Modeling Processor Idle Times in MPSoC Platforms to Enable Integrated DPM, DVFS, and Task Scheduling Subject to a Hard Deadline[*]

Amirhossein Esmaili
Department of Electrical Engineering
University of Southern California
Los Angeles, California
esmailid@usc.edu

Mahdi Nazemi
Department of Electrical Engineering
University of Southern California
Los Angeles, California
mnazemi@usc.edu

Massoud Pedram
Department of Electrical Engineering
University of Southern California
Los Angeles, California
pedram@usc.edu

ABSTRACT

Energy efficiency is one of the most critical design criteria for modern embedded systems such as multiprocessor system-on-chips (MPSoCs). Dynamic voltage and frequency scaling (DVFS) and dynamic power management (DPM) are two major techniques for reducing energy consumption in such embedded systems. Furthermore, MPSoCs are becoming more popular for many real-time applications. One of the challenges of integrating DPM with DVFS and task scheduling of real-time applications on MPSoCs is the modeling of idle intervals on these platforms. In this paper, we present a novel approach for modeling idle intervals in MPSoC platforms which leads to a mixed integer linear programming (MILP) formulation integrating DPM, DVFS, and task scheduling of periodic task graphs subject to a hard deadline. We also present a heuristic approach for solving the MILP and compare its results with those obtained from solving the MILP.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; **Real-time systems**; • **Theory of computation** → **Scheduling algorithms**; **Integer programming**; • **General and reference** → *Design*; *Performance*;

KEYWORDS

Task Scheduling, Energy Optimization, DVFS, DPM, Real-time MPSoCs

1 INTRODUCTION

Energy consumption is one of the most important design criteria of computing devices, ranging from portable embedded systems to servers in data centers. Furthermore, with growing demand for high performance in embedded systems, architectures such as multiprocessor system-on-chip (MPSoC) are becoming more popular for many real-time applications. In order to reduce energy consumption in such embedded systems, two main techniques are used, namely, dynamic voltage and frequency scaling (DVFS) and dynamic power management (DPM). In DVFS, operating voltage and clock frequency of processors are adjusted based on workload characteristics. With DPM, processors are switched to a low power state (sleep mode) when they are not used for execution of any tasks (idle time/interval). This leads to the reduction of static power consumption. However, switching to a sleep mode has non-negligible time and energy overhead, and it only causes energy savings when the idle time of a processor is longer than a threshold called *break-even* time [1].

There have been many research studies regarding reducing the energy consumption using DVFS and/or DPM. A major portion of these studies only considers DVFS for the energy optimization on single and multiprocessor platforms [2–4]. Ref [5] has focused on DPM and has proposed an energy-efficient scheduling relying on minimizing the number of processor switching and maximizing the usage of energy-efficient cores in heterogeneous platforms. Some other papers have integrated scheduling of tasks with DVFS and then, at the final phase, have applied DPM wherever it was possible [6]. However, with the increase in static power portion of the total power consumption of systems [7], both DPM and DVFS should be integrated with scheduling of tasks for the sake of energy optimization. Reference [8] has combined DPM and DVFS for minimizing energy consumption of a uniprocessor platform performing periodic hard real-time tasks with precedence constraints. A major challenge of integrating DPM with the scheduling of tasks in a multiprocessor platform is formulating idle intervals and their associated energy consumption in the total energy consumption of the these platforms. The authors in [9] have developed an energy-minimization formulation for a multiprocessor system considering both DVFS and DPM and solves it via mixed integer linear programming (MILP). However, one major assumption in their formulation is that the processor assignment for the tasks to be scheduled is known in advance. Furthermore, they only consider inter-task DVFS, i.e., the frequency of the processor stays constant for the entire duration of the execution of a task. However, when there is a set of discrete frequencies available for task execution (as that is the case in [9] and also our work as we will see in Section 2.4), allowing intra-task DVFS and the usage of a combination of discrete frequencies for execution of tasks can result in more energy savings [1].

In this paper, by proposing a method for modeling idle intervals in a multiprocessor system, we present an energy optimization MILP formulation integrating both DVFS and DPM with scheduling of real-time tasks with precedence and time constraints. By solving the MILP, for each task, we obtain the optimum processor assignment, execution start time, and the distribution of its workload among available frequencies of the processor. To the best of our knowledge, this is the first work that integrates both DVFS and DPM with scheduling of real-time periodic dependent tasks

[*]Codes and scripts for this work are available from https://github.com/Amirhossein-Esmaili/Energy_Aware_Task_Scheduling_in_MPSoCs

in a formulation that provides optimum values for all the aforementioned results simultaneously in a multiprocessor platform. We also present a heuristic approach for solving the model and compare its results with those obtained from solving the MILP. The rest of the paper is organized as follows: Section 2 explains the models used for the problem formulation and presents the formal problem statement. Section 3 presents the proposed method and MILP formulation. Section 4 provides the results. Finally, Section 5 concludes the paper and discusses future work.

2 MODELS AND PROBLEM DEFINITION

2.1 Voltage and Frequency Change Overhead

The frequency change for modern processors takes around tens of microseconds depending on the amount and (up or down) direction of the frequency change. According to [10], the frequency downscaling for Intel Core2 Duo E6850 processor takes approximately between 10 to 60 microseconds depending on the amount of the frequency change. In contrast, the transition to and from sleep modes of modern processors usually takes in the order of a few milliseconds. Therefore, for our modeling, we ignore the latency overhead of switching frequencies compared to that of transition to and from sleep modes of a processor. The energy overhead associated with frequency change is also small and neglected in our modeling.

2.2 Task Model

Tasks to be scheduled are modeled as a task graph which itself is a directed acyclic graph (DAG) represented by $G(V, E, T_d)$, in which V denotes the set of tasks (we have a total of n tasks), E denotes data dependencies among tasks, and T_d denotes the period of the task graph (i.e., tasks in the task graph are repeated after T_d). Each task graph should be scheduled before the arrival of the next one (i.e., T_d acts as a hard deadline for scheduling of tasks). In this paper, the workload of each task is represented by the total number of processor cycles required to perform that task completely. For task u ($u = 1, 2, ..., n$), this workload is represented by W_u.

2.3 Energy Model

For modeling the processor power consumption during executing a task with frequency f, similar to [4], the following model would be exploited:

$$P = af^{\alpha} + bf + c, \tag{1}$$

in which af^{α} represents dynamic power portion, and $bf + c$ represents static power portion of total processor power consumption. α indicates the technology-dependent dynamic power exponent; usually ≈ 3. a is a constant that depends on the average switched capacitance and the average activity factor. Therefore, energy consumption in one clock cycle, when executing a task with frequency f, is obtained via the following formulation:

$$E_{cycle} = af^{\alpha-1} + b + \frac{c}{f}. \tag{2}$$

For modeling the processor energy consumption during an idle time, E_{idle} function is used according to the formulation presented in (3). Here, for the illustration purposes, we only use one sleep mode for switching to and waking up from, and power consumption during this sleep mode is considered to be zero (It is straightforward to extend the work to support multiple sleep modes each associated with a different non-zero power consumption):

$$E_{idle}(I) = \begin{cases} c \times I & 0 \le I < T_{be} \\ E_{sw} & T_{be} \le I < T_d \\ 0 & I = T_d \end{cases}, \tag{3}$$

where I represents the idle time, c is the frequency-independent component of power consumption (by setting f to zero in (1)), and E_{sw} is the switching energy overhead for both switching to the sleep mode and waking up from it. T_{be} represents break-even time and is obtained as follows:

$$T_{be} = \max(T_{sw}, \frac{E_{sw}}{c}), \tag{4}$$

where $\frac{E_{sw}}{c}$ represents the minimum amount the idle time should be so that switching to the sleep mode and waking up from it causes energy savings, and T_{sw} is the physical time needed for both switching to the sleep mode and waking up from it. T_{be} is the maximum of these two values. Furthermore, the third term in (3) conveys the fact that if no task is assigned to a processor or equivalently $I = T_d$, that processor is not used for scheduling of tasks and thus does not contribute to the total energy consumption at all. Therefore, our model explores the possibility of scheduling the task graph on a subset of K available processors if it results in energy savings.

2.4 Problem Statement

Using the combination of DVFS and DPM, where each of these techniques can be done for each processor independently, we are looking for energy-optimized scheduling of the task graph represented by $G(V, E, T_d)$ on a platform comprising of K homogeneous processors subject to a hard deadline. Each processor supports a set of m distinct frequencies: $\{f_1, f_2, ..., f_m\}$. We are considering a non-preemptive scheduling method. Therefore, when the execution of a task starts on each of the processors, it continues until the task completion without any interruption. Consequently, for each task, we are looking for optimum values for: processor assignment for the task, task execution start time, and distribution of the total number of required processor cycles for the complete execution of the task among m available frequencies.

3 PROPOSED METHOD

3.1 Constraints of the Proposed Scheduling Model

In this section, we formulate constraints of the proposed scheduling model. Duration of task u ($u = 1, 2, ..., n$) is formulated as follows:

$$Dur_u = \sum_{i=1}^{m} \frac{N_{u,i}}{f_i}, \tag{5}$$

where $N_{u,i}$ indicates number of processor cycles performed at f_i ($i = 1, 2, ..., m$) for the execution of task u. Therefore:

$$\sum_{i=1}^{m} N_{u,i} = W_u, \quad N_{u,i} \ge 0. \tag{6}$$

According to (2) and (5), energy consumption during the execution of task u can be formulated as follows:

$$E_{task}(u) = \sum_{i=1}^{m}(N_{u,i}.(af_i^{\alpha-1} + b + \frac{c}{f_i})). \tag{7}$$

To ensure each task finishes its execution before T_d, for $u = 1, 2, ..., n$, we have:

$$S_u + Dur_u \leq T_d, \quad S_u \geq 0, \tag{8}$$

where S_u represents start time of the execution of task u. Furthermore, the precedence constraint is formulated as follows:

$$S_u + Dur_u \leq S_v, \quad \forall e(u, v) \in E. \tag{9}$$

Here, we do not consider any inter-task communication cost associated with $e(u, v)$ for sending output data of task u to input data of task v (The model can be easily extended to incorporate this cost).

For processor assignment for task u to processor k, $k = 1, 2, ..., K$, we introduce the decision variable of $P_{k,u}$ which is defined as follows:

$$P_{k,u} = \begin{cases} 1 & \text{if task u is assigned to processor k} \\ 0 & \text{otherwise} \end{cases}. \tag{10}$$

Therefore, we have the following constraint:

$$\sum_{k=1}^{K} P_{k,u} = 1, \quad \text{for } u = 1, 2, ..., n. \tag{11}$$

One other important constraint that needs to be satisfied is that the execution of tasks assigned to the same processor shall not overlap each other (non-preemptive scheduling). For this, we define an auxiliary decision variable called $O_{k,u,v}$ representing ordering of tasks. For $k = 1, 2, ..., K$; $u = 1, 2, ..., n$; $v = 1, 2, ..., n, v \neq u$; we define:

$$O_{k,u,v} = \begin{cases} 1 & \text{if task u is scheduled immediately} \\ & \text{ before task v on processor k} \\ & \\ 0 & \text{otherwise} \end{cases}. \tag{12}$$

In addition, if task v is the first task assigned to processor k, we define $O_{k,0,v}$ to be 1 (and is 0 otherwise). On the other hand, if task u is the last task assigned to processor k, we define $O_{k,u,n+1}$ to be 1 (and is 0 otherwise). Furthermore, if there is no task assigned to processor k, we define $O_{k,0,n+1}$ to be 1 (and is 0 otherwise). Accordingly, using (12) and the definitions provided for $O_{k,0,v}$, $O_{k,u,n+1}$ and $O_{k,0,n+1}$, we have the following constraints for $k = 1, 2, ..., K$:

$$\sum_{\substack{v=1 \\ v\neq u}}^{n+1} O_{k,u,v} = P_{k,u}, \quad \text{for } u = 0, 1, ..., n \tag{13}$$

$$\sum_{\substack{u=0 \\ u\neq v}}^{n} O_{k,u,v} = P_{k,v}, \quad \text{for } v = 1, 2, ..., n+1. \tag{14}$$

According to (13), if task u is assigned to processor k ($P_{k,u} = 1$), either there is one and only one task scheduled immediately after task u on processor k or task u is the last task assigned to processor k. Similarly, according to (14), if task v is assigned to processor k ($P_{k,v} = 1$), either there is one and only one task scheduled immediately before task v on processor k or task v is the first task assigned to processor k. In both (13) and (14), $P_{k,0}$ and $P_{k,n+1}$ are defined as 1 for all $k = 1, 2, ..., K$.

For non-preemptive scheduling we should have:

$$\sum_{k=1}^{K}\sum_{u=1}^{n}\left((S_u + Dur_u).O_{k,u,v}\right) \leq S_v, \quad \text{for } v = 1, 2, ..., n, v \neq u, \tag{15}$$

which can be formulated as the following linear constraint:

$$\begin{aligned} S_u + Dur_u - (1 - O_{k,u,v}) \times T_d &\leq S_v, \\ \text{for } u &= 1, 2, ..., n, \\ \text{for } v &= 1, 2, ..., n, v \neq u, \\ \text{for } k &= 1, 2, ..., K. \end{aligned} \tag{16}$$

3.2 Modeling Idle Intervals

Using $O_{k,u,v}$ variables introduced in Section 3.1, we can conveniently model idle intervals in an MPSoC platform. Specifically, for each task v ($v = 1, 2, ..., n$), we formulate the amount of the idle time before servicing task v on the processor to which task v is assigned. When task v is not the first task scheduled on the processor to which it is assigned, the idle time before servicing task v can be written as follows:

$$\begin{aligned} I_v = (1 - \sum_{k=1}^{K} O_{k,0,v}) \\ \times (S_v - \sum_{k=1}^{K}\sum_{\substack{u=1 \\ u\neq v}}^{n}((S_u + Dur_u).O_{k,u,v})). \end{aligned} \tag{17}$$

If task v is the first task scheduled on any of K processors, the first term in multiplication in (17) causes I_v to be zero. In that case, idle time before servicing task v on the processor k to which the task is assigned is obtained using the following:

$$\begin{aligned} I'_k = (T_d - \sum_{u=1}^{n}((S_u + Dur_u).O_{k,u,n+1})) \\ + \sum_{v=1}^{n}(O_{k,0,v} \times S_v). \end{aligned} \tag{18}$$

In (18), the second term in summation represents the idle time on processor k before servicing its first assigned task in the current period. On the other hand, The first term in the summation in (18) represents the idle time on that processor after servicing its last assigned task in the previous period. This interval should also be taken into account when calculating the amount of idle time before servicing first task scheduled on processor k. If there is no task assigned to processor k at all, (18) would give the value of T_d for I'_k.

3.3 Objective Function

Subject to constraints formulated so far, we are trying to minimize the following objective function which represents the total energy consumption:

$$\sum_{u=1}^{n} E_{task}(u) + \sum_{v=1}^{n} E_{idle}(I_v) + \sum_{k=1}^{K} E_{idle}(I'_k). \tag{19}$$

The objective function of (19), alongside the formulated constraints, forms a mixed integer programming over the positive real variables of S_u and $N_{u,i}$; and the Boolean decision variables of $P_{k,u}$ and $O_{k,u,v}$. The number of these variables in our problem are n, nm, nk, and $(n+1)^2k - nk$, respectively.

However, due to formulations presented for idle time intervals in (17) and (18), and the concave piece-wise behavior of E_{idle} function in (3) in a minimization problem, it is a non-linear non-convex programming ($E_{task(u)}$ in (19) is linear with respect to positive real variables of $N_{u,i}$ and this term does not contribute to the nonlinearity of the problem).

For linearizing (17) and (18), we use the lemma mentioned in [9], where this lemma is stated as follows: *Given constants s_1 and s_2, if P_1 and P_2 are two constraint spaces where P_1 is $\{[t,b,x] \mid t = bx,\ -s1 \le x \le s2,\ b \in \{0,1\}\}$, and P_2 is $\{[t,b,x] \mid -bs_1 \le t \le bs_2,\ t+bs_1-x-s_1 \le 0,\ t-bs_2-x+s_2 \ge 0,\ b \in \{0,1\}\}$, then, P_1 and P_2 are equivalent.* Proof of this lemma is given in [9]. With this lemma, we can substitute multiplication of a Boolean decision variable and a bounded real variable, with a newly introduced bounded real variable and three added linear constraints indicated in $P2$). Using this lemma multiple times, we can reach linear representations for idle time interval formulations in (17) and (18) at the end.

Furthermore, $E_{idle}(I_v)$ in (19) can be written as follows:

$$E_{idle}(I_v) = S_v.(E_{sw}) + (1 - S_v).(c \times I_v), \tag{20}$$

where S_v is a Boolean decision variable which is 1 when $T_{be} \le I_v < T_d$ and is 0 otherwise ($I < T_{be}$). Therefore, this decision variable represents switching and whether we put the processor during I_v in the sleep mode or not. Since I_v represents the amount of idle time before servicing task v on the processor to which task v is assigned when task v is not the first task on that processor, I_v can never be T_d. Therefore, we do not need to formulate the third term of (3) in (20). Corresponding constraint for S_v ($v = 1, 2, ..., n$), is written as follows:

$$\frac{I_v - T_{be}}{T_d} \le S_v \le \frac{I_v}{T_{be}}, \quad S_v \in \{0,1\}. \tag{21}$$

For $E_{idle}(I'_k)$, a similar formulation like (20) can be used except that we need another Boolean decision variable called U_k which represents whether we assign any task to processor k or not. When U_k is 0, it means processor k is not used at all for scheduling the task graph and thus does not contribute to the energy consumption in (19). Therefore, U_k is 1 when we assign one or more tasks to processor k ($I'_k < T_d$) and is 0 otherwise ($I'_k = T_d$, or equivalently: $T_d \le I'_k \le T_d$). Accordingly, $E_{idle}(I'_k)$ in (19) can be written as follows:

$$E_{idle}(I'_k) = U_k.[S'_k.(E_{sw}) + (1 - S'_k).(c \times I'_k)], \tag{22}$$

where S'_k represents whether we switch the processor during I'_k to the sleep mode or not (similar to S_v). The usage of U_k in (22) allows formulating the third term of (3). Corresponding constraints for S'_k and U_k ($k = 1, 2, ..., K$) are written as follows:

$$\frac{I'_k - T_{be}}{T_d} \le S'_k \le \frac{I'_k}{T_{be}}, \quad S'_k \in \{0,1\}, \tag{23}$$

$$\frac{I'_k - T_d}{Td} \le U_k \le \frac{I'_k}{T_d}, \quad U_k \in \{0,1\}. \tag{24}$$

In order to linearize (20) and (22), we again use the aforementioned lemma. However, for (22), where we have a multiplication of two Boolean decision variables, we also need the following lemma: *If P_1 and P_2 are two constraint spaces where P_1 is $\{[z,x,y] \mid z = xy,\ x \in \{0,1\},\ y \in \{0,1\}\}$, and P_2 is $\{[z,x,y] \mid z \le x,\ z \le y,\ x+y-z \le 1,\ x \in \{0,1\},\ y \in \{0,1\}\}$, then, P_1 and P_2 are equivalent.* Using these lemmas and methods for linearizing the objective function of (19), the energy-optimized scheduling problem expressed in Section 2.4 is modeled as an MILP formulation.

4 RESULTS

4.1 Experiment Setup

In order to solve the formulated MILP, we use IBM ILOG CPLEX Optimization Studio [11]. The platform on which simulations are performed is a computer with a 3.2 GHz Intel Core i7-8700 Processor and 16 GB RAM. Using [9] for obtaining energy model parameters, the frequency-independent component of processor power consumption, which is represented by c in (1), is obtained as $276\,mW$. Each processor can operate independently of other processors at either $f_1 = 1.01\,GHz$, $f_2 = 1.26\,GHz$, $f_3 = 1.53\,GHz$, $f_4 = 1.81\,GHz$, $f_5 = 2.1\,GHz$. For these frequencies, frequency-dependent component of processor power consumption, which is represented by $af^{\alpha} + bf$ in (1), is $430.9\,mW$, $556.8\,mW$, $710.7\,mW$, $896.5\,mW$, and $1118.2\,mW$, respectively. Using curve fitting, we obtain $a = 23.8729$, $b = 401.6654$, and $\alpha = 3.2941$ in (1). E_{sw} and T_{sw} are set as $385\,\mu J$ and $5\,ms$. Here, We consider a architecture with 4 processors. Simulations are performed on 8 task graphs randomly generated using TGFF [12], which is a randomized task graph generator widely used in the literature to evaluate the performance of scheduling algorithms. Detailed information for each task graph is presented in Table 1. For studied task graphs, the average workload of each task is set to 2×10^6 cycles (around $1\,ms$ execution time under maximum frequency). The maximum in-degree and out-degree for each node is set to 2 and 3, respectively. The number of tasks in studied random task graphs ranges from 7 to 28.

To evaluate the advantage of our modeling of idle intervals in multiprocessor systems, we consider two cases: 1) A baseline case which uses only the first term of (19) as the objective function alongside with constraints of (5) to (14), and (16). In other words, in this baseline case, we do not use any idle time-related terms in the objective function or constraints. 2) Using (19) as the objective function alongside all constraints and linearization techniques mentioned in Section 3 (this case is our proposed method). In the baseline case, switching to the sleep mode during an idle time is done, if possible, after the scheduling is finished (i.e., DPM is not integrated with DVFS and scheduling in the baseline case). Therefore, the baseline case is an integrated Scheduling and Clock-and-voltage scaling followed by mode Transition algorithm (iSC+T). The second case, which is our proposed method, is referred to as an integrated Scheduling, Clock-and-voltage scaling, and mode Transition algorithm (iSCT).

4.2 Effect of Modeling Idle Intervals

According to Table 1, including the energy consumption of modeled idle intervals in the objective function causes an average energy saving of 15.34% (up to 25.21%) for iSCT versus iSC+T. To better

Table 1: Task Graphs Characteristics and Corresponding Energy Consumption Values Obtained from iSCT versus iSC+T

Task Graph	No. of Tasks	Total Workload of Tasks in Processor Cycles ($\times 10^6$)	Td (ms)	Total Energy Consumption (mJ)		iSCT versus iSC+T Energy Saving (%)
				iSC+T	iSCT	
TGFF1	7	15.89	8	12.67	10.45	17.52
TGFF2	11	18.69	12	13.60	12.06	11.32
TGFF3	14	34.39	10	25.99	22.70	12.66
TGFF4	15	31.89	12	28.08	21.00	25.21
TGFF5	16	34.88	12	27.35	23.02	15.83
TGFF6	18	33.46	14	25.38	21.98	13.40
TGFF7	22	44.94	22	33.74	29.39	12.89
TGFF8	28	56.81	18	42.95	37.00	13.85

Table 2: Idle Intervals Characteristics and No. of Used Processors for iSCT versus iSC+T

Task Graph	No. of Idle Intervals		Total Idle time (ms)		No. of Used Processors	
	iSC+T	iSCT	iSC+T	iSCT	iSC+T	iSCT
TGFF1	5	3	21.62	24.00	3	1
TGFF2	4	3	35.79	36.00	4	1
TGFF3	6	3	17.53	21.22	4	2
TGFF4	10	3	27.16	29.00	4	2
TGFF5	7	3	25.21	29.00	4	2
TGFF6	8	3	34.13	34.13	4	2
TGFF7	10	3	58.63	58.64	4	2
TGFF8	10	3	34.87	36.96	4	2

observe the contribution of modeling idle intervals in an MPSoC platform, for each scheduled task graph, the total number of idle intervals on all processors, and total time of these idle intervals are shown in Table 2 for both iSCT and iSC+T. Furthermore, for each scheduled task graph, the number of used processors for the scheduling of that task graph, out of maximum 4 processors, is also presented in Table 2 for both iSCT and iSC+T.

According to Table 2, while for all scheduled task graphs, the total time of idle intervals are higher or the same for iSCT compared to iSC+T, the number of idle intervals for iSCT are notably fewer than the number of idle intervals for iSC+T (on average fewer than half). Therefore, by including the energy consumption of modeled idle intervals in the objective function of (19), instead of having a number of distributed short idle intervals, we will have fewer merged longer idle intervals. This results in more opportunities for switching the processors to the sleep mode during idle intervals and thus more energy savings (as indicated in Table 1). In fact, for task graphs studied in this paper, the percentage of idle intervals that are longer than T_{be}, and thus we can switch the processor to the sleep mode, are 91.67 % and 32.31 % for iSCT and iSC+T, respectively.

On the other hand, as indicated in Table 2, iSCT explores the possibility of the usage of a subset of 4 processors if it results in energy savings. For discussed task graphs in this paper, iSCT always uses fewer than 4 processors for the scheduling. The unused processors do not contribute to the energy consumption at all (we do not need to switch them to the sleep mode and wake them up in every time period). This can be helpful in terms of energy efficiency, particularly when the energy overhead of switching processors is relatively high. Reference [9] cannot take advantage of this since the processor assignment for tasks to be scheduled is assumed to be known in advance and it is not integrated in their MILP formulation.

On the platform we performed simulations, iSCT and iSC+T approaches on average generated results for studied task graphs in less than 69 and 1 minutes, respectively. Since we are considering scheduling of a periodic task graph, these simulations are done offline only once for one period of the task graph. The obtained scheduling can be programmed to a MPSoC for real-time scheduling of each arriving period of the task graph.

4.3 A Heuristic Approach to Solve the Model

Here, we propose a two-stage heuristic algorithm to solve the formulated model: 1) We first determine and fix the values of $O_{k,u,v}$ and $P_{k,u}$ variables using a polynomial-time list scheduling algorithm. 2) Using fixed $O_{k,u,v}$ and $P_{k,u}$ values, the number of variables in the original MILP problem reduces significantly. Also, (17) and (18) will become linear formulations in the first place, and we do not anymore need to use the first lemma presented in Section 3.3 multiple times to linearize them. This further reduces the number of variables of the MILP problem considerably (since each time using of this lemma adds one set of real variables, plus three sets of constraints). Then, the new formulated problem with considerably fewer number of variables, which is still an MILP due to the usage of (20) and (22) in the objective function of (19), will be solved to obtain values of S_u and $N_{u,i}$.

For the first stage, we use a variant of heterogeneous earliest finish time (HEFT) algorithm [13]. While this algorithm aims for heterogeneous platforms, it can be applied to a homogeneous platform similar to our paper as well. Basically, in this algorithm, tasks are ordered according to their *upward rank*, which is defined recursively for each task as follows:

$$rank_{up}(u) = Dur^*_u + \max_{v \in succ(u)} (rank_{up}(v)), \tag{25}$$

where here, Dur^*_u is the duration of the task u when all of its workload is executed using maximum available frequency, and $succ(u)$ is the set of immediate successors of task u. Ranks of the tasks are computed recursively starting from *exit tasks* of the task graph (exit tasks are the ones with out-degree of zero). The upward rank of exit tasks are equal to their corresponding Dur^* values. Basically, $rank_{up}(u)$ indicates the length of critical path from task u to exit tasks, including Dur^*_u itself.

After the calculation of ranks for all the tasks, a task list is generated by sorting the tasks in the decreasing order of their ranks. Tie-breaking is done randomly. Then, tasks are scheduled on processors based on the order of the task list. Each task can only be scheduled after a time called *ready_time* of that task, which indicates the time that the execution of all immediate predecessors of that task has completed. For each task, we look for the first idle interval on each processor after the task ready_time, with the amount of at least Dur^* of that task, and assign the task to the processor which gives us the earliest finish time. Since the task list sorted by the decreasing order of ranks gives a topological sorting of the DAG [13], when we choose a task for scheduling, its predecessors have already been scheduled. The time complexity of HEFT algorithm is $O(|E| \times K)$ where $|E|$ denotes the number of edges of the DAG and K denotes the number of processors [13].

Using HEFT in the first stage of our heuristic approach, we determine the processor assignment for each task ($P_{k,u}$), and ordering of tasks on each processor ($O_{k,u,v}$). Note that obtained start times

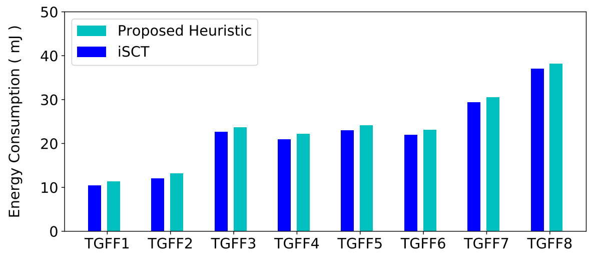

Figure 1: Energy Consumption obtained from the proposed heuristic approach and iSCT for different task graphs

for tasks after the first stage just show relative ordering of tasks on each processor. Also, we only used maximum frequency in the first stage. Next, in the second stage, we solve the newly derived MILP, which has been obtained after fixing $O_{k,u,v}$ and $P_{k,u}$ values in the first stage, and has considerably fewer number of variables compared to the original MILP. This gives us the values for S_u and $N_{u,i}$ variables. On average, on the platform we performed simulations, solving the newly derived MILP provided results for studied task graphs in less than 2 seconds, which is considerably less than the simulation time of solving the original MILP.

Fig. 1 shows a comparison between the energy consumption obtained from iSCT, and the energy consumption obtained from solving the problem using the proposed heuristic approach. According to Fig. 1, the heuristic method provides close estimates compared to the optimum solution. The values of energy consumption obtained from the heuristic approach are on average 5.66% higher than the optimum solution.

5 CONCLUSIONS AND FUTURE WORK

In this paper, by proposing a method for modeling idle intervals in multiprocessor systems, we presented an energy optimization MILP formulation integrating both DVFS and DPM with scheduling of real-time tasks with precedence and time constraints. By solving the MILP, for each task, we obtain the optimum processor assignment, execution start time, and the distribution of its workload among available frequencies of the processor. Results show the effectiveness of our modeling of idle intervals in MPSoCs in terms of energy efficiency. We also presented a heuristic approach for solving the MILP which provided close results compared to optimum results. It is worth mentioning that although our proposed model focuses on MPSoCs, it can also be applicable to servers in data centers by using proper energy model parameters of those platforms.

For future work, workload of tasks can be investigated to represent more than just the processor cycle count; e.g., the memory requirement, or the possibility of executing the entire or part of a task on GPUs can be modeled and investigated. Also, obtaining a variant of the proposed model for heterogeneous processors could be another potential future direction.

REFERENCES

[1] M. E. Gerards and J. Kuper. Optimal dpm and dvfs for frame-based real-time systems. *ACM Transactions on Architecture and Code Optimization (TACO)*, 9(4):41, 2013.

[2] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pages 225–232. IEEE, 2001.

[3] X. Huang, K. Li, and R. Li. A energy efficient scheduling base on dynamic voltage and frequency scaling for multi-core embedded real-time system. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 137–145. Springer, 2009.

[4] M. E. Gerards, J. L. Hurink, and J. Kuper. On the interplay between global dvfs and scheduling tasks with precedence constraints. *IEEE Transactions on Computers*, 64(6):1742–1754, 2015.

[5] T. Nakada, H. Yanagihashi, H. Nakamura, K. Imai, H. Ueki, T. Tsuchiya, and M. Hayashikoshi. Energy-aware task scheduling for near real-time periodic tasks on heterogeneous multicore processors. In *Very Large Scale Integration (VLSI-SoC), 2017 IFIP/IEEE International Conference on*, pages 1–6. IEEE, 2017.

[6] K. Srinivasan and K. S. Chatha. Integer linear programming and heuristic techniques for system-level low power scheduling on multiprocessor architectures under throughput constraints. *INTEGRATION, the VLSI journal*, 40(3):326–354, 2007.

[7] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo. Applying real-time interface and calculus for dynamic power management in hard real-time systems. *Real-Time Systems*, 47(2):163–193, 2011.

[8] P. Rong and M. Pedram. Power-aware scheduling and dynamic voltage setting for tasks running on a hard real-time system. In *Design Automation, 2006. Asia and South Pacific Conference on*, pages 6–pp. IEEE, 2006.

[9] G. Chen, K. Huang, and A. Knoll. Energy optimization for real-time multiprocessor system-on-chip with optimal dvfs and dpm combination. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(3s):111, 2014.

[10] S. Park, J. Park, D. Shin, Y. Wang, Q. Xie, M. Pedram, and N. Chang. Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(5):695–708, 2013.

[11] Ibm ilog cplex optimization studio, version 12.8. Available from: https://www.ibm.com/products/ilog-cplex-optimization-studio.

[12] R. P. Dick, D. L. Rhodes, and W. Wolf. Tgff: task graphs for free. In *Proceedings of the 6th international workshop on Hardware/software codesign*, pages 97–101. IEEE Computer Society, 1998.

[13] H. Topcuoglu, S. Hariri, and M.-y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.