Throughput-Constrained DVFS for Scenario-Aware Dataflow Graphs

Morteza Damavandpeyma[1], Sander Stuijk[1], Twan Basten[1,2], Marc Geilen[1] and Henk Corporaal[1]
[1]Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands
[2]Embedded Systems Institute, Eindhoven, The Netherlands
{m.damavandpeyma, s.stuijk, a.a.basten, m.c.w.geilen, h.corporaal}@tue.nl

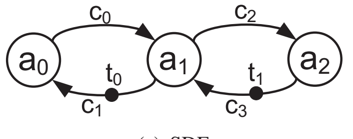
***Abstract*—Dynamic behavior of streaming applications can be effectively modeled by scenario-aware dataflow graphs (SADFs). Many streaming applications must provide timing guarantees (e.g., throughput) to assure their quality-of-service. For instance, a video decoder which is running on a mobile device is expected to deliver a video stream with a specific frame rate. Moreover, the energy consumption of such applications on handheld devices should be as low as possible. This paper proposes a technique to select a suitable multiprocessor DVFS point for each mode (scenario) of a dynamic application described by an SADF. The technique assures strict timing guarantees while minimizing energy consumption. The technique is evaluated by applying it to several streaming applications. It solves the problem faster than the state of the art technique for dataflow graphs. Moreover, the DVFS controller devised using the proposed technique is more compact and reduces energy consumption compared to the controller devised using the counterpart technique.**

I. Introduction

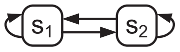
Scenario-aware dataflow graphs (SADFs) [1] are introduced to model dynamic behavior of streaming applications. These applications are applied iteratively on indefinite input streams. The dataflow graphs capture such an iterative behavior; the execution of the model for a single input instance is called an *iteration*. For example, in a video decoder which processes a video stream, the operations involved in processing one video frame specify one iteration. An SADF is composed of several application scenarios (modes) and a finite state machine (FSM). The behavior of the application in each scenario is modeled through a synchronous dataflow graph (SDF) [2]. Fig. 1 depicts an example SADF with two scenarios s_1 and s_2 which use the same SDF (see Fig. 1(a)), but with different execution times for the actors (tasks) in each of the scenarios (see Fig. 1(c)). The FSM of the SADF specifies the possible scenario occurrence orders (e.g., Fig. 1(b)).

Streaming applications, such as signal processing and multimedia applications, should meet strict timing requirements. Furthermore, energy consumption is an important design criterion for such applications. Dynamic voltage and frequency scaling (DVFS) [3] is used to develop low power/energy implementations. This paper presents a technique to determine for each application scenario an energy-aware frequency setting while satisfying a throughput constraint. It is assumed that the application (SADF) is already mapped and scheduled to a platform with multiple processing elements. The technique from [4] can be used to include mapping and scheduling information into the SADF model. The switching cost of DVFS is considered in our analysis.

This work was supported in part by the Dutch Technology Foundation STW, project NEST 10346.



(a) SDF.



(b) FSM.

	a_0	a_1	a_2
scenario s_1	50	35	45
scenario s_2	40	15	40

(c) Actor execution times in scenario s_1 and s_2.

Fig. 1. SADF with two scenarios s_1 and s_2.

Ref. [5] addresses the same problem as we do. In the DVFS controller of [5] devised for an SADF, a power mode (i.e., DVFS operating point) is specified for each possible state of the application. *Timestamps* are used to distinguish between states; timestamps capture the miss-aligned completion of the iterations on a platform with multiple processing elements. In [5], an initial state is selected as starting point; for each possible scenario transition from that state a low-power mode that satisfies the timing requirement for the upcoming iteration is considered as desired power mode for that specific scenario switch. This can lead to a new state or a recurrent state. In case of a new state, the exploration should be continued for the new state. The exploration is stopped if all discovered states are recurrent. This way, all possible states within the given power modes are traversed; the authors of [5] categorize their approach within the *state-space based* techniques. Our approach is distinguishable from [5] for several reasons: (1) in [5], only one iteration is considered in power mode selection which can result in a greedy slack distribution within just that iteration; this prevents fair slack distribution over multiple iterations. In contrast, we do power mode selection over all iterations involved in all *critical timing cycles*; a critical timing cycle is defined as a cycle that limits the throughput. In our approach, slack can be used across multiple iterations which generally provides higher energy savings. (2) the state-space based DVFS technique can result in many distinguishable states. This makes the analysis a time consuming procedure.

TABLE I
ACTOR EXECUTION TIMES OF THE EXAMPLE PARAMETRIC SADF.

	pe_1	pe_2		
scenario	a_0	a_1	a_2	scale factor sets
s_1	$10p_{1,1}$	$7p_{1,2}$	$9p_{1,2}$	$\text{SFS}_1 = \{1, 2, 3, 4, 5\}$
s_2	$8p_{2,1}$	$3p_{2,2}$	$8p_{2,2}$	$\text{SFS}_2 = \{1, 2, 3, 4, 5\}$

In our approach, the critical timing cycles are identified and resolved by choosing suitable frequency settings for the processing elements that execute the actors involved in those cycles; processing elements not involved in any critical cycle can operate at their lowest frequency. The analysis time of our technique on four realistic benchmark models is much smaller than the analysis time of the state-space based technique on the same applications. (3) our DVFS controller is more compact than the one from [5] reducing the storage requirement. For a WLAN application, our technique only requires storing 4 sets of processor frequencies (equal to the number of scenarios) while the state-space based technique requires for the same application a controller with 277 frequency sets.

Our technique requires to identify which actor firings in which scenarios are part of the critical timing cycle and on which processing elements those actors are executed. For this reason, the SADF model is extended to a parametric SADF model to accommodate the processor clock cycle periods (i.e. inverse of the frequencies) in the model. In this model, instead of using concrete values (e.g., as shown in Fig. 1(c)), linear expressions (e.g., as shown in Tab. I) provide the actor execution times in terms of some parameters (scale factors). The processors are also explicitly modeled with some initial tokens. This allows us to identify when an iteration has ended on a processor to allow switching to another DVFS operating point. The timing expressions of critical cycles in a parametric SADF reveal which processors are involved in the critical cycles and how much their clock cycle periods (or frequencies) contribute to the length of these timing cycles. This information is used to choose energy-aware frequencies per application scenario to ensure a throughput-constrained solution. The frequency choices are made at design-time and are used at run-time to enable DVFS on iteration boundaries. Our technique is applied to four streaming applications: an MPEG4 video decoder [1], an MP3 audio decoder [6], a WLAN receiver [7] and the baseband (physical layer) processing of the Long Term Evolution (LTE) standard [8]. In all cases, our proposed technique is faster than the technique from [5] and it provides significant energy saving compared to the counterpart technique; furthermore, our technique constructs more compact DVFS controllers.

The remainder of the paper is structured as follows. Sec. II sketches the basic concepts. Sec. III presents our technique. The experimental results are given in Sec. IV. Sec. V discusses related work and Sec. VI concludes.

II. PRELIMINARIES

This section presents the basic terms and approaches required to follow this paper.

A. Synchronous Dataflow Graphs (SDFs)

An SDF is a directed graph (A, C). A node $a \in A$, called *actor*, represents a function (task) of the application and the time required to read/write its input/output data. An edge $c \in C$, called *channel*, captures (data) dependencies between actors. Fig. 1(a) depicts an example SDF with 3 actors and 4 channels. *Tokens* model data communicated through channels. Channels may contain initial tokens, depicted with a solid dot. The example SDF contains two initial tokens labeled t_0 and t_1. An essential property of SDFs is that every time an actor *fires* (executes) it consumes the same amount of tokens from its input channels and produces the same amount of tokens on its output channels. These amounts are called the *rates* (indicated next to the channel ends when the rates are larger than 1). The rates determine how often actors have to fire with respect to each other such that the distribution of tokens over all channels is in balance. This property is captured in the *repetition vector* [2] of an SDF. Fixed rates allow SDFs to execute in a periodic form, which is called an *iteration*. In one iteration each actor is fired as often as indicated in the repetition vector of the SDF. As an essential property of SDFs, the initial token distribution is achieved after one iteration. Consistency (i.e., the existence of a repetition vector) and absence of deadlock are practically, necessary conditions for SDFs which can be verified efficiently [9], [10]. Any SDF that is not consistent requires unbounded memory to execute or deadlocks. So, we only consider consistent and deadlock free SDFs.

B. Max-Plus Analysis for SDFs

An actor $a \in A$ can be associated with a value $\tau_a \in \mathbb{N}_0$ that represents the execution time of the actor plus the time required to read/write its input/output data. One iteration of an SDF graph resets the token distribution to the initial token setting. These tokens may be produced at different points in time. A *token timestamp* vector is defined to specify the production time of tokens. The notation γ_k $(k \in \mathbb{N})$ is used to accommodate the production time of the tokens needed in the k^{th} iteration of the graph. Consider γ_0 as the initial token timestamp vector of the SDF. Assume that all entries in γ_0 are set to zero. For each SDF, a characteristic Max-Plus matrix $G|_{n\times n}$ $(n = |\gamma|)$ exists that can be used to calculate timestamp vectors [11]. An entry $G[i, j] \in G$ corresponds to the minimum time distance from the j^{th} token in the previous iteration to the i^{th} token in the current iteration. The characteristic Max-Plus matrix can be determined using the technique presented in [12]. Consider $\tau_{a_0} = 50$, $\tau_{a_1} = 35$ and $\tau_{a_2} = 45$ time units as sample execution times for the three actors in our example SDF (see Fig. 1(a)). The characteristic matrix for our example SDF is:

$$G = \begin{array}{c c} & \begin{array}{cc} t_0 & t_1 \end{array} \\ \begin{array}{c} t_0 \\ t_1 \end{array} & \begin{pmatrix} 85 & 35 \\ 130 & 80 \end{pmatrix} \end{array}$$

As an example, the matrix G specifies that the minimum time distance from token t_0 in the k^{th} iteration to token t_1 in

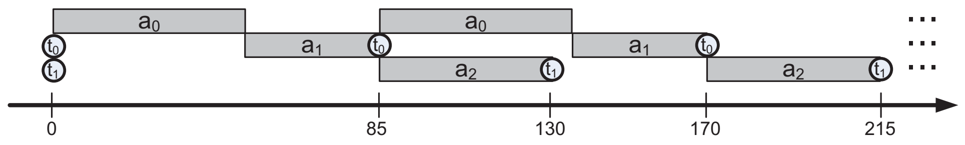


Fig. 2. Execution of the example SDF.

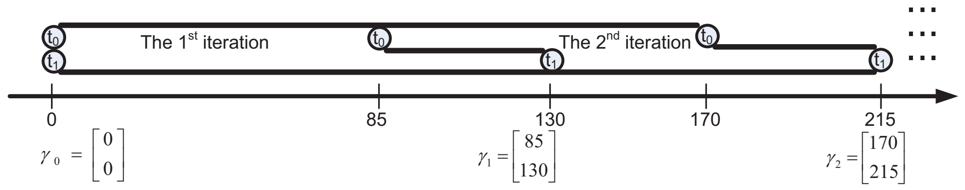


Fig. 3. The evolution of the token timestamp vector for the example SDF.

the $(k+1)^{th}$ iteration is 130 time units via $a_0 - a_1 - a_2$ (130 time units is the result of $\tau_{a_0} + \tau_{a_1} + \tau_{a_2}$).

The evolution of the token timestamp vector can be determined by using Max-Plus matrix multiplication as follow:

$$\gamma_{k+1} = G \cdot \gamma_k \tag{1}$$

The next example shows how the timestamp vector γ_1 is computed using Eqn. 1:

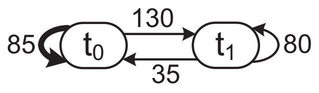
$$\gamma_1 = \begin{pmatrix} 85 & 30 \\ 130 & 80 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \max\{85+0, 30+0\} \\ \max\{130+0, 80+0\} \end{pmatrix} = \begin{pmatrix} 85 \\ 130 \end{pmatrix}$$

Any timestamp vector γ_k $(k \in \mathbb{N})$ can be determined by iteratively performing the Max-Plus multiplication of Eqn. 1. Fig. 2 shows the execution of the example SDF for two iterations; from this execution, the evolution of the token timestamp vector can also be obtained (see Fig. 3).

The technique presented in [6] explains how the throughput of an SDF graph can be determined using the Max-Plus characteristic matrix of the graph; to calculate throughput, the Max-Plus automaton graph (MPAG) is built using the characteristic Max-Plus matrix. The corresponding MPAG for the example SDF is shown in Fig. 4. In an MPAG, a node is created for each initial token in the SDF and if $G[i,j]$ is not equal to $-\infty$ an edge with weight $G[i,j]$ is added from the node of the j^{th} token to the node of the i^{th} token. The $-\infty$ value for an element $G[i,j]$ means that there is no dependency from the j^{th} token to the i^{th} token. A cycle which is limiting the throughput is called a *critical cycle* of the SDF. A critical cycle of the SDF can be discovered by applying a maximum cycle mean (MCM) analysis on the MPAG. The MCM of the MPAG is the inverse of the throughput. The edge related to $G[0,0]$, which is shown with a bold arrow, determines the throughput which is equal to $1/85$ iterations/time unit in our example SDF.

C. Scenario-Aware Dataflow Graphs (SADFs)

The behavior of scenarios in an SADF can be expressed by using SDFs. An example SADF with two scenarios s_1 and



$$\begin{tikzcd} t_0 \arrow[loop left, "85"] \arrow[r, bend left, "130"] & t_1 \arrow[l, bend left, "35"] \arrow[loop right, "80"] \end{tikzcd}$$

Fig. 4. MPAG of the example SDF.

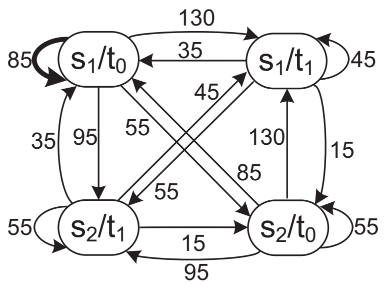


Fig. 5. Max-Plus automaton graph of the example SADF.

s_2 is shown in Fig. 1; s_1 and s_2 are using the same SDF (see Fig. 1(a)), but with different execution times for actors in each of the scenarios (see Fig. 1(c)). Scenario transitions in the SADF are modeled by an FSM. A scenario is specified for each state of the FSM. Each edge in the FSM indicates a scenario transition. The corresponding SDF of an FSM state is executed for one iteration when that FSM state is being activated. Dependencies between consecutive executions of the same or different scenarios are modeled by initial tokens. Initial tokens are labeled to indicate intra-scenario dependencies among different SDFs [13].

For each scenario, a characteristic Max-Plus matrix can be specified. The following shows the characteristic Max-Plus matrices for scenarios s_1 and s_2 of our example SADF:

$$G_{s_1} = \begin{pmatrix} 85 & 35 \\ 130 & 80 \end{pmatrix} \qquad G_{s_2} = \begin{pmatrix} 55 & 15 \\ 95 & 55 \end{pmatrix}$$

An MPAG can be built for each of the Max-Plus matrices using the approach explained in Sec. II-B. The MPAGs of all scenarios can be merged into a single MPAG by using the technique of [6]. Fig. 5 shows the MPAG for our example SADF. In brief, a node is added to the MPAG for each token in the scenario graph of an FSM state (e.g., node s_1/t_0 for token t_0 in scenario s_1 in Fig. 5). If $G_M[y,x]$ is not equal to $-\infty$ and there is a state transition from a state in the FSM that executes scenario N to a state that executes scenario M, an edge with weight $G_M[y,x]$ is added from node N/t_x to node M/t_y in the MPAG. The critical timing cycle of the SADF can be determined by applying MCM analysis on the MPAG. The cycle denoted by the bold arrow shows the critical cycle for our example SADF; the calculated MCM is 85 time units and as a result the throughput of the example SADF is limited to $1/85$ iterations/time unit.

D. Parametric SADF: modeling processor frequencies

The execution time of an actor running on a processing element linearly depends on the clock cycle period of the underlying processing element. Hence, the execution time of the actor can be expressed by using a linear expression. Let $PE = \{pe_1 \dots pe_n\}$ be the set of processing elements and $S = \{s_1 \dots s_m\}$ the set of all scenarios in an SADF. The aim of this paper is to assign for each scenario $s_i \in S$ a set of scale factors (parameters) denoted by $p_{i,1} \dots p_{i,n}$ to minimize the energy consumption while satisfying a certain throughput constraint; $p_{i,j}$ expresses the scale factor (parameter) with which the clock cycle period of the processing element pe_j is scaled in scenario s_i (i.e., the actor execution times scale with the same factor). A processing element usually operates in specific discrete clock cycle points. Hence, we assume that those discrete points are known for each of the processing elements. SFS_j represents the finite set that contains all possible scale factors for processing element pe_j. Let min_j (max_j) be the smallest (largest) scale factor amongst all values in the set SFS_j.

A *parametric SADF* is defined as a model in which the execution times of the actors are expressed in terms of the scale factors. For example, consider the SADF of Fig. 1 in which actor a_0 is mapped to processing element pe_1 and actors a_1 and a_2 are mapped to processing element pe_2. Tab. I contains the execution time of the actors in terms of the scale factors of the processing elements. For each processing element two parameters (i.e., scale factors) are specified because two scenarios exist in the example SADF (e.g., $p_{1,1}$ and $p_{2,1}$ for pe_1). Let F be a set which contains all parameters (e.g., $P = \{p_{1,1}, p_{1,2}, p_{2,1}, p_{2,2}\}$ for the example parametric SADF).

Sec. III contains the proposed technique to determine the suitable scale factors (parameters). For that technique, the expression of the critical timing cycle must be determined for a set of concrete parameters. A set of concrete parameters specifies *a parameter point* represented by $\overline{p}$. A parameter point $\overline{p}$ is a concrete point in the $n \times m$ dimensional parameter space (n and m are respectively the number of processing elements and the number of scenarios); $\overline{p}$ specifies a concrete value $\overline{p}_{i,j} \in \mathrm{SFS}_j$ for each $p_{i,j} \in P$.

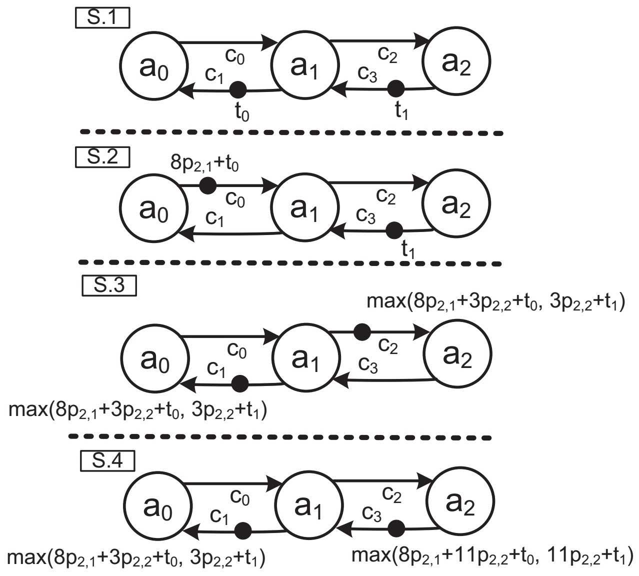

S.1
a0 a1 a2 c0 c1 c2 c3
t0 t1
S.2
8p2,1+t0
t1
S.3
max(8p2,1+3p2,2+t0, 3p2,2+t1)
max(8p2,1+3p2,2+t0, 3p2,2+t1)
S.4
max(8p2,1+3p2,2+t0, 3p2,2+t1)
max(8p2,1+11p2,2+t0, 11p2,2+t1)

Fig. 6. Symbolic execution of scenario graph s_2 of the example SADF (t_0 and t_1 represent the token labels).

E. Critical cycle identification

A parametric throughput analysis technique for SADFs is presented in [14]; this technique builds the MPAG of an SADF to capture the dependencies among initial tokens of consecutive iterations. The technique of [14] can be used for parametric SADFs to determine a throughout expression for a parameter point $\overline{p}$. In [14], a symbolic execution - up to one iteration - of the scenario graphs of the SADF is used to determine information required to construct the MPAG. The symbolic execution of scenario s_2 in our example parametric SADF is shown in Fig. 6. The initial graph is shown in step $S.1$. Consecutive firings of actors a_0, a_1 and a_2 - up to one iteration - are shown in steps $S.2$, $S.3$ and $S.4$. Step $S.4$ indicates that the graph has reset to the initial token distribution after one iteration. The token timestamps in step $S.4$ capture the symbolic dependencies of the initial tokens with respect to their initial values when scenario s_2 is being executed. For instance, the timestamp $max(8p_{2,1}+3p_{2,2}+t_0, 3p_{2,2}+t_1)$ of the reproduced token t_0 (on channel c_1) in step $S.4$ implies that t_0 can be reproduced if $8p_{2,1} + 3p_{2,2}$ time unites has elapsed after production of token t_0 in the previous iteration and $3p_{2,2}$ time units has elapsed after production of token t_1 in the previous iteration. This information is used to construct the MPAG of the SADF in the way that [6] suggests. The symbolic MPAG of the example SADF is shown in Fig. 7. In this figure, a node is placed for each initial tokens of the scenarios. The edges are added using the information acquired from the symbolic execution of the scenario graphs. For example in Fig. 7, an edge from node s_2/t_0 to node s_2/t_1 with weight of $8p_{2,1} + 3p_{2,2}$ is added to the MPAG. The MPAG is evaluated for a given parameter point $\overline{p}$ to form a concrete MPAG. The result of the evaluation for $\overline{p} : \{p_{1,1} = 1, p_{1,2} = 2, p_{2,1} = 3, p_{2,2} = 1\}$ is shown in Fig. 7

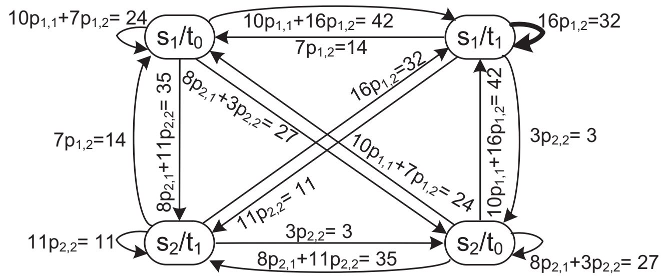
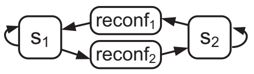
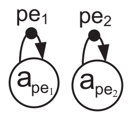


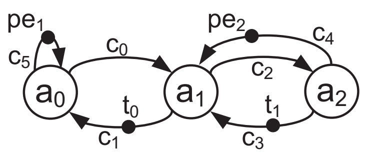
Fig. 7. Symbolic MPAG of the example SADF.



(a) Scenario FSM of reconfiguration.



(b) Reconfiguration SDF.



(c) Mapped scenario graph.

Fig. 8. Modeling mapping and reconfiguration in the example SADF.

as the numbers that follow from the edge terms. The critical timing cycle can be found by applying a maximum cycle mean (MCM) analysis on the concrete MPAG. The bold arrow in Fig. 7 shows the critical timing cycle in our example graph. The symbolic terms of each edge in this cycle are used to determine the critical cycle expression for the given parameter point $\overline{p}$ (i.e., $16p_{1,2}$ for our example). This critical timing cycle implies that with the chosen parameter point, the throughput of the application is limited by the frequency of processor pe_2. Increasing its frequency when executing scenario s_1 may increase the throughput of the application.

III. Proposed technique

Power consumption in VLSI circuits depends linearly on the operating frequency and quadratically on the supply voltage of the processing elements [15]. While lowering the voltage supply, the maximal possible operating frequency also reduces. Hence, lowering the voltage and frequency could reduce the total energy consumption quadratically at linear time cost [16]. So, in our technique, the execution times are expressed linearly in terms of the processor clock cycle periods (i.e. inverse of the frequencies) and energy consumption is expressed quadratically in terms of the processor frequencies.

A. Overview

This section presents a multiprocessor frequency assignment technique for dynamic applications modeled by SADF graphs in such a way that a strict throughput requirement is guaranteed. In the following subsection, we show how to capture the timing delays resulting from DVFS in an SADF model. Our DVFS assignment technique starts with the minimum energy option; in other words, it assigns the lowest possible frequency (or the highest clock cycle period) to each processor running in an application scenario. The initial setting may violate the required throughput. So, the technique checks whether or not the initial setting satisfies the throughput; if the throughput is satisfied, the initial setting is reported as final solution. In the other case, our technique finds a critical timing cycle in the application which violates the throughput. Then, the frequency of the processors involved in the critical cycle are increased to make that cycle fit within the required throughput. Even after resolving the first critical cycle, some other timing cycles may exist which are violating the throughput. Those timing cycles are similarly resolved one after another until all of the timing cycle in the SADF model respect the required throughput. Sec. III-C discusses our technique in detail.

B. Modeling voltage & frequency scaling in SADF

DVFS can be viewed upon as a reconfiguration at run-time. This can be effectively modeled in SADFs as a *reconfiguration scenario*. In the FSM of the SADF, an intermediate state must be placed before switching to another original FSM's state. Two *reconfiguration states* are added to the FSM of our example SADF to capture the reconfiguration steps (see Fig. 8(a)). For a reconfiguration state, a reconfiguration scenario is defined. A reconfiguration scenario captures any step involved in the reconfiguration operation. For instance in our example SADF which is mapped to a platform with two processing elements, the reconfiguration requires setting the frequencies and voltages for two processing elements. The SDF of the reconfiguration scenario for each of the processing elements contains an actor with a self-edge; on that self-edge, a token with a label indicating the related processing element models the processing resource dependency between consecutive iterations (see Fig. 8(b)). The execution time of the added actors are set to the DVFS delay. This way, the overhead of switching between different frequency points on iteration boundaries is considered. Mapping should also be modeled in the scenario graphs of the SADF. The technique from [4] is used for this purpose. Fig. 8(c) shows such a modeling for the scenario graph of the example SADF; the processor tokens in this SDF establish the resource dependencies among all scenarios. This effect is shown graphically in Fig. 9; in this figure, an example scenario transition from scenario s_1 to scenario s_2 is depicted. The time required to capture the DVFS delay is modeled by a reconfiguration scenario shown with $Reconfig_1$ in Fig. 9. Hence processor tokens pe_1 and pe_2 can be released (to be used by s_2) after the DVFS setting completion on both of the processing elements.

C. Clock cycle period settings under a throughput constraint

The used SADF is a parametric model according to Sec. II-D. Algorithm 1 contains our heuristic technique to identify the scale factors (parameters) which result in energy savings under a throughput constraint. G represents the given parametric SADF and $Period$ represents the inverse of the required throughput. The set SFS of scale factors is also

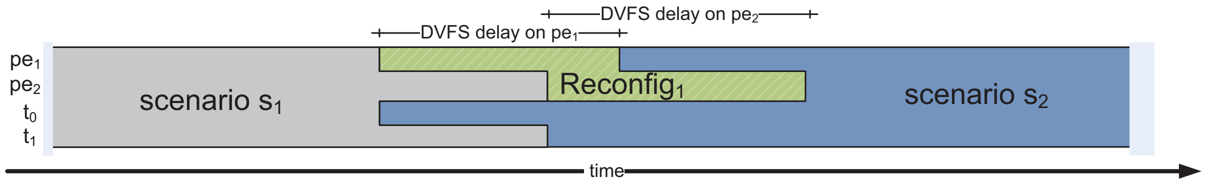


Fig. 9. Processor tokens to model resource dependency across scenarios.

required as an input to the algorithm. Each $\text{SFS}_j \in \text{SFS}$ contains all concrete values that a parameter $p_{i,j}$ can obtain (i.e., all allowed scale factors for processor pe_j). $Solution$ is a set which contains the final concrete values for all parameters (as the output of the algorithm). For the example SADF, 65 time units is specified as the timing constraint (i.e, $Period = 65$). The DVFS delay is assumed to be 1 time unit in our example.

Algorithm 1: Throughput-constrained DVFS for SADF

```
  input : Parametric SADF $G$
  input : Timing constraint $Period$
  input : Scale factor sets $\text{SFS} = \{\text{SFS}_1 \dots \text{SFS}_n\}$
  output: Scale factor set $Solution$
1 if feasibilityCheck($G$, $min_1 \dots min_n$, $Period$) $\neq$ "Feasible" then
2     return $Solution \leftarrow \emptyset$
3 n $\leftarrow$ number of processing elements
4 m $\leftarrow$ number of scenarios in $G$
5 for $j \leftarrow 1$ to n do
6     for $i \leftarrow 1$ to m do
7         $\overline{pa}_{i,j} = max_j$
8 while true do
9     /* step 1 */
10    $criticalCycleExpr_1 \leftarrow$ getCriticalCycle($G$, $\overline{pa}$)
11    if $|criticalCycleExpr_1| > Period$ then
12        $\overline{pb} \leftarrow$ resolveCycle($criticalCycleExpr_1$, $\overline{pa}$, SFS, $Period$)
13    else
14        return $Solution \leftarrow \overline{pa}$
15    /* step 2 */
16    $criticalCycleExpr_2 \leftarrow$ getCriticalCycle($G$, $\overline{pb}$)
17    if $|criticalCycleExpr_2| > Period$ then
18        $\overline{pc} \leftarrow$ resolveCycle($criticalCycleExpr_2$, $\overline{pb}$, SFS, $Period$)
19    else
20        return $Solution \leftarrow \overline{pb}$
21    /* re-initialization */
22    for $j \leftarrow 1$ to n do
23        for $i \leftarrow 1$ to m do
24            if $\overline{pb}_{i,j} \neq \overline{pc}_{i,j}$ then
25                $\overline{pa}_{i,j} = \overline{pc}_{i,j}$
```

As a first step, the existence of any valid solution for the given scale factors is checked by the function feasibilityCheck (lines 1-2 in Algorithm 1); this is checked by extracting the critical timing cycle for the case that all parameters (scale factors) are set to their minimum values (i.e. the highest processor frequencies). The problem is not feasible in case the length of the critical cycle is larger than the timing constraint (i.e., $Period$); otherwise, the analysis is continued to find the desired solution.

Initially, all parameters in the parametric SADF are set to their maximum value (lines 5-7 in Algorithm 1); this assures the lowest energy consumption, although this may not satisfy the timing requirement. The technique iteratively refines the initial parameter setting to obtain a parameter point that meets the timing constraint. The repetitive part of the algorithm (lines 8-25 in Algorithm 1) is composed of two main steps followed by a re-initialization step. In each of the main steps, a critical cycle of the SADF for the given parameter point is extracted by function getCriticalCycle; the approach explained in Sec. II-E is used for this. In case of multiple critical cycles with equal length, one of them is chosen arbitrarily. The other cycles can be processed in later repetitions of the algorithm. The extracted critical cycle must be resolved by choosing a proper scale factor for the parameters involved in the critical cycle; our scale factor selection approach, which is abstracted by function resolveCycle in the algorithm, picks a parameter point amongst all possible combinations of the involved parameters in the critical cycle to achieve the lowest energy consumption while still meeting the timing requirement. The complexity of resolveCycle depends on the number of parameters that contribute to the critical cycle (denoted by ρ) and the number of possible clock cycle (or frequency) points for each of the processing elements (denoted by π). So, the number of parameter points required to be verified by resolveCycle is equal to π^ρ because each parameter in the critical cycle can get π different values. The value of π depends on the platform property; the value of ρ depends on the application property and at worst case ρ can be equal to the number of parameters in the parametric SADF model. Hence, resolveCycle has an exponential complexity. However, our experiments on several real applications reveal that only a few parameters contribute to the critical cycle (i.e., in practice ρ is small); hence, verifying all parameter points can be done quickly (refer to Sec. IV).

Figure 10 depicts the application of our algorithm to the example SADF. At step 1 of the 1^{st} repetition, the critical timing cycle with expression $10p_{1,1} + 7p_{1,2}$ is extracted when all parameters are initialized with a value of 5 time units. This cycle violates the required timing constraint of 65 time units. Hence, the parameters $p_{1,1}$ and $p_{1,2}$ must be set in a way that this cycle gets bounded within the required period. All parameter combinations of $p_{1,1}$ and $p_{1,2}$ are shown in Fig. 10(a) (left-top); the points marked with solid black dots are not valid selections as they violate the timing constraint. Among the rest of the parameter points, $p_{1,1} = 3$ and

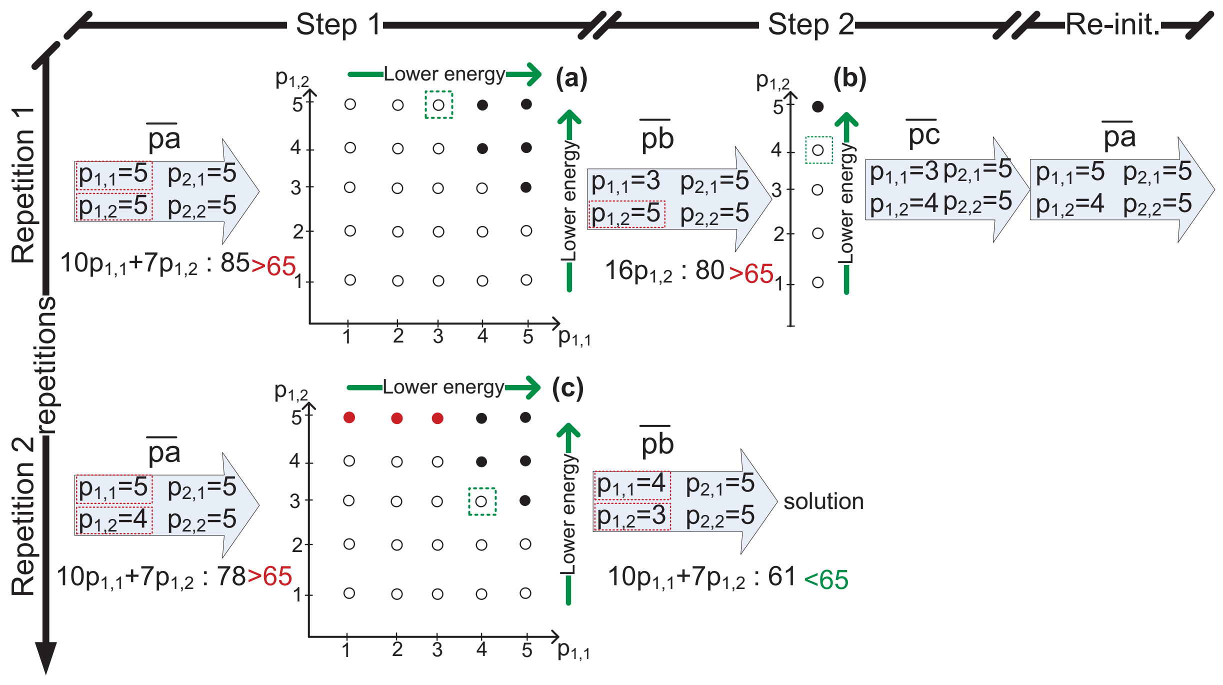


Fig. 10. Applying our technique to the example SADF.

$p_{1,2} = 5$ are selected since this choice assures the lowest energy consumption at this stage. In this example, we assume that the platform is homogeneous and processing elements consume equal amounts of energy when they are operating at the same frequency. The resulting parameter point after step 1 (i.e., parameter point $\overline{pb}$) is fed to step 2. The first critical cycle now has been resolved by step 1; the next critical cycle can be extracted in step 2. The new critical cycle found in step 2 can be resolved similarly as step 1. As shown in step 2 of the 1^{st} repetition in Fig. 10, only parameter $p_{1,2}$ contributes to the second critical cycle; so, $p_{1,2}$ is reduced to value 4 in order to resolve the critical cycle found in step 2. The outcome of the second step is the parameter point $\overline{pc}$. Resolving a critical cycle by one step cannot enlarge other critical cycles found in prior steps; because in function resolveCycle, we restrict our choice to parameter points in which all parameters have values smaller than or equal to the ones in the input parameter point argument of resolveCycle.

Both of the main steps decrease parameters in order to shrink the identified critical cycles. The re-initialization step (lines 22-25 in Algorithm 1) provides an opportunity for parameters to avoid unnecessary frequency increase for some processing elements involved in critical timing cycles when possible. Consider that $\overline{pa}$ is the parameter point to be used in the subsequent algorithm repetition. The re-initialization step updates the parameters in $\overline{pa}$ with the values of the parameters whose values have been reduced in step 2. Parameters which were only reduced in step 1 of the current repetition keep their original values. In this way, they are reconsidered in the next algorithm repetitions. The reason for not changing the parameter values for those parameters that were only changed in the first step is that the critical cycle of the first step may have been affected by the adaptations made in the second step. As a result of the re-initialization step, the information obtained in one repetition is used in subsequent repetitions to provide better parameter point selection. In the second repetition of the algorithm for our example SADF, resolving the first critical cycle $10p_{1,1} + 7p_{1,2}$ is performed with the knowledge that parameter $p_{1,2}$ should get a value smaller than 5 because of the critical cycle $16p_{1,2}$; the solid red dots in Fig. 10(c) display this effect. Hence, step 1 in the second repetition of the algorithm reduces $p_{1,1}$ to 4 (instead of 3 after step 1 of the 1^{st} repetition). The critical cycle identified by step 2 in the second repetition is already smaller than the required $Period$ and the algorithm stops further analysis and reports the current parameter point (i.e., $\overline{pb}$ in the second repetition) as the final solution.

The algorithm stops either after step 1 or step 2 (line 15 or line 20 in the algorithm) whenever the length of the critical cycle in one of those steps is smaller than the timing requirement; if not, the re-initialization step fixes the values of those parameters which are reduced in step 2 for the subsequent repetitions. This ensures that by each repetition of the algorithm some parameters get smaller and eventually the timing requirement is met. In our algorithm, two main steps perform parameter point selection, each considering the information from one critical cycle. Increasing the number of main steps in our algorithm allows considering more critical cycles in our parameter point selection; however, more steps implies more analysis time. Empirically, the number of main steps is set to two in our heuristic.

TABLE III
EXPERIMENTAL RESULTS.

	the technique from [5]		*our technique*			
Benchmark	**#states**	**analysis times (ms)**	**#rep**	**analysis times (ms)**	ρ	**energy gain**
MPEG4 dec. (9 scenarios)	over 300k	not finished in 3 days	4	104	2	N/A
MPEG4 dec. clustered (4 scenarios)	189 415	47 192 700	2	16	2	34 %
MP3 dec.	over 700k	not finished in 3 days	9	416	3	N/A
MP3 dec. quantized	4370	162 550	9	416	3	9%
WLAN	277	3 796	3	10	8	10%
LTE	65	456	3	8	3	44%

TABLE II
THE SPECIFICATION OF BENCHMARK APPLICATIONS.

Benchmark	**#sce.**	**#p.e.**	π	**#par**	**timing constraint**
MPEG4 dec.	9	4	2	36	20 frames/sec.
MP3 dec.	5	3	2	15	20 frames/sec.
WLAN	4	3	2	12	250k OFDM symbols/sec.
LTE	5	2	5	10	100 symbols/sec.

IV. EXPERIMENTAL RESULTS

The proposed DVFS technique for SADFs is compared to the related work (i.e., [5]) which uses an approach based on state-space exploration. This comparison is performed because [5] is the closest approach to our technique in the literature and we solve a similar problem for (scenario-aware) dataflow graphs. A set of realistic applications is used for this comparison: an MPEG4 video decoder [1], an MP3 audio decoder [6], a WLAN receiver [7] and the baseband (physical layer) processing of the Long Term Evolution (LTE) standard [8]. Tab. II shows for each of the applications the number of application scenarios ($\#sce.$), the number of processing elements ($\#p.e.$) in the platform to which the application is mapped, the number of the available frequency points (π), the number of parameters in the parametric SADF ($\#par$) and the timing constraint of each application. The overhead of DVFS is set to a value taken from [17], [18]. Hence, 10 ns is used as delay of DVFS in our experiments. All experiments are performed on an Intel core i7 (3 GHz) with 4GB of RAM running Linux.

The complexity of both our DVFS technique and the state-space based technique from [5] is determined by the number of scenarios. Scenarios of an application can be clustered to form a less complex model. For instance, we clustered 9 scenarios in the MPEG4 decoder into a smaller model with 4 scenarios. Analyzing a smaller model can be done faster than the original model; but, analysis of such a clustered model can deteriorate the accuracy of the analysis, in the end leading to suboptimal frequency settings. An approach to further limit the number of states distinguished by the state-space based technique from [5] is to quantize the execution time of the actors; but, timing quantization can also affect the accuracy of the model and as a result the outcome of the DVFS analysis.

Initially, we apply our technique and the technique from [5] to the original models of the benchmark applications. Results are shown in Tab. III. Besides analysis times, the number of unique states identified for each of the benchmarks are reported when applying the state-space based DVFS. As our technique is a repetitive algorithm, the number of repetitions is reported for our technique. For our technique, the maximum number of parameters found in any of the critical cycles of the SADF model is also reported (ρ); the small values for ρ in our results show that the function $resolveCycle$ can quickly find a suitable parameter point for critical cycles to make them fit within the throughput constraint while looking for an energy efficient option. For the original model of the MPEG4 decoder and the MP3 decoder, the state-space based DVFS does not find solutions within a reasonable time (i.e., 3 days). The number of states of an application can drastically increase; hence, analyzing all states may not be a practical approach. Our technique finds solutions in a fraction of a second.

To make the state-space based technique applicable to the MPEG4 decoder, we perform scenario clustering; 9 scenarios in the original model are clustered into 4 separate scenarios. The results of applying our technique and the state-space based technique on the clustered model of the MPEG4 decoder are also shown in Tab. III. For the MP3 decoder, the execution time of the actors are quantized to limit the state space. Our technique can be applied on both the MPEG4 decoder and MP3 decoder without any scenario clustering nor any timing quantization; the state-based technique only works on the coarser models. The results in Tab. III show that our technique is also on the coarser models much faster than the state-space based DVFS.

The memory required to store the DVFS controller devised by our technique depends on the number of scenarios in the SADF, while for the state-space based technique, the memory size depends on the number of discovered states. The results in Tab. III (the second column) show that more compact DVFS controllers are achievable by using our technique.

We also estimate the energy consumption of an SADF running on a platform with multiple processing elements. A long sequence of scenario iterations (i.e., 200k scenario iterations in our experiments) is fed to the DVFS controller devised by each of the two techniques. The energy consumption is calculated per iteration. Each experiment is performed 10 times with a different seed for the scenario sequence generator; the results reported in Tab. III, last column, are averages of those 10 experiments. The results show that our technique offers solutions with less energy consumption compared to the technique from [5]. The reason why our technique offers lower energy consumption is because our algorithm considers critical cycles that may run across multiple iterations when assigning clock frequencies. In this way timing slack of one or several iterations can be effectively exploited through all iterations of a critical cycle. As a result, operating frequencies of processing elements can get lowered.

Fig. 11 gives some concrete energy results for the MPEG4 decoder. The figure confirms the energy savings obtained by our technique for the model with 4 scenarios. It also shows, however, that a more refined model with 9 scenarios allows a further reduction in energy consumption. Our technique scales better to finer-grained models than the state-based technique.

We also performed some experiments for processing elements with different numbers of the frequency points. By increasing the number of frequency points, we provide frequencies with higher resolutions. To assess the gain of dynamic switching, our technique is also used to find a static solution in which each processing element runs at a fixed frequency. Applying our technique to a parametric SADF in which one parameter (scale factor) is specified per processing element across all scenarios determines a static solution. Fig. 12 depicts the results when our technique, the static approach and the technique from [5] are used to devise a DVFS controller for the WLAN application. Increasing the number of frequency points increases the number of states for the application which is why the technique from [5] is not capable of finding any solutions for the cases with more than two frequency points for the WLAN application; the analysis times are shown in Fig. 12(a). Energy consumption values are shown in Fig. 12(b); the values are normalized using the largest value. Our technique assures 10% to 41% lower energy consumption compared to the state-space based technique when the number of frequency points increases from 2 to 10. Increasing the number of frequency points provides more refined frequencies to save more energy. Fig. 12(b) also shows that both DVFS techniques provide less energy consumption compared to the static technique. The analysis time of our technique rises by increasing the number of frequency points because this increases the number of parameter points to be verified in our algorithm. However, the analysis time of our technique is not too high to make the analysis infeasible. The static approach based on our technique is fast because the number of parameters in the parametric SADF is limited to only the number of processing elements. Our algorithm scales reasonably well because typically only a limited number of parameters are involved in critical timing cycles.

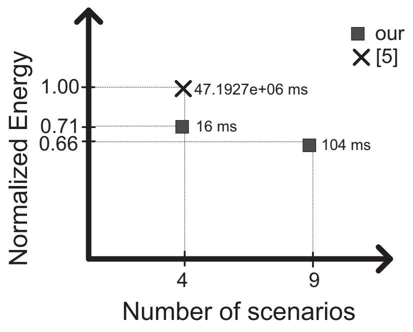
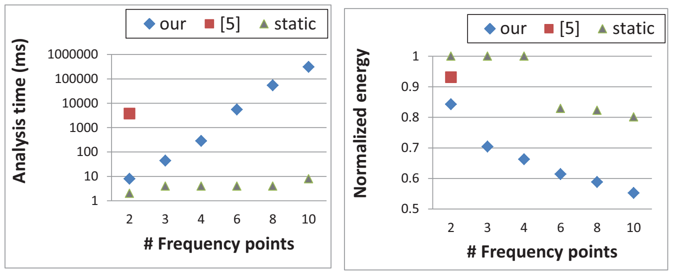


Fig. 11. Impact of scenario clustering on energy consumption and analysis time.



(a) Analysis time. (b) Normalized energy.

Fig. 12. Results of WLAN for different number of frequency points.

V. Related work

Related DVFS approaches can be viewed from three different angles: (1) design policy; (2) application model; (3) solution granularity. The *design policy* determines whether the DVFS choices are made at *run-time* [19], [20] or *design-time* [21], [22]; the first type predicts the workload and adjusts the supply voltage and the operating frequency of the underlying processing elements at run-time. The second ones assume that the workload of applications are known at design-time. Run-time approaches suffer from extra timing and energy overheads. Design-time approaches can result in pessimistic solutions if the behavior of the applications is not captured properly. The application models to devise a DVFS controller may be *static* or *dynamic*; in static models (e.g., SDF and task graphs) [21], [22], actors (tasks) use the worst case as their execution time (i.e., WCET). Dynamic models (e.g., SADF) [5], [23] capture execution time variation. Static models are simple and easy to analyze which make them suitable for static applications. Dynamic models are favorable for more dynamic applications (e.g., modern multimedia applications). The solution granularity determines how often a voltage and frequency switch can occur in a design. In a *fine-grained* solution [21], [22], a DVFS may happen before executing actors (tasks); in a *coarse-grained* solution [5], the DVFS may happen before executing iterations (e.g., processing a video frame). The overhead of DVFS, the size of the actors and the size of the iteration determine which granularity is suitable for a system.

Our technique analyzes SADFs to devise a coarse-grain DVFS controller at design time for dynamic throughput-constrained applications. SADFs are suitable to capture the dynamic behavior of applications. We offer a coarse-grained DVFS solution to avoid overhead because of the frequent voltage and frequency switches in fine-grained solutions. However, fine-grained solutions like [21] can be beneficial when the overhead is negligible. Some run-time approaches (e.g., [24]) use WCET information to perform better run-time power management. Our technique can also be used to provide initial information for such run-time techniques to further tune the final solution and loosen the run-time overhead; we leave further run-time refinement of our technique as future work.

Among the related work, [5] is the most similar to ours. The technique of [5] suffers from state-space explosion and it requires much time for the analysis; our technique finds solutions for all of our benchmark applications in seconds to minutes, depending on the number of scenarios and frequency points. Moreover, using [5] results in pessimistic solutions because of the greedy approach used in its voltage and frequency selection. Our technique considers all iterations involved in the critical timing cycles to effectively utilize slack; in this way, workloads are balanced across multiple iterations and frequencies can be lowered. As a result, our technique assures better solutions in terms of energy consumption. The DVFS controller devised by our technique is more compact than the DVFS controller of [5]; this can save considerable memory space to store the controller.

The authors of [25] develop a game theory-based technique to synthesize a controller for SADFs; they optimize throughput by modifying the scheduling policy. The resulting controller of [25] is optimized for a single design metric (i.e., only throughput), while our technique reduces energy consumption under a throughput constraint. Note that our technique is a heuristic which provides sub-optimal solutions and the technique of [25] finds optimal solutions. Extending the game theory-based technique of [25] to consider two design metrics is a relevant, but challenging problem which requires further research. Ref. [23] presents a technique to detect application scenarios at design-time; it also devises a pro-active voltage scaling by predicting the scenario sequences. However, the technique of [23] is not applicable to multiprocessor platforms.

VI. Conclusion

A technique is developed to synthesize a DVFS controller for SADFs to reduce the energy consumption while meeting a throughput requirement. The SADF model is extended to a parametric model in order to capture the processor frequencies of the platform to which the application is mapped. The proposed technique uses a symbolic version of a Max-Plus automaton graph analysis to identify the critical timing cycles. Initially, the application is set to the lowest possible energy mode. Then, the critical cycles that are violating the timing requirement are repetitively resolved by refining the processor frequencies. Our analysis is faster than the state of the art technique for dataflow graphs; the experiments show that our technique furthermore constructs more compact DVFS controllers with lower energy consumption.

References

[1] 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 *MEMOCODE'06*, pp. 185 –194.

[2] S. Bhattacharyya, P. Murthy, and E. Lee, "Synthesis of embedded software from synchronous dataflow specifications," *J. VLSI Signal Processing*, vol. 21, pp. 151–166, 1999.

[3] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low power CMOS digital design," *IEEE J. Solid State Circuits*, vol. 27, pp. 473–484, 1995.

[4] M. Damavandpeyma, S. Stuijk, T. Basten, M. Geilen, and H. Corporaal, "Modeling static-order schedules in synchronous dataflow graphs," in *DATE'12*, pp. 775–780.

[5] J. Zimmermann, O. Bringmann, and W. Rosenstiel, "Analysis of multi-domain scenarios for optimized dynamic power management strategies," in *DATE'12*, pp. 862 –865.

[6] M. Geilen and S. Stuijk, "Worst-case performance analysis of synchronous dataflow scenarios," in *CODES+ISSS'10*, pp. 125–134.

[7] O. Moreira, "Temporal analysis and scheduling of hard real-time radios running on a multi-processor," Ph.D. dissertation, TU Eindhoven, 2012.

[8] D. Martín-Sacristán, J. F. Monserrat, J. Cabrejas-Peñuelas, D. Calabuig, S. Garrigas, and N. Cardona, "On the way towards fourth-generation mobile: 3GPP LTE and LTE-advanced," *EURASIP J. Wirel. Commun. Netw.*, vol. 2009, pp. 4:1–4:10, Mar. 2009.

[9] S. Bhattacharyya, P. Murthy, and E. Lee, *Software Synthesis from Dataflow Graphs*. Kluwer Academic Publishers, 1996.

[10] E. Lee and D. Messerschmitt, "Synchronous data flow," *IEEE Proceedings*, vol. 75, no. 9, pp. 1235 –1245, 1987.

[11] F. Baccelli, G. Cohen, G. J. Olsder, and J. pierre Quadrat, *Synchronization and Linearity*. John Wiley & Sons.

[12] M. Geilen, "Synchronous dataflow scenarios," *ACM Trans. Embed. Comput. Syst.*, vol. 10, no. 2, pp. 16:1–16:31, Jan. 2011.

[13] M. Geilen, J. Falk, C. Haubelt, T. Basten, B. Theelen, and S. Stuijk, "Performance analysis of weakly-consistent scenario-aware dataflow graphs," TU Eindhoven, Tech. Rep. ESR-2011-03, 2011.

[14] M. Damavandpeyma, S. Stuijk, M. Geilen, T. Basten, and H. Corporaal, "Parametric throughput analysis of scenario-aware dataflow graphs," in *ICCD'12*, pp. 219 –226.

[15] T. Burd and R. Brodersen, "Energy efficient CMOS microprocessor design," in *HICSS'95*, p. 288.

[16] T. Burd, T. Pering, A. Stratakos, and R. Brodersen, "A dynamic voltage scaled microprocessor system," *IEEE J. Solid-State Circuits*, vol. 35, pp. 1571–1580, 2000.

[17] M. Meijer, J. de Gyvez, and R. Otten, "On-chip digital power supply control for system-on-chip applications," in *ISLPED'05*, pp. 311 – 314.

[18] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *HPCA'08*, pp. 123 –134.

[19] A. Alimonda, S. Carta, A. Acquaviva, A. Pisano, and L. Benini, "A feedback-based approach to DVFS in data-flow applications," *TCAD*, vol. 28, no. 11, pp. 1691–1704, 2009.

[20] P. Choudhury, P. P. Chakrabarti, and R. Kumar, "Online dynamic voltage scaling using task graph mapping analysis for multiprocessors," in *VLSID'07*, pp. 89–94.

[21] A. Nelson, O. Moreira, A. Molnos, S. Stuijk, B. T. Nguyen, and K. Goossens, "Power minimisation for real-time dataflow applications," in *DSD'11*, pp. 117–124.

[22] D. Shin and J. Kim, "Power-aware scheduling of conditional task graphs in real-time multiprocessor systems," in *ISLPED'03*, pp. 408–413.

[23] S. V. Gheorghita, T. Basten, and H. Corporaal, "Scenario selection and prediction for DVS-aware scheduling of multimedia applications," *J. Signal Process. Syst.*, vol. 50, no. 2, pp. 137–161, 2008.

[24] P. Vivet, E. Beigne, H. Lebreton, and N.-E. Zergainoh, "On line power optimization of data flow multi-core architecture based on vdd-hopping for local DVFS," in *PATMOS'10*, 2011, pp. 94–104.

[25] Y. Yang, M. Geilen, T. Basten, S. Stuijk, and H. Corporaal, "Playing games with scenario- and resource-aware sdf graphs through policy iteration," in *DATE'12*, pp. 194–199.