Analytic Clock Frequency Selection for Global DVFS

Marco E.T. Gerards, Johann L. Hurink, Philip K.F. Hölzenspies, Jan Kuper, Gerard J.M. Smit
University of Twente, Department of EEMCS
P.O. Box 217, 7500 AE Enschede, The Netherlands
m.e.t.gerards@utwente.nl

***Abstract*—Computers can reduce their power consumption by decreasing their speed using Dynamic Voltage and Frequency Scaling (DVFS). A form of DVFS for multicore processors is *global* DVFS, where the voltage and clock frequency is shared among all processor cores. Because global DVFS is efficient and cheap to implement, it is used in modern multicore processors like the IBM Power 7, ARM Cortex A9 and NVIDIA Tegra 2. This theory oriented paper discusses energy optimal DVFS algorithms for such processors.**

There are no known provably optimal algorithms that minimize the energy consumption of nontrivial real-time applications on a global DVFS system. Such algorithms only exist for single core systems, or for simpler application models. While many DVFS algorithms focus on tasks, this theoretical study is conceptually different and focuses on the amount of *parallelism*. We provide a transformation from a multicore problem to a single core problem, by using the amount of parallelism of an application. Then existing single core algorithms can be used to find the optimal solution.

Furthermore, we extend an existing single core algorithm such that it takes static power into account.

***Keywords*—Dynamic voltage and frequency scaling, energy minimization, mathematical programming, parallel processing**

I. Introduction

As the power consumption of computing devices has increased exponentially [1], energy consumption has become one of the most important design criteria for these devices. For portable embedded devices this holds even more, since the energy-density of batteries does not grow with the same rate as the energy consumption of these devices.

In this paper, we present algorithms for minimizing the energy consumption of a multicore system that executes a real-time application with precedence constraints. We focus on Dynamic Voltage and Frequency Scaling (DVFS) to decrease the energy consumption. DVFS allows for the decrease of the clock frequency and voltage of the processor, leading to a decrease of the power consumption, at the cost of an increased latency. Mathematical techniques and algorithms are frequently used to determine optimal clock frequencies. Such techniques are referred to as algorithmic power management [1], [2]. Based on a survey of such techniques by Irani and Pruhs [1], the current paper can be qualified as an algorithmic power management paper; we use algorithms to find clock frequencies that globally minimize the energy consumption.

This work is supported through NWO project EASY.

For multicore processors, there are two main flavors of DVFS, namely local DVFS and global DVFS. While local DVFS changes the clock frequency per core, global DVFS makes these changes for the entire chip. For this reason, the optimal solutions to the local and global DVFS problems are not interchangeable. Global DVFS is in practice the most common of these techniques, since it is cheaper to implement [3], [4]. Examples of modern processors and systems that use global DVFS are the Intel Itanium, the PandaBoard (dual-core ARM Cortex A9), IBM Power7 and the NVIDIA Tegra 2 [3], [5]–[7]. Because global DVFS is often used, we focus on global DVFS alone.

The aperiodic real-time applications that we consider consist of many tasks with an execution order that is restricted by precedence constraints. Each task has an execution time, an arrival time and a deadline; our solution method does not impose any restrictions on these properties of the tasks. Before such an application can be executed, its tasks need to be scheduled. Note that both the scheduling policy and clock frequency selection have an influence on the energy consumption.

We focus on determining optimal clock frequencies for the entire run-time of an application. Hereby, we assume that a feasible schedule for the tasks of the application is *given*. Since we consider global DVFS, the concrete assignment of clock frequencies does not influence the relative order of the execution of tasks and therefore does not influence the feasibility of a solution with respect to precedence constraints.

Optimal clock frequency selection is a nontrivial problem, since it may be efficient to finish some tasks early, such that later tasks can run on a lower clock frequency as several authors have demonstrated [8], [9]. This property makes the problem a global problem *over time*. Furthermore, the amount of *parallelism* (i.e., number of active cores at a given time) is important when calculating the optimal clock frequencies [10].

Conceptually, this paper is different from many DVFS papers. While many papers choose a clock frequency for tasks, we choose a clock frequency that depends on the amount of parallelism, while respecting constraints for individual tasks. This is based on the following intuition.

Since low clock frequencies have a higher energy efficiency, it is better to increase the clock frequency when only a few cores are active (increasing the energy only for few cores) and decrease the clock frequency when more cores are active (decreasing the energy for more cores). This may lead to a reduction of the total energy consumption. However, since the energy consumption depends in a superlinear and convex

way on the clock frequency [8], [9], [11], [12], very high clock frequencies lead to extreme high costs. Hence, the clock frequency when only very few cores are active should not be arbitrarily high meaning that a trade-off is required. While this trade-off has been formally studied in a simplified setting [10], we study it in the context of a nontrivial application with arrival time and deadline restrictions.

We present an algorithm that finds the optimal clock frequencies by generalizing the aforementioned intuition. The approach presented in this paper solves the multicore problem by first transforming it to a single core problem (depending on the amount of parallelism), then solving this single core problem, and finally transforming the solution of the single core problem back to the original problem. We combine several existing techniques from the literature to motivate our transformation and to solve our problem.

The main contributions of our paper are as follows.

- An overview of existing literature to motivate the reduction, and the application of specific approaches from the literature to solve the single core problem (Section II).
- A method to reduce the (multicore) global DVFS problem to a single core problem (Section IV).
- An optimal algorithm for the single core problem that takes static power into account (Section V).

Although our solution is only optimal under the modeling assumptions mentioned in Section III, the insights from this paper are also useful for other systems.

II. Related work

Several papers discuss optimal DVFS for multiprocessor/multicore systems with precedence constraints and a single deadline for the entire application [13], [14]. In contrast, we allow independent arrival times and deadlines for all tasks in our application. Applications that can be modeled and optimized using our model are, for example, streaming applications such as audio- and video decoders [9] and telecommunication applications, where frames and/or datagrams all have individual deadlines and are processed by parallel tasks.

In our previous research [15], we study a simplified real-time system with precedence constraints, where all tasks arrive when the application begins and a *single* deadline for the entire application is used. In that research, we already prove that optimal scheduling for a restricted version of the real-time system that we consider in the current paper is NP-hard. The main topics of our previous article are the *interaction* between scheduling and clock frequency selection for a restricted real-time system with a single deadline for the entire application and energy efficient scheduling, in contrast to the current paper that focuses on *clock frequency selection* and discusses the more complex and very general real-time system with precedence constraints, arbitrary arrival times, deadlines and work.

Cho and Melhem [10] show how for a restricted model the optimal clock frequencies depend on the amount of parallelism. Although we do not use the same model, their observation, that there is an interplay between parallelism and energy consumption, is also used throughout our paper. Pruhs et al. [16] present the power inequality, which can be used to relate the optimal clock frequencies of tasks to each other in a local DVFS system, but provide no method to actually calculate the optimal clock frequencies. We focus on global DVFS alone for a general real-time system and do provide optimal clock frequencies.

Yang et al. [17] use a similar observation in their paper on global DVFS for frame-based systems. For these kinds of systems, they show how the optimal clock frequencies depend on the amount of parallelism in an application. It was proven [15], [17], that when at time t_1 there are n active cores and at time t_2 there are m active cores, the ratio between the optimal clock frequencies is $\sqrt[\alpha]{m/n}$ (for some constant α to be discussed). We use this factor in a substitution of variables to transform our multiprocessor problem to an equivalent single core problem. This reduction is one of the main contributions of the current paper, since based on this substitution we can use single core algorithms to solve our multicore problem.

Yao et al. [8] study optimal scheduling and DVFS for a real-time single core system with arbitrary arrival times and deadlines, and present an algorithm called YDS. The base for YDS is the observation that the dynamic power consumption depends on the clock frequency, the voltage (which depends on the clock frequency), and YDS assumes that the capacitance of the processor is constant. Kwon and Kim [18] extend the YDS algorithm such that it takes variable switched capacitances into account. They use a substitution of variables to translate their problem to a problem that can be solved by YDS. We use the idea of a substitution of variables in our paper to translate the multicore problem to a single core problem.

Irani et al. [19] show that, when static power is present, the YDS algorithm is still effective. Although increasing the clock frequency decreases the static energy consumption for the period when the tasks are executed, it also creates additional idle periods during which the processor again consumes the same amount of static energy. This explains why DVFS remains an effective technique as long as the dynamic power is not negligible. Le Sueur and Heiser [20] confirm this observation experimentally. We use the observation by Irani et al. [19] to derive an algorithm that takes static energy into account.

We rewrite our problem to a single core problem with a fixed order for the tasks, which makes the problem easier to solve, i.e., we can use a specialized algorithm that takes this ordering into account and has a lower time complexity than YDS. Instead of using YDS, we use an algorithm that is based on the algorithm by Huang and Wang [9] (with a quadratic time complexity) and we extend this algorithm such that it takes static power into account.

To the best of our knowledge, there are no papers that present algorithms that derive the optimal clock frequencies (local or global DVFS) for a real-time system with arbitrary arrival times, deadlines and work.

III. Model

In this paper we research the theoretical effects of DVFS on a multicore system that executes a nontrivial real-time application, and follow the commonly accepted modeling assumptions from the literature. Section III-A considers the power model of a global DVFS processor. Afterward, in Section III-B, we discuss how to subdivide a multicore schedule into so-called pieces.

A. System model

The power consumption of a multicore processor consists of the dynamic power consumption and the static power consumption. We assume that dynamic power is only used when a core is used for calculation [10], [17], i.e., when a core is *active*. Because of this, the dynamic power consumption of the system depends on the number of active cores. We use the dynamic power model from Yang et al. [17] to express the power consumption $p^D = c_1 f^\alpha$ of a single core, where f denotes the clock frequency, and c_1 and α are technology dependent constants; usually $\alpha \approx 3$. This model implicitly assumes that the voltage is linearly related to the clock frequency, a common assumption in the power management literature (see the survey articles by Chen and Kuo [11], and Zhuravlev et al. [12]).

The static power of a processor is often approximated by a linear function of the voltage [21]. Because we assume that the voltage is linearly related to the clock frequency, we model the static power as $p^S(f) = c_2 + c_3 f$.

The total power of the multicore processor as a function of the clock frequency, called the *power function*, is given by

$$p_m(f) = m c_1 f^\alpha + c_2 + c_3 f,$$

where m is the number of active cores. The power function is convex, a property that is often used when calculating the optimal clock frequencies [8], [9]. This power model is approximate, which makes it possible to study the effect of parallelism using closed form formulas [10]. Such models are very common in the literature [8]–[13], [16].

For now, we assume that the clock frequency for the multicore system can be changed at any time, and we make the common assumption that continuous clock frequencies are available [8], [13], [16], [17]. The function that maps time to a clock frequency, called the *clock frequency function*, is denoted by $\varphi : \mathbb{R}^+ \to \mathbb{R}^+$.

The total energy consumption is obtained by integrating power over time, i.e., when exactly m cores are active during the time interval $[t_1, t_2]$ the energy consumption for this time interval is

$$E = \int_{t_1}^{t_2} p_m(\varphi(\tau)) d\tau.$$

Note that the clock frequency does not influence the energy that is due to the term c_3, and for ease of notation we exclude c_3 from out equations in the remainder of the paper.

Below a certain clock frequency that is called the *critical clock frequency*, the static power becomes dominant. clock frequencies below f^{crit} are inefficient under circumstances that are discussed in Section V.

Although the speed of the processor itself scales linearly with the clock frequency, this is not always true for memory. Cho and Melhem [10] use a model similar to ours and considered the influence of operations of which the speed does not depend on the clock frequency (like memory access). They show that a model that takes such operations into account can be transformed to a model where the speed scales linearly with the clock frequency. Hence, we may assume that the speed scales linearly with the clock frequency and refer interested readers to the paper by Cho and Melhem [10].

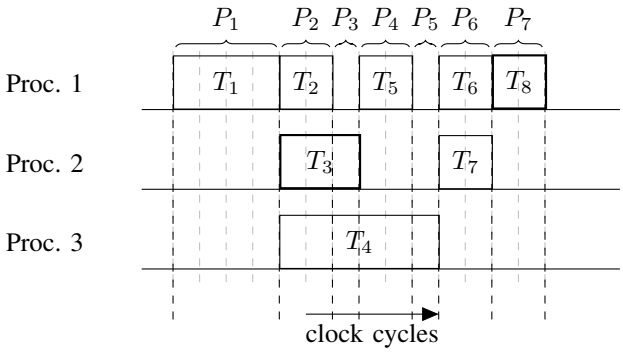

Figure 1: A feasible schedule for Example 1

Due to the convexity of the power function, the clock frequency is rarely changed in the optimal solution. Yao et al. [8] observed that the number of clock frequency changes of the optimal solution grows slowly with the number of tasks. This means that our optimization techniques keeps the number of clock frequency changes (and the overhead) low. According to the recent article by Park et al. [21], the time and energy overhead of DVFS is comparable with context switching overhead. The transition delay overhead is at most $62.68\mu s$ [21] on an Intel Core2 Duo E6850. We assume that the tasks are, in this respect, large enough, and because of this we can neglect the overhead of changing clock frequencies.

B. Pieces

An application consists of multiple tasks. Each task corresponds to a certain amount of *work*, measured in clock cycles. A task can have precedence constraints with other tasks, an arrival time and a deadline. As in the article by Li [13], we assume that the cores communicate through highly efficient communication mechanisms (e.g., shared memory). Since we have assumed that a schedule is given, we do not formalize the definitions of tasks. Instead, we informally start with an example of an application.

Example 1. *Consider an application that consists of* $N = 8$ *tasks with precedence constraints. The amount of work of the tasks* $T_1, \ldots T_8$ *is given by* 4, 2, 3, 6, 2, 2, 2 *and* 2 *respectively. Tasks* T_3 *and* T_8 *have the deadlines* 30 *and* 150 *respectively, tasks* T_3, T_4 *and* T_8 *have arrival times* 19, 5 *and* 140 *respectively, the other tasks have no deadline and are available from the beginning.*

In the context of this example, the exact precedence constraints are not relevant, only the fact that they create "gaps" in the schedule. A possible feasible schedule for the application is depicted in Figure 1, in which the tasks with a deadline are highlighted.

We assume without loss of generality that the given schedule does not contain any period where *all* cores are idle. Since only static power is consumed during these processor-wide idle periods, our assumption does not change the optimal solution.

The following lemma states that the average speed is optimal if the real-time constraints are met, and the number of active cores remains constant.

Lemma 1. *Given a work interval $[a, b]$ during which exactly m cores are active, and an execution time t for this work, then using the clock frequency $\frac{b-a}{t}$ is optimal for this interval.*

Proof: The infinite version of Jensen's inequality states:

$$p_m\left(\frac{1}{t_2-t_1}\int_{t_1}^{t_2}\varphi(\tau)d\tau\right) \leq \frac{1}{t_2-t_1}\int_{t_1}^{t_2} p_m(\varphi(\tau))d\tau.$$

Multiplying this equation by $t_2 - t_1$ directly leads to the result of the lemma. ∎

Irani et al. [19] used Jensen's inequality similarly to prove a single core version of this lemma.

When using this lemma, t has to be chosen such that the real-time constraints are met. Furthermore, the choice of t may influence the energy consumption of the entire schedule, not only of the interval $[a, b]$.

Note, that only at the start or completion of the execution of a task, the number of active cores can change. Furthermore, timing constraints such as arrival times or deadlines are also related to these start and completion times. Therefore, we are only interested in work intervals (a, b) where no tasks start or complete their execution.

The following corollary shows that it is optimal to use a single constant clock frequency for these intervals.

Corollary 1. *Given work interval (a, b) during which no task starts or ends. Then there is an optimal clock frequency function φ that is constant during the execution of the work in $[a, b]$.*

Proof: This is a direct consequence of Lemma 1. ∎

Because the optimal clock frequency is constant on the interval $[a, b]$ as specified in Corollary 1, we subdivide the schedule into such intervals. We choose these intervals such that they are as large as possible and call these intervals *pieces*.

Definition 1 (Piece)**.** *A piece is a maximal interval $[a, b]$ such that no task starts or finishes in (a, b).*

A given schedule uniquely subdivides into K pieces, and let the k-th piece be denoted by P_k. For ease of notation, we use the index set $\mathcal{K} = \{1, \ldots, K\}$.

Let m_k denote the number of active cores during piece P_k; this number of active cores during piece P_k cannot change, because no task starts or ends its execution during this piece. Furthermore, let the number of clock cycles for which piece P_k has to be executed be denoted by w_k. Hence $w_k m_k$ work is executed for this piece. The total work W can be obtained by summing the work of all tasks, but can now also be expressed in terms of pieces by $W = \sum_{k=1}^{K} w_k m_k$.

Let the begin time of piece P_k be denoted by t_k^b and its completion time be denoted by t_k^c. The actual values for t_k^b and t_k^c depend on the clock frequencies of the pieces $P_1, \ldots, P_k$, which is discussed later. For each piece P_k we can assign an arrival time a_k, which is the latest arrival time among all tasks that start at the beginning of this piece. This makes sure that the piece is not started before the arrival time of any task in this piece. Similarly, for piece P_k a deadline, denoted by d_k, can be derived; this is the earliest deadline among all tasks that are completed at the end of this piece.

Example 2 (Continued from Example 1)**.** *Figure 1 also shows the subdivision of the schedule from Example 1 into seven pieces ($P_1, \ldots, P_7$). A new piece starts whenever a task starts or finishes its execution. The number of clock cycles the pieces are active (w_k) is given by $w_1 = 4$, $w_2 = 2$, $w_3 = 1$, $w_4 = 2$, $w_5 = 1$, $w_6 = 2$ and $w_7 = 2$. The number of active cores is given by $m_1 = 1$, $m_2 = 3$, $m_3 = 2$, $m_4 = 2$, $m_5 = 1$, $m_6 = 2$ and $m_7 = 1$. Piece P_3 has deadline $d_3 = 30$ (deadline of task T_3) and piece P_7 has deadline $d_7 = 150$ (deadline of task T_8). The arrival time of piece P_2 is $a_2 = 19 = \max\{19, 5\}$. For piece P_7, the arrival time is $a_7 = 140$.*

During the execution of piece P_k the power function p_{m_k} is used, since during the entire execution of piece P_k exactly m_k cores are active. To determine the energy consumption, we consider the static and dynamic power separately.

The application begins at some given time t^B, and the power consumption of the processor is accounted for until some time t^C. This means that the static energy consumption is $c_2(t^C - t^B)$. For the begin time, we take t_1^b, while for the end time we assume that either $t^C = d_K$ or $t^C = t_K^c$; both situations are discussed when we present a solution to the problem.

Based on Corollary 1, we assign a constant clock frequency f_k to each piece P_k. For a given frequency assignment f_k to piece P_k ($k \in \{1, \ldots, K\}$), the dynamic energy consumption of piece P_k is the power ($m_k c_1 f_k^\alpha$) times the duration of piece P_k at the chosen clock frequency $\left(\frac{w_k}{f_k}\right)$ giving $m_k c_1 f_k^{\alpha-1} w_k$. To obtain the total dynamic energy consumption we have to sum this over all pieces.

The total energy consumption can now be expressed in terms of pieces and consists of the static and dynamic energy.

$$E = c_2(t^C - t^B) + \sum_{k=1}^{K} m_k c_1 f_k^{\alpha-1} w_k.$$

IV. Optimization model

Based on the discussion in the previous section, the problem considered in this paper reduces to energy minimization, under constraints such as ordering constraints (piece P_k is executed before piece P_{k+1}), arrival times and deadlines. More precisely, we get the following mathematical optimization problem.

Optimization Problem 1.

$$\min_{\substack{f_1,\ldots,f_K \\ t_1^b,\ldots,t_N^b}} \; c_2(t^C - t^B) + \sum_{k=1}^{K} m_k c_1 f_k^{\alpha-1} w_k \tag{1}$$

$$\text{s.t.} \quad t_k^b + \frac{w_k}{f_k} \leq d_k, \qquad \text{for all } k \in \mathcal{K}, \tag{2}$$

$$t_k^b \geq a_k, \qquad \text{for all } k \in \mathcal{K}, \tag{3}$$

$$t_k^b + \frac{w_k}{f_k} \leq t_{k+1}^b, \qquad \text{for all } k \in \mathcal{K}, \tag{4}$$

$$f_k \geq 0, \qquad \text{for all } k \in \mathcal{K}, \tag{5}$$

$$t_K^b + \frac{w_K}{f_K} \leq t^C. \tag{6}$$

The energy consumption is given as a cost function (1), which has to be minimized. The constraint (2) enforces that all pieces meet their deadline, (3) ensures that pieces do not begin before their arrival time, (4) enforces that piece P_k is finished before piece P_{k+1} starts, while (5) guarantees causality. The last constraint (6) makes sure that the application is not finished after the time for which the static energy consumption is accounted for.

For this problem, standard approaches from the literature cannot be used, due to the multicore aspect with weights m_k in the cost function that result from the number of active cores. We rewrite this problem to an (intuitively) easier to analyze problem. For this, we substitute the variables for clock frequencies and work. This substitution cancels out some terms, and the values m_k—making the problem a multicore problem—disappear from the equations. The idea behind the substitution is based on the ratio $\sqrt[\alpha]{m_a/m_b}$ between optimal clock frequencies of a multicore processor, described in several papers [15], [17].

The substitution of variables is as follows.

$$\hat{f}_k = f_k \sqrt[\alpha]{m_k}, \tag{7}$$

$$\hat{w}_k = w_k \sqrt[\alpha]{m_k}. \tag{8}$$

Substitution into the cost function (energy) gives:

$$E = c_2(t^C - t^B) + \sum_{k=1}^{K} c_1 \hat{f}_k^{\alpha-1} \hat{w}_k,$$

while substitution into the deadline constraint gives:

$$t_k^b + \frac{w_k}{f_k} = t_k^b + \frac{\hat{w}_k}{\hat{f}_k} \leq d_k.$$

This leads to the following optimization problem:

Optimization Problem 2.

$$\min_{\substack{f_1,\ldots,f_K \\ t_1^b,\ldots,t_N^b}} c_2(t^C - t^B) + \sum_{k=1}^{K} c_1 \hat{f}_k^{\alpha-1} \hat{w}_k$$

$$\begin{aligned}
\textit{s.t.} \quad & t_k^b + \frac{\hat{w}_k}{\hat{f}_k} \leq d_k, && \textit{for all } k \in \mathcal{K}, \\
& t_k^b \geq a_k, && \textit{for all } k \in \mathcal{K}, \\
& t_k^b + \frac{\hat{w}_k}{\hat{f}_k} \leq t_{k+1}^b, && \textit{for all } k \in \mathcal{K}, \\
& \hat{f}_k \geq 0, && \textit{for all } k \in \mathcal{K}, \\
& t_K^b + \frac{\hat{w}_K}{\hat{f}_K} \leq t^C.
\end{aligned}$$

Problem 2 is of interest because it has the same form as a well-known single core problem (real-time system with agreeable deadlines) from the literature [9], [22]. In this single core problem, the variable $\hat{w}_k$ is the work of task $\hat{T}_k$, while $\hat{f}_k$ is the optimal clock frequency of task $\hat{T}_k$. Furthermore, deadlines and arrival times are again given by a_k and d_k and the execution order is predetermined. As this relation is used throughout this paper, we give a formal definition.

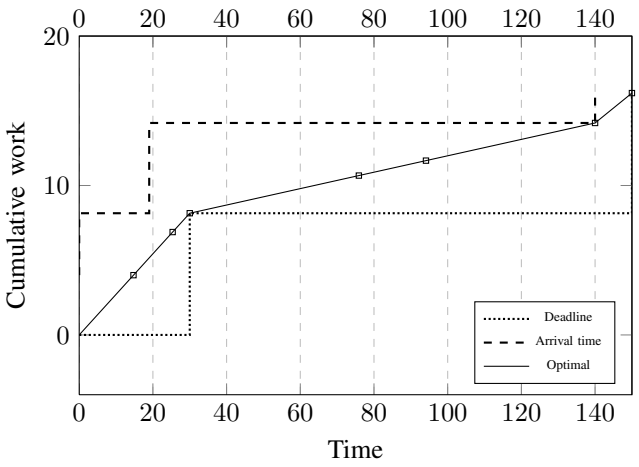

Figure 2: Optimal clock frequencies for the equivalent single core problem

Definition 2 (Equivalent single core problem)**.** *The multicore global DVFS problem with K pieces $(P_1, \ldots, P_K)$ is equivalent to the single core problem with K tasks $(\hat{T}_1, \ldots, \hat{T}_K)$, where each piece P_k is represented by a task $\hat{T}_k$ with work $\hat{w}_k = w_k \sqrt[\alpha]{m_k}$, clock frequency $\hat{f}_k = f_k \sqrt[\alpha]{m_k}$, and the same completion times and begin times.*

Based on this, Problem 1, can be transformed to the equivalent single core problem, this problem is then solved and the resulting solution is transformed back to obtain the optimal clock frequencies and begin times for the pieces. In the following, we give an example of this important transformation and solution technique.

Example 3 (Continued from Example 2)**.** *For illustration purposes, we use the power function $p_m(f) = mf^3$. The number of clock cycles $w_1 = 4$, $w_2 = 2$, $w_3 = 1$, $w_4 = 2$, $w_5 = 1$, $w_6 = 2$ and $w_7 = 2$, together with the parallelism described by $m_1 = 1$, $m_2 = 3$, $m_3 = 2$, $m_4 = 2$, $m_5 = 1$, $m_6 = 2$ and $m_7 = 1$ are transformed to $\hat{w}_1 = 4$, $\hat{w}_2 = 2\sqrt[3]{3}$, $\hat{w}_3 = \sqrt[3]{2}$, $\hat{w}_4 = 2\sqrt[3]{2}$, $\hat{w}_5 = 1$, $\hat{w}_6 = 2\sqrt[3]{2}$ and $\hat{w}_7 = 2$. Hence, the global DVFS problem with 7 pieces is now transformed to the equivalent single core problem with 7 tasks.*

For now, we can use any convex problem solver (e.g., CVX [23]) to solve this problem, which gives the solution: $\hat{f}_1 = \hat{f}_2 = \hat{f}_3 = 0.2715$, $\hat{f}_4 = \hat{f}_5 = \hat{f}_6 = 0.0549$ and $\hat{f}_7 = 0.2$. The next section provides an efficient algorithm to this problem, meaning that a general convex problem solver is not required anymore. Figure 2 shows the relation between the variables of the equivalent single core problem. The graph "Optimal" shows the cumulative workload that has been executed at a given time. The slope of this graph is the optimal clock frequency. The completion times of the tasks are indicated using squares. This graph must stay above the graph "Deadline", otherwise a deadline is missed. For example, at time 30, exactly $4 + 2\sqrt[3]{3} + \sqrt[3]{2} \approx 8.1444$ work must have been done. Similarly, the graph "Optimal" has to stay below the graph "Arrival time", which depicts the arrival times of the equivalent single

core problem. A well-known property of the solution of the single core problem is that—due to convexity of the power function p_1—the number of clock frequency changes is kept to a minimum. The solution $\hat{f}_1, \ldots, \hat{f}_K$, as shown in Figure 2, meets this property: if other clock frequencies would be used they are either too high, too low, or would imply unnecessary changes of the clock frequency.

The optimal clock frequencies for the original multicore global DVFS problem are obtained by transforming the optimal solution for the equivalent single core problem $\hat{f}_1, \ldots, \hat{f}_K$ back to $f_1, \ldots, f_K$. After this transformation, the optimal clock frequencies for the original global DVFS problem are given by $f_1 = 0.2715$, $f_2 = 0.1882$, $f_3 = 0.2155$, $f_4 = 0.0436$, $f_5 = 0.0549$, $f_6 = 0.0436$ and $f_7 = 0.2000$. During the execution of task T_4 the clock frequency changes four times.

In the next section, we show how to calculate the optimal clock frequencies using an efficient algorithm.

V. Optimal solution

In the previous section, we reduced the global DVFS problem to an equivalent single core problem with static power, but we did not provide a solution to this problem yet. While we can use any convex optimization tool to solve our problem, there are a few reasons to develop a tailored method. Firstly, such an approach is specific to our problem, and should therefore be more efficient. Secondly, the approach, presented in the following provides insights that can be the base for online algorithms or algorithms that use slack in an optimal way.

With respect to static energy, we consider two subproblems. Section V-A considers the case where the static energy consumption has to be taken into account for a given fixed period of time ($t^C = d_K$), hence the solution does not influence the static energy consumption. We refine this result in Section V-B, where we extend the results such that it can be used when static energy has to be accounted for only until the last task has finished its execution ($t^C = t^c_K$).

A. Fixed static energy

In this subsection we assume that the processor is active until the deadline of the last task $\hat{T}_K$. This means that the static energy consumption can no longer be influenced by the selected clock frequencies, i.e., we can assume $c_2 = 0$ [19].

For readability, we refer to the workload after transformation as tasks in contrast to pieces, since then the terminology in this section matches that of the literature on single core DVFS. Recall that task T_n is the n-th task in the multicore problem, while task $\hat{T}_k$ is the k-th task in the equivalent single core problem and corresponds to piece P_k in the multicore problem.

In the single core problem that results from the aforementioned reduction, the order of the tasks is given. To solve the single core problem, we adopt the "RecursiveSmoothing" algorithm of Huang and Wang [9], resulting in the function optfreq (see Algorithm 1). For correctness and optimality of this algorithm, we refer to the article of Huang and Wang [9].

The idea behind Algorithm 1 is as follows. Unnecessary clock frequency fluctuations have to be eliminated in the optimal solution, because otherwise the clock frequencies of consecutive tasks can be replaced by a common clock frequency leading to a decrease of the energy consumption (see Lemma 1). This means that the clock frequency is only changed when a task $\hat{T}_k$ arrives or when a task meets its deadline.

Algorithm 1 Optimal clock frequencies

$(\hat{f}_x, \ldots, \hat{f}_z)$ = **Function** optfreq (x,z,t^B,t^C)
  $F = \dfrac{\sum_{i=x}^{z} \hat{w}_i}{t^C - t^B}$

  $\mathrm{y} := \underset{j \in \{x,\ldots,z\}}{\operatorname{argmax}} \max \Big(t^B + \dfrac{\sum_{i=x}^{j} \hat{w}_i}{F} - d_j,\ a_j - t^B - \dfrac{\sum_{i=x}^{j} \hat{w}_i}{F} \Big)$

  if $t^B + \dfrac{\sum_{i=x}^{y} \hat{w}_i}{F} - d_y > 0$ **then** {y misses its deadline}
    $(\hat{f}_x, \ldots, \hat{f}_y)$ = optfreq (x,y,t^B, d_y)
    $(\hat{f}_{y+1}, \ldots, \hat{f}_z)$ = optfreq ($y+1$,z,d_y, t^C)
  else if $a_y - t^B - \dfrac{\sum_{i=x}^{y} \hat{w}_i}{F} > 0$ **then** {y violates arrival}
    $(\hat{f}_x, \ldots, \hat{f}_{y-1})$ = optfreq (x,$y-1$,t^B, a_y)
    $(\hat{f}_y, \ldots, \hat{f}_z)$ = optfreq (y,z,a_y, t^C)
  else {no task is infeasible}
    $(\hat{f}_x, \ldots, \hat{f}_z) = (F, \ldots, F)$
  end if
  return $(\hat{f}_x, \ldots, \hat{f}_z)$

This idea is implemented by the algorithm as follows. First a candidate solution with a constant clock frequency F is determined for the complete interval, hence F is chosen such that all tasks are executed between t^B and t^C. However, some tasks can miss their deadline or are required to begin too early. To avoid unnecessary clock frequency fluctuations, the task with the greatest deadline/arrival time violation is determined, this task is denoted by $\hat{T}_y$. The algorithm enforces the begin or completion time for this task such that it does not violate a constraint anymore. This divides the set of tasks into two parts, namely the tasks scheduled before the critical task $\hat{T}_y$ and the tasks scheduled after it. The algorithm is then recursively applied to both groups of tasks and the optimal solution follows.

B. Variable static energy

In this section, we assume that the processor is switched off after the last task is completed, i.e., we choose $t^C = t^c_K$. This means that it may pay off to increase the clock frequency for the last tasks, such that the processor can be turned off and the static energy consumption decreases.

Before we solve this problem, we first restrict our attention to a much simpler problem, namely the single core problem with $a_k = 0$ for all $k \in \{1, \ldots, K\}$ (i.e., arrival times are not taken into account). In this relaxed problem, it can be assumed without loss of generality that a task $\hat{T}_{k+1}$ starts immediately when task $\hat{T}_k$ is finished, since that minimizes the static energy consumption. Then the problem becomes as follows.

Algorithm 2 Solution to Problem 3

$(\hat{f}_x, \ldots, \hat{f}_z)$ = **Function** optfreq2(ℓ,K,t^B,t^C)
  $j = \ell$
  while $j \leq K$ **do**
    $h = \max\left(\arg\max_{k\in\{j,\ldots,N\}} \sum_{i=j}^{k} \frac{w_i}{d_k - d_{j-1}}\right)$
    $\hat{f}_j, \ldots, \hat{f}_h = \max\left(f^{\text{crit}}, \sum_{i=j}^{h} \frac{w_i}{d_h - d_{j-1}}\right)$
    $j = h + 1$
  end while

Optimization Problem 3.

$$\min_{\substack{f_1,\ldots,f_K \\ t_1^b,\ldots,t_N^b}} c_2(t^C - t^B) + \sum_{k=1}^{K} c_1 \hat{f}_k^{\alpha-1} \hat{w}_k$$

$$\text{s.t.} \quad \sum_{n=1}^{k} \frac{\hat{w}_k}{\hat{f}_k} \leq d_k, \qquad \text{for all } k \in \mathcal{K},$$
$$\hat{f}_k \geq 0, \qquad \text{for all } k \in \mathcal{K},$$
$$t_K^b + \frac{\hat{w}_K}{\hat{f}_K} \leq t^C.$$

In the optimal solution for this problem there is only one idle period during which the processor is turned off, namely after the last task of the application. Increasing the clock frequency also increases the length of period during which the processor is off, and with it the static energy consumption decreases. As a consequence, no clock frequency $f_k < f^{\text{crit}}$ should be used, since increasing f_k decreases the energy consumption.

Summarizing, for the optimal solution, fluctuations of the clock frequency have to be avoided, and no clock frequencies below f^{crit} are used. The optimal solution to this problem can be determined using Algorithm 2 (i.e., by calling $\text{optfreq2}(1, K, t^B, t^C)$). This algorithm chooses the lowest clock frequencies, which leads to a schedule respecting the deadlines of the tasks, while avoiding unnecessary clock frequency changes.

Theorem 1. *Algorithm 2 gives the optimal solution to Problem 3.*

Proof: We show that the solution $\hat{f}_1, \ldots, \hat{f}_K$ (we denote the respective clock frequency function by $\hat{\varphi}(\tau)$), that is produced by Algorithm 2 is optimal. The optimal solution is unique since the cost function is a strictly convex function which is minimized on a closed convex set.

Assume the theorem is false and the unique optimal solution is given by $\bar{f}_1, \ldots, \bar{f}_K$ (we denote the respective clock frequency function by $\bar{\varphi}(\tau)$). Since using any clock frequency $\bar{f}_k < f^{\text{crit}}$ is not optimal, we assume that $\bar{f}_k \geq f^{\text{crit}}$.

Now, consider the smallest m such that $\hat{f}_m \neq \bar{f}_m$. The two possible scenarios are considered separately:

(i) $\bar{f}_m > \hat{f}_m$:
In this case we consider the smallest $n > m$, such that $\bar{f}_n < \hat{f}_n$. Such a value does exist, since otherwise the optimal solution requires more energy than the solution found by Algorithm 2, which is a contradiction. It holds that $\bar{f}_m > \hat{f}_m > \hat{f}_n > \bar{f}_n$, since the sequence $\hat{f}_1, \ldots, \hat{f}_K$ produced by Algorithm 2 is non-increasing.
For all tasks $\hat{T}_m, \ldots, \hat{T}_{n-1}$, we have $\bar{t}_k^c < \hat{t}_k^c \leq d_k$. We now take a value $t > 0$, such that $\bar{f}_m t \leq w_m$, and $\bar{f}_n t \leq w_n$, and $t < \max_{k\in\{m,\ldots,n-1\}} d_k - \bar{t}_k^c$.
We consider two small portions of work $w'_m = \bar{f}_m t$ from task $\hat{T}_m$, and $w'_n = \bar{f}_n t$ from task $\hat{T}_n$. For w'_m and w'_n, we change the clock frequencies to $f = \frac{1}{2}\bar{f}_m + \frac{1}{2}\bar{f}_n$.
The total execution time of these two portions of work remains $2t$, while the energy consumption becomes.

$$\begin{aligned} p_1(f)2t &= p_1\left(\frac{1}{2}\bar{f}_m + \frac{1}{2}\bar{f}_n\right) \\ &< \frac{1}{2}p_1(\bar{f}_m)2t + \frac{1}{2}\bar{p}_1(\bar{f}_n)2t \\ &= p_1(\bar{f}_m)t + p_1(\bar{f}_n)t. \end{aligned}$$

Here, the strict inequality is due to the strict convexity of the single core power function p_1.
Note that we have chosen the portions of work and the clock frequencies such that no deadline is violated, while the energy consumption of the optimal solution decreases. This contradicts the assumption that the solution is optimal.

(ii) $\bar{f}_m < \hat{f}_m$:
Note, that the solution from Algorithm 2 is constant for tasks $\hat{T}_m, \ldots, \hat{T}_n$ (for some $n \geq m$), where $\hat{t}_n^c = d_n$. Hence, there must be some time $\bar{t}_m^b < t \leq d_n$ such that: $\int_{\bar{t}_m^b}^{t} \hat{\varphi}(\tau)d\tau = \int_{\bar{t}_m^b}^{t} \bar{\varphi}(\tau)d\tau$, otherwise task $\hat{T}_n$ misses its deadline in the optimal solution $\bar{f}$.
But this means that the optimal solution can be improved by using the constant clock frequency $\hat{f}_m$ on the interval $[\bar{t}_m^b, t]$, which contradicts the assumption that the solution is optimal.

Both cases contradict the assumption that Algorithm 2 is not optimal, which proves the theorem. ∎

In the following, we combine Algorithm 1 and Algorithm 2 to solve the original problem *with* arrival times. In this optimal solution there is some task $\hat{T}_\ell$ which is the last task that finishes exactly on the arrival time of the next task, i.e. $t_\ell^c = a_{\ell+1}$. When no such task exists, we take $\ell = 0$.

Clearly, the processor is active from the start of task $\hat{T}_1$ until the arrival of task $\hat{T}_{\ell+1}$. The static energy consumption during this period is constant. Hence, the results from Section V-A can be used; the optimal clock frequencies for tasks $\hat{T}_1, \ldots, \hat{T}_\ell$ can be determined using optfreq (1,ℓ,a_1,$a_{\ell+1}$).

For tasks $\hat{T}_{\ell+1}, \ldots, \hat{T}_K$ it holds by definition that $t_i^c > a_{i+1}$. Hence, when calculating the optimal solution, the arrival times do not have to be considered. This means that we can use Algorithm 2 to find the optimal clock frequencies.

Now it has become straightforward to calculate the optimal clock frequencies. For a given ℓ, Algorithm 1 and Algorithm 2 can be used for tasks $\hat{T}_1, \ldots, \hat{T}_\ell$ and tasks $\hat{T}_{\ell+1}, \ldots, \hat{T}_K$ respectively. The value ℓ is determined by iterating over all possible values of ℓ, namely $0, \ldots, K-1$, and choosing the value that gives a feasible solution with the lowest cost.

VI. Conclusions

The energy consumption can be reduced by using higher clock frequencies when fewer cores are active. The superlinear relation between clock frequency and power imposes a bound on the amount of increase of the clock frequency.

We have translated the multicore global DVFS problem to an equivalent single core problem. For this, we subdivide a given schedule into so-called *pieces*. The workload of a piece is multiplied by $\sqrt[\alpha]{m}$, where m is the number of active cores of the piece. After this transformation, all references to the amount of parallelism disappear from the problem. We can consider these transformed pieces as tasks in an equivalent single core problem. This is one of our major contributions, since it enables the use of single core DVFS techniques for multicore global DVFS systems.

All processors consume static power. For this, we considered two different scenarios: (i) static power is consumed until the deadline of the last task, (ii) static power is consumed until the last task is finished. For single core systems, the first problem was solved in the literature, while the second problem was solved using an algorithm that we introduced. This algorithm is a further contribution to the theory of single core DVFS.

Future Work

In future work we like to address the problem where the workload is *not* known before the tasks are executed. Zitterell and Scholl [24] solved this problem for the single core case. We want to derive a similar result for global DVFS, by using our transformation of the multicore problem to a single core problem.

References

[1] S. Irani and K. R. Pruhs, "Algorithmic problems in power management," *SIGACT News*, vol. 36, no. 2, pp. 63–76, jun 2005. [Online]. Available: http://doi.acm.org/10.1145/1067309.1067324

[2] K. Pruhs, "Green computing algorithmics," in *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, oct. 2011, pp. 3–4.

[3] 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*, vol. 54, no. 8, pp. 1282–1294, 2011.

[4] P. Chaparro, J. González, G. Magklis, C. Qiong, and A. González, "Understanding the thermal implications of multi-core architectures," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 18, no. 8, pp. 1055–1065, aug. 2007.

[5] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd, "Power7: IBM's next-generation server processor," *Micro, IEEE*, vol. 30, no. 2, pp. 7–15, march-april 2010.

[6] A. Kandhalu, J. Kim, K. Lakshmanan, and R. R. Rajkumar, "Energy-aware partitioned fixed-priority scheduling for chip multi-processors," in *17th International Conference on Embedded and Real-Time Computing Systems and Applications*, vol. 1. Los Alamitos, CA, USA: IEEE Computer Society, 2011, pp. 93–102.

[7] D. Zhang, D. Guo, F. Chen, F. Wu, T. Wu, T. Cao, and S. Jin, "Tl-plane-based multi-core energy-efficient real-time scheduling algorithm for sporadic tasks," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, pp. 47:1–47:20, jan 2012. [Online]. Available: http://doi.acm.org/10.1145/2086696.2086726

[8] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proceedings of IEEE 36th Annual Foundations of Computer Science*, 1995, pp. 374–382.

[9] W. Huang and Y. Wang, "An optimal speed control scheme supported by media servers for low-power multimedia applications," *Multimedia Systems*, vol. 15, no. 2, pp. 113–124, 2009.

[10] S. Cho and R. G. Melhem, "On the interplay of parallelization, program performance, and energy consumption," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 3, pp. 342–353, mar 2010. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2009.41

[11] J.-J. Chen and C.-F. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms," in *Embedded and Real-Time Computing Systems and Applications, 2007. RTCSA 2007. 13th IEEE International Conference on*, 2007, pp. 28–38.

[12] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto, "Survey of energy-cognizant scheduling techniques," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 7, pp. 1447–1464, 2013.

[13] K. Li, "Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers," *Computers, IEEE Transactions on*, vol. 61, no. 12, pp. 1668–1681, 2012.

[14] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz, "Bounding energy consumption in large-scale MPI programs," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing - SC '07*, 2007, pp. 1–9.

[15] M. E. T. Gerards, J. L. Hurink, and J. Kuper, "On the interplay between global DVFS and scheduling tasks with precedence constraints," 2013, (under review).

[16] K. Pruhs, R. van Stee, and P. Uthaisombut, "Speed scaling of tasks with precedence constraints," in *Approximation and Online Algorithms*, T. Erlebach and G. Persinao, Eds. Springer Berlin / Heidelberg, 2006, vol. 3879, ch. Lecture Notes in Computer Science, pp. 307–319, 10.1007/11671411_24. [Online]. Available: http://dx.doi.org/10.1007/11671411_24

[17] C.-Y. Yang, J.-J. Chen, and T.-W. Kuo, "An approximation algorithm for energy-efficient scheduling on a chip multiprocessor," in *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, ser. DATE '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 468–473. [Online]. Available: http://dx.doi.org/10.1109/DATE.2005.51

[18] W.-C. Kwon and T. Kim, "Optimal voltage allocation techniques for dynamically variable voltage processors," *ACM Transactions on Embedded Computing Systems*, vol. 4, no. 1, pp. 211–230, 2005.

[19] S. Irani, S. Shukla, and R. Gupta, "Algorithms for power savings," *ACM Trans. Algorithms*, vol. 3, no. 4, pp. 41:1–41:23, nov 2007. [Online]. Available: http://doi.acm.org/10.1145/1290672.1290678

[20] L. Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the 2010 International Conference on Power Aware Computing and Systems, HotPower'10*. USENIX Association, 2010, pp. 1–8.

[21] S. Park, J. Park, D. Shin, Y. Wang, Q. Xie, M. Pedram, and N. Chang, "Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, no. 5, pp. 695–708, 2013.

[22] E. Bampis, C. Dürr, F. Kacem, and I. Milis, "Speed scaling with power down scheduling for agreeable deadlines," *Sustainable Computing: Informatics and Systems*, vol. 2, no. 4, pp. 184–189, 2012.

[23] M. Grant and S. Boyd, "Graph implementations for nonsmooth convex programs," in *Recent Advances in Learning and Control*, V. Blondel, S. Boyd, and H. Kimura, Eds. Springer-Verlag Limited, 2008, vol. 371, ch. Lecture Notes in Control and Information Sciences, pp. 95–110.

[24] T. Zitterell and C. Scholl, "A probabilistic and energy-efficient scheduling approach for online application in real-time systems," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: ACM, 2010, pp. 42–47. [Online]. Available: http://doi.acm.org/10.1145/1837274.1837287