Adaptive Dynamic Power Management for Hard Real-time Pipelined Multiprocessor Systems

Gang Chen[1], Kai Huang[1,2], Alois Knoll[1]
[1]Insititute of Robotics and Embedded Systems, Technical University Munich, Germany
[2]School of Mobile Information Engineering, Sun Yat-sen University, China
{cheng,huangk,knoll}@in.tum.de

***Abstract*—Energy efficiency is a critical design concern for embedded systems. Dynamic power management (DPM) schemes in Multiprocessor System on Chips (MPSoCs) has been wildly used to explore the idleness of processors and dynamically reduce the energy consumption by putting idle processors to low-power states. In this paper, we explore how to effectively apply dynamic power management in adaptive manner to reduce leakage power consumption for coarse-grained pipelined systems under hard real-time requirements. At each adaptive point, a system transformation is proposed to model the pipeline system with unfinished events as multi-stream system. By using extended pay-burst-only-once principle, the service curves for corresponding stream can be computed as a constraint for a minimal resource demand and energy minimization problem can be formulated with respect to the resource demands at each adaptive point. One light-weight heuristic, called balance workload scheme (BWS), is proposed in this paper to solve the minimization problem. Simulation results using real-life applications are presented to demonstrate the effectiveness of our approach.**

I. Introduction

To achieve high performance and energy efficiency, multi-core architectures are widespread in modern computer systems. When multi-core architectures are powered by batteries, reducing the power consumption is one of the major design goals, because an energy-efficient design enables slower depletion of batteries and results in lower chip temperatures that improve performance and reliability.

Among multi-core architectures, pipelined multiprocessor architectures are believed to be one of promising computing paradigms for embedded system design, which can, in principle, provide high throughput and low energy consumption [2], [11], [12]. In pipelined multiprocessor architectures, processors are connected in a pipelined fashion via shared memory (e.g., FIFOs). A streaming application can be split into a sequence of functional blocks that are computed by a pipeline of processors where clock/power-gating techniques can be applied to achieve energy efficiency.

Designing the scheduling policy for the pipeline stages under the requirements of both energy efficiency and timing guarantee is non-trivial due to the conflict objectives between energy efficiency and timing guarantee. Most of the previous work on this topic [26], [4], [12], [2] cannot be applied to hard real-time system with non-deterministic workloads. To deal with non-deterministic workloads, the current state-of-the-art approach [3] computes periodic time-driven turn on/off patterns for pipeline stages in offline to minimize the leakage power consumption while guarantee the end-to-end deadline. However, the off-line approach is not energy-efficient compared to adaptive approach, because the slacks caused by runtime variability of execution time and event arrivals cannot be explored in the static approach.

This paper explores how to apply dynamic power management in adaptive manner to reduce leakage power consumption for coarse-grained pipelined systems under hard real-time requirements. We consider a pipeline architecture, where each processor have active, standby, and sleep modes with different power consumptions, and a streaming application with end-to-end deadline requirement. The streaming application can be split into a sequence of coarse-grained functional blocks which are mapped to a pipeline architecture for processing. The workload of the streaming application is abstracted as an event stream. The event arrivals of the stream are modeled as the arrival curves in interval domain [15], [22]. At each adaptive point, dynamic power management scheme should decide *when* and *which* power state should be selected for a processor to reduce the energy consumption of the system under the hard real-time constraints. The decision is a challenging one due to the following facts. First, at one time instant for scheduling decision, scheduling decision need to guarantee the timing requirement of the new events in the first stage, as well as the on-going events stored in the system. Second, the time and energy overheads are involved in a transition between active mode and sleep mode. Third, the current decision should be consistent with the decision made in the last point.

In this paper, we propose one integrated approach to resolve these concerns. The integrated approach could adaptively regulate the delay of the processors according to the workload and the current state of the stages, and procrastinate the events as late as possible. Comparing to the state-of-the-art work in [3], the adaptive approach can efficiently explore the slacks by using dynamic counter techniques [14] and adaptively checking FIFO state, and can achieve significant energy savings with respect to the off-line approach in [3]. The contributions of this paper are summarized as follows:

- We propose one system transformation to model the system with event stored in FIFOs as multi-stream system, which enables us to analysis the system timing efficiently.
- We extend the pay-burst-only-once principle for the transformed multi-stream system and offer the proof for its correction. By inversely using this extended the pay-

burst-only principle, the service curves for the corresponding stream can be computed as a constraint for a minimal resource demand.
- We derive a formulation of the minimization problem based on the needed resource of individual stages of the pipeline architecture at each adaptive time instant for scheduling decision.
- We propose one light weight scheme, called balance workload scheme, to find one optimal decision at each adaptive point.
- We conduct simulation using the real-life application to demonstrate the effectiveness of our approach.

The rest of the paper is organized as follows: Section II reviews related work in the literature. Section III presents basic models and the definition of the studied problem. Section IV describes the motivation and the proposed approach. Experimental evaluation is presented in Section V and Section VI concludes the paper.

II. Related Work

Pipelined computing is a promising paradigm for embedded system design, which can in principle provide high performance and low energy consumption. Pipelined multiprocessor systems are wildly applied as a viable platform for high performance implementation of multimedia applications [21], [20]. Energy optimization for pipelined multiprocessor systems is an interesting topic where a number of techniques have been proposed in the literature. Carta et al. [2] and Alimonda et al. [1] proposed a feedback-control technique for dynamic voltage/frequency scaling (DVFS) in a pipelined MPSoC architecture with soft real-time constraints, aimed at minimizing energy consumption with throughput guarantees. Javaid et al. [12] proposed a adaptive pipelined MPSoC architecture and a run-time balancing approach based on workload prediction to achieve energy efficiency. Authors in [11] proposed a dynamic power management scheme for the adaptive pipelined MPSoCs. In this work, the duration of idle periods is determined based on the future workload prediction and is used to select an appropriate power state for the idle processor. However, above approaches are under the soft real-time constraints. When coming to the hard real-time systems, these approaches are not applicable.

There are also many works [5], [26], [27] for hard real-time systems. Yang et al. [26] presents an integer linear programming (ILP) formulation for the problem of frequency assignment of a set of periodic independent tasks on heterogeneous multi-processor system. Zhang et al. [27] proposed a two-phase framework that integrates task scheduling and voltage selection to minimize the energy consumption of real-time dependent tasks on MPSoCs. But these methods require the precise timing information, such as periodical real-time events. However, in practice, this precise timing information of task arrivals might not be determined in advance. In the above studies, there is no guarantee that a event will arrive in time. Therefore, these approaches can not be applied to guarantee the worst-case deadline in embedded systems where violating deadlines could be disastrous. Unlike previous works, we focus on improving energy-efficiency in hard real-time embedded systems while guarantee the system satisfy the worst-case deadline constraint.

To model irregular event arrivals, Real-Time Caculus (RTC) [22], which is based on Network Calculus [15], can be applied to abstract task arrivals into time interval domain. Considering the DVFS system, Maxiaguine et al. [17] computed a safe frequency at periodical interval to prevent buffer overflow of a system. By adopting RTC models, Chen et al. [6] explored the schedulability for online DVFS scheduling algorithms. Combining optimistic and pessimistic DVFS Scheduling, Perathoner et al. [19] presented an adaptive scheme for the scheduling of arbitrary event streams. When only consider dynamic power management (DPM), Huang et al. [10] presented a algorithm to find periodic time-driven patterns to turn on/off processor for energy saving. On-line algorithms are proposed in [9], [14] to adaptively control the power mode of a system, procrastinating the processing of arrived events as late as possible. In one algorithm in [9], a bound of event arrivals is computed based on historical information of event arrivals in the recent past. Instead of using historical information, dynamic counter technique [14] is used to predict the future workload. Unfortunately, above approaches can only be applied to single processor. In the context of multiple processors system, the authors in [3] recently presented one off-line approach to compute a set of periodic time-driven turn on/off patterns for the pipelined multiprocessor systems. However, the approach in [3] cannot explore slack generated at runtime to reduce the energy consumption further. Nevertheless, how to apply dynamic power management at runtime is not yet clear. In this paper, we present an adaptive approach to determine energy-efficient scheduling at runtime with hard real-time constraints for pipelined multiprocessor systems using the arrival curve model.

III. System Models and Problem Definition

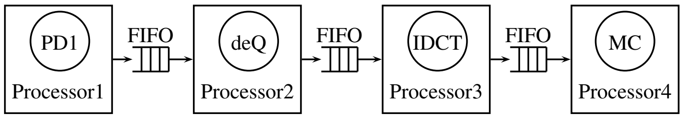
A. Hardware Model

We consider the system with the pipeline architecture showed in Fig. 1(a). Sub-tasks of a partitioned application are mapped and executed in different processors, which are connected via FIFOs. Each processor in the pipelined system has three power consumption modes, namely active, standby, and sleep modes, as shown in Fig. 1(b). To serve events, the processor must be in the active mode with power consumption P_a. When there is no event to process, the processor can switch to sleep mode with lower power consumption P_σ. However, mode-switching from sleep mode to active mode will cause additional energy and latency penalty, respectively denoted as $E_{sw,on}$ and $t_{sw,on}$. To prevent the processor from frequent mode switches, the processor can stay at standby mode with power consumption P_s, which is less than P_a but more than P_σ, i.e. $P_a > P_s > P_\sigma$. Moreover, the mode-switch from active (standby) mode to sleep mode will cause energy and time overhead, respectively denoted by $E_{sw,sleep}$ and $t_{sw,sleep}$.

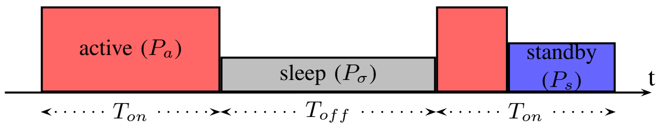
Consider the overhead of switching system from active mode to sleep mode, the system break-even time T_{BET} denotes the minimum time length that the system stays at sleep mode. If the interval that the system can stay at sleep mode is smaller than T_{BET}, the mode-switch mode overheads are larger than the energy saving. Therefore, switching mode is not worthwhile. Break-even time T_{BET} can be defined as:

$$T_{BET} = \max(t_{sw}, \frac{E_{sw}}{P_s - P_\sigma}) \qquad (1)$$

where $t_{sw} = t_{sw,on} + t_{sw,sleep}$ and $E_{sw} = E_{sw,on} + E_{sw,sleep}$.



(a) H.263 decoder on pipeline hardware architecture



(b) Power model of a processor

Fig. 1. System model

B. Energy Model

The analytical processor energy model in [16], [25], [13] is adopted in this paper, whose accuracy has been verified with SPICE simulation. The dynamic power consumption of the core on one voltage/frequency level (V_{dd}, f) can be given by:

$$P_{dyn} = C_{eff} \cdot V_{dd}^2 \cdot f \qquad (2)$$

where V_{dd} is the supply voltage, f is the operating frequency and C_{eff} is the effective switching capacitance. The cycle length t_{cycle} is given by a modified alpha power model.

$$t_{cycle} = \frac{L_d \cdot K6}{(V_{dd} - V_{th})^\alpha} \qquad (3)$$

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.

$$V_{th} = V_{th1} - K_1 \cdot V_{dd} - K_2 \cdot V_{bs} \qquad (4)$$

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:

$$P_{sta} = L_g \cdot (V_{dd} \cdot I_{subn} + |V_{bs}| \cdot I_j) \qquad (5)$$

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

$$I_{subn} = K_3 \cdot e^{K_4 V_{dd}} \cdot e^{K_5 V_{bs}} \qquad (6)$$

where K_3, K_4 and 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 at active mode and at stand-by mode, i.e., P_a and P_s, under one voltage/frequency (V_{dd}, f) can be respectively computed as:

$$P_a = P_{dyn} + P_{sta} + P_{on} \qquad (7)$$

$$P_s = P_{sta} + P_{on} \qquad (8)$$

where P_{on} is an inherent power needed for keeping the processor on.

C. Event Model

This paper considers streaming applications that can be split into a sequence of tasks. As shown in Fig. 1(a), a H.263 decoder is represented as four tasks (i.e., PD1, deQ, IDCT, MC) implemented in a pipeline fashion [18]. To model the workload of the application, the concept of arrival curve $\alpha(\Delta) = [\alpha^u(\Delta), \alpha^l(\Delta)]$, originated from Network Calculus [15], is adopted. $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ provides the upper and lower bounds on the number of arrival events for the stream S in any time interval Δ. Many other traditional timing models of event streams can be unified in the concept of arrival curves. For example, a periodic event stream can be modeled by a set of step functions where $\bar{\alpha}^u(\Delta) = \lfloor \frac{\Delta}{p} \rfloor + 1$ and $\bar{\alpha}^l(\Delta) = \lfloor \frac{\Delta}{p} \rfloor$. For a sporadic event stream with minimal inter arrival distance p and maximal inter arrival distance p', the upper and lower arrival curve is $\bar{\alpha}^u(\Delta) = \lfloor \frac{\Delta}{p} \rfloor + 1$, $\bar{\alpha}^l(\Delta) = \lfloor \frac{\Delta}{p'} \rfloor$, respectively. Moreover, a widely used model to specify an arrival curve is the PJD model, where the arrival curve is characterized by period p, jitter j, and minimal inter arrival distance d. In PJD model, the upper arrival curve can be determined as $\bar{\alpha}^u(\Delta) = \min\{\lceil \frac{\Delta + j}{p} \rceil, \lceil \frac{\Delta}{d} \rceil\}$. Fig. 2 depicts arrival curves for the above cases.

Analogous to arrival curves that provide an abstract event stream model, a tuple $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$ defines an abstract resource model which provides an upper and lower bounds on the available resources in any time interval Δ. Further details are referred to [22]. Note that arrival curves are event-based which specifies the number of the events of the steam in one interval time, while service curves are based on the amount of computation time. Therefore, service curve β has to be transformed to $\bar{\beta}$ to indicate the number of the events of the stream that processor can processed in specified interval time. Suppose that the execution time of an event is c, the transformation of the service curves can be done by $\bar{\beta}^l = \lfloor \frac{\beta^l}{c} \rfloor$ and $\bar{\beta}^u = \lfloor \frac{\beta^u}{c} \rfloor$. With these definitions, a processor with lower service curve $\bar{\beta}^{Gl}(\Delta)$ is said to satisfy the deadline D for the event stream specified by $\alpha^u(\Delta)$, if the following condition holds.

$$\bar{\beta}^{Gl}(\Delta) \geq \alpha^u(\Delta - D), \forall \Delta \geq 0 \qquad (9)$$

D. Problem Statement

This paper explores how to use dynamic power management in runtime manner to effectively minimize the energy consumption for coarse-grained pipelined multiprocessor systems under hard real-time requirement. Intuitively, energy saving can be achieved by (a) tuning as many as possible processors in the pipeline to sleep mode, and (b) keeping each processor staying at sleep mode as long as possible. However, according to the definition of break-even time T_{BET}, switching from/to

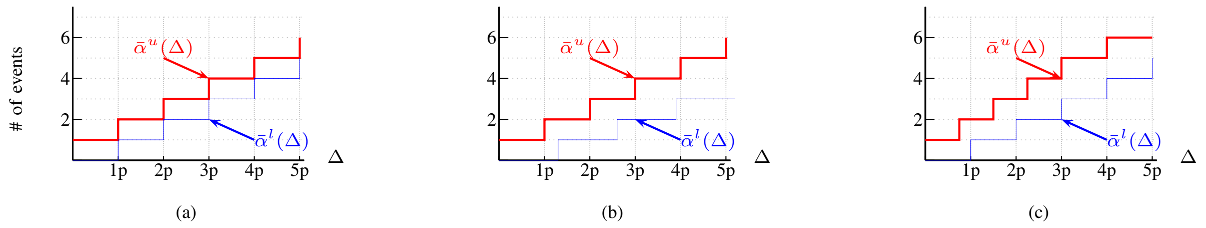


Fig. 2. Examples for arrival curves, where (a) periodic events with period p, (b) events with minimal inter-arrival distance p and maximal inter-arrival distance $p' = 1.3p$, and (c) events with period p, jitter $j = p$, and minimal inter-arrival distance $d = 0.75p$.

the sleep mode is not worthwhile when the sleeping interval is shorter than T_{BET}. On the other hand, prolonging the sleep mode might cause the current or future events violate their timing constraints. Thus, we need to make decisions for: (a) which processors in the pipeline should be selected to switch to the sleep mode, and (b) when to turn them back to the active mode to serve events to guarantee the deadline constraints. Thus, the problem studied in this paper is defined as follows.

Given pipelined platform with m stages, an event stream $\mathcal{S}$ processed by this pipeline, and an end-to-end deadline requirement D, we are to compute a adaptive dynamic power management at each time instant to minimize the energy consumption, which decides (a) which processors in the pipeline should be selected to switch to the sleep mode, and (b) when to turn on and off the processors. At the same time, the computed dynamic power management should guarantee that the worst-case end-to-end delay of any event trace constrained by the given event stream $\mathcal{S}$ does not exceed D.

IV. Proposed Approach

This section presents one adaptive approach to reduce leakage power of pipelined multiprocessor systems, which lies in an inverse use of the extended pay-burst-only-once principle. At one time instant for scheduling decisions (i.e., turn on or turn off the processors), scheduling decision should guarantee the timing requirement of the event trace in future as well as on-going events stored in system. To effectively model the system that contains unfinished events, we propose a novel system transformation, by which the whole system can be transferred as the multi-stream system where one stream is used to maintain the event trace in the first stage and the other streams are used to model the unfinished events in system. Based on this system transformation, we can compute one service curve for the corresponding stream as a constraint for the minimal resource demand by using the extended pay-burst-only-once principle. The energy minimization problem is then formulated with respect to the resource demands for individual pipeline stages. We propose one light-weight heuristic, called balanced workload scheme (BWS), to solve this minimization problem. In this paper, balanced workload scheme (BWS) is implemented in periodic manner to regulate the delay of the stages. Finally, we discuss how to determine the size of FIFOs between processors.

A. Motivation

In contrast to the work in [3], which computes a set of periodic power management for each processor in off-line, we propose one on-line approach to minimize the energy consumption, which could adaptively switch power state of the processor in pipelined multiprocessor systems according to the current workload and the state of the stages. Compared to static approach presented in [3], our approach could achieve energy savings by the following facts.

Firstly, the *execution slack* usually occurs due to difference between worst-case assumptions made in the offline analysis and the actual online behavior of the system. The off-line approach is based on the assumption that each job executes for its worst-case execution time (WCET). However, due to the inherent variability of execution time, most of the jobs in a real scenario finish their execution earlier than their WCET c_w, thus generate *execution slack* $c_w - \bar{c}$ ($\bar{c}$ denotes the execution time in a real scenario).

Secondly, the real-time system is analyzed by the off-line approach with the assumption that the task arrives in worst-case pattern, i.e., the upper bound $\alpha^u(\Delta)$ and lower bound $\alpha^l(\Delta)$ on the number of arrival tasks for the stream S in any time interval Δ. However, this worst-case arrival pattern rarely happens in hard real-time systems. Jobs are released with a variable delay bounded by the arrival curve $\alpha(\Delta)$. For brevity, the slack generated due to this variable delays is termed as *dynamic slack*.

The slacks mentioned above can be explored and managed explicitly in our approach, leading to significant energy savings with respect to the off-line approach in [3]. By adaptively monitoring the filling level of FIFOs between processors, our approach can explicitly identify the *execution slack*. On the other hand, by using the dynamic counter techniques [14], our approach can adaptively predict the event arrival in future. This adaptive prediction scheme, which efficiently explores and manages *dynamic slack*, procrastinates the events as late as possible without violating the timing constraints.

B. Real-time Calculus Routines

1) Service Curve: Without loss of generality, a pipelined system with m heterogeneous stages ($m \geq 2$) is considered. Fig. 3 presents 3-stage pipeline as example. At one adaptive time instant, we are to regulate the delay τ_i adaptively according to the workload to reduce leakage power consumption for pipelined multiprocessor system under hard real-time constraints. For each stage, the service curve at each adaptive point can be modeled as a bounded delay function, which can be defined as follows.

$$\beta_i^{Gl}(\Delta) = \max(0, (\Delta - \tau_i)) \tag{10}$$

The transformed service curve $\bar{\beta}_i^{Gl}$ can be approximated as:

$$\bar{\beta}_i^{Gl}(\Delta) = \lfloor \frac{\beta_i^{Gl}(\Delta)}{c_i} \rfloor \geq \frac{1}{c_i}(\Delta - \tau_i - c_i) \tag{11}$$

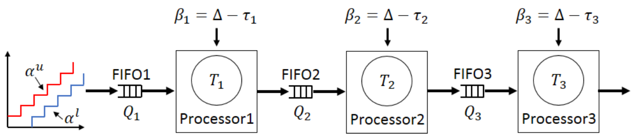


Fig. 3. 3-stage pipeline.

2) Demand Curve for Unfinished Events Stored in System: We denote the set of unfinished events at time t as $E_i(t)$. Note that although the absolute deadline $D_{i,j}$ for event $e_{i,j} \in E_i(t)$ does not change, the relative deadline is not D_i anymore. It varies according to the relative distance from t. Suppose that events in $E_i(t)$ are indexed as $e_{i,1}, e_{i,2}, ..., e_{i,|Ei(t)|}$ from the earliest deadline to the latest. The demand curve $\alpha_i^F(\Delta, t)$ for $E_i(t)$ can be defined as:

$$\alpha_i^F(\Delta, t) = \begin{cases} j, & D_{i,j} - t < \Delta \leq D_{i,j+1} - t, \\ |\boldsymbol{E_i}(t)|, & \Delta > D_{i,|\boldsymbol{E_i}(t)|} - t, \end{cases} \tag{12}$$

3) Future Prediction: To explore the *dynamic slack* efficiently, we use dynamic counter technique presented in [14] to conservatively bound the future workload. According to [14], the number of events arriving in the time interval $[t, t + \Delta]$ can be bounded tight by $\mu(\Delta, t)$, which is determined by dynamic counters. Due to the simplicity of the dynamic counter, dynamic counters can be easily implemented as part of the hardware with negligible overhead [8], [14].

C. System Transformation

In this paper, we are to find one adaptive power management scheme to reduce leakage power consumption for pipelined multiprocessor system under hard real-time constraints. However, it is not a easy task to determine adaptive power management for each adaptive time instant to guarantee hard real-time constraints for the pipeline. We take 3-stage pipeline system as an example, as shown in Fig. 3. At each time instant, there might be some unfinished events stored in system waiting for process. At the same time, the system also needs to process the new event entering to the first stage. Thus, power management decision at each time instant should guarantee the timing requirement of the new event trace in the first stage as well as unfinished events stored in system. In addition, unfinished events stored in system prevent us to adopt pay burst only once principle directly. According to [15], [3], the approach without using pay-burst-only-once principle will suffer from pessimistic result as well as costly computation.

In this section, we propose one novel system transformation, which enables us to analyze the system timing efficiently. The main idea is that the whole system is transferred as one multi-stream system where one stream is used to maintain the event trace in the first stage and the other streams are used to model the workload for the unfinished events. For $m-$stage system, unfinished events in stage p_i $(2 \leq i \leq m)$ can be represented by one special leaky-bucket stream S_i with $\alpha(\Delta) = b_i + r_i \cdot \Delta$, where the burst b_i is the number of the unfinished events (i.e., the events stored in $FIFO_i$ and the event processing on stage p_i), denoted by $Q_i + 1$ in Fig. 4, and leaky rate r_i is 0. The stream S_1 in the first stage can be represented as the predicted arrival curve $\mu(\Delta, t)$ pulsed with the one burst $Q_1 + 1$ represented by unfinished events in stage p_1. As shown in Fig. 4, at each adaptive time instant, the 3-stage pipeline system can be transferred as the system with 3 streams (i.e., S_1, S_2, and S_3). With this system transformation, we can compute one service curve for the corresponding stream as a constraint for the minimal resource demand by using the extended pay-burst-only-once principle (See Section IV-D).

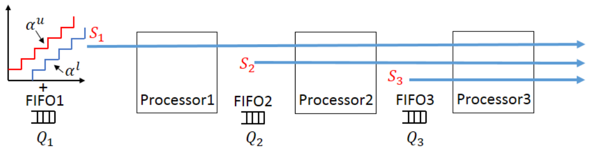


Fig. 4. System transformation.

D. Problem Formulation

After system transformation, the pipeline at each adaptive point can be represented as an aggregate scheduling system. Then, a new extended pay-burst-only-once principle is proposed to compute the end-to-end service curve for stream S_i in an aggregate scheduling system. Based on this new extended pay-burst-only-once principle, the energy minimization problem is then formulated with respect to the resource demands for individual pipeline stages at each adaptive point.

In [7], the extended pay-burst-only-once principle has been extended to compute the end-to-end service curve for stream S_i in an aggregate scheduling *only* for the case with FIFO service curve elements and single leaky bucket constrained arrival curve. However, the original arrival curve bound $\alpha(\Delta)$ and the predicted arrival curve $\mu(\Delta, t)$ at the adaptive time instant t may not be constrained as the form of leaky bucket in the real world. In addition, the events are scheduled as non-preemptive first-come-first-serve manner in this system, not with FIFO service curve. Thus, the approach presented in [7] cannot be applied. In this paper, a new extended pay-burst-only-once principle is proposed to derive end-to-end service curve for stream S_i for the transferred multi-stream system.

Lem. 1: At one adaptive time instant, the adaptive m-stage system can be represented as m-stream system. The end-to-end service curve $\beta_i(\Delta)$ for stream S_i $(i = 1, ..., m)$ with service curve elements with the rate-latency format $\beta^{R,T}$ and arrival curves $\alpha(\Delta)$, can be computed by:

$$\beta_i(\Delta) = \min_{j=i}^{m}(R_j) \cdot (\Delta - \sum_{j=i}^{m} T_j - \sum_{j=i+1}^{m} \frac{Q_j + 1}{\min_{k=j}^{m}(R_k)}) \tag{13}$$

where Q_j is the number of the stored events of $FIFO_j$ at the one adaptive time instant.

Proof: According to the derivations in [7], we obtain (14) for the stream S_i for the time pairs $t_{j+1}-t_j \geq 0$ in general sense.

$$R_i^{m+1}(t_{m+1}) - R_i^1(t_1) \geq \sum_{j\in\jmath_i} \beta^{R_j,T_j}(t_{j+1}-t_j) - \sum_{k\in\kappa_i}\sum_{j\in\jmath_{i,k}} (R_k^{j+1}(t_{j+1}) - R_k^j(t_j)) \tag{14}$$

where R_i^j denotes the arrival function of stream S_i on the stage j (the case of $j = m+1$ indicates the output of the stream), $\jmath_i$ denotes the stage set that the stream S_i goes through (i.e., the path of the stream S_i), κ_i denotes the interference stream set that uses the complete path or some part of the path of the stream S_i, ,and $\jmath_{i,k}$ denotes all stages of a sub-path that are passed by both stream S_i and stream S_k ($k > i \geq 1$).

According to system transformation and non-preemptive first-come-first-serve schedule, the stream S_i is interferenced by the stream S_j when $j \geq i+1$ holds. Thus, we have (15).

$$\kappa_i = \{S_{i+1}, \cdots, S_m\} \tag{15}$$

In addition, according to system transformation, the stream S_i goes through from the ith stage p_i to the final stage p_m. Thus, we can obtain (16) and (17).

$$\jmath_i = \{p_i, \cdots, p_m\} \tag{16}$$

$$\jmath_{i,k} = \{p_k, \cdots, p_m\} \tag{17}$$

At one adaptive point, the stage may have already fetched one event from its FIFO and started to process this event. Consider this case, we can derive (18) with (17) and $k \geq i+1$.

$$\sum_{j\in\jmath_{i,k}} R_k^{j+1}(t_{j+1}) - R_k^j(t_j) = R_k^{m+1}(t_{m+1}) - R_k^k(t_k) \leq Q_k + 1 \tag{18}$$

Similar to the derivation in [7], with (15), (16) and (18), the $\inf_{(t_{j+1}-t_j>0)|j\in\jmath_i}$ of (14) can be derived to the form that is given in (13). □

By using the approximated lower bound of (11), we can state below theorem with Lem. 1.

Thm. 1: At each adaptive time instant t, assume demand curve for future events arrival and unfinished events stored in stage p_i in m stage pipeline system under end-to-end deadline D constraint can be defined as $\mu(\Delta - D, t)$ and $\alpha_i^F(\Delta,\, t)$, respectively. The pipelined system satisfies an end-to-end deadline D, if the following condition holds for each stream S_i ($1 \leq i \leq m$).

$$\min_{j=i}^{m}(\frac{1}{c_j}) \cdot (\Delta - \sum_{j=i}^{m}(\tau_j + c_j) - \sum_{j=i+1}^{m}\frac{Q_i+1}{\min\limits_{k=j}^{m}(\frac{1}{c_k})}) \geq \alpha_{S_i}^D(\Delta, t) \tag{19}$$

where $\alpha_{S_i}^D(\Delta)$ is the demand curve for S_i, which can be defined as:

$$\alpha_{S_i}^D(\Delta, t) = \begin{cases} \mu(\Delta - D, t) + \alpha_i^F(\Delta,\, t), & i = 1, \\ \alpha_i^F(\Delta,\, t), & i \geq 2, \end{cases} \tag{20}$$

Proof: By using the approximated lower bound of (11), the lower bound of end-to-end service curve $\beta_i^{Gl}(\Delta)$ for stream S_i ($i = 1, ..., m$) can be derived as the form of the right hand side of (19) according to Lem. 1. For each stream, (19) can guarantee the end-to-end deadlines of the future events as well as the stored events are no more than D. □

For each stream S_i, (19) needs to be satisfied to guarantee the deadline of both the future events and the stored events. The lower bound of end-to-end service curve $\beta_i^{Gl}(\Delta)$ for the stream S_i (i.e., the left hand side of the inequality (19)) can be considered as a bounded-delay function $bdf_i(\Delta, \rho_i, b_i) = max(0, \rho_i(\Delta - b_i))$ with slope $\rho_i = \min_{j=i}^{m}(\frac{1}{c_j})$ and bounded-delay $b_i = \sum\limits_{j=i}^{m}(\tau_j + c_j) + \sum\limits_{j=i+1}^{m}\frac{Q_i+1}{\min\limits_{k=j}^{m}(\frac{1}{c_k})}$. To satisfy the inequality (19), the maximum bounded-delay can be determined as follows.

$$b_{S_i}^{max} = \sup\{b : bdf_i(\Delta, \rho_i, b) \geq \alpha_{S_i}^D(\Delta, t), \forall\Delta \geq 0\} \tag{21}$$

Thus, the end-to-end deadline constraints can be formulated as (22). The constraint set (22) can be organized as form of triangle. For stream S_i, the constraint has $m+1-i$ variables in the left hand of (22).

$$\forall S_i : \sum_{j=i}^{m}\tau_j \leq b_{S_i}^{max} - \sum_{j=i}^{m}c_j - \sum_{j=i+1}^{m}\frac{Q_i+1}{\min\limits_{k=j}^{m}(\frac{1}{c_k})} \tag{22}$$

The right hand side of (22) is constant. For brevity, for the rest of the paper, the right hand side of (22) is denoted as $\lambda(S_i)$.

The *deadline constraint* set (22) with triangle form can guarantee the deadline requirements for the pipeline. In next step, one decision should be made to decide which processor can switch to sleep state and which cannot. Due to the definition of the break even time T_{BET}, the current state of the processors should be taken into account to make such decision. If the processor currently stays at the active or idle state, the processor can be selected to enter sleep state only when $\tau \geq T_{BET}$ holds. If the processor currently stays at sleep state, the processor can stay at sleep state without any constraints. Therefore, we need to consider the current state of stages to make the decision.

At one adaptive time instant, the stage can be divided into active set Φ_a and sleep set Φ_s according to the current state that the stages are in. Active set Φ_a and sleep set Φ_s denote the stage set that stay at active state (or idle state) and the stage set that stay at sleep state, respectively. Let binary variable e_i denote the state switch decision for active stage set Φ_a in the adaptive time instant: $e_i = 1$ if the stage p_i ($p_i \in \Phi_a$) switch from active state to sleep state and 0 otherwise. The binary variable e_i depends on the break even time constraints: (a) $T_{BET} \leq \tau_i \Rightarrow e_i = 1$; (b) $T_{BET} > \tau_i \Rightarrow e_i = 0$. The sleep interval of the active stage can be formulated as $\overline{\tau}_i = e_i \cdot \tau_i$, which is a quadratic item. For brevity, we call these constraints as *active set constraints*

For the sleep set Φ_s, the on-going sleep interval τ_i^0, i.e., the duration from the point of entering sleep state to current point, may not be greater than T_{BET}. Thus, to make the current decision be consistent to the former decision, processors need to stay at sleep mode for more than $\max(T_{BET} - \tau_i^0, 0)$. We call this additional sleep interval consistency interval $CI_i = \max(T_{BET} - \tau_i^0, 0)$. The *consistency constraints* can be defined as:

$$\forall p_i \in \Phi_s : \tau_i \geq CI_i \tag{23}$$

At each adaptive time instant, we are to greedily maximize the total sleep interval of all processors. The objective function can be formulated as:

$$\sum_{p_i \in \Phi_a} \bar{\tau}_i + \sum_{p_i \in \Phi_s} \tau_i \tag{24}$$

Up to now, the adaptive power management problem can be formulated as a set of optimization problem at different time instant, which maximizes the objective (24) with respect to *deadline constraints*, *active set constraints*, and *consistency constraints*.

Above optimization problem contains quadratic item $\bar{\tau}_i = e_i \cdot \tau_i$ and integer variables. Solving such a mixed integer quadratic programming (MIQP) problem is quite time-consuming. To address this issue, one light weight scheme, called balance workload scheme (BWS), is presented in Section IV-E to maximize the total sleep interval of all processors while guaranteeing the hard real-time constraints.

E. Proposed Heuristic

In this section, we present one fast heuristic, called balance workload scheme (BWS), to find sub-optimal solution at each adaptive time instant. At each adaptive time instant, balance workload scheme (BWS) tries to find the first-step decision to turn as many as possible processors into sleep mode. Based on the first-step decision, we make the second-step decision to keep the processor sleeping as long as possible.

1) Determine Sleep Stages: For the stage in active set Φ_a in current point, we need to determine which stage will enter into sleep stage and which stage will stay at active state, i.e., *active set constraints*. For the stage in sleep set Φ_s, sleep interval should be more than CI_i to make the current decision be consistent to the former decision, i.e., *consistency constraints*. As the first-step decision, we are to determine which stages in Φ_a should switch to sleep mode. This decision tries to switch as many as possible processors to sleep mode to balance the workload among processors.

The pseudo code of the algorithm is depicted in Algo. 1, which determines sleep stages set π_s and active stages set π_a. To satisfy constraints for sleep set Φ_s in (23), sleep intervals τ_i^s for stages in Φ_s are initially assigned to CI_i and later on might be prolonged by Algo. 2. Thus, we assign sleep decision variable e_i as 1 and put sleep set Φ_s into π_s at this moment (Line 2–Line 5 in Algo. 1). Then, (22) can be updated as (25).

$$\forall S_i : \sum_{p_j \in \Phi_a \cap \jmath_i} \tau_j \leq \lambda(S_i) - \sum_{p_j \in \Phi_s \cap \jmath_i} \tau_j \tag{25}$$

Note that, for multiple inequality (25) with the same variables (i.e., the same set $\Phi_a \cap \jmath_i$), the constants in the right hand of different inequality (25) can be merged by min operation. With this operation, multiple inequality (25) with the same variables can be merged as one inequality. Similar to (22), the constraint set (25) can also be organized as form of triangle (Line 7 in Algo. 1). We denote $\bar{\lambda}(\Phi_a, i)$ as the constant in the the right hand of i variables inequality (25) and $\Phi_{a,i}$ as the corresponding stage subset with i variables. For example, assume 4-stage pipeline with $\Phi_a = \{p_1, p_3, p_4\}$ and $\Phi_s = \{p_2\}$, the constraint set (25) is organized as form of triangle and is represented as follow.

$$\tau_4 \leq \bar{\lambda}(\Phi_a, 1) = \lambda(S_4)$$
$$\tau_3 + \tau_4 \leq \bar{\lambda}(\Phi_a, 2) = \min(\lambda(S_3), \lambda(S_2) - CI_2)$$
$$\tau_1 + \tau_3 + \tau_4 \leq \bar{\lambda}(\Phi_a, 3) = \lambda(S_1) - CI_2$$

For the constraint (25) with i variables, the number of the processor we can turn off can be bounded as (26). We denote this ith bound as $\psi(\Phi_a, i)$.

$$\sum_{p_j \in \Phi_{a,i}} e_j \leq \min(\left\lfloor \frac{\bar{\lambda}(\Phi_a, i)}{T_{BET}} \right\rfloor, i) \tag{26}$$

According to the triangle form constraint set (26), we first find the minimum bound $\psi_m(\Phi_a)$ and the corresponding index $I_m(\psi_m)$ (Line 8 in Algo. 1). Then, active set Φ_a can be divided into two part $\Phi_{a,I_m(\psi_m)}$ and $\Phi_a - \Phi_{a,I_m(\psi_m)}$. For stages in $\Phi_{a,I_m(\psi_m)}$, we need to select $\psi(\Phi_a, I_m(\psi_m))$ stages to turn off. To balance the workload between stages, the backlog Q_i is considered as the priority to select the stage to turn off. The smaller Q_i is, the higher priority to turn off the stage has. With this criterion, we select $\psi(\Phi_a, I_m(\psi_m))$ stages $\Phi^H_{a,I_m(\psi_m)}$ with higher priority to put into π_s and the remaining part $\Phi_{a,I_m(\psi_m)} - \Phi^H_{a,I_m(\psi_m)}$ with lower priority to put into π_a. (Line 9 and Line 10 in Algo. 1). Note that the constraints (26) with less than $I_m(\psi_m)$ variables are always satisfied due to $\psi_{i \geq I_m(\psi_m)}(\Phi_a, i) \geq \psi(\Phi_a, I_m(\psi_m))$.

Algorithm 1 Determine Sleep Stages

```
Input: Active Set $\Phi_a$, Sleep Set $\Phi_s$, Consistency Constant
    Set $CI$, Deadline Constant Set $\lambda$
Output: Sleep Stages $\pi_s$, Active Stages $\pi_a$
1:  $\pi_s \leftarrow \phi$; $\pi_a \leftarrow \phi$;
2:  for Stage $p_i \in \Phi_s$ do
3:      $\tau_i \leftarrow CI_i$;
4:      $\pi_s \leftarrow \{\pi_s, p_i\}$;
5:  end for
6:  while $\Phi_a \neq \phi$ do
7:      Update constant set $\bar{\lambda}(\Phi_a, i)$ and $\psi(\Phi_a, i)$ according
        to (25) and (26);
8:      Find $\psi_m(\Phi_a) = \min_{i=1}^{|\Phi_a|}(\psi(\Phi_a, i))$ and the corresponding
        index $I_m(\psi_m)$;
9:      $\pi_s \leftarrow \{\pi_s, \Phi^H_{a,I_m(\psi_m)}\}$;
10:     $\pi_a \leftarrow \{\pi_a, \Phi_{a,I_m(\psi_m)} - \Phi^H_{a,I_m(\psi_m)}\}$;
11:     Update $\Phi_a \leftarrow \Phi_a - \Phi_{a,I_m(\psi_m)}$ ;
12: end while
```

2) Prolong Sleep Interval: In the next step, we need to determine the sleep interval for each stages based on the decision $\{\pi_a, \pi_s\}$ obtained from Algo. 1. For the stages in π_a, we assign the delay τ_i as 0 directly (Line 2 in Algo. 2). For the stages in π_s, the delay τ_i is represented as $\tau_i = \tau_i^0 + \tau_i^e$, where τ_i^0 denotes the initial delay values to guarantee the feasibility of the constraints, which can be determined by the routine in Algo. 2 (i.e., Line 4 to Line 10), and τ_i^e denotes the extended delay. The constraints (22) can be updated as (27).

$$\forall S_i : \sum_{p_j \in \pi_s \cap \jmath_i} \tau_j^e \leq \lambda(S_i) - \sum_{p_j \in \pi_s \cap \jmath_i} \tau_j^0 \qquad (27)$$

Similar to (25), the constraint set (27) can also be organized as form of triangle. We denote $\bar{\lambda}^e(\pi_s, i)$ as the constant in the the right hand of i variables inequality (27) and $\pi_{s,i}$ as the corresponding stage subset with i variables. Using the similar scheme in Algo. 1, we firstly find the minimum constant $\bar{\lambda}^e_m(\pi_s)$ and the corresponding index $I_m(\bar{\lambda}^e_m)$ (Line 13 in Algo. 2). Then, stage set π_s can be divided into two part $\pi_{s,I_m(\bar{\lambda}^e_m)}$ and $\pi_s - \pi_{s,I_m(\bar{\lambda}^e_m)}$. For stages in $\pi_{s,I_m(\bar{\lambda}^e_m)}$, the total delay $\bar{\lambda}^e_m(\pi_s)$ is assigned equally among τ_j^e.

Algorithm 2 Prolong Sleep Interval

Input: Active Set Φ_a, Sleep Set Φ_s, Consistency Constant Set CI, Deadline Constant Set λ, Sleep Stages π_s, Active Stages π_a

Output: Sleep Interval Set τ

```
 1: for Stage $p_i \in \pi_a$ do
 2:     $\tau_i \leftarrow 0$;
 3: end for
 4: for Stage $p_i \in \pi_s$ do
 5:     if $p_i \in \Phi_a$ then
 6:         $\tau_i^0 \leftarrow T_{BET}$;
 7:     else
 8:         $\tau_i^0 \leftarrow CI_i$;
 9:     end if
10: end for
11: while $\pi_s \neq \phi$ do
12:     Update constant set $\bar{\lambda}^e(\pi_s, i)$ according to (27);
13:     Find $\bar{\lambda}^e_m(\pi_s) = \min_{i=1}^{|\pi_s|}(\bar{\lambda}^e(\pi_s, i))$ and the corresponding index $I_m(\bar{\lambda}^e_m)$;
14:     Assign delay $\bar{\lambda}^e(\pi_s, i)$ equally to $\tau_j^e$ among stages in $\pi_{s,I_m(\bar{\lambda}^e_m)}$;
15:     $\pi_s = \pi_s - \pi_{s,I_m(\bar{\lambda}^e_m)}$;
16: end while
```

3) Put It All Together: Based on Algo. 1 and Algo. 2, balanced workload scheme (BWS) can be represented as Algo. 3. At one adaptive time instant, balanced workload scheme (BWS) can make the decision, which can make as many as possible stages enter sleep state and make them stay at sleep state as long as possible. The feasibility of the decision made by balanced workload scheme (BWS) at each adaptive instance can be guaranteed by Lem. 2. For simplicity, we implement balanced workload scheme (BWS) in periodic manner in this paper. Note that balanced workload scheme can also be easily extended to be executed in other manners, e.g., event-triggered manner.

Lem. 2: At one adaptive time instant, sleep interval τ obtained from Algo. 3 is feasible.

Proof: The solution obtained from Algo. 1 conforms to the triangle form constraints (26). Thus, the solution obtained from Algo. 1 is feasible. In Algo. 2, the initial delay value τ_i^0 can guarantee the feasibility of the sleep decision obtained from Algo. 1, i.e., consistence constraint and break-even constraint. The while loop (Line 11 to Line 16 in Algo. 2) can guarantee the extended delay τ_i^e conforms to end-to-end deadline (22) accoding to the definition of (27). □

Algorithm 3 Balanced Workload Scheme (BWS)

Input: Active Set Φ_a, Sleep Set Φ_s, Consistency Constant Set CI, Deadline Constant Set λ

Output: Sleep Interval Set τ

```
1: Obtain the sleep decision $\pi_s$ and $\pi_a$ according to Algo. 1;
2: Obtain sleep interval $\tau$ according to Algo. 2;
```

TABLE I
CONSTANTS FOR 70NM TECHNOLOGY [16], [25].

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

F. Worst Case FIFOs Size Analysis

To prevent the FIFO overflow during the implementation of the proposed dynamic power management, we need to analyze the worst case FIFO size requirement in off-line. This information could guild us to assign share memory between stages before the balance workload scheme is implemented in run-time and avoid the FIFO overflow. For ith stage in m stages pipeline system, the worst case FIFO size can be obtained by turning on the other stages all the time and procrastinating ith stage as late as possible. According to (21), the maximum bounded-delay can be determined and then the service curve of ith stage can also be determined. Using the analysis approach in [24], the worst-case FIFOs size can be determined.

V. EXPERIMENTAL EVALUATIONS

This section provides simulation results for the proposed adaptive dynamic power management scheme. The pipeline simulator is implemented in MATLAB using RTC-toolbox from [23]. We implement balanced workload scheme (BWS) in periodic manner and set the activation period as 5ms.

A. Simulation Setup

The experiments are conducted based on classical energy model of 70nm technology processor in [16], [25], [13], whose accuracy has been verified with SPICE simulation. Tab. I lists the energy parameter under 70nm technology [16], [25], [13]. According to [13], executing at $V_{dd} = 0.7V$ is more energy efficient than executing at lower voltages levels. To achieve the minimize the overall energy consumption of the system, we assume that the processor runs at this critical frequency level when the processor is in the active state. From [25], [13], body bias voltage V_{bs} is obtained as $-0.7V$. From [13], P_{on} related to idle power can be obtained as $100mW$ and

TABLE II
POWER PARAMETERS

V_{dd}	P_a	P_s	P_σ	E_{sw}	t_{sw}
$0.7V$	$656mW$	$390mW$	$50\mu W$	$483\mu J$	$10ms$

the power consumption in sleep mode P_σ is set as $50\mu W$. According to energy model in Section III-B, we can calculate the corresponding active power P_a and stand-by power P_s under voltage level $V_{dd} = 0.7V$. In [13], we can obtain energy overhead E_{sw} of state transition as $483\mu J$. We set time overhead t_{sw} of state transition as $10ms$. Tab. II lists all the power parameters used in the experiment.

The H.263 decoder shown in Fig. 1(a) is used as the test application. The execution time of each subtask in H.263 decoder application can refer from [18]. An event stream is specified by the PJD model with period p, jitter j, and minimal inter-arrival distance d. The period p and the jitter j of the H.263 decoder application are respectively set as $100ms$ and $150ms$ with varying the end-to-end deadline. The relative deadline D of the stream is defined as $D = \gamma \cdot p$ and varies according to the deadline factor γ. To compare the impact of different algorithms, we simulate traces with a 10sec time span. The traces are generated by the RTC tools [23] and conformed to the arrival curve specifications. It is worthy noting that a worst-case execution time c_w is associated with the transformed service curve, as stated in Section III-C. To model the variability of execution time of the tasks, the actual execution time c_a are randomly selected from $[c_b, c_w]$, where the best-case execution time c_b is defined as $c_b = \alpha \cdot c_w$ and varies according to the execution time factor α.

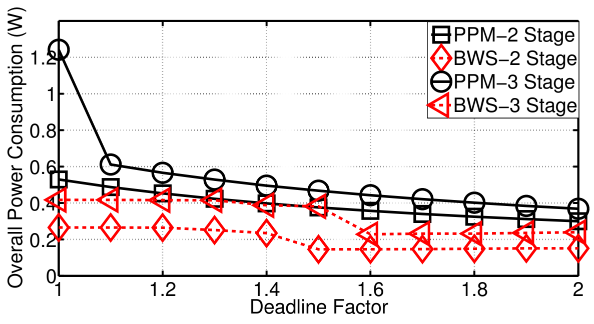


Fig. 5. Overall power consumption under different end-to-end deadline constraint .

B. Results

Firstly, we show the effectiveness of the proposed adaptive power management scheme (BWS) comparing to the periodic power management scheme (PPM) in [3] under different end-to-end deadline constraint. Cases of 2-stage and 3-stage pipeline architectures with homogeneous 70-nm processors are evaluated. To demonstrate how the proposed scheme can effectively explore *dynamic slacks*, we firstly remove the variability of the execution time of the tasks by setting the factor α as 1. We vary the deadline factor γ from 1 to 2 with step 0.1. The simulation results are shown in Fig. 5. From Fig. 5, we can make the following observations: (1) The overall power consumptions of both scheme decrease as the end-to-end deadline increases. This is expected because the loose end-to-end deadline requirement could create more opportunities of entering the sleep state and longer sleep time. (2) Adaptive power management scheme (BWS) outperforms periodic power management scheme (PPM) on both pipeline architectures. In case of no *execution slack*, adaptive power management scheme (BWS) can on average achieve 50% and 35% energy savings on 2-stage pipeline and 3-stage pipeline, respectively. This indicates our approach can efficiently explore *dynamic slack* to achieve energy savings comparing to periodic power management scheme (PPM).

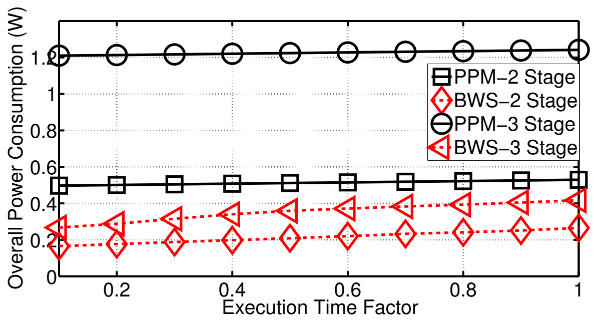


Fig. 6. Overall power consumption with the variability of execution time.

Next, we show how the proposed power management scheme can efficiently explore *execution slack*. H.263 application with deadline factor $\gamma = 1$ is evaluated in 2-satge and 3-stage pipeline architectures with homogeneous 70-nm processors. We vary execution time factor α from 0.1 to 1 with step 0.1. The smaller α is, the more variable the execution time of task is. We randomly generate different actual execution time for each event and then put them into simulator. Fig. 6 shows how the overall power consumption changes when execution time factor α varies. As shown in Fig. 6, power consumption of the proposed approach (BWS) react with the variability of execution time of the tasks. The increment of α, which indicates the variability spaces of execution time of the tasks decrease, will result in the increment of overall power consumption of the proposed approach. This is caused by the fact that the filling level of FIFOs among processors could response to this variability of execution time of tasks. By adaptively monitoring the filling level of FIFOs (See (19) in Thm. 1), our approach can explicitly identify the *executionn slack* to achieve the energy savings. Comparing to the case $\alpha = 1$, the proposed approach can achieve 21% and 13% energy savings at $\alpha = 0.5$ for 2-stage and 3-stage pipeline, respectively. In addition, we can observe periodic power management scheme (PPM) fails to response to the variability of execution time in both architectures. Comparing to the case $\alpha = 1$, PPM can only achieve 3% and 1% energy savings at $\alpha = 0.5$ for 2-stage and 3-stage pipeline, respectively.

Finally, we demonstrate the efficiency of the proposed schemes by reporting overall power consumption and the com-

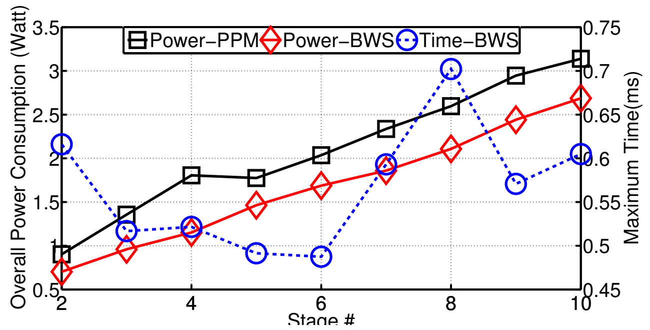


Fig. 7. Power consumption and computation time.

putation time for different pipeline architectures. We test our approaches by up to 10-stage heterogeneous pipeline by using obove stream setting and collect the maximum computation time for each architecture. The worst case execution time of subtasks mapped on each stage are randomly generated between 20ms and 40ms. We set the execution time factor α as 0.5. The end-to-end deadline for the test case with different stage number is determined by $n \cdot 60$, where n is the stage number. Fig. 7 shows the power consumption and the maximum computation time on different architectures. We can see that the proposed approach (BWS) outperforms PPM approach. In addition, as shown in the figure, the proposed approach (BWS) require a small computation time (less than 0.75ms), which makes our algorithms applicable online.

VI. Conclusion

This paper presents one adaptive power management approach to reduce the leakage power consumption for pipelined systems. Targeting the streaming application with non-deterministic workload arrivals under hard real-time constraints, the proposed approach adaptively regulates the delay of the processors according to the workload while guarantee the end-to-end deadline requirement. In addition, the proposed approach can efficiently explore the slacks generated at runtime to achieve energy savings. Simulation results demonstrate the effectiveness of our approaches.

References

[1] A. Alimonda, S. Carta, A. Acquaviva, A. Pisano, and L. Benini. A feedback-based approach to dvfs in data-flow applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2009.

[2] S. Carta, A. Alimonda, A. Pisano, A. Acquaviva, and L. Benini. A control theoretic approach to energy-efficient pipelined computation in mpsocs. *ACM Transactions on Embedded Computing Systems*, 2007.

[3] G. Chen, K. Huang, C. Buckl, and A. Knoll. Energy optimization with worst-case deadline guarantee for pipelined multiprocessor systems. In *Design, Automation and Test in Europe (DATE)*, 2013.

[4] 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*, 2014.

[5] J.-J. Chen and T.-W. Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *Proceedings of the 2007 IEEE/ACM International Conference on Computer-aided Design (ICCAD)*, 2007.

[6] J.-J. Chen, N. Stoimenov, and L. Thiele. Feasibility analysis of on-line dvs algorithms for scheduling arbitrary event streams. In *RTSS 2009: Proceedings of the 2009 30th IEEE Real-Time Systems Symposium (RTSS)*, 2009.

[7] M. Fidler. Extending the network calculus pay bursts only once principle to aggregate scheduling. In *Quality of Service in Multiservice IP Networks*. 2003.

[8] K. Huang, G. Chen, C. Buckl, and A. Knoll. Conforming the runtime inputs for hard real-time embedded systems. In *Proceedings of the 49th Design Automation Conference (DAC)*, 2012.

[9] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. Buttazzo. Adaptive dynamic power management for hard real-time systems. In *2009 30th IEEE Real-Time Systems Symposium (RTSS 2009)*, 2009.

[10] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. Buttazzo. Periodic power management schemes for real-time event streams. In *Proceedings of the 48th IEEE International Conference on Decision and Control (CDC)*, 2009.

[11] H. Javaid, M. Shafique, J. Henkel, and S. Parameswaran. System-level application-aware dynamic power management in adaptive pipelined mpsocs for multimedia. In *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2011.

[12] H. Javaid, M. Shafique, S. Parameswaran, and J. Henkel. Low-power adaptive pipelined mpsocs for multimedia: An h.264 video encoder case study. In *Proceedings of 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2011.

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

[14] K. Lampka, K. Huang, and J.-J. Chen. Dynamic counters and the efficient and effective online power management of embedded real-time systems. In *Proceedings of the 9th IEEE/ACM International Conference on the Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2011.

[15] J. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer, 2001.

[16] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw. 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)*, 2002.

[17] S. Maxiaguine, A. Chakraborty and L. Thiele. Dvs for buffer-constrained architectures with predictable qos-energy tradeoffs. In *Proceedings of the 2005 IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2005.

[18] H. Oh and S. Ha. Hardware-software cosynthesis of multi-mode multi-task embedded systems with real-time constraints. In *Proceedings of 10th International Symposium on Hardware/Software Codesign (CODES+ISSS)*, 2002.

[19] S. Perathoner, K. Lampka, N. Stoimenov, L. Thiele, and J.-J. Chen. Combining optimistic and pessimistic dvs scheduling: An adaptive scheme and analysis. In *Proceedings of the 2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2010.

[20] S. L. Shee, A. Erdos, and S. Parameswaran. Heterogeneous multiprocessor implementations for jpeg: a case study. In *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis (CODES+ISSS)*, 2006.

[21] S. L. Shee and S. Parameswaran. Design methodology for pipelined heterogeneous multiprocessor system. In *Proceedings of the 44th annual Design Automation Conference (DAC)*, 2007.

[22] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proceedings of the 2000 IEEE International Symposium on Circuits and Systems*, 2000.

[23] E. Wandeler and L. Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.

[24] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse. System architecture evaluation using modular performance analysis - A case study. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(6):649–667, Oct. 2006.

[25] W. Wang and P. Mishra. Leakage-aware energy minimization using dynamic voltage scaling and cache reconfiguration in real-time systems. In *the 23rd International Conference on VLSI Design (VLSID)*, 2010.

[26] Y. Yu and V. Prasanna. Power-aware resource allocation for independent tasks in heterogeneous real-time systems. In *Proceedings of 9th International Conference on Parallel and Distributed Systems (ICPADS)*, 2002.

[27] Y. Zhang, X. S. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *Proceedings of the 39th annual Design Automation Conference (DAC)*, 2002.