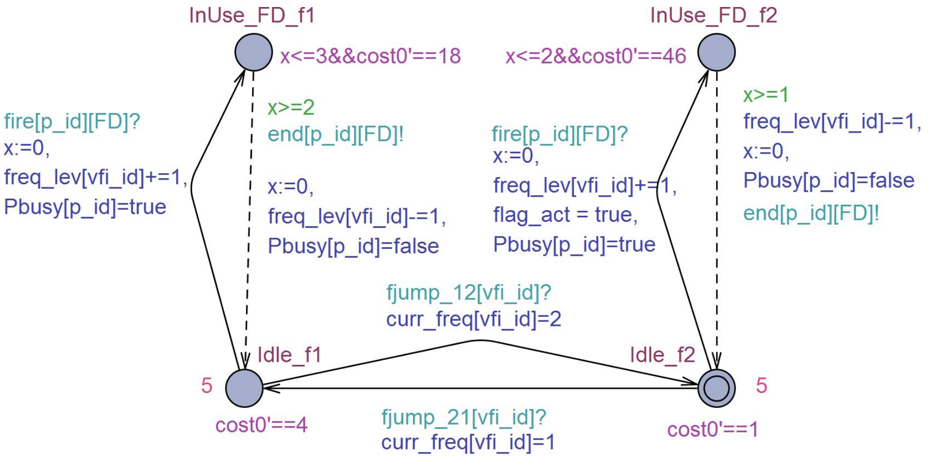


Fig. 5: Uppaal Stratego model *Processor* for process p_id in VFI vfi_id (restricted to frequencies f_1, f_2 and transitions for actor FD)

quency switches from f_ℓ to f_k (for $k = \ell \pm 1$). It synchronizes with $Processor_i$ for all $\pi_i \in \Pi_y$ that are in the same VFI y, through the parameterized broadcast action fjump_lk[y]. The automaton *Scheduler* checks whether all processors in the switched VFI y are in the idle location (this excludes switching frequencies in the middle of a task execution), and are running at the same frequency. This is done by testing the global counters freq_lev[y]=0 and curr_freq[y]=ℓ.

To avoid traces with infinite switching between Idle locations (Zeno behavior), we needed some extra restrictions. We defined that each processor starts in the highest frequency (by setting the initial location in *Processor*). We further ensure that a processor can only switch frequencies after firing an actor (at the highest frequency, using flag_act). Furthermore, to switch frequencies, the scheduler must wait for 2 time units (denoted by $x >= 2$ where x is a local clock). We also have 5 a *Rate of Exponential* in location Initial .

3.2 Learning and Optimization using Uppaal Stratego

Recall the repetition vector γ, which captures the number of actor firings in a single iteration. Usually, one is interested in a periodic cycle, which consists of a number of iterations. In this paper, we restrict to the execution of one iteration. So for each actor $a \in A$, $\gamma(a)$ denotes the number of times a must fire. For example, the repetition vector γ of the example SDF graph given in Section 3 is $\gamma(\langle \mathrm{FD}, \mathrm{VLD}, \mathrm{IDC}, \mathrm{RC}, \mathrm{MC}\rangle) = \langle 1, 5, 5, 1, 1\rangle$. We capture the states after firing according to the repetition vector γ by predicate Q:

$$Q := \bigwedge_{a \in A} \{\mathsf{counter\_a} = \gamma(a)\}$$

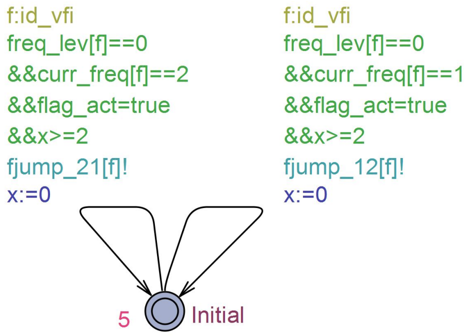


Fig. 6: Uppaal Stratego model for the Scheduler

We now summarize our method to synthesize an energy-optimal controller from an SDF graph G and a HPM $\mathcal{P}$ with n processors that meets the throughput-constraint T using Uppaal Stratego, by the following steps:

1. Generate stochastic hybrid games from G and $\mathcal{P}$ according to the translation in Section 3.1, resulting in $A_G \| \mathit{Processor}_1 \| \ldots \| \mathit{Processor}_n \| \mathit{Scheduler}$. This network of timed automata forms the input to Uppaal Stratego.
2. Synthesize the most permissive safe strategy that finishes one iteration within time T, by running the following query, where clock variable *time* is never reset, but just used to observe the overall time progress.

$$\textbf{strategy}\ \texttt{Safe} = \textbf{control} : \texttt{A} <> (\texttt{Q and time} <= \texttt{T})$$

3. Obtain a near-optimal strategy with respect to energy consumption that finishes within time T, by running the following query:

$$\textbf{strategy}\ \texttt{OptSafe} = \textbf{minE} : (\sum_{\pi_\texttt{i} \in \Pi} \texttt{cost}_\texttt{i})\ [<= \texttt{T}] :<> (\texttt{Q})\ \textbf{under}\ \texttt{Safe}$$

4. To get an impression of the minimal energy needed when the throughput constraint would be ignored, one may optionally run the following query:

$$\textbf{strategy}\ \texttt{Opt} = \textbf{minE} : (\sum_{\pi_\texttt{i} \in \Pi} \texttt{cost}_\texttt{i})\ [<= \texttt{T}] :<> (\texttt{Q})$$

5. For all strategies $S \in \{\mathit{Safe}, \mathit{OptSafe}, \mathit{Opt}\}$ one can compute the average energy consumption from a number of simulation runs (say 100) by the following query:

$$\textbf{E}[<= \texttt{T}; 100]\ (\textbf{max} : \sum_{\pi_\texttt{i} \in \Pi} \texttt{cost}_\texttt{i})\ \textbf{under}\ \texttt{S}$$

A number of 10 simulations for executing actor $a, b, ... \in A$ under the strategy S can be visualized with the query

$$\textbf{simulate 10}\ \{\texttt{counter\_a}, 2 + \texttt{counter\_b}, ...\}\ \textbf{under}\ \texttt{S}$$

Voltage(V)	Frequency(MHz)	$\mathbf{P_{idle}}$(W)	$\mathbf{P_{occ}}$(W)
1.2	1400	0.1	4.6
1.00	1032.7	0.4	1.8

Table 2: Platform description

4 Experimental Evaluation via MPEG-4 Decoder

4.1 Modeling the MPEG-4 Decoder

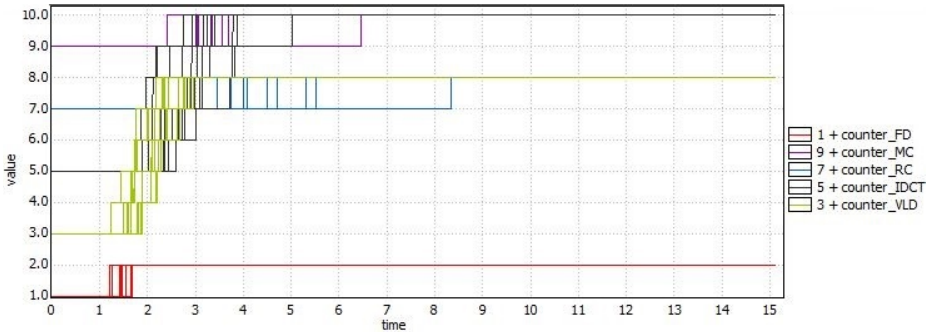
Let us consider the example of an MPEG-4 decoder capable of processing five macroblocks as shown in Figure 1, mapped on Exynos 4210 processors $\Pi = \{\pi_1, \ldots, \pi_n\}$. Table 2 shows two DVFS levels (MHz) $\{f_1, f_2\} \in F$ taken from Table 1 and corresponding experimental power consumption. We assume that the (best/worst) execution times of all actors $a \in A$ at lower frequency level, i.e., f_1 are rounded to the next integer. As $f_1 = 0.738 \times f_2$, $\tau_{best}(a, f_1) = \lceil \frac{\tau_{best}(a, f_2)}{0.738} \rceil$ and $\tau_{worst}(a, f_1) = \lceil \frac{\tau_{worst}(a, f_2)}{0.738} \rceil$.

The UPPAAL STRATEGO models of this example are shown in Figures 4, 5 and 6. For easier understanding, the models are shown with respect to one actor only, i.e., FD $\in A$. Figure 4 shows the automaton A_G which models the actor FD, and its incoming $\mathit{In}(FD)$ and outgoing $\mathit{Out}(FD)$ edges. The automata $\mathit{Processor}_1, \ldots, \mathit{Processor}_n$ model the processors $\Pi = \{\pi_1, \ldots, \pi_n\}$, as shown in Figure 5. Figure 6 presents the automaton of *Scheduler*. As clocks in UPPAAL STRATEGO can only take integer values, all power consumption values are multiplied by 10 in Figure 5.

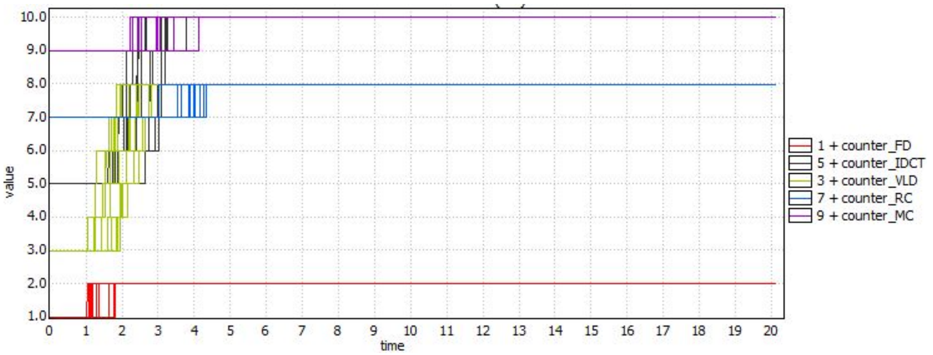
Initially, all processors $\mathit{Processor}_1, \ldots, \mathit{Processor}_n$ are in the idle location at the highest frequency level, Idle_f2. The idle power consumption $P_{idle}(\pi, f_2) = 0.4$ W is annotated as an invariant in the location Idle_f2.

Let us consider that the MPEG-4 Decoder in Figure 1 is mapped on 4 Exynos 4210 Processors. For the constraint of finishing an iteration within 15 ms (67 frames per second (fps)), Figure 7 shows 10 random runs for each controller, i.e., `Safe`, `Opt`, and `OptSafe` respectively. Figure 7a shows the strategies that achieve 67 fps without optimising energy consumption. Whereas, Figure 7b shows the strategies having minimal energy consumption without any constraint on the throughput. In particular, Figure 7c shows the strategies that after learning the strategy `Safe`, guarantee to be both energy-optimal and achieve 67 fps.

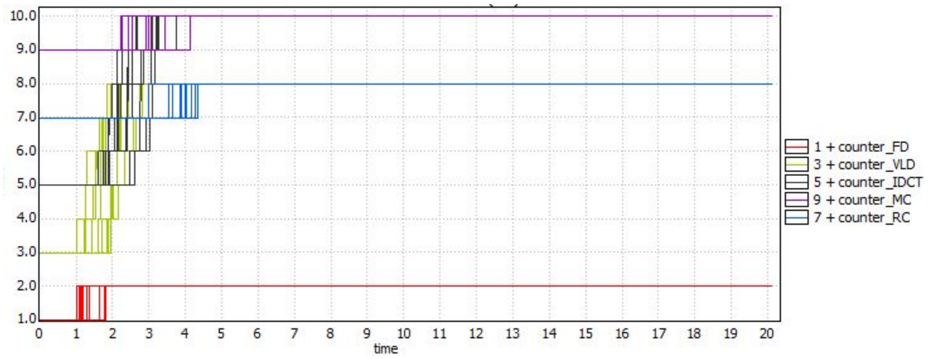
We also generated various strategies for the MPEG-4 decoder on a varying number of processors. For the constraint of 67 fps, the evaluation of the energy consumption under different controllers explained in subsection 3.2 are summarised in Table 3. If we analyse the `OptSafe` strategy in Table 3, we observe that achieving the same number of frames per second at fewer processors lowers the energy consumption. The reason is the high slack at the higher number of processors, and therefore the processors stay idle for most of the time. As a



(a) Runs under `Safe`



(b) Runs under `Opt`



(c) Runs under `OptSafe`

Fig. 7: Uppaal Stratego simulations for MPEG-4 decoder under `Safe`, `Opt`, and `OptSafe` Controllers

result, the static energy surpasses the dynamic energy. For instance, the energy consumption is decreased by 5.4%, when moving from 5 to 4 processors. Since the strategy `Opt` is not guaranteed to finish within the deadline, we will not consider it in the rest of the paper.

Processors	Safe	Opt	OptSafe
5	42.76	41.64	42.21
4	40.87	39.79	39.94
3	39.61	38.39	38.57
2	38.57	37.08	37.31
1	36.96	35.21	35.47

Table 3: Energy estimations with adaptive execution times

Processors	Safe	OptSafe	Optimal
5	67.26	59.7	55.4
4	66.9	58.01	53.8
3	63.83	56.2	53.4
2	62.41	54.5	54.5
1	64.07	63.4	63.4

Table 4: Comparison of energy estimations with worst-case execution times

4.2 Comparison with UPPAAL CORA

In this subsection, we compare the approach presented in this paper (stochastic hybrid games) with the priced timed automata based approach in [2]. The work in [2] like us, computes the energy-optimal schedules for SDF applications running on multiprocessor platforms. However, in comparison to our approach of using stochastic hybrid games, the problem of finding the schedules is encoded as a reachability property over priced timed-automata models. This is then checked by the model checker UPPAAL CORA [7]. For comparison, we take the example of the MPEG-4 decoder in Figure 1. We assume that the SDF graph is mapped on Exynos 4210 processors.

First, we removed the stochastic features in our stochastic hybrid models by considering worst-case execution times of the actors only. For the constraint of 67 fps, columns 2-3 in Table 4 show the energy consumption (mWs), calculated using the approach of this paper, against the varying number of processors given in column 1. The results are given for the strategies `Safe` and `OptSafe`. For the same throughout constraint, column 4 shows the energy consumption calculated using the approach presented in [2]. Note that the optimal results from CORA are slightly better than the near-optimal results from STRATEGO.

As said earlier, the biggest strength of UPPAAL STRATEGO is the ability to handle uncertainty in the environment, and to compute an adaptive strategy. We utilized this feature by having best- and worst-case execution time in our

Processors	**Adaptive Time**		**Worst-Case Time**		
	Safe	**OptSafe**	**Safe**	**OptSafe**	**Optimal**
5	42.76	42.21	67.26	59.7	55.4
4	40.87	39.94	66.9	58.01	53.8
3	39.61	38.57	63.83	56.2	53.4
2	38.57	37.31	62.41	54.5	54.5
1	39.69	35.47	64.07	63.4	63.4

Table 5: Comparing adaptive with worst-case execution times

Processors	**Safe**		**OptSafe**		**Optimal**	
	Memory	Time	Memory	Time	Memory	Time
5	2622.06	50354.44	2660.15	75831.63	71.13	144.52
4	634.14	5911.04	695.42	8487.192	29.46	7.27
3	165.86	327.85	224.75	505.09	21.46	0.71
2	86.6	5.32	123.21	18.38	19.75	0.31
1	79.28	0.02	115.55	1.96	19.62	0.09

Table 6: Comparing time and memory consumption of the Stratego and Cora-tools

Uppaal Stratego model. On the other hand, Uppaal Cora considers worst-case execution times only. Table 5 compares this strength of Uppaal Stratego with Uppaal Cora, by combining Table 3 and 4. In Table 5, columns 2-3 shows the energy consumption, when having best- and worst-case (adaptive) execution times. Columns 3-5 copy the results from Table 4, considering worst-case execution times only. Clearly, the adaptive strategy saves energy.

Table 6 compares the computation time (sec) and the memory consumption (MB), of both methods, when having worst-case execution times. As we can see, for our running example, Uppaal Cora is more efficient in terms of memory and time, in particular when analysing systems with more cores.

5 Conclusion

5.1 Discussion

We set out an alternative method to map and schedule streaming applications specified in SDF onto heterogeneous multi-processor hardware. We conducted an experiment in Uppaal Stratego, and compared it with a previous approach in Upaal Cora. We will discuss some limitations and strengths of the approach with Uppaal Stratego.

The most important issue is that Uppaal Cora provides a concrete schedule (in the form of a timed trace), from which a concrete implementation can be derived. This is clearly not possible in Uppaal Stratego, since it deals with stochastic systems. However, apart from the visualization, we have found no way to retrieve the computed strategy in a form that can be used to synthesize real controller code. In [1, 2] we used the traces also to find out the number of iterations needed to get into the periodic phase of the schedule. This would be needed to really compute the long-term throughput of streaming applications like an MPEG-4 decoder.

Next, we found some limitations in the input language of Uppaal Stratego. Some of them are inherent due to the use of statistical model checking engine. In particular, the absence of handshake synchronization led us to add more and more global variables and guards, in order to keep the components synchronized despite using broadcast communication only. Another issue is that we could not add discrete jumps in costs on transitions. This means that we couldn't model the costs of switching between frequencies (as specified by P_{tr}), as we do in our original Cora models. Another consequence is that we had to modify the model to avoid Zeno-runs between infinitely switching frequencies in idle mode, which now happen "for free".

We also compared the results of Cora's optimal schedule, and Stratego's near-optimal strategies. The good news is that the estimated near-optimal results by Stratego are quite close to those computed by Cora (at most 10% deviation). We also had hoped that Stratego would be faster, due to the use of statistical model checking and learning, but this was not the case. Actually, the computations in Stratego took considerably longer time than in Cora.

We want to stress that the several drawbacks that we encountered are inherent due to the stronger capabilities of Stratego. Using Stratego, we can handle uncertainty in the environment, and compute an adaptive strategy based on the actual behavior so far. We already exploited this by distinguishing the best- and worst-case execution time of the actors in the MPEG-4 decoder. Stratego assumes that the actual time is distributed uniformly in this interval.

This feature shows the distinguishing power of Stratego and it provides an enormous potential for energy savings compared to the traditional approach. The traditional approach will always take into account the worst-case execution time, even when this is not realistic at all. This could result in an over-dimensioning of the system, leading to idle-time intervals. This is energy-inefficient, because those intervals could have been used to lower the clock frequency in the busy intervals. A truly adaptive strategy would thus make more efficient use of the same resources.

5.2 Related Work

The state-of-the-art method of applying DVFS on SDF graphs is proposed in [24, 31]. These papers consider dynamic power usage, but they ignore static power usage, which is non-negligible in modern processors. The work in [24] requires the expensive transformation of SDF graphs to equivalent Homogeneous SDF (HSDF) graphs, which we avoid in our approach. Also, their work is not applicable to platforms with a limited number of processors. The approach in [31] considers that each task is executed as soon as it is enabled, unlike real-life applications where this is not possible due to limitations on the available number of processors. A recent stochastic approach [20] introduces exponential Scenario-Aware SDF models, which add mode switches and exponential delays to SDF graphs. They compute minimal and maximal expected values, but cannot derive concrete schedules. Another direction [29] takes into account strategies for battery schedules, in order to minimize battery wear and optimize for a system's life time. In [3], we integrated task scheduling on heterogeneous hardware with battery strategies, as an approach to energy-self-supporting systems. Finally, an interesting alternative is to use mean-payoff games to synthesize a controller for resource- and scenario-aware SDF graphs, which reacts to environment behavior [30]. They use policy iteration algorithms to optimize for throughput, in contrast to our approach, where the goal is to optimize for energy consumption.

5.3 Research Perspectives

In this paper, we have presented a method of synthesizing energy-optimal controllers for dataflow applications mapped on heterogeneous platforms using Uppaal Stratego. We further have compared this approach with an approach with Uppaal Cora, in terms of modeling effort, results and computation times. For deterministic systems, currently Uppaal Cora provides a more expressive input language, faster algorithms, and ready to use results in the form of a concrete schedule, when compared to Uppaal Stratego. However, Uppaal Cora cannot handle environment uncertainty, which leads to assuming worst-case behavior everywhere, and potentially energy-wasteful schedulers. Uppaal Stratego provides the means to compute adaptive schedules, optimizing for energy in an uncertain environment.

A future research direction is to carry on from the results achieved in this paper and explore the possibilities of battery-aware scheduling of SDF graphs. As real-life batteries are hybrid in nature, Uppaal Stratego is a natural choice to model them. Then using the combination of symbolic synthesis, statistical model checking, and reinforcement learning, we can synthesize battery-aware controllers.

Acknowledgement

The authors are grateful to Jakob Haahr Taankvist and Peter Gjøl Jensen for their valuable help to understand Uppaal Stratego. The authors would also like to thank the anonymous reviewers for their helpful and constructive comments that greatly contributed to improving the final version of the paper.

References

1. W. Ahmad, R. de Groote, P. K. F. Hölzenspies, M. Stoelinga, and J. van de Pol. Resource-constrained optimal scheduling of synchronous dataflow graphs via timed automata. In *Proceedings of 14th International Conference on Application of Concurrency to System Design, (ACSD)*, pages 72–81, 2014.
2. W. Ahmad, P. K. F. Hölzenspies, M. Stoelinga, and J. van de Pol. Green computing: Power optimisation of VFI-based real-time multiprocessor dataflow applications. In *Proceedings of 18th Euromicro Conference on Digital Systems Design (DSD)*, pages 271–275, 2015.
3. W. Ahmad, M. Jongerden, M. Stoelinga, and J. van de Pol. Model checking and evaluating QoS of batteries in MPSoC dataflow applications via hybrid automata. In *Proceedings of 16th International Conference on Application of Concurrency to System Design (ACSD)*, pages 114–123, 2016.
4. W. Ahmad, B. M. Yildiz, A. Rensink, and M. Stoelinga. *Evaluating the Tools on the Face Detection and Recognition Case Study*, chapter 2, pages 4–6. FP7 EU Project SENSATION, Deliverable D1.4, 2016.
5. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
6. G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems*, volume 3185. 2004.
7. G. Behrmann, K. G. Larsen, and J. I. Rasmussen. Optimal scheduling using priced timed automata. *SIGMETRICS Performance Evaluation Review*, 32(4):34–40, 2005.
8. L. Benini, A. Bogliolo, and G. D. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):299–316, 2000.
9. F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR 2005 – Concurrency Theory*, volume 3653. 2005.
10. A. David, D. Du, K. G. Larsen, A. Legay, M. Mikučionis, D. Bøgsted Poulsen, and S. Sedwards. Statistical Model Checking for Stochastic Hybrid Systems. *ArXiv e-prints*, 2012.
11. A. David, P. G. Jensen, K. G. Larsen, A. Legay, D. Lime, M. G. Sørensen, and J. H. Taankvist. On time with minimal expected cost! In *Automated Technology for Verification and Analysis*, volume 8837, pages 129–145. 2014.
12. A. David, P. G. Jensen, K. G. Larsen, M. Mikučionis, and J. H. Taankvist. Uppaal Stratego. In *Proceedings of 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 206–211. 2015.
13. A. David, K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen. Uppaal SMC tutorial. *International Journal on Software Tools for Technology Transfer*, 17(4), 2015.
14. V. Devadas and H. Aydin. On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications. *IEEE Transactions on Computers*, 61(1):31–44, 2012.
15. M. E. T. Gerards, J. L. Hurink, and J. Kuper. On the interplay between global DVFS and scheduling tasks with precedence constraints. *IEEE Transactions on Computers*, 64:1742–1754, 2014.
16. 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 Transactions on Computers*, 61(12):1682–1696, 2012.

17. T. Henzinger. The theory of hybrid automata. In *Proceedings of Eleventh Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 278–292, 1996.
18. S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proceedings of ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pages 38–43, 2007.
19. S. Irani and K. R. Pruhs. Algorithmic problems in power management. *ACM SIGACT News*, 36(2):63–76, 2005.
20. J.-P. Katoen and H. Wu. Exponentially timed SADF: compositional semantics, reductions, and analysis. In *Proceedings of 14th ACM/IEEE International Conference on Embedded Software (EMSOFT)*, pages 1:1–1:10, 2014.
21. E. A. Lee and D. G. Messerschmitt. Synchronous data flow. In *Proceedings of the IEEE*, volume 75(9), pages 1235–1245, 1987.
22. J. F. Manwell and J. G. McGowan. Lead acid battery storage model for hybrid energy systems. *Solar Energy*, 50(5):399 – 405, 1993.
23. 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. *The Computer Journal*, 54(8):1282–1294, 2011.
24. A. Nelson, O. Moreira, A. Molnos, S. Stuijk, B. T. Nguyen, and K. Goossens. Power minimisation for real-time dataflow applications. In *Proceedings of 14th Euromicro Conference on Digital System Design (DSD)*, pages 117–124, 2011.
25. 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. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(5):695–708, 2013.
26. T. D. ter Braak, K. Sunesen, W. Ahmad, M. Stoelinga, J. van de Pol, J.-P. Katoen, and H. Wu. *Evaluating the Tools on the Face Detection and Recognition Case Study*, chapter 2, pages 6–23. FP7 EU Project SENSATION, Deliverable D4.4, 2016.
27. B. Theelen, M. Geilen, T. Basten, J. Voeten, S. Gheorghita, and S. Stuijk. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *Proceedings of 4th ACM/IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE)*, pages 185–194, 2006.
28. M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proceedings of 1st USENIX Conference on Operating Systems Design and Implementation*, 1994.
29. E. R. Wognsen, B. R. Haverkort, M. Jongerden, R. R. Hansen, and K. G. Larsen. A score function for optimizing the cycle-life of battery-powered embedded systems. In *Proceedings of 13th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, pages 305–320, 2015.
30. Y. Yang, M. Geilen, T. Basten, S. Stuijk, and H. Corporaal. Playing games with scenario- and resource-aware SDF graphs through policy iteration. In *Proceedings of Design, Automation and Test in Europe (DATE)*, pages 194–199, 2012.
31. J. Zhu, I. Sander, and A. Jantsch. Energy efficient streaming applications with guaranteed throughput on MPSoCs. In *Proceedings of 8th ACM/IEEE International Conference on Embedded software (EMSOFT)*, pages 119–128, 2008.
32. S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto. Survey of energy-cognizant scheduling techniques. *IEEE Transactions on Parallel and Distributed Systems*, 24(7):1447–1464, 2013.