A

Energy Optimization for Real-Time Multiprocessor System-on-Chip with Optimal DVFS and DPM Combination

GANG CHEN, Technical University Munich, Germany
Kai Huang, Technical University Munich, Germany
Alois Knoll, Technical University Munich, Germany

Energy optimization is a critical design concern for embedded systems. Combining DVFS+DPM is considered as one preferable technique to reduce energy consumption. There have been optimal DVFS+DPM algorithms for periodic independent tasks running on uni-processor in the literature. Optimal combination of DVFS and DPM for periodic dependent tasks on multi-core systems is however not yet reported. The challenge of this problem is that the idle intervals of cores are not easy to model. In this paper, a novel technique is proposed to directly model the idle intervals of individual cores such that both DVFS and DPM can be optimized at the same time. Based on this technique, the energy optimization problem is formulated by means of mixed integrated linear programming. We also present techniques to prune the exploration space of the formulation. Experimental results using real-world benchmarks demonstrate the effectiveness of our approach compared to existing approaches.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-based System**]: Real-time and embedded systems

General Terms: Design, Scheduling, Energy

Additional Key Words and Phrases: Energy Optimization, DVFS, DPM, Real-Time MPSoCs

ACM Reference Format:
Gang Chen, Kai Huang, Alois Knoll, 2013. Energy Optimization for Real-Time Multiprocessor System-on-Chip with Optimal DVFS and DPM Combination. *ACM Trans. Embedd. Comput. Syst.* XX, XX, Article A (June 2013), 21 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

With increasing requirement for high-performance, multi-core architectures such as MPSoCs (Multiprocessor System-om-Chip) are believed to be the major solution for future embedded systems, e.g., electronic vehicle [Lukasiewycz et al. 2012]. Chip makers have released several MPSoCs, e.g., ARM Cortex-A15 MPCore [ARM 2012], Intel Atom processors [Intel 2009], and Marvell ARMADA MV78460 multi-core processors [Marvell 2012]. Many real-time applications, especially streaming applications, can be executed on multiple processors simultaneously to achieve parallel processing. When real-time applications are executed on multi-core architectures, minimizing the energy consumption is one of the major design goals, because an energy-efficient design will increase the reliability and decrease the heat dissipation of the system.

This work has been partly funded by German BMBF projects ECU (grant number: 13N11936) and Car2X (grant number: 13N11933).
Author's addresses: G. Chen and K. Huang and A. Knoll, Technical University Munich, Boltzmannstrae 3, 85748 Garching, Germany.

DOI: http://dx.doi.org/10.1145/0000000.0000000

Power consumption of processors mainly comes from dynamic power consumption due to switching activity and static power consumption due to the leakage current [Jejurikar et al. 2004]. In micrometer CMOS technology, dynamic power dominates the power consumption of processors. As the number of processing cores on a chip increases (e.g., Intel has produced 48-core x86 Processor as Single-chip Cloud Computer [Intel 2012]), chip density increases, which leads to a dramatic increase of static power. According to the International Technology Roadmap for Semiconductors [ITRS 2011], leakage power increases its dominance of total power consumption as semiconductors progress toward 32nm. Thus, both dynamic power consumption and static power consumption need to be considered in overall energy optimization of the system.

To reduce dynamic power and static power consumption, two main mechanisms can be employed, i.e., Dynamic Voltage Frequency Scaling (DVFS) and Dynamic Power Management (DPM), respectively. DVFS reduces the dynamic power consumption by dynamically adjusting voltage and frequency of a processor [Jha 2001]. The disadvantage of this technique is the lack of means to reduce static power consumption. There are two types of DVFS polices, i.e., intra-task DVFS and inter-task DVFS. Intra-task DVFS allows the processor frequencies can be changed within single task, while inter-task DVFS only allows frequencies can be changed at inter-task boundaries. We consider inter-task DVFS in this paper. On the other hand, DPM explores idle intervals of a processor and switches the processor to a sleep mode with low static power consumption to reduce the static power [Jha 2001]. The limitations of DPM are that mode-switching in processor causes additional energy and latency penalty. Actually, it is worthwhile to switch the processor to sleep mode only when the idle interval is longer than a certain threshold called break-even time [Chen et al. 2013; Cheng and Goddard 2006].

In principle, DVFS and DPM counteract each other with respect to energy reduction and a trade-off between them plays a critical role in energy consumption reduction [Gerards and Kuper 2013]. Concerning DVFS, lower frequencies result in lower dynamic power consumption, which however prolong the task execution time and shorten idle intervals. Therefore, DVFS techniques in general reduce the opportunities of reducing static power. On the other hand, although running the system at higher frequencies can create longer sleep intervals and reduce more static power, DPM will cost more dynamic power and mode-switch overheads. In the last decade, the researchers used to believe that, between DVFS and DPM, DVFS should be exploited before DPM [Jha 2001; Srinivasan and Chatha 2007]. However, as transistor technology is shifting toward sub-micron domains (e.g. Intel has shift its manufacturing technologies into 22nm in 2011 [Intel 2011]), the static power increases exponentially and becomes comparable or even greater than dynamic power. According to [Huang et al. 2011], the static power accounts for as much as 50% percentage of the total power dissipation for high-end processors in 90 nm technologies. Thus, DPM technology becomes more and more important for energy-efficient design. Applying DVFS before DPM will result in a sub-optimal solution. A motivation example will further elaborate this issue in Section 4. Therefore, the motivation of our work is to globally integrate DVFS and DPM and find the best trade-off to reduce the total energy consumption of a system.

In this paper, we present a novel energy optimization technique which optimally integrates DVFS and DPM in real-time multi-core systems. We consider multi-core systems that DVFS and DPM can be applied for each core independently and each core operates at several discrete voltage and frequency levels. For a given set of applications represented as directed acyclic graphics and a mapping of the applications on the MPSoC, our approach can generate an optimal time-triggered non-preemptive schedule and an optimal frequency assignment for each task by which the total energy consumption of MPSoCs is minimized. Based on this optimal schedule, the tasks can be

scheduled with insertion of voltage-setting and mode-switching instructions. Energy-efficient code can be generated by integrating this optimal approach into compilers and real-time OSs. The contributions of our work can be summarized as:

— The challenge of globally integrating DVFS and DPM is that the idle interval of the processors cannot be modeled because the scheduling cannot be determined in the optimization stage. In contrast to the work in [Srinivasan and Chatha 2007], which firstly only integrates DVFS with scheduling and then separately applies DPM as a final stage to generate system level low power designs, we propose a key technique to directly model the idle interval of individual processor and integrate DVFS and DPM within scheduling.
— We develop an energy-minimization formulation that globally integrates DVFS and DPM and solve it by means of mixed integer linear programming. In contrast to the work in [Wang et al. 2011] based on genetic algorithm, our approach can guarantee to find the optimum.
— A refinement technique is presented to prune the design space of our formulation.
— We conduct simulations using real-life applications and demonstrate the effectiveness of our approach by extensive experiments, comparing with [Srinivasan and Chatha 2007].

The rest of the paper is organized as follows: Section 2 reviews related work in the literature. Section 3 presents basic models and the definition of studied problem. Section 4 presents the motivation example and Section 5 describes the proposed approach. Experimental evaluation is presented in Section 6 and Section 7 concludes the paper.

2. RELATED WORK

DVFS is one of most effective techniques for energy optimization and has been used for more than a decade. A lot of DVFS scheduling techniques for MPSoCs has been proposed in the literature. For the independent frame-based task set, Chen and Kuo [Chen and Kuo 2005] proposed approximation algorithm based on a Kuhn-Tucker optimality condition on homogeneous multiprocessor. Hung et al. [Hung et al. 2006] explored energy-efficient scheduling of periodic independent real-time tasks in a heterogeneous multiprocessor. Based on level-packing, Xu et al. [Xu et al. 2012] address energy minimization problem for parallel independent task systems with discrete operation modes and under timing constraints. For the dependent tasks on MPSoC, Zhang et al. [Zhang et al. 2002] proposed a two-phase framework that integrates task scheduling and voltage selection to minimize energy consumption of real-time dependent tasks on MPSoC. Based on list scheduling, Gruian et al. [Gruian and Kuchcinski 2001] proposed a DVS scheduling technology which scales down the delays of all tasks by the ratio of the timing constraint over the critical path length.

Combined DVS+DPM approach is considered as one preferable techniques for low power systems [Jha 2001]. However, only a few of literatures have considered the combination of DPM and DVS. For uniprocessors, Devadas and Aydins [Devadas and Aydin 2012] addressed energy minimization combining DVFS and DPM only for independent frame-based tasks, where DVFS is employed on uniprocessor and DPM is used for peripheral devices. Based on [Devadas and Aydin 2012], the state-of-the-art work in [Gerards and Kuper 2013] presented a schedule for independent frame-based tasks that globally minimizes the energy consumption. However, these approaches can not be applied to multi-core platform. In the context of MPSoCs, Bhatti et al. [Bhatti et al. 2011] proposed a machine-learning mechanism to adapt on runtime to the DVFS and DPM policy for independent tasks to achieve energy efficiency. For dependent tasks, Srinivasan et al. [Srinivasan and Chatha 2007] presented a mixed-integer linear programming(MILP) formulation, which integrates DVFS along with pipelining schedul-

ing, and applies DPM the the final design step. This work applies DVFS before DPM, which will result in suboptimal solution. Based on task pipelining, Wang et al. [Wang et al. 2011] proposed a two-phase approach to optimize the energy consumption. In the first phase, a periodic dependent task graph is transformed into a set of independent tasks by retiming technology. In the second phase, a scheduling algorithm based on genetic algorithm is proposed to optimize the energy consumption. However, this work can not find the exact optimal solution as it is based on genetic algorithm. Besides, the approaches in [Srinivasan and Chatha 2007; Wang et al. 2011] can only handle one task graph.

In contrast to prior work, we consider the problem for MPSoCs that globally integrates DVFS and DPM with scheduling, rather than applying DVS before DPM [Srinivasan and Chatha 2007], and generate an exact optimal scheduling for dependent tasks with minimizing the overall energy consumption.

3. MODELS AND PROBLEM DEFINITION

3.1. Hardware Model

In this paper, we consider a typical multi-core system [Wang et al. 2011; Wang et al. 2010] with local memories, as shown in Fig. 1. The multi-core system consists of M cores $P = \{p_1, p_2, \cdots, p_M\}$. Every core is connected via a high bandwidth shared bus. Bus arbiter implements a given bus protocol and assigns bus access rights to individual cores. In this paper, we adopt a time-triggered bus based on time-division multiple access (TDMA) protocol.

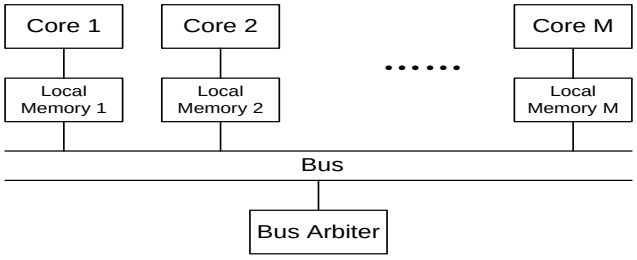


Fig. 1. Hardware model.

3.2. Task Model

We consider the functionality of the entire system as an *applications set* A, which consists of a set of independent periodic *applications*. An *application* $J \in A$ is modeled as a directed acyclic task graph $G(V, E, H)$, where vertexes V denote the set of tasks T to be executed, the edges E represent data dependencies between tasks and H denotes the period of the *application*. The deadline D of the *application* is equal to its period. We use w_{ij} to denote the worst case execution time (WCET) of task $T_i \in V$ under frequency f_j and $W_i = \{w_{i1}, w_{i2}, ..., w_{is}\}$ to denote the WCET profile of task T_i, where s is the total number of available frequency levels. In this paper, we abstract communication between two tasks mapped on different cores as special task, which mapped on bus and whose execution time can be modeled as its communication overhead. Thus, we can also integrate communication task into task graph.

Time-triggered scheduling can offer a fully deterministic real-time behavior for safety-related systems. Current practice in many safety-critical domains, such as electric vehicle [Lukasiewycz et al. 2012] and avionics systems [Lin et al. 2007], favors a

time-triggered approach [Baruah and Fohler 2011]. In this paper, we consider a periodic time-triggered non-preemptive scheduling policy. We use R to denote the set of the profiles for all tasks in *applications set* A. A task profile $r_i \in R$ is defined as a tuple $r_i =< W_i, s_i, h_i, d_i >$, where s_i, h_i, d_i are respectively the start time, period, and deadline of T_i. Note that the start time s_i is an unknown variable, which is determined by *scheduling* S. The tasks belonging to the same *application* share the same period and deadline.

3.3. Energy Model

In this paper, we assume cores in a MPSoC can support both DVFS and DPM. For DVFS, each core has s different voltage/frequency levels, which are denoted as $\{(V_1, f_1), (V_2, f_2), \cdots, (V_s, f_s)\}$. The voltage/frequency levels are sorted in ascent order. For each frequency level, there is a power consumption associated, and thus we have a set of power values $\{P_1, P_2, \cdots, P_s\}$ corresponding to voltage/frequency levels and $P_1 < P_2 < \cdots < P_s$. We adopt inter-task DVFS in this paper [Srinivasan and Chatha 2007; Wang et al. 2011; Zhong and Xu 2008], hence the frequency of the core stays constant for entire duration of a task's execution. Besides, We adopt the same assumption as [Li and Wu 2012; Singh et al. 2013; Srinivasan and Chatha 2007; Zhang et al. 2002] and assume that the energy consumption of the task is determined by the assigned frequency. Thus, the task has an uniform energy consumption during the entire execution time. Note that our approach can also be extended to the case that the energy consumption is associated with the tasks.

The analytical processor energy model in [Martin et al. 2002; Wang and Mishra 2010; Jejurikar et al. 2004] is adopted in this paper, whose accuracy has been verified by SPICE simulation. The dynamic power consumption of the core on one voltage/frequency level (V_{dd}, f) can be given by:

$$P_{dyn} = C_{eff} \cdot V_{dd}^2 \cdot f \tag{1}$$

where V_{dd} is the supply voltage, f is the operating frequency and C_{eff} the effective switching capacitance. The cycle length t_{cycle} is given by a modified alpha power model which is verified by SPICE simulation [Martin et al. 2002; Wang and Mishra 2010; Jejurikar et al. 2004].

$$t_{cycle} = \frac{L_d \cdot K_6}{(V_{dd} - V_{th})^\alpha} \tag{2}$$

where K_6 is technology constant and L_d is estimated by the average logic depth of all instructions critical path in the processor. The threshold voltage V_{th} is given below.

$$V_{th} = V_{th1} - K_1 \cdot V_{dd} - K_2 \cdot V_{bs} \tag{3}$$

where V_{th1}, K_1, K_2 are technology constants and V_{bs} is the body bias voltage.

The static power is mainly contributed by the subthreshold leakage current I_{subn}, the reverse bias junction current I_j and the number of devices in the circuit L_g. It can be presented by:

$$P_{sta} = L_g \cdot (V_{dd} \cdot I_{subn} + |V_{bs}| \cdot I_j) \tag{4}$$

where the reverse bias junction current I_j is approximated as a constant and the subthreshold leakage current I_{subn} can be determined as:

$$I_{subn} = K_3 \cdot e^{K_4 V_{dd}} \cdot e^{K_5 V_{bs}} \tag{5}$$

where K_3, K_4, K_5 are technology constants. To avoid junction leakage power overriding the gain in lowering I_{subn}, V_{bs} should be constrained between 0 and -1V. Thus,

the power consumption of the processor under each voltage/frequency (V_{dd}, f) can be computed as:

$$P = P_{dyn} + P_{sta} + P_{on} \tag{6}$$

where P_{on} is an inherent power needed for keeping the processor on, which related to idle power.

Considering the overhead of switching the processor between active mode and sleep mode, the processor break-even time T_{BET} indicates the minimum time length that the processor should stay at sleep mode. If the interval at which the processor can stay at sleep mode is smaller than T_{BET}, the mode-switch mode overheads are larger than the energy saving. Therefore, mode-switch is not worthy. The break-even time T_{BET} can be defined as follows:

$$T_{BET} = \max\left(t_{sw}, \frac{E_{sw} - P_{sleep} \cdot t_{sw}}{P_{idle} - P_{sleep}}\right) \tag{7}$$

Where t_{sw} and E_{sw} denote the total state transition time and energy overhead, respectively. P_{idle} and P_{sleep} respectively represent the idle power and sleep power and we have $P_{idle} > P_{sleep}$.

Given a time-triggered schedule S, the total energy consumption $E_t(S)$ in one hyper-period can be represented as follows:

$$E_t(S) = E_d(S) + E_i(S) + E_s(S) + E_{ov}(S) \tag{8}$$

Where $E_d(S)$ is the total energy consumption when the processor is executing tasks, $E_i(S)$ is the total energy consumption when the cores stay at idle mode, $E_s(S)$ is the total sleep consumption when the cores stay at sleep mode, and $E_{ov}(S)$ is the energy consumption due to the overhead of mode-switches.

The energy consumption of executing task T_i running at frequency f_j is:

$$E_d(T_i, f_j) = P_j \cdot w_{ij} \tag{9}$$

When the idle interval of the core is less than the break even time T_{BET}, i.e., $t_{idle} < T_{BET}$, the core should not enter sleep mode and should stay at idle mode with high energy consumption P_{idle}.

$$E_i = P_{idle} \cdot t_{idle} \tag{10}$$

The energy consumed in the sleep mode, E_s, is calculated by

$$E_s = P_{sleep} \cdot t_{sleep} \tag{11}$$

where the sleep interval t_{sleep} should be longer than the break even time T_{BET}, i.e, $t_{sleep} \geq T_{BET}$, and P_{sleep} is the power consumption when the core stays at the sleep mode.

3.4. Problem Statement

Given an *applications set* A with task profile R, a multi-core architecture with M cores $P = \{p_1, p_2, \cdots, p_M\}$, each core with s discrete voltage/frequency levels, and task mapping $\Pi\{T \rightarrow P\}$, our goal is to find a voltage assignment for each task and a time-triggered scheduling S so that the total energy $E_{total}(S)$ is minimized while the timing constraints of all applications are guaranteed.

4. MOTIVATION

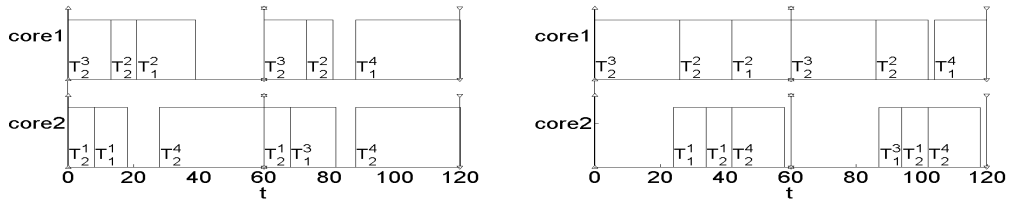
In this section, we present a motivation example to show that the strategy of applying DVFS before DPM cannot generate the optimal solution.

Table I. Task profiles and mapping.

	T_1^1	T_1^2	T_1^3	T_1^4	T_2^1	T_2^2	T_2^3	T_2^4
Mapping	p_2	p_1	p_2	p_1	p_2	p_1	p_1	p_2
WCET[ms]	5	9	7	16	4	8	13	16

Table II. Frequency Assignment and Overall Power Consumption.

	T_1^1	T_1^2	T_1^3	T_1^4	T_2^1	T_2^2	T_2^3	T_2^4	$P(W)$
SubOPT	L	L	L	L	L	H	H	L	0.82
OPT	L	L	H	H	L	L	L	H	0.74



(a) Applying DVFS before DPM. (b) Optimal DVFS and DPM combination.

Fig. 2. Schedules.

Consider two applications $J_1 = \{T_1^1, T_1^2, T_1^3, T_1^4\}$ and $J_2 = \{T_2^1, T_2^2, T_2^3, T_2^4\}$ with dependencies $\{T_1^1 \rightarrow T_1^2, T_1^1 \rightarrow T_1^3, T_1^2 \rightarrow T_1^4, T_1^3 \rightarrow T_1^4\}$ and $\{T_2^1 \rightarrow T_2^4, T_2^2 \rightarrow T_2^4, T_2^3 \rightarrow T_2^4\}$, respectively. For simplicity, communication overhead is not considered in this motivation example. The period of the application J_1 and J_2 are respectively 120ms and 60ms. The MPSoC hardware has a dual-core architecture $P = \{p_1, p_2\}$. Each core has two voltage/frequency levels, i.e., high level f_H and low level f_L. The normalized frequency is 1 at the high level and 0.5 at the low level. The dynamic power at the high level and low level are 0.4W and 0.1W, respectively. The static power at the high level and low level are 0.16W and 0.12W, respectively. P_{on} which is related to idle power is 0.15W. The time overhead and energy overhead of run-sleep mode switch are 25ms and 1mJ, respectively. The task profiles and mapping are listed in Tab. I.

We compare two different schemes, i.e., applying DVFS before DPM (SubOPT) and globally integrating DVFS and DPM (OPT). The frequency assignment and total power consumption P is listed in Tab. II and the schedules of these two schemes are shown in Fig. 2. OPT can achieve about 9.1% energy savings w.r.t SubOPT. In Fig. 2(a), all intervals are smaller than the break-even time and the system should turn on all the time. In SubOPT, DVFS assigns the frequency as low as possible, which increases the execution time of the task and shortens the idle time intervals of the system. In contrary, OPT can avoid this. In Fig. 2(b), we can see that, by increasing some task's frequencies, some idle intervals are extended large enough and core 2 can enter into sleep mode. Besides, by adjusting the frequency, some tasks in core 1 could run in lower frequency for further power savings.

5. PROPOSED APPROACH

This section presents our mixed integer linear programming (MILP) approach for integrating DVFS and DPM into task scheduling. We start with a MILP formulation that focuses only on the scheduling problem. Then, we introduce a key technique to model the idle interval of the cores and integrate DPM into the formulation. Based on the observation that the MILP formulation may suffer from the state explosion, we develop a refinement, the so-called execution windows analysis, to reduce the exploration space of the formulation.

5.1. Time-Triggered Task Scheduling

In this paper, we consider time-triggered non-preemptive schedule. For each task T_i with the profile $< W_i, s_i, h_i, d_i >$, the k-th instance of task T_i starts at $s_i + k \cdot h_i$. W_i contains the WCETs of the task T_i under different frequency settings. We use a set of binary variables c_{ij} to describe the frequency assignment of the task T_i: $c_{ij} = 1$ if the task T_i executes with frequency f_j and $c_{ij} = 0$ otherwise. In this case, the actual WCET of T_i can be obtained as $\sum_{j=1}^{s} c_{ij} w_{ij}$, where s is the total number of available different discrete frequency levels. As we adopt the inter-task DVFS, each task can be assigned only one frequency.

$$\sum_{k=1}^{s} c_{ik} = 1 \tag{12}$$

To formulate the scheduling problem by means of MILP, we have to cope with the task dependency, deadlines, and non-preemption. We present our formulation as follows.

Let ξ denote the overheads for dynamic frequency scaling and task switch. The data dependency $T_j \rightarrow T_i$ requires the start time of T_i to be no earlier than the finish time of T_j. Note that T_i or T_j can also be the communication task.

$$s_j + \sum_{k=1}^{s} c_{jk} w_{jk} + \xi \leq s_i \tag{13}$$

For deadline constraint, task T_i has to finish no later than its deadline:

$$s_i + \sum_{k=1}^{s} c_{ik} w_{ik} + \xi \leq d_i \tag{14}$$

The non-preemptive constraint requires that any two tasks mapped to the same core must not overlap in time, as well as the communication tasks in the bus. T_p^i denotes the i-th instance of task T_p. Let binary variable $z_{p\widetilde{p}}^{ij}$ denote the execution order of task T_p^i and $T_{\widetilde{p}}^j$: $z_{p\widetilde{p}}^{ij} = 1$ if T_p^i finishes before the start of $T_{\widetilde{p}}^j$, and 0 otherwise. $z_{p\widetilde{p}}^{ij}$ and $z_{\widetilde{p}p}^{ji}$ $(p \neq \widetilde{p})$ contradict each other, i.e., $z_{p\widetilde{p}}^{ij} + z_{\widetilde{p}p}^{ji} = 1$. H_r and $H_{p\widetilde{p}}$ denote the hyper-period of all tasks and the hyper-period of only task T_p and $T_{\widetilde{p}}$ (i.e., LCM of periods of T_p and $T_{\widetilde{p}}$), respectively. $TC(T_p)$ denotes the set of tasks that are mapped to the same core as T_p does. The non-preemption constraint can thereby be expressed as follows.
$\forall T_p, T_{\widetilde{p}} \in TC(T_p), i = 0, ..., (\frac{H_{p\widetilde{p}}}{h_p} - 1), j = 0, ..., (\frac{H_{p\widetilde{p}}}{h_{\widetilde{p}}} - 1)$:

$$i \cdot h_p + s_p + \sum_{k=1}^{s} c_{pk} w_{pk} - (1 - z_{p\widetilde{p}}^{ij}) H_r + \xi \leq j \cdot h_{\widetilde{p}} + s_{\widetilde{p}} \tag{15}$$

$$j \cdot h_{\widetilde{p}} + s_{\widetilde{p}} + \sum_{k=1}^{s} c_{\widetilde{p}k} w_{\widetilde{p}k} - z_{p\widetilde{p}}^{ij} H_r + \xi \leq i \cdot h_p + s_p \tag{16}$$

The constraints (15) and (16) ensure that either the instance of T_p runs strictly before the instance of $T_{\widetilde{p}}$, or vice verse. Noting that (15) and (16) can also be applied to non-preemptive constraint of communication tasks in the bus.

5.2. DPM Representation

5.2.1. Challenge. The challenge of globally integrating DVFS and DPM is that the idle interval of the cores is difficult to model, which is determined by the scheduling. How-

ever, the scheduling cannot be determined in the optimization stage. To illustrate this challenge, we give a simple example in Fig. 3. Task T_1, T_2, T_3, and $T4$ run at the same core with periods of h, h, $2h$, and h, respectively. We can make following observations.

— The idle interval of the task is determined by its closest task. For example, the idle interval I_1 of task T_1 is determined by the task T_3, while I_2 is determined by the task T_2. However, it is unknown which two task are the closest tasks in the time axis because the execution order is not known yet.
— The idle interval of the last task in one hy-period is determined by the first task in the following hy-period. For example, the idle interval of the last task T_4 is combined by I_3 and I_4. Thus, we cannot represent the idle intervals in one hy-period.

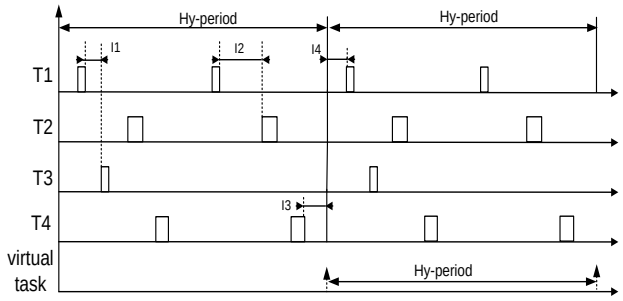


Fig. 3. Example.

In this section, we will present a novel technique to represent DPM efficiently. At first, we construct the execution order matrix O by reusing the scheduling decision variables $z_{p\widetilde{p}}^{ij}$ in Section 5.1. Then, based on the constructed execution order matrix O, we propose a novel approach to represent which two tasks instances are closest to each other, i.e., to represent which two tasks are scheduled to form the idle interval. With this approach, the idle interval of the task can be modeled. Besides, we introduce the virtual task concept to model idle interval of the last task instance in one hy-period. In the end, by taking the break-even time constraint into consideration, the decision variables that determine whether the system can enter sleep mode can be modeled as linear items.

5.2.2. DPM Formulation. In Section 5.1, binary variable $z_{p\widetilde{p}}^{ij}$ has been adopted to model the execution order of two tasks. To model the execution order of tasks, we reuse this binary variable $z_{p\widetilde{p}}^{ij}$ to construct a matrix O, so called execution order decision matrix (EODM), which is defined as:

Definition 5.1. Assume there are N tasks which need to decide their execution order and z^{ij} denote the execution order of task instance T_i and $T_j (i \neq j)$: $z^{ij} = 1$ if the task instance T_i finishes before the task T_j, and 0 otherwise. An $N \times N$ execution order decision matrix O can be determined as: (1) $i = j \Rightarrow o_{ij} = 0$ (2) $i \neq j \Rightarrow o_{ij} = z^{ij}$

z^{ij} and z^{ji} $(i \neq j)$ contradict each other, i.e., $z^{ij} + z^{ji} = 1$. Correspondingly, we have $o_{ij} + o_{ji} = 1$. According to Def. 5.1, execution order decision matrix O can be constructed based on binary variable $z_{p\widetilde{p}}^{ij}$ in Section 5.1. For the scheduling presented in Fig. 3, there are 7 task instances in one hy-period. To order 7 tasks, the 7×7 execution order decision matrix $O_{7\times7}$ can be determined as follows. And the

row and column of the matrix is indexed by task instance $\{T_1^1, T_1^2, T_2^1, T_2^2, T_3^1, T_4^1, T_4^2\}$. For brief description, TS_i denote the i-th task instance in the matrix. For example, task instances $\{T_1^1, T_1^2, T_2^1, T_2^2, T_3^1, T_4^1, T_4^2\}$ in the matrix are represented as $\{TS_1, TS_2, TS_3, TS_3, TS_4, TS_5, TS_6\}$. We can see task instance T_1^1 finishes before the other instances from the first row of the matrix.

$$O_{7\times7} = \begin{pmatrix} & T_1^1 & T_1^2 & T_2^1 & T_2^2 & T_3^1 & T_4^1 & T_4^2 \\ T_1^1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ T_1^2 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ T_2^1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ T_2^2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ T_3^1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ T_4^1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ T_4^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{17}$$

To represent the task execution order, we give out the following lemma.

LEMMA 5.2. *$ST(TS_i)$ and $FT(TS_i)$ denote the start time and finish time of task instance TS_i. Given an execution order decision matrix O with N task instances and compute decision variable matrix $A = O(1-O)^T$ with the element $a_{ij} = \sum_{k=1}^{N} o_{ik}(1-o_{jk})$. If $a_{ij} = 1$ holds, then TS_j is the closest task instance of TS_i and the idle interval I_i after task instance TS_i finishes can be represented as $I_i = ST(TS_j) - FT(TS_i)$.*

PROOF. $a_{ij} = \sum_{k=1}^{N} o_{ik}(1-o_{jk}) = \sum_{k\neq j}^{N} o_{ik} \cdot o_{kj} + o_{ij}$. $o_{ik} \cdot o_{kj} = 1$ represents that task instance TS_k starts after TS_i but before TS_j. When $\sum_{k\neq j}^{N} o_{ik} \cdot o_{kj} \neq 0$ holds, it means it exists task instance T_k that starts after the finish time of TS_i but before the start time of TS_j. In this case, TS_j must start after TS_i, i.e., $o_{ij} = 1$. Thus $a_{ij} \geq 2$. For the case of $\sum_{k\neq j}^{N} o_{ik} \cdot o_{kj} = 0$, it means there is no task instance starting after TS_i but before TS_j. In this case, if $o_{ij} = 1$ holds, TS_j is the closest task instance of TS_i, i.e., $a_{ij} = 1$. Otherwise, TS_j finish before TS_i, i.e., $a_{ij} = 0$. □

For the quadratic item $o_{ik} \cdot o_{kj}$ in Lem. 5.2, we can define intermediate variable $t_{ikj} = o_{ik} \cdot o_{kj}$ and it can be linearized as follows.

$$\begin{aligned} 0 \leq t_{ikj} &\leq o_{ik} \\ 0 \leq t_{ikj} &\leq o_{kj} \\ o_{ik} + o_{kj} - 1 \leq t_{ikj} &\leq 1 \end{aligned}$$

To demonstrate its correctness, we present one simple example. Based on $O_{7\times7}$, determination matrix $A_{7\times7}$ is calculated as:

$$A_{7\times7} = \begin{pmatrix} & T_1^1 & T_1^2 & T_2^1 & T_2^2 & T_3^1 & T_4^1 & T_4^2 \\ T_1^1 & 0 & 4 & 2 & 5 & \underline{\mathbf{1}} & 3 & 6 \\ T_1^2 & 0 & 0 & 0 & \underline{\mathbf{1}} & 0 & 0 & 2 \\ T_2^1 & 0 & 2 & 0 & 3 & 0 & \underline{\mathbf{1}} & 4 \\ T_2^2 & 0 & 0 & 0 & 0 & 0 & 0 & \underline{\mathbf{1}} \\ T_3^1 & 0 & 3 & \underline{\mathbf{1}} & 4 & 0 & 2 & 5 \\ T_4^1 & 0 & \underline{\mathbf{1}} & 0 & 2 & 0 & 0 & 3 \\ T_4^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{18}$$

From the matrix $A_{7\times7}$, we can locate the closest task for all tasks except T_4^2. For example, the closest task instance of T_1^1 is T_3^1. We also notice that there is no following task executed after T_4^2, since T_4^2 is the last executed task instance in the hy-period. Thus, the idle interval of T_4^2 cannot be determined. To represent the idle interval of T_4^2,

virtual task concept is proposed. The period of the *virtual task* is hy-period of the tasks and the execution time is zero. It starts when one hy-period starts or ends. As shown in Fig. 3, there are two *virtual tasks* in one hy-period, which are respectively denoted as VTS and VTE. VTS and VTE are at the start and end of the hy-period, respectively. By adding VTS and VTE into execution order decision matrix O and recomputing the determination matrix A, the closest instance of T_4^2 and VTS can be determined as VTE and T_1^1, respectively. Thus, I_3 and I_4 can be modeled.

To this end, we have found a principle to model the idle intervals after task finishes. However, this principle is still not obvious and it is not linear. To make it more simple for DPM representation, we present an two-step technique to transform this idle intervals representation into a 0-1 representation. Before introducing this transformation, we present some properties about the relationship between execution order decision matrix O and decision matrix A.

PROP. 1. *Execution order decision matrix O and decision matrix A have following relationships.*
(1) o_{ij} is 0-1 variable and a_{ij} is integer variable bounded in $[0, N-1]$
(2) $o_{ij} = 0 \Leftrightarrow a_{ij} = 0$
(3) $o_{ij} = 1 \Leftrightarrow a_{ij} \neq 0$

Based on the the relationships between execution order decision matrix O and decision matrix A presented in Prop. 1, two-step transformation is presented as follows.
Step 1: Define intermediate 0-1 variable matrix B. The element b_{ij} in the matrix B is determined by: (a) $2 \leq a_{ij} \leq N-1 \Rightarrow b_{ij} = 1$; (b) $a_{ij} \leq 1 \Rightarrow b_{ij} = 0$;

In the step 1, the value of 0-1 variable matrix B is determined by the value of the variable matrix A. To linearize this determination, we present the following lemma.

LEMMA 5.3. *Assume x and b are integer variable and 0-1 variable, respectively. s and s_0 are integer constants with $0 \leq x \leq s$ and $s_0 < s$. Given the determinations: $(a) s_0 \leq x \leq s \Rightarrow b = 1$; (b) $x \leq s_0 - 1 \Rightarrow b = 0$; Then, this determinations can be linearized as the following constraint.*

$$\frac{x - (s_0 - 1)}{s} \leq b \leq \frac{x}{s_0}$$

According to Prop. 1, $o_{ij} - b_{ij} = 1$ only happens when $a_{ij} = 1$ holds, and 0 otherwise. Thus, 0-1 variable $o_{ij} - b_{ij}$ can be used to decide whether TS_j is the closest task instance of TS_i.
Step 2: The 0-1 variable matrix $O - B$ can be used to represent the closest task of each task. If TS_j is the closest task instance of TS_i, $o_{ij} - b_{ij} = 1$ holds. Otherwise, $o_{ij} - b_{ij} = 0$ holds. Thus, we can represent the idle interval I_i of TS_i directly, which can be determined as follows.

$$I(TS_i) = \begin{cases} \sum\limits_{TS_j \neq VTE} (o_{ij} - b_{ij})(ST(TS_j) - FT(TS_i)) & TS_i \neq VTE \\ \sum\limits_{TS_j \neq VTE} (o_{ji} - b_{ji})(ST(TS_i) - FT(TS_j)) & TS_i = VTE \end{cases} \tag{19}$$

In (19), $I(VTS)$ and $I(VTE)$ respectively represent the first and last idle interval in one hy-period, e.g., I_4 and I_3 in Fig. 3. Denote TS_k as the last task instance in the task instance set $TSS = \{TS | TS \neq VTS, TS \neq VTE\}\}$. As the closest task of TS_k is VTE, we can get $I(TS_k) = 0$ according to (19). Thus, we can represent idle interval sets $S1 = \{I(TS_i) | TS_i \in TSS\}$ for the first $N-1$ task instances except the last task instance TS_k. It is worthy noting that $I(TS_k) = 0$ has no influence on the object function. Instead of $I(TS_k) = 0$, $I(VTS) + I(VTE)$ can be used to calculate the idle interval of the last task

instance. For example, in Fig. 3, the idle interval of T_4^2 can be computed as $I_3 + I_4$. For brevity, $I(VTS) + I(VTE)$ is denoted as $I(TS_{N+1})$.

In (19), $ST(TS_j)$ and $FT(TS_i)$ are linear items and $(FT(TS_j) - ST(TS_i))$ is bounded in $[-H_r, H_r]$ according to time-triggered scheduling. The item $o_{ij} - b_{ij}$ is 0-1 variable. Define intermediate variable $t_{ij} = (o_{ij} - b_{ij})(FT(TS_j) - ST(TS_i))$ and then it can be linearized according to Lem. 5.4.

LEMMA 5.4. *Given constant* $s_1, s_2 > 0$ *and two constraint spaces* $P_1 = \{[t, b, x] | t = bx,\ -s_1 \leq x \leq s_2,\ b \in \{0, 1\}\}$ *and* $P_2 = \{[t, b, x] | -b \cdot s_1 \leq t \leq b \cdot s_2,\ t + b \cdot s_1 - x - s_1 \leq 0, t - b \cdot s_2 - x + s_2 \geq 0, b \in \{0, 1\}\}$*, then* $P_1 \rightleftharpoons P_2$

PROOF. $\mathbf{P_1} \Rightarrow \mathbf{P_2}$: We obtain $-b \cdot s_1 \leq t \leq b \cdot s_2$ according to $t = bx$ and $-s_1 \leq x \leq s_2$. Based on $-s_1 \leq x \leq s_2$ and $b \in \{0, 1\}$, we can obtain $(b-1)(x - s_2) \geq 0$ and $(b-1)(x + s_1) \leq 0$. Hence, $t - b \cdot s_2 - x + s_2 \geq 0$ and $t + b \cdot s_1 - x - s_1 \leq 0$ hold. $\mathbf{P_2} \Rightarrow \mathbf{P_1}$: If $b = 0$ holds, we can prove that $t = 0$ and $-s_1 \leq x \leq s_2$ according to the definition of P_2. If $b = 1$ holds, we can obtain $-s_1 \leq t = x \leq s_2$ from P_2. Thus, $P_2 \rightleftharpoons P_1$. □

To show this transformation flows in details, we present an example in Fig. 3. In the first step, the 0-1 variable matrix $B_{8\times8}$ can be determined as follows.

$$\begin{pmatrix} & VTS & T_1^1 & T_1^2 & T_2^1 & T_2^2 & T_3^1 & T_4^1 & T_4^2 & VTE \\ VTS & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ T_1^1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ T_1^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ T_2^1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ T_2^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ T_3^1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ T_4^1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ T_4^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ VTE & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{20}$$

In the second step, the 0-1 variable matrix $O_{8\times8} - B_{8\times8}$ can be determined as follows. From the representation, the closest task can be directly represented as 0-1 variable and each one task has only one closest task except virtual task VTE. Thus, the idle interval of the core can be formulated as (19).

$$\begin{pmatrix} & VTS & T_1^1 & T_1^2 & T_2^1 & T_2^2 & T_3^1 & T_4^1 & T_4^2 & VTE \\ VTS & 0 & \underline{\mathbf{1}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ T_1^1 & 0 & 0 & 0 & 0 & 0 & \underline{\mathbf{1}} & 0 & 0 & 0 \\ T_1^2 & 0 & 0 & 0 & 0 & \underline{\mathbf{1}} & 0 & 0 & 0 & 0 \\ T_2^1 & 0 & 0 & 0 & 0 & 0 & 0 & \underline{\mathbf{1}} & 0 & 0 \\ T_2^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \underline{\mathbf{1}} & 0 \\ T_3^1 & 0 & 0 & 0 & \underline{\mathbf{1}} & 0 & 0 & 0 & 0 & 0 \\ T_4^1 & 0 & 0 & \underline{\mathbf{1}} & 0 & 0 & 0 & 0 & 0 & 0 \\ T_4^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \underline{\mathbf{1}} \\ VTE & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{21}$$

To represent the decision of entering sleep mode, m_i is used to denote the mode-switch decision for each interval $I(TS_i)$. The core can enter into the sleep state only when the idle interval $I(TS_i)$ is not shorter than the break-even time (i.e., $I(TS_i) \geq T_{BET}$). Thus, the value of m_i is determined by (a) $I(TS_i) \geq T_{BET} \Rightarrow m_i = 1$; (b) $I(TS_i) < T_{BET} \Rightarrow m_i = 0$. It is obvious that the idle interval I_i is bounded by the period of task instance TS_i, denoted as $h(TS_i)$. Similar to Lem. 5.3, above determination can be

transformed to the linear formulation.

$$\frac{I(TS_i) - T_{BET}}{h(TS_i)} < m_i \leq \frac{I_i}{T_{BET}} \tag{22}$$

Thus, the idle ans sleep interval of individual core in one hy-period can be represented as follows.

$$t_{sleep} = \sum_{i=1}^{N+1} m_i \cdot I(TS_i) \tag{23}$$

$$t_{idle} = \sum_{i=1}^{N+1} (1 - m_i) \cdot I(TS_i) \tag{24}$$

It is obvious that the varibale I_i is bounded by the period of task instance TS_i, i.e., $0 \leq I(TS_i) \leq h_i$. m_i is 0-1 variable, thus (23) and (24) can be linearized according to Lem. 5.4

5.3. Objective Function

The energy overhead of mode-switch can be determined as follows:

$$E_{ov} = \sum_{i=1}^{N+1} m_i \cdot E_{sw} \tag{25}$$

Up to now, we have presented the formulation for DVFS +DPM integration with task scheduling. In this paper, we are to minimize the total energy consumption in one hyper-period and the following object is used:

$$E = \sum_{\forall T_i} \frac{H_r}{h_i} \sum_{j=1}^{s} c_{ij} w_{ij} p_j + P_{sleep} t_{sleep} + P_{idle} t_{idle} + \sum_{i=1}^{N} m_i E_{sw} \tag{26}$$

5.4. Refinement

To determine the closest task instance for one task instance, we need to check the timing information of every task instance in one hy-period. Thus, the total number of variables used increases quadratically with the number of task instances in one hy-period, resulting in dramatically increased exploration space for the MILP. To maintain the scalability of the approach, it is important to develop techniques that can reduce the exploration space. Here, we propose a refinement approach based on execution window analysis, which can be used to determine which task instances have chance to construct the idle intervals. By this approach, we only need to check the task instances which are possible to execute in this execution windows, rather than checking every task instance in one hy-period.

In the following, we outline how to determine execution window for each task and how the execution window can be used to determine the possible task instance set that need to check. The worst-case execution window of task instance T_p^i is determined by the earliest start time $s_p^{min} + i \cdot h_p$ and its latest start time $s_p^{max} + i \cdot h_p$, as shown in Fig. 4. s_p^{min} and s_p^{max} denote the lower bound and the upper bound of the start time s_p for task T_p, respectively. Algo. 1 represents the details of computing the bounds for start time for each task.

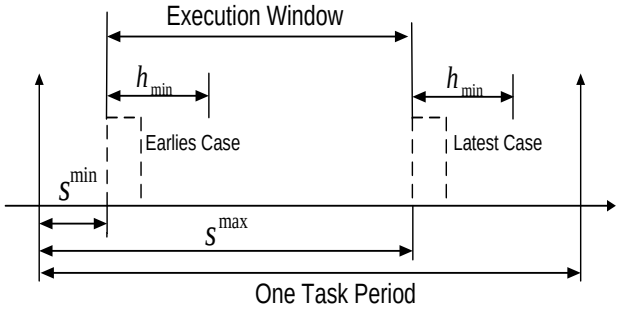


Fig. 4. Execution window.

ALGORITHM 1: Find Bounds for Start Time

Input: Directed Cyclic Task Graph $G(V, E, H)$ and WCET w_{ps} of each task T_p under highest frequency f_s.
Output: $[s_p^{min}, s_p^{max}]$ of the start time s_p of each task T_p.
for *each task* T_p **do**
  $s_p^{min} \leftarrow 0; s_p^{max} \leftarrow d_p - w_{ps}$;
end
for *each task* $T_p \in V$ *from the header of* G *to the end of* G **do**
  while $T_{\widetilde{p}}$ *is the child of* T_p **do**
    $s_{\widetilde{p}}^{min} \leftarrow \max(s_{\widetilde{p}}^{min}, s_p^{min} + w_{ps})$;
  end
end
for *each task* $T_p \in V$ *from the end of* G *to the header of* G **do**
  while $T_{\widetilde{p}}$ *is the parent of* T_p **do**
    $s_{\widetilde{p}}^{max} \leftarrow \min(s_{\widetilde{p}}^{max}, s_p^{max} - w_{\widetilde{p}s})$;
  end
end

Once the bound [s_p^{min}, s_p^{max}] of the start time is found, we can compute the execution window $EW(T_p^i)$ for each task instance T_p^i.

$$EW(T_p^i)_{min} = s_p^{min} + i \cdot h_p \tag{27}$$

$$EW(T_p^i)_{max} = s_p^{max} + i \cdot h_p \tag{28}$$

According to worst case execution window, we can determine whether two task instance $T_{\widetilde{p}}^i$ and T_p^i are possible to overlap each other. Thus, the value of some decision variables o_{ij} can be fixed according to (29), which can be used to reduce the scale of the execution order matrix O and the number of the following intermediate variables (e.g., a_{ij} and b_{ij}) .

$$o_{ij} = \begin{cases} 1 & EW(T^i)_{max} + w^i + \xi \leq EW(T^j)_{min} \\ 0 & EW(T^j)_{max} + w^j + \xi \leq EW(T^i)_{min} \\ var & \text{otherwise} \end{cases} \tag{29}$$

Besides, worst case execution window analysis can also be used to determine whether task instance $T_{\widetilde{p}}^i$ is possible to be the closest task instance T_p^i. Denote $h_{min}(p_i)$ as the minimum period of the tasks on the core p_i. The idle interval on the core p_i must be less than $h_{min}(p_i)$. The possible closest task instance set $PCTI(T_p^i)$ of task instance

T_p^i can be determined as follows.

$$PCTI(T_p^i) = \{TS | EW(TS)_{min} < EW(T_p^i)_{max} + h_{min} \ \wedge \ EW(TS)_{max} > EW(T_p^i)_{min}\} \tag{30}$$

Thus, we only need to explore the task instances in $PCTI(T_p^i)$ to construct determination variable in determination matrix A, rather than all the task instances. Then, we can formulate DPM using the techniques present in Section 5.2.

6. PERFORMANCE EVALUATIONS

This section presents the case studies. We use some real-life benchmarks in the simulation. The energy parameters of the processor are collected from [Jejurikar et al. 2004; Wang and Mishra 2010; Martin et al. 2002] with 70 nm technology. The CPLEX [CPLEX 2010] solver is used to solve the MILP problems. All experiments are conducted on a computer with 2.3GHz Intel 8-core CPU and 16GB memory.

6.1. Experiment Setup

To evaluate the effectiveness of our approach, we conduct the experiments on 10 task graphs: three FFT benchmarks from MiBench [Guthaus et al. 2001], two consumer application benchmark form Embedded Systems Synthesis Benchmarks(E3S) [Vallerio and Jha 2003] largely based on the date from the Embedded Microprocessor Benchmark Consortium (EMBC), five task graphs from TGFF [Dick et al. 1998]. The task graphs for FFT is obtained based on the implementation. By employing Cooley-Tukey algorithm [Henry and Nazhandali 2009], N-point FFT can be split into two parallel $\frac{N}{2}$-point FFT. We modified the FFT benchmark in MiBench into three different task graphs with 4 tasks, 7 tasks, and 10 tasks, respectively. The task graph with 4 tasks is obtained by splitting N-point FFT into two $\frac{N}{2}$ FFT tasks. The graph with 7 tasks is obtained by splitting N-point FFT into one $\frac{N}{2}$ FFT tasks and two $\frac{N}{4}$ FFT tasks. Similarly, we construct the graph with 10 tasks by splitting N-point FFT into four $\frac{N}{4}$ FFT tasks. For simplicity, FFT-N-M denotes task graph with M tasks for N-point FFT. We run each task in FFT benchmark on SimpleScalar cycle-accurate simulation platform [SimpleScalar 2003] to obtain its execution time(in cycles). Two consumer application benchmarks in E3S, i.e., consumer-1 and consumer-2, are embedded consumer electronic applications. In these two benchmarks, consumer-1 application contains 7 tasks including tasks like JPEG compression, high pass gray-scale filter, and RGB to YIQ conversion, etc. And consumer-2 application contains 5 tasks including tasks like JPEG decompression, RGB to CYMK conversion, and display, etc. Besides, we use TGFF to generate 5 period task graphs by adopting example input files that come with the software package. kbasic-1 and kbasic-2 are generated by kbasic example input file and respectively contain 8 and 10 tasks. kseries-parallel-1 and kseries-parallel-2 are obtained from kseries_parallel example input file and respectively have 8 and 16 tasks. robtst with 13 tasks is obtained from robtst example input file. We consider eight combinations of these applications. Details of the combinations are shown in Tab. III.

We consider a 4-core and 8-core architectures for our experiment. The experiments are conducted based on classical energy model of 70nm technology processor in [Martin et al. 2002; Wang and Mishra 2010; Jejurikar et al. 2004], whose accuracy has been verified by SPICE simulation. Tab. IV lists the energy parameter under 70nm technology [Martin et al. 2002; Wang and Mishra 2010]. We assume that the processor operates at five voltage levels in the range of $[0.65V, 0.85V]$ with 50 mV steps. From [Wang and Mishra 2010; Jejurikar et al. 2004], body bias voltage V_{bs} is obtained as $-0.7V$. According to energy model in Section 3.3, we can calculate the corresponding frequency f, dynamic power P_{dyn} and static power P_{sta} under different voltage level, as shown

Table III. Task graph sets.

	Task graph	Task#
Set1	FFT-65536-4,FFT-131072-7, FFT-262144-10	21
Set2	consumer-1,FFT-65536-10, FFT-32768-4	21
Set3	FFT-65536-10,consumer-2, kseial-pallel-1	23
Set4	kbasic-1,kbasic-2	18
Set5	robtst,kbasic-1,FFT-65536-4	25
Set6	kseial-pallel-1,kseial-pallel-2	24
Set7	FFT-65536-4,kbasic-1,kseial-pallel-1	20
Set8	FFT-131072-4,consumer-1, consumer-2,kseial-pallel-1	24

Table IV. Constants for 70nm technology [Martin et al. 2002; Wang and Mishra 2010].

Const	Value	Const	Value	Const	Value
K_1	0.063	K_6	5.26×10^{-12}	V_{th1}	0.244
K_2	0.153	K_7	-0.144	I_j	4.8×10^{-10}
K_3	5.38×10^{-7}	V_{dd}	[0.5,1]	C_{eff}	0.43×10^{-9}
K_4	1.83	V_{bs}	[-1,0]	L_d	37
K_5	4.19	α	1.5	L_g	4×10^6

Table V. Dynamic power consumption and static power consumption for 70nm processor.

V_{dd} (V)	0.85	0.8	0.75	0.7	0.65
f (GHz)	2.10	1.81	1.53	1.26	1.01
P_{dyn} (mW)	655.5	498.9	370.4	266.7	184.9
P_{sta} (mW)	462.7	397.6	340.3	290.1	246

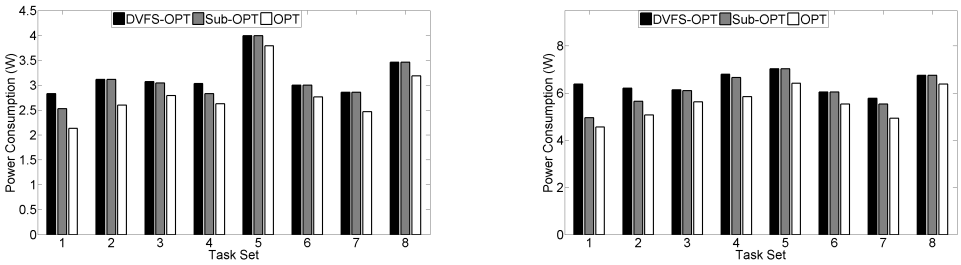
in Tab. V. From [Wang and Mishra 2010], P_{on} related to idle power can be obtained as $276mW$ and the power consumption in sleep mode P_{sleep} is set as $80\mu W$. The energy overhead E_{sw} of state transition is set as $385\mu J$. The total state transition time t_{sw} and the overhead of dynamical frequency scaling ξ are obtained by referring to the sleep mode timing specification of the commercial processor [Marvell 2009].

To evaluate the performance of our technique, we compared the energy consumption with the following technique:

— Optimal DVFS (`Dvfs-OPT`): Tasks are assigned the optimal frequency without considering DPM. The processor stays at the idle mode for all the idle interval.
— Applying DVFS before DPM (`SubOPT`): Similar to [Srinivasan and Chatha 2007], we implement optimal DVFS as the first step to get the frequency assignment and applies DPM as the final design step. For fairness comparison, we use our DPM technique to get the final result.
— Optimal DVFS-DPM (`OPT`): Integrate DVFS and DPM globally with scheduling.

6.2. Results

Fig. 5 shows the overall power consumptions of the three compared techniques for 8 task sets on 4-core and 8-core architectures. To increase the workload on 8-core architecture, we implement the simulation by duplicating the number or decreasing the period of task graph. From the simulation result, we can see the scheme of applying DVFS before DPM (`SubOPT`) fails to achieve power savings at most benchmark set in both architectures. This is because optimal DVFS in the first step will result lower frequency assignment, which will prolong the execution time of tasks and, at the same time, reduce the opportunity of entering the sleep state. Comparing to `Dvfs-OPT`, the



(a) 4-core architecture (b) 8-core architecture

Fig. 5. Overall power consumption.

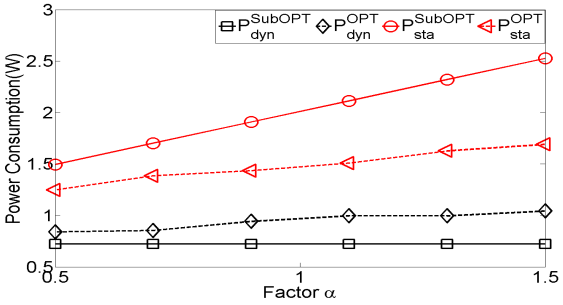
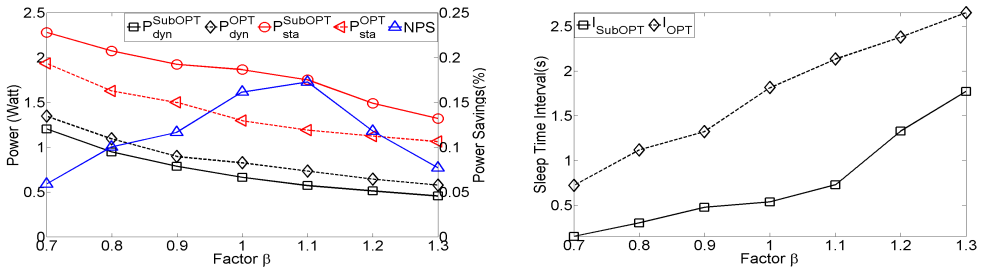


Fig. 6. Power consumption with P_{on} varying.

scheme of applying DVFS before DPM (SubOPT) can only on average achieve 2.2% and 4.7% power savings on 4-core and 8-core architectures, respectively. Besides, we can observe that our approach (OPT) is more energy-efficient than the scheme of applying DVFS before DPM (SubOPT). Our approach (OPT) , which integrates DVFS and DPM globally with scheduling, can on average achieve 10.5% (up to 16.0%) and 8.9% (up to 12.2%) power savings with respect to SubOPT on 4-core and 8-core architectures, respectively.

Next, we conduct the experiment to show the impact of P_{on} to the effectiveness of our approach. α is denote as the factor that varies the P_{on} with respect to its original setting, i.e, $P_{on} = \alpha \cdot P_{on}^{org}$, where P_{on}^{org} is the original setting. We vary the factor α from 0.5 to 1.5 with fixed step size 0.2. The dynamic power consumption P_{dyn}, static power consumption P_{sta} are compared between SubOPT and OPT. Fig. 6 illustrates the results for benchmark set 7 on 4-core architecture. Note that, when P_{on} increases, the leakage power increases its dominance of the total power consumption. From the results, we can make the following observations: (1) By creasing the frequency of the processor to create longer idle intervals, OPT consume more dynamic power than SubOPT. At the same time, OPT achieve significant leakage power savings, which results the overall power consumption of OPT is smaller than SubOPT. (2) The leakage power of SubOPT increases linearly with respect to P_{on} while its dynamic power becomes constant. It means that when the leakage power increases its dominance, the techniques of applying DVFS before DPM cannot increase the opportunities of entering sleep modes to reduce the leakage power. In contrast, OPT can optimally deal with the trade-off between dynamic power and leakage power to reduce overall power consumption. When P_{on} increases, OPT could increase the frequencies, which results in slightly increase of dynamical power, to avoid the rapid growth of leakage power.



(a) Power consumption with periods varying. (b) Sleep time interval with periods varying.

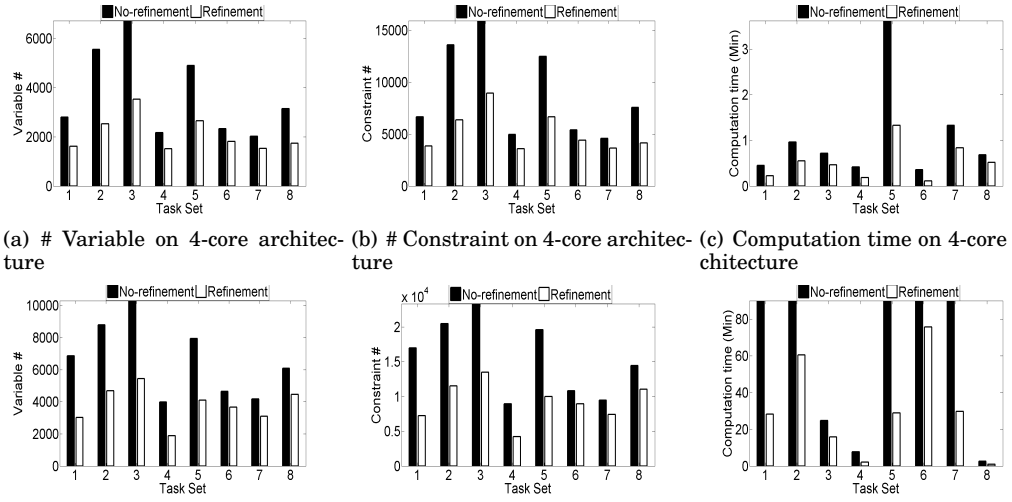
Fig. 7. Performance with periods varying.

Then, we discuss the impact of the period setting to the effectiveness of our approach. Similar to the definition of the factor α, we also define the factor β that varies the periods of each period with respect to its original setting. We vary the factor β from 0.7 to 1.3 with fixed step size 0.1. The dynamic power consumption P_{dyn}, static power consumption P_{sta}, and sleep time interval I are compared between SubOPT and OPT. Fig. 7 illustrates the results for benchmark set 1 on 4-core architecture. In Fig. 7, dynamic power consumption P_{dyn} and static power consumption P_{sta} of both techniques decreases as the period of the application increases. This is expected because the bigger period of the application can prolong both the execution time and the idle intervals. One interesting observation is that the overall power savings of OPT with respect to SubOPT increases when $\beta < 1.1$ and decreases when $\beta \geq 1.1$. This is caused by the fact that, in SubOPT, most tasks of the application has been assigned to lowest frequency when the periods of the applications are bigger enough. All increased period are contributed to prolong the idle intervals, which results in the sleep time interval of SubOPT increase rapidly (as shown in Fig. 7(b)). Thus, leakage power of SubOPT will decrease rapidly, which also results in the power savings of OPT decreasing.

In the end, we conduct experiments to show the efficiency of our refinement technique. We adopt the same task graph set as above. We compare the numbers of variables and constraints in the MILP problem as well as the solving time. Fig. 8 shows the results on both 4-core and 8-core architectures for approaches with refinement and without refinement. Noting that refinement approach can achieve the same optimal solution as the approach without refinement. The MILP solver is set to have 90 minutes time budget to execute for each task graph set. On 4-core architecture, the refinement approach can generate the results in 2 minutes. On 8-core architecture, the non-refinement approach fail to generate the results on 5 task sets due to expired execution time, while the refinement approach can generate the results in 80 minutes. Besides, the refinement approach can on average achieves 39.82% reduction on the number of variables and 37.82% reduction on the number of constraints. The results show refinement technique can significantly reduce the MILP problem size.

7. CONCLUSION AND FUTURE WORK

This paper presents an energy optimization technique for scheduling real-time tasks on MPSoCs based platforms with optimally DVFS and DPM combination. A key technique is proposed to directly model the idle interval of individual processor. Based on this technique, an integrated solution for optimal DVFS and DPM combination problem is presented based on mixed integer linear programming. Our technique can generate an optimal time-triggered non-preemptive schedule for each task and an optimal frequency assignment for each task to minimize the total energy consumption of MPSoCs.



(a) # Variable on 4-core architecture (b) # Constraint on 4-core architecture (c) Computation time on 4-core architecture

(d) # Variable on 8-core architecture (e) # Constraint on 8-core architecture (f) Computation time on 8-core architecture

Fig. 8. No-refinement vs refinement.

Based on this optimal schedule, the tasks can be scheduled with insertion of voltage-setting instructions. Besides, we develop a novel technique that can significantly reduce the exploration space of MILP. Proof-of-concept simulation results demonstrate the effectiveness of our approach compared with existing approaches.

For the next step, we are interested in implementing the proposed approach on realistic system and evaluating its performance. Now, a new MPSOCs based on time-triggered architecture is presented in [Salloum et al. 2012] specially for safety-critical embedded systems. Besides, there are also available time-triggered embedded systems in the industry, e.g., TTE processor [TTE system 2007]. Our optimization process can be used to produce energy-aware time-triggered non-preemptive schedules for such time-triggered MPSOCs.

Furthermore, another interesting future work would be to support voltage-frequency island partitioned system. Approach to support voltage-frequency island partitioned system should meet two kinds of constraints: (1) cores within one island should share the same frequency. (2) cores within one island should enter into sleep mode at the same time as well as switching to active mode.

REFERENCES

ARM 2012. ARM Cortex-A15 serious. `http://www.arm.com/products`. (2012).

S. Baruah and G. Fohler. 2011. Certification-Cognizant Time-Triggered Scheduling of Mixed-Criticality Systems. In *Proceedings of 2011 IEEE 32nd Real-Time Systems Symposium (RTSS)*.

MuhammadKhurram Bhatti, Ceile Belleudy, and Michel Auguin. 2011. Hybrid power management in real time embedded systems: an interplay of DVFS and DPM techniques. *Real-Time Systems* (2011).

Gang Chen, Kai Huang, Christian Buckl, and Alois Knoll. 2013. Energy optimization with worst-case deadline guarantee for pipelined multiprocessor systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*.

Jian-Jia Chen and Tei-Wei Kuo. 2005. Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics. In *Proceedings of International Conference on Parallel Processing (ICPP)*.

Hui Cheng and Steve Goddard. 2006. Online energy-aware I/O device scheduling for hard real-time systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*.

CPLEX 2010. IBM ILOG CPLEX Optimizer. `http://www.ibm.com/software/`. (2010).

V. Devadas and H. Aydin. 2012. On the Interplay of Voltage/Frequency Scaling and Device Power Management for Frame-Based Real-Time Embedded Applications. *IEEE Trans. Comput.* (2012).

R.P. Dick, D.L. Rhodes, and W. Wolf. 1998. TGFF: task graphs for free. In *Proceedings of the 6th International Workshop on Hardware/Software Codesign*.

Marco E. T. Gerards and Jan Kuper. 2013. Optimal DPM and DVFS for frame-based real-time systems. *ACM Transactions on Architecture and Code Optimization (TACO)*, Article 41 (2013), 23 pages.

F. Gruian and K. Kuchcinski. 2001. LEneS: task scheduling for low-energy systems using variable supply voltage processors. In *Proceedings of the 2001 Asia and South Pacific Design Automation Conference (ASP-DAC)*.

M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. 2001. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the 2001 IEEE International Workshop on Workload Characterization (WWC)*.

Michael B. Henry and Leyla Nazhandali. 2009. Hybrid Super/Subthreshold Design of a Low Power Scalable-Throughput FFT Architecture. In *Proceedings of the 2009 International Conference on High Performance Embedded Architectures and Compilers (HiPEAC)*.

Kai Huang, Luca Santinelli, Jian-Jia Chen, Lothar Thiele, and GiorgioC. Buttazzo. 2011. Applying real-time interface and calculus for dynamic power management in hard real-time systems. *Real-Time Systems* (2011).

Chia-Mei Hung, Jian-Jia Chen, and Tei-Wei Kuo. 2006. Energy-Efficient Real-Time Task Scheduling for a DVS System with a Non-DVS Processing Element. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium (RTSS)*. 303 –312.

Intel 2009. Intel Atom Processor. `http://www.intel.com/processors/atom.html`. (2009).

Intel 2011. Intel 22nm Technology. `http://www.intel.com/silicon-innovations/`. (2011).

Intel 2012. Intel Single-Chip Cloud Computer (SCC). `http://www.intel.com/content/www/us/en/research`. (2012).

ITRS 2011. International Technology Roadmap for Semiconductors. `http://www.itrs.net/reports.html`. (2011).

R. Jejurikar, C. Pereira, and R. Gupta. 2004. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of 2004 41st ACM/IEEE Design Automation Conference (DAC)*.

N.K. Jha. 2001. Low power system scheduling and synthesis. In *Proceedings of 2001 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*.

Dawei Li and Jie Wu. 2012. Energy-Aware Scheduling for Frame-Based Tasks on Heterogeneous Multiprocessor Platforms. In *Proceedings of the 2012 41st International Conference on Parallel Processing (ICPP)*.

C.E. Lin, Hung-Ming Yen, and Yu-Shang Lin. 2007. Development of Time Triggered hybrid data bus System for small aircraft digital avionic system. In *Proceedings of IEEE/AIAA 26th Digital Avionics Systems Conference (DASC)*.

Martin Lukasiewycz, Sebastian Steinhorst, Florian Sagstetter, Wanli Chang, Peter Waszecki, Matthias Kauer, and Samarjit Chakraborty. 2012. Cyber-Physical Systems Design for Electric Vehicles. In *Proceedings of the 2012 Euromicro Conference on Digital System Design(DSD)*.

Steven M. Martin, Krisztian Flautner, Trevor Mudge, and David Blaauw. 2002. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design (ICCAD)*.

Marvell 2009. Marvell PXA270 Processor. `http://www.marvell.com/application-processors/pxa-family/`. (2009).

Marvell 2012. Marvell ARMADA. `http://www.marvell.com/`. (2012).

Christian El Salloum, Martin Elshuber, Oliver Hoftberger, Haris Isakovic, and Armin Wasicek. 2012. The ACROSS MPSoC – A New Generation of Multi-core Processors Designed for Safety-Critical Embedded Systems. In *Proceedings of 2012 15th Euromicro Conference on Digital System Design (DSD)*.

SimpleScalar 2003. SimpleScalar LLC. `http://www.simplescalar.com`. (2003).

Amit Kumar Singh, Anup Das, and Akash Kumar. 2013. Energy optimization by exploiting execution slacks in streaming applications on multiprocessor systems. In *Proceedings of the 50th Annual Design Automation Conference (DAC)*.

Krishnan Srinivasan and Karam S. Chatha. 2007. Integer linear programming and heuristic techniques for system-level low power scheduling on multiprocessor architectures under throughput constraints. *Integration,the VLSI Journal* (2007).

TTE system 2007. Reliable time-triggered Processor. `http://www.tte-systems.com/products/`. (2007).

K.S. Vallerio and N.K. Jha. 2003. Task graph extraction for embedded system synthesis. In *Proceedings of the 16th International Conference on VLSI Design (VLSID)*.

Weixun Wang and P. Mishra. 2010. Leakage-Aware Energy Minimization Using Dynamic Voltage Scaling and Cache Reconfiguration in Real-Time Systems. In *Proceedings of the 23rd International Conference on VLSI Design (VLSID)*.

Yi Wang, Duo Liu, Zhiwei Qin, and Zili Shao. 2010. Memory-Aware Optimal Scheduling with Communication Overhead Minimization for Streaming Applications on Chip Multiprocessors. In *Proceedings of the 2010 31st IEEE Real-Time Systems Symposium (RTSS)*.

Yi Wang, Hui Liu, Duo Liu, Zhiwei Qin, Zili Shao, and Edwin H.-M. Sha. 2011. Overhead-aware energy optimization for real-time streaming applications on multiprocessor System-on-Chip. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* (2011).

Huiting Xu, Fanxin Kong, and Qingxu Deng. 2012. Energy Minimizing for Parallel Real-Time Tasks Based on Level-Packing. In *Proceedings of the 2012 18th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*.

Yumin Zhang, Xiaobo Hu, and D.Z. Chen. 2002. Task scheduling and voltage selection for energy minimization. In *Proceedings of the 39th Design Automation Conference (DAC)*. DOI:http://dx.doi.org/10.1109/DAC.2002.1012617

Xiliang Zhong and Cheng-Zhong Xu. 2008. System-wide energy minimization for real-time tasks: Lower bound and approximation. *ACM Transactions on Embedded Computing Systems (TECS)* (2008).

Received XXX XXX; revised XXX XXX; accepted XXX XXX