Minimizing Expected Energy Consumption for Streaming Applications With Linear Dependencies on Chip Multiprocessors

Ahmed Abousamra, Rami Melhem, Daniel Mossé
University of Pittsburgh
Computer Science Department
{abousamra, melhem, mosse}@cs.pitt.edu

***Abstract*—Dynamic voltage scaling (DVS) is a widely applied power management mechanism in real-time systems. We propose an algorithm for scheduling periodic hard real-time streaming applications with linear dependencies and known probability distributions of computational requirements on chip multiprocessors (CMP). The goal of the scheduling is to minimize the *expected* energy consumption while satisfying two quality of service (QoS) requirements: throughput and response time. Our experiments show significant energy savings (up to 55%) over scheduling when only the worst case computational requirements are known. In addition, while dynamically reclaiming processor idle time across multiple processors yields small benefit when scheduling is based on the probability distribution of computational requirements, it results in significant energy savings when scheduling for the worst case, especially for applications with short deadlines.**

I. Introduction

Dynamic voltage scaling (DVS) has become a widely applied energy conservation technique in real-time systems, and many processors now implement DVS (e.g., Intel's XScale [21] and AMD's Athlon 64 [2]). In addition, power issues, as well as processor-memory speed gap and increased difficulty in extracting more instruction-level-parallelism (ILP), are making chip multiprocessors (CMP) replace single-core processors for general-purpose computing and embedded devices. For example, Intel and AMD offer quad core processors [1], [9], and Sun offers eight core processor [17].

With the number of cores on a chip expected to rise, many real-time applications are, or will soon be, running on CMPs. The problem of scheduling real-time applications on CMP or clusters of computers is not new [5], [13], [16], [25] and requires (i) mapping the application task graph to processor cores, and (ii) determining the execution speed of each task. In [23], Xu et al. considered scheduling a periodic hard real-time streaming application on a CMP with the goal of minimizing energy consumption for the ***worst case*** computational requirements, while satisfying the two QoS requirements: throughput and response time.

Workloads, however, often exhibit large variability in computational requirements [6], [15]. Therefore, scheduling for the worst case execution requirements misses the opportunity of achieving potentially greater energy savings. We are motivated by this observation to consider the same problem, but in addition use knowledge of the tasks' probability distribution of computational requirements to minimize the ***expected*** energy consumption while also satisfying the same two QoS requirements. Specifically, we consider scheduling periodic hard-real time application with inter-arrival time T (throughput requirement is $\frac{1}{T}$), deadline (response time) D, and linear precedence constraints, i.e., linear task graph, on a CMP, where we can visualize the processors as stages of an execution pipeline. We are interested in $T \leq D$ since it permits using one or more pipeline stages (processors). Our work on linear task graphs is already complex and is a first step towards CMP scheduling of general task graphs.

An algorithm of particular interest to us is Practical Inter-Task DVS (PITDVS) [24]. It schedules a set of periodic hard real-time tasks that share the same period, T, and deadline, D, and having $D = T$, to run on a single processor using knowledge of the probability distributions of the computational requirements of the tasks. The goal of PITDVS is to schedule the tasks so as to minimize the ***expected*** energy consumption. Because PITDVS only solves the problem when $T = D$, but not when $T < D$, our initial attempt was to develop an algorithm, which we call Extended PITDVS (EPITDVS) (see section V-A), that uses PITDVS to partition the

time D among the tasks, then map (i.e., assign) them to processor cores such that the execution time on any core is at most T. Simulation results did not show significant energy savings (see Section VI). This prompted us to develop a better algorithm – using dynamic programming – which we call Probabilistic1D (see Section V-B), whose computed schedules achieved energy savings in over 99% of the experiments, and also achieved the highest energy savings in over 94% of all experiments.

The rest of the paper is organized as follows. Section II describes related work. Application and system models are described in Section III. Section IV describes previously developed related algorithms. We present our proposed algorithm in Section V. Experimental setup and results are described in Section VI. Section VII concludes the paper.

II. Related Work

DVS is an effective energy saving scheme since it allows quadratic energy savings at the expense of only a linear increase in execution time. As a result, much work in the last decade suggested different schemes for applying DVS to conserve energy while meeting application deadlines or causing a small degradation of the user interactive experience [7], [8], [11], [12], [14], [26]; most of this work is geared towards single processor. In multiprocessor and multicore systems, tasks are first mapped to cores or processors based on precedence constraints described by general [23], [27] or conditional task graphs [13], [16], [19]. Task scheduling, that is, time allocation and frequency selection, is based on either the tasks' worst case computational requirements [23], [27], or on the probability distribution of their computational requirements [4], [20], [24]. For independent tasks, where any task can be assigned to any processor, the work in [28] reclaims the time unused by a task to reduce the execution speed of future tasks, a technique we refer to as *cross-stage idle time reclamation* (see Section V-C).

Although we use the probability distribution of computational requirements of the application tasks to do the mapping and scheduling of tasks to cores, our problem differs from prior work in that we not only have a response time constraint but also a throughput constraint.

Closely related to our work are the scheduling algorithms GRACE [26], PACE [12], PPACE, OITDVS, PITDVS2 [24], Scheduling1D, and Scheduling2D [23]. GRACE, PACE, and PPACE are *stochastic intra-task* DVS schemes, i.e., based on knowledge of the deadline and the probability distribution of computational requirements of the *single* task running on *one* processor, dynamically change the frequency of the processor at run time to save energy while guaranteeing the task meets its deadline. OITDVS is an optimal *stochastic inter-task* DVS scheme for conserving energy for a *set* of periodic tasks, run on *one* processor, that share the same period and their deadlines are equal to the end of the period. OITDVS relies on the probability distribution of computational requirements of the tasks to allot time to each task. PITDVS2 is a practical version of OITDVS, which approximates the computed continuous frequency using two adjacent discrete processor frequencies [10]; we use PITDVS2 as part of our proposed algorithm. For periodic streaming applications with linear and general task graphs, the work in [23] proposes the Scheduling1D and Scheduling2D algorithms for scheduling the tasks on a CMP with the goal of minimizing the *worst case* energy consumption of the application.

III. Models and Problem Definition

a) Application Model: We consider a periodic application composed of n tasks, $\tau_1, ..., \tau_n$, with linear precedence constraints. The inter-arrival time (period) of the application is T, and deadline is D. The application can be modeled as a directed linear task graph $G(V,E)$, with exactly one source and one sink. Vertex $v_i \in V, i = 1, ..., n$, represents task τ_i. The worst case number of execution cycles of τ_i is denoted by c_i, and the probability density function of its number of execution cycles is given by p_i. The directed edge $e_{ij} \in E$ represents dependency between tasks τ_i and τ_j, i.e., τ_j cannot begin to execute until τ_i finishes execution. Because we are only considering linear task graphs, there is exactly one outgoing edge from each vertex $v_j \in V$ to $v_{j+1} \in V$, $j = 1, 2, 3, ..., n-1$. A probability density function of communication volume w_{ij} is associated with each edge $e_{ij} \in E$. The communication volume determines the time and energy penalties of communication when τ_i and τ_j are not scheduled to run on the same core, otherwise, these penalties are 0.

As in [23], [25], we adopt a linear communication cost model. When transferring B bits of data, the communication delay is $t_p + \lambda B$ and the communication energy is γB, where t_p is the propagation delay, λ is the reciprocal of the operating data rate of the interconnection network, and γ is the energy spent to transfer one bit of data. We assume fixed data rate, that is, λ and γ are fixed.

We assume the program for each task follows stream programming style [18], where for each task instance the program first receives the data it needs through

communication with its predecessor task, then executes the task, and finally sends data to the successor task.

b) System Model: Power for each core has two components: static and dynamic. When a processor core is on, it is either (i) idle, consuming P_{idle} power, or (ii) executing some task at some frequency f, consuming total power $P(f)$, which includes the power for the memory as well as other components that are in the same clock domain of the processor. Each core can be turned on and off. When a core is off, it does not consume energy. As shown in [23], it is not always optimal to use all available cores due to the static power.

Each processor core provides M discrete frequencies, $f_1 < f_2 < ... < f_M$. Dynamic change of frequency incurs time and energy penalties [3]; time penalty is proportional to the difference between the two frequencies, while energy penalty is proportional to the difference between the squares of the two frequencies. If task τ_i executes for c_i' cycles at frequency f, then it executes for $\frac{c_i'}{f}$ time and consumes $P(f) \times \frac{c_i'}{f}$ energy, for those c' cycles portion of execution. The assumption that the execution time is linearly proportional to the execution frequency is justified when the cache (or memory) of a core is clocked at the same frequency as the core, which would be the case in a pipelined architecture.

c) Problem Definition: Given a periodic application as described above, our goal is to schedule the application tasks to run on the cores of a CMP described above, with the goal of minimizing the application *expected* energy consumption, while satisfying the QoS requirements: throughput, $\frac{1}{T}$, and response time, D. Scheduling tasks means mapping (assigning) tasks to execute on cores and computing the frequencies at which to run each task (frequencies can be dynamically changed at run time to save energy, for example, to reclaim slack time).

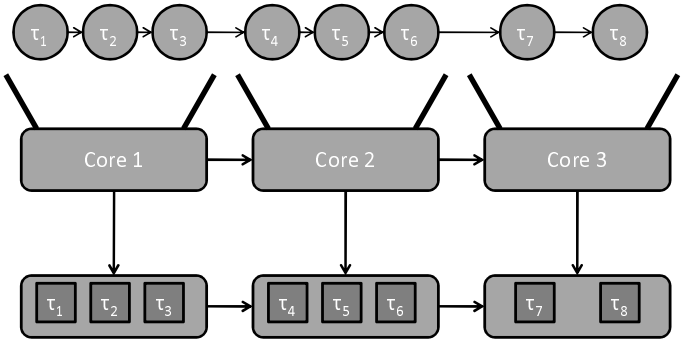

Fig. 1: Example of mapping linear task graph to cores arranged in a pipeline.

Due to the linear nature of the application task graph, we can visualize the cores running the tasks as being stages of an execution pipeline (only consecutive tasks to be mapped to the same core).

The throughput requirement limits the execution time on any core (pipeline stage) to at most T, including the time for communication with the next stage, while the response time requirement sets D as the upper limit of the sum of the execution times of pipeline stages. We focus on $T \leq D$ because it allows more flexibility in choosing the (one or more) cores on which to execute the application tasks.

IV. Solution to Related Scheduling Problems

In this section, we review related algorithms from [23], [24]. The algorithm in Section IV-A schedules linearly dependent tasks $\tau_1, ..., \tau_n$ on a pipeline in a way that minimizes the *worst case* energy consumption. The algorithm in Section IV-B schedules $\tau_1, ..., \tau_n$ onto a single processor in a way that minimizes the *expected* energy consumption. These two algorithms are relevant to our work presented in Section V, which schedules $\tau_1, ..., \tau_n$ on a pipeline in a way that minimizes the *expected* energy consumption.

A. Scheduling1D

Given a linear task graph, $G(V,E)$, of a periodic hard real-time application, along with knowledge of the *worst case* computational and communication requirements of each task, application period T, and deadline D, Scheduling1D [23] aims to schedule the application tasks to run on a CMP conforming to the models described in Section III with the goal of minimizing the *worst case* energy consumption while satisfying the two QoSs: throughput, $\frac{1}{T}$, and response time, D.

Scheduling1D is a dynamic programming algorithm that computes all feasible solutions, that is, all feasible mappings of tasks to processors using all the processors supported discrete frequencies, to find the optimal one. Since the number of solutions is clearly exponential in the worst case, the algorithm applies an approximation technique to limit the solutions search space guaranteeing polynomial running time and finding a solution within a $(1+\varepsilon)$ of the optimal energy consumption, where ε is a user-parameter.

Scheduling1D is based on the recursive structure of the optimal solution for linear task graphs. Assuming the time available to execute tasks τ_i to τ_n is t, the optimal scheduling of the tasks τ_i through τ_n is represented by the vector-valued function $E_i(t) = [e,q,j,d]$, where e

```
1.  E_i(t).e = ∞
2.  for j = i to n do
3.     c = Σ_{k=i}^{j} c_k
4.     for s = 1 to M do
5.        d = c/f_s + t_p + λ w_{j,j+1}
6.        if d ≤ T then
7.           // e' is the energy for the 1st stage
8.           e' = (P(f_s) − P_idle) c/f_s + P_idle T
9.           e = e' + γ w_{j,j+1} + E_{j+1}(t − d).e
10.          if e < E_i(t).e then
11.             E_i(t).e = e
12.             E_i(t).q = E_{j+1}(t − d).q + 1
13.             E_i(t).j = j
14.             E_i(t).d = d
```

Fig. 2: Computing $E_i(t)$

denotes the minimum energy consumption executing the tasks τ_i through τ_n when servicing a single instance of the application, q denotes the optimal number of stages (processor cores) for the tasks τ_i through τ_n, j indicates that tasks $\tau_i, \tau_{i+1}, ..., \tau_j$ are mapped to the first of the q stages, and d is the time used for the first stage plus the communication delay from the first stage to the next stage. The constraint of the throughput requirement implies that d should be smaller than or equal to T. Obviously, the optimal energy consumption and optimal scheduling will be obtained from $E_1(D)$. We use the notation $E_i(t).e$ to denote the e value of $E_i(t)$, and use a similar notation to refer to the q, j, and d values of $E_i(t)$.

Figure 2 shows the pseudo-code for computing $E_i(t)$ assuming that $E_{i+1}(t-d), ..., E_n(t-d)$ are known for any value of d. For each $j = i, ..., n$, the algorithm considers the possibility of mapping $\tau_i, ..., \tau_j$ to one stage executing at one of possible frequencies, f_s, which will consume the time d computed in line 5. In line 10, the least energy solution for $E_i(t)$ is selected using the recursive knowledge of $E_{j+1}(t-d)$. The recursive procedure starts with $E_n(t)$, which is obtained by mapping only τ_n into a single stage (processor). Finally, it should be noted that in [23], time is discretized so that the function $E_n(t)$ is defined at discrete times – implied by the M discrete processor frequencies – rather than being a continuous function. Consequently, each of the functions $E_i(t)$ is also defined at discrete times implied by the discretization of $E_{i+1}(t), ..., E_n(t)$ and the M discrete frequencies. For more details see [23].

B. Practical Inter-Task DVS (PITDVS)

Given a set of periodic tasks $\tau_1, ..., \tau_n$, that share the same period, T, and whose deadlines are equal to the end of that period, along with their order of execution, and the probability distribution function of the computational requirements of each of them, PITDVS [24] aims to schedule the tasks to run on *a single* processor core with the goal of minimizing their expected energy consumption. The algorithm has two components, one executed offline, and the other executed online. The offline part of the algorithm, called Optimal Inter-Task DVS (OITDVS), is a dynamic programming algorithm that determines the fraction β_i of the time T that should be allocated to task τ_i in order to minimize the expected execution energy of $\tau_1, ..., \tau_n$. It should be noted that the computed fractions $\beta_1, ..., \beta_n$ are independent of the time T, and are used during run-time to set the speed for the tasks. Specifically, after finishing the execution of $\tau_1, ..., \tau_{i-1}$, if t is the time remaining until the deadline, then τ_i should be scheduled to run at frequency $f = \frac{c_i}{\beta_i t}$. However, the online algorithm must approximate the computed frequency $f = \frac{c_i}{\beta_i t}$ to be one of the available M frequencies, and must also guarantee that all tasks can meet the deadline even if they all need to execute for their worst case cycles. In [24], PITDVS2 achieved the best results, as it approximated the computed continuous frequency using two adjacent discrete frequencies [10].

V. The Proposed Scheduling Algorithms

In this section, we present our proposed scheduling algorithms for the problem defined in Section III-0c. Our goal is to use knowledge of the probability distributions of the application tasks computational requirements to map them to pipeline stages and determine the execution time to allot to each stage.

A. Extended Practical Inter-Task DVS (EPITDVS)

Our first algorithm is a natural extension of PITDVS (Section IV-B), by extending it to use multiple resources (i.e., cores). The algorithm has an offline and online components; the offline component works as follows: (1) Run OITDVS, the offline part of PITDVS, which uses the probability distributions of the number of execution cycles to compute the fraction t_i of D to allot to each task $\tau_i, i = 1, ..., n$, with the restriction that no task is allowed time more than T — because of the throughput constraint. (2) Consider the tasks in their order $\tau_1, \tau_2, ..., \tau_n$, and map to the first stage tasks $\tau_1, ..., \tau_j$, such that $t_1 + ... + t_j \leq T$ and $t_1 + ... + t_{j+1} > T$. Continue the process by mapping to the second stage tasks $\tau_{j+1}, ..., \tau_k$,

such that $\tau_{j+1}+...+t_k \leq T$ and $\tau_{j+1}+...+t_{k+1} > T$, and so on until all tasks are mapped to stages. (3) Re-run OITDVS for the tasks mapped to each stage to determine the fraction of the stage time to allot to each task. At run-time, each stage applies the PITDVS2 online algorithm to schedule its mapped (assigned) tasks.

Steps 1 and 2 of the algorithm use the probability distributions of the number of execution cycles to find a balanced mapping of the tasks to pipeline stages. Step 3, also uses the probability distributions to re-allocate times to the tasks mapped to each of the pipeline stages. Experimental results showed that this two-phase process does not provide a significant improvement in energy consumption over Scheduling1D, which although uses only knowledge of worst case execution times, uses dynamic programming to simultaneously map tasks to stages and allocate time to tasks within each stage. Therefore, in the next section we devised a new algorithm, which applies a dynamic programming approach to solve the problem in one phase using probability distributions of execution times.

B. Probabilistic1D

Probabilistic1D is composed of an offline and online components. The offline component computes a schedule by mapping tasks to pipeline stages, and for each stage, determines the fraction of the execution time to allot to each of its mapped tasks. In order to save energy, the online component dynamically adjusts the frequency of each task based on t', the time remaining until the deadline of the stage it is running on, while guaranteeing all the remaining tasks of the stage can finish execution within t', even if they all require their worst case number of execution cycles.

1) The offline scheduling component: The offline scheduling part of Probabilistic1D is a dynamic program that explores different feasible task mappings to pipeline stages, and for each feasible mapping, computes the total expected application energy consumption as the sum of the expected energy consumptions for executing the individual pipeline stages. The chosen schedule is the one with the least expected energy consumption.

Given a subset of tasks mapped to a stage, the expected stage energy consumption varies depending on the stage allowed execution time. Since the throughput requirement constrains the stage time to be at most T, we use T as the stage allowed execution time[1].

Probabilistic1D recursively computes solutions for mapping tasks $\tau_i,...,\tau_n$ to pipeline stages. Specifically, for each $i, i = n,...,1$, a set E_i of tuples is computed such that each tuple (e_q,q) in E_i represents the solution with the least expected energy consumption, e_q, when $\tau_i,...,\tau_n$ are mapped to a pipeline with q stages. Given that the number of stages in the pipeline cannot exceed the number of tasks, i, and enforcing the deadline constraint requires the number of stages not to exceed $\lfloor \frac{D}{T} \rfloor$, the set E_i will be of the form $E_i = \{(e_q,q); q = 1,...,min\{i, \lfloor \frac{D}{T} \rfloor\}\}$

The recursive computation of E_i using $E_{i+1},...,E_n$ is shown in lines 4-10 of the algorithm given in Figure 3 with Line 1 defining $E_{n+1} = \{(0,0)\}$ to terminate the recursion. Specifically, for each value of j, $i \leq j \leq n$, the algorithm examines the mapping in which $\tau_i,...,\tau_j$ are mapped to one pipeline stage, in conjunction with E_{j+1}, the best computed mappings of $\tau_{j+1},...,\tau_n$, to pipeline stages. Line 5 makes sure that tasks $\tau_i,...,\tau_j$ can fit on one stage, by testing whether stage time T is long enough to allow $\tau_i,...,\tau_j$ to execute for their worst case cycles, as well as allow enough time for communication between τ_j and τ_{j+1} across pipeline stages. Line 6 computes the expected energy consumption (explained below), for servicing a single instance of the application, of the stage where $\tau_i,...,\tau_j$ are mapped. For each solution in E_{j+1}, line 9 checks whether it is possible to add one more stage without violating the deadline. If so, the newly computed solution is added to E_i if it is the best solution obtained so far using $q+1$ stages. This is represented in line 10 by the operator $\oplus$ which is defined formally as follows:

$$E_i \oplus (e_{new},q) = \begin{cases} E_i \cup \{(e_{new},q)\} \text{ ,if } \nexists (e_{old},q) \in E_i \\ E_i \cup \{(e_{new},q)\} - \{(e_{old},q)\} \text{ ,if} \\ \quad \exists (e_{old},q) \in E_i \text{ , } e_{old} > e_{new} \\ E_i \text{ ,otherwise} \end{cases}$$

a) Computing expected energy consumption of a stage:: Before we describe how to compute expected energy consumption of a stage, we need to describe p_i, the probability density function of the number of execution cycles of τ_i. The execution cycles of τ_i are divided into bins of equal number of cycles. Let bin_i denote the number of those bins. For task τ_i, we define $p_i(l)$ as the probability that τ_i runs for a number of cycles

[1]We experimented with values other than T for stage allowed execution time, but setting stage time equal to T produced the best results with respect to percentage of energy savings, as well as the percentage of experiments in which the application consumed less energy when scheduled using Probabilistic1D compared to when scheduled using the baseline algorithm, Scheduling1D.

1. $E_{n+1} = \{(0,0)\}$
2. for $i = n$ down to 1 do
3. // Compute E_i
4. for $j = i$ to n do
5. if $f_M \times \sum_{k=i}^{j} c_k + \lambda w_{j,j+1} \leq T$ then
6. let $e^{'}$ = expected energy consumption when
7. $\tau_i, ..., \tau_j$ are mapped to one stage
8. $\forall$ tuples $u = (e,q) \in E_{j+1}$
9. if $(q+1)T <= D$ then
10. $E_i = E_i \oplus (e^{'} + \gamma w_{j,j+1} + e, q+1)$

Fig. 3: Offline (scheduling) component of Probabilistic1D

that falls within bin l. If each bin contains b_i cycles, then $cycle_i(l) = lb_i$ denotes the number of the last cycle that falls in bin l.

Consider a pipeline stage s, to which tasks $\tau_i, ..., \tau_j$ are mapped. To compute expected energy consumption of s, we first run *OITDVS* to partition the stage time, T, among the tasks. Let t_k denote the time alloted for τ_k, $i \leq k \leq j$. Define the function $f(t,c)$ to return the processor frequency to execute c cycles within time t. In essence, $f(t,c)$ approximates $\frac{c}{t}$ to one of the M supported processor frequencies. We use the following equation to compute the expected energy consumption of s:

$$\sum_{k=i}^{j} \sum_{l=1}^{bin_k} p_k(l) \times P(f(t_k, c_k)) \times \frac{cycle_k(l)}{f(t_k, c_k)} \quad (1)$$

In equation 1, the energy consumed when executing task τ_k is computed as the expected value of the energies consumed when τ_k executes for a number of cycles that falls into bins 1, 2, ..., bin_k. Specifically, if c, the number of cycles executed by τ_k, falls into bin l, $1 \leq l \leq bin_k$, then the energy consumed is equal to the product of the power (the second term in equation 1) and the execution time (the third term in equation 1).

Observe that $|E_n| = 1$, and $|E_{n-1}| \leq 2$, since we may either map τ_{n-1} to its own stage separate from τ_n, or we may map it to the same stage with τ_n. Similarly, $|E_{n-2}| \leq 3$, because we may map τ_{n-2} to its own stage or map τ_{n-2} to the first stage of each solution $u \in E_{n-1}$. This is due to the fact that only consecutive tasks in the linear task graph can be mapped to the same stage. Therefore, we have, $|E_i| \leq |E_{i+1}| + 1$, $i = 1, 2, ..., n-1$, and $|E_n| = 1$. It then follows that the number of solutions explored by Probabilistic1D is $O(n^2)$.

2) The online scheduling component: The online part of Probabilistic1D applies the online part of PITDVS2 to the individual pipeline stages in order to dynamically adjust the frequency of the tasks of each stage. As mentioned above, a continuous frequency, f, is computed for the next task, $\tau_k, k = 1, ..., n$, to run on a stage based on c_k, remaining stage time, and $\sum_{j=k+1}^{l} c_j$, where $\tau_{k+1}, ..., \tau_l$ are tasks mapped to the same stage of τ_k. f is then approximated using one or two adjacent discrete frequencies of the M frequencies supported by the processor.

C. Cross-Stage Idle Time Reclamation (CSITR)

Due to the variability of the workload of the different application instances, at run time, the online components of the algorithms PITDVS, EPITDVS, and Probabilistic1D, all apply dynamic voltage scaling to each pipeline stage in order to vary the processor frequency at which to run each stage task; effectively reclaiming the slack time on each stage.

However, it is possible that any stage, s, would finish processing a given application instance, I_i, earlier than its deadline. Instead of s sitting idle consuming static power, we can reclaim its idle time by starting to execute the tasks of the next application instance, I_{i+1}, as soon as the previous pipeline stage, $s-1$, finishes processing I_{i+1}. This way, when s starts to execute the application instance I_{i+1}, it would be allowed an execution time longer than the offline computed execution time (in the case of Probabilistic1D, stage execution time is fixed to T as mentioned above). This in turn allows the online part of scheduling algorithm to reclaim the increased stage slack time by running tasks at even lower frequencies, thus saving more energy. To evaluate the effectiveness of the CSITR technique, in our simulations we compare application execution energy consumption when the algorithms are run with and without reclaiming processor idle time.

VI. Evaluation

A. Experimental Setup

In this section we evaluate our proposed algorithm, Probabilistic1D, through simulations. We compare it to EPITDVS and to the baseline state-of-the-art algorithm: Scheduling1D without cross-stage idle time reclamation. In addition, we evaluate the effect of cross-stage idle time reclamation on Probabilistic1D by comparing the energy savings achieved in each case relative to the baseline algorithm.

TABLE I: XScale Speed Settings and Power Consumptions

Speed (MHz)	150	400	600	800	1000
Voltage (V)	0.75	1.0	1.3	1.6	1.8
Power (mW)	80	170	400	900	1600

a) Application Task Graphs:: We experiment with a synthetically generated workload. We run two sets of experiments. In one set, the application task graph consists of 5 tasks, and in the other it consists of 10 tasks. For each set, we experiment with uniform and bimodal distributions of task computational requirements with a minimum workload of 12,500 cycles and a maximum of 1,250,000 cycles per task. We partition task computational requirements cycles into bins of equal number of cycles; we used 100 bins.

We use different application period, T, and deadline, D, value pairs. For the 5-task applications, we use periods of 2.5 msec to 12.5 msec with a step of 1 msec, and deadlines of 10 msec to 25 msec, also with a step of 1 msec. For the 10-task applications, we use periods of 5 msec to 25 msec, and deadlines of 20 msec to 50 msec, both with a step of 1 msec. We only run experiments where $D \geq T$. The approximation factor, ε, used in Scheduling1D is 0.05. Each experiment is defined by the specific values of T and D, as well as the number of application tasks and their probability distributions of computational requirements. For each run, we simulate the execution of 1000 application instances and we compute the average percentage of energy savings of the schedules of Probabilistic1D and EPITDVS, both with cross-stage idle time reclamation, relative to the schedule of Scheduling1D without cross-stage idle time reclamation.

b) Communication Model:: We assume a similar communication medium as in [23]. To compute λ and γ (section III-0a), we assume a transmission rate of 20 Gbytes/s and transmission power is set to 20% of maximum processor power when the link is fully utilized. Communication energy and time penalties are only incurred for communication between two tasks mapped to two consecutive pipeline stages (processor cores). In our simulation, we use a normal probability distribution with small variance to generate the amount of communication data.

c) Processor Model:: We simulate the Intel XScale processor (Table I) [22]. We assume processor static power is 40 mW when the processor is idle (i.e., half the power consumed when the processor is running at its lowest frequency). Dynamic change of frequency incurs time and energy penalties (Section III-0b).

B. Experimental Results

Considering all the conducted experiments, we found that in over 94% of them, the schedules of Probabilistic1D with cross-stage idle time reclamation achieved the greatest percentage of energy savings, and only in less than 1% of the experiments did the schedules of Probabilistic1D, also with cross-stage idle time reclamation, consume more energy compared to the baseline algorithm. These are much better results than those achieved by the schedules of EPITDVS with cross-stage idle time reclamation. EPITDVS schedules achieved the greatest percentage of energy savings in less than 5% of the experiments, and consumed more energy than the baseline in 37% of the conducted experiments.

We noticed the results of the two sets of experiments – corresponding to using applications with 5 and 10 tasks – exhibited similar trends, so we only show graphs for experiments of a 10-task application with tasks having a bimodal probability distribution of computational requirements.

Figures 4, 5, and 6 show the average percentage of energy savings – relative to the baseline – for the schedules of Probabilistic1D, EPITDVS, and Scheduling1D, all with cross-stage idle time reclamation (CSITR) applied, respectively.

For a better comparison of the performance of the algorithms, we take a 2D cross-section of the plots in Figures 4, 5, and 6 at values of $T = 5, 9, 19$ msec, corresponding to short, intermediate, and long inter-arrival times, respectively, and plot them in Figures 7, 8, and 9, respectively. From these figures, we find that for a fixed T, the average percentage of energy savings of all algorithm tends to go down when the deadline, D, is increased. This is expected since in this case Scheduling1D has more time to execute each instance of the application, therefore it schedules the stages (processors) to run slower to consume less energy, and although the schedules of, for example, Probabilistic1D, still consume less energy, the relative energy savings are smaller in this case. Also, from the same figures, we note that as D is increased, the schedules of Scheduling1D *with* cross-stage idle time reclamation may consume more energy (we observed a penalty of at most 1% more energy) relative to the baseline schedule of Scheduling1D *without* cross-stage idle time reclamation. This is due to time and energy penalties associated with dynamic change of processors' frequencies – which are not incurred by

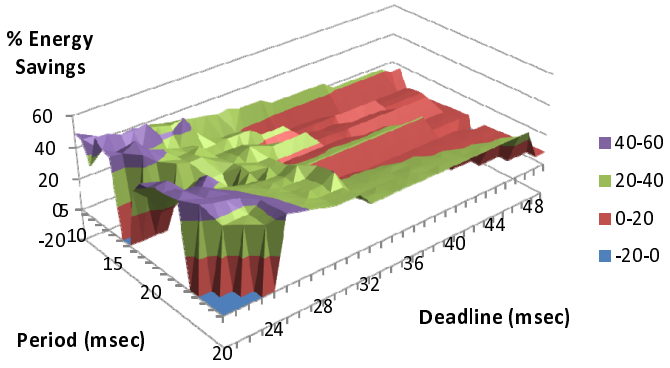

Fig. 4: % Energy Savings of Probabilistic1D with CSITR relative to Scheduling1D without CSITR.

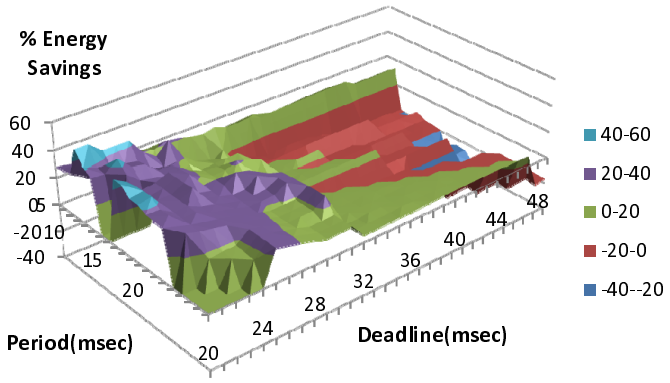

Fig. 5: % Energy Savings of EPITDVS with CSITR relative to Scheduling1D without CSITR.

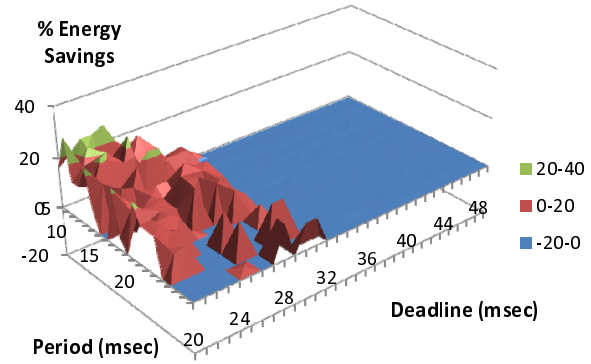

Fig. 6: % Energy Savings of Scheduling1D with CSITR relative to Scheduling1D without CSITR.

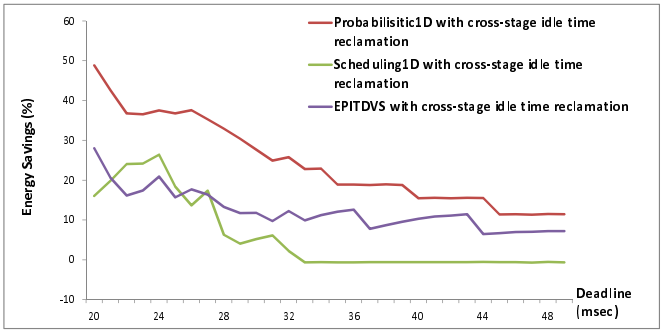

Fig. 7: % Energy Savings relative to Scheduling1D without CSITR for $T = 5$ msec

the baseline since it always uses the offline computed frequency.

Now, we consider how cross-stage idle time reclamation affects the average percentage of energy savings of the schedules produced by Probabilistic1D and Scheduling1D. Figures 10, 11, and 12, plot the percentage of energy savings of the schedules of Probabilistic1D *with and without* cross-stage idle time reclamation, as well as the schedules of Scheduling1D with cross-stage idle time reclamation. These plots were generated for a 10-task periodic application using the same tasks probability distributions described above, and for inter-arrival times $T = 5, 9, 19$ msec, respectively.

For the schedules of Probabilistic1D, we found that the average improvement in the percentage of energy savings when applying cross-stage idle time reclamation was less than 3% than when it was not applied. This indicates that the offline schedules computed by Probabilisitic1D were able to capture the variability of the application workload to a high degree of accuracy that they minimize the time processors are idle.

For the schedules of Scheduling1D, we note that for relatively short deadlines, cross-stage idle time reclamation allows Scheduling1D to save more energy (up to 30% in our experiments) compared to the baseline. However, for longer deadlines, Scheduling1D with cross-stage idle time reclamation usually performs slightly worse (consumes less than 1% more energy) due to the time and energy penalties incurred when processor frequency is dynamically changed.

VII. Conclusion and Future Work

We consider the problem of scheduling a periodic hard real-time application with linear precedence constraints on a CMP using knowledge of the probability distribution of computation requirements of its constituting tasks. The goal of the scheduling is to minimize the expected energy consumption. Our experimental results show that our proposed scheduling algorithm, Probabilistic1D, achieves as much as 55% average energy savings over scheduling for the worst case execution. Our experiments also demonstrate the benefits of cross-stage processor idle time reclamation at run time. For future work, we would like to extend Probabilisitic1D to handle periodic hard real-time applications with general

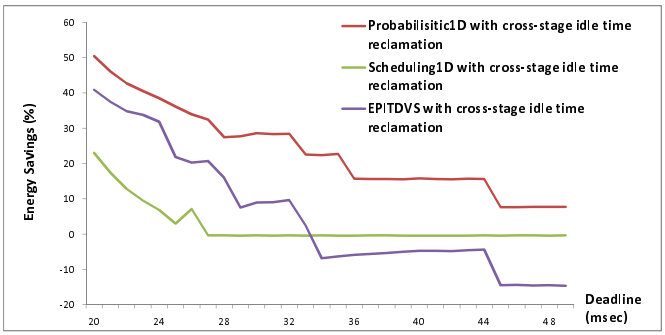

Fig. 8: % Energy Savings relative to Scheduling1D without CSITR for $T = 9$ msec

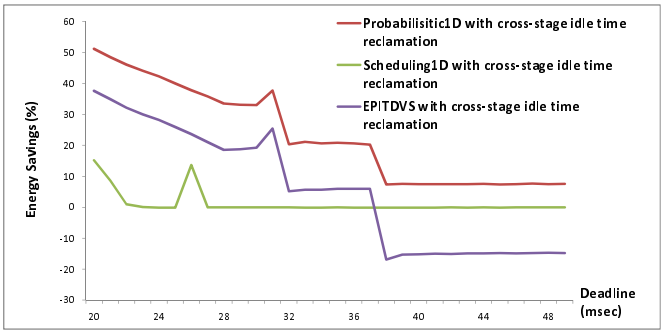

Fig. 9: % Energy Savings relative to Scheduling1D without CSITR for $T = 19$ msec

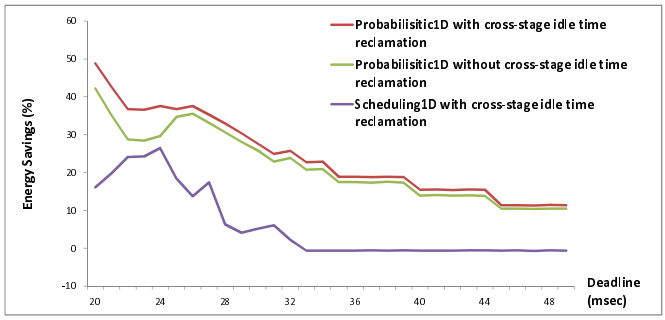

Fig. 10: Effect of CSITR on Probabilistic1D and Scheduling1D, $T = 5$ msec

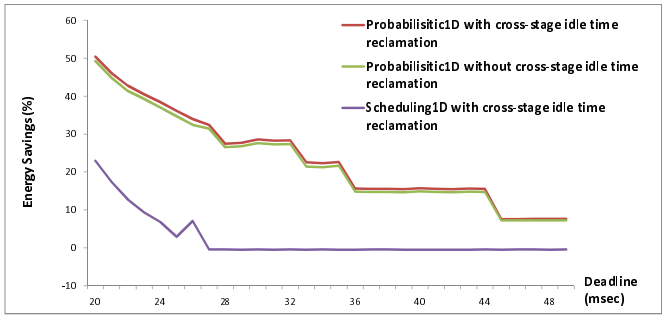

Fig. 11: Effect of CSITR on Probabilistic1D and Scheduling1D, $T = 9$ msec

precedence constraints.

REFERENCES

[1] AMD, "Quad-core amd opteron processor," http://multicore.amd.com/us-en/AMD-Multi-Core.aspx.

[2] ——, "Mobile amd athlon 4 processor model 6 cpga data sheet," 2001, http://www.amd.com.

[3] T. D. Burd and R. W. Brodersen, "Design issues for dynamic voltage scaling," in *ISLPED*, 2000, pp. 9–14.

[4] J.-J. Chen, "Expected energy consumption minimization in dvs systems with discrete frequencies," in *SAC*, 2008, pp. 1720–1725.

[5] E. N. Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," in *PACS*, 2002, pp. 179–196.

[6] R. Ernst and W. Ye, "Embedded program timing analysis based on path clustering and architecture classification," in *International Conference on Computer Aided Design*, nov 1997, pp. 598–604. [Online]. Available: citeseer.ist.psu.edu/ernst97embedded.html

[7] K. Flautner, S. K. Reinhardt, and T. N. Mudge, "Automatic performance setting for dynamic voltage scaling," in *MOBICOM*, 2001, pp. 260–271.

[8] F. Gruian, "Hard real-time scheduling for low-energy using stochastic data and dvs processors," in *ISLPED '01: Proceedings of the 2001 international symposium on Low power electronics and design*. New York, NY, USA: ACM, 2001, pp. 46–51.

[9] Intel, "Intel core2 quad processors," http://www.intel.com/products/processor/core2quad/.

[10] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *ISLPED*, 1998, pp. 197–202.

[11] C. M. Krishna and Y.-H. Lee, "Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems," *IEEE Trans. Comput.*, vol. 52, no. 12, pp. 1586–1593, 2003.

[12] J. R. Lorch and A. J. Smith, "Pace: A new approach to dynamic voltage scaling," *IEEE Transactions on Computers*, vol. 53, no. 7, pp. 856–869, 2004.

[13] P. Malani, P. Mukre, and Q. Qiu, "Power optimization for conditional task graphs in dvs enabled multiprocessor systems," in *VLSI-SoC*, 2007, pp. 230–235.

[14] J. A. Pouwelse, K. Langendoen, and H. J. Sips, "Energy priority scheduling for variable voltage processors," in *ISLPED*, 2001, pp. 28–33.

[15] C. Rusu, R. Xu, R. G. Melhem, and D. Mossé, "Energy-efficient policies for request-driven soft real-time systems," in *ECRTS*, 2004, pp. 175–183.

[16] D. Shin and J. Kim, "Power-aware scheduling of conditional task graphs in real-time multiprocessor systems," in *ISLPED*, 2003, pp. 408–413.

[17] Sun, "Sun ultrasparc t1 processor," http://www.sun.com/processors/UltraSPARC-T1/.

[18] W. Thies, M. Karczmarek, and S. P. Amarasinghe, "Streamit: A language for streaming applications," in *CC*, 2002, pp. 179–196.

[19] D. Wu, B. M. Al-Hashimi, and P. Eles, "Scheduling and mapping of conditional task graphs for the synthesis of low power embedded systems," in *DATE*, 2003, pp. 10 090–10 095.

[20] C. Xian, Y.-H. Lu, and Z. Li, "Dynamic voltage scaling for multitasking real-time systems with uncertain execution time,"

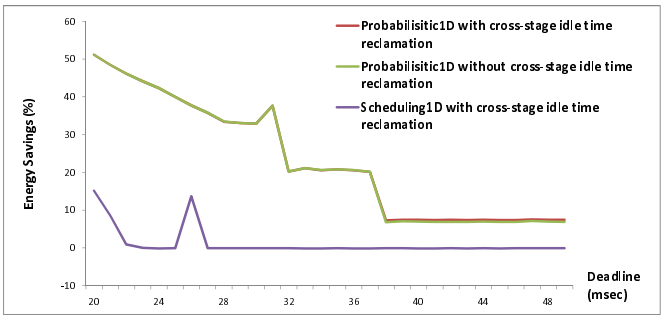

Fig. 12: Effect of CSITR on Probabilistic1D and Scheduling1D, $T = 19$ msec

IEEE Trans. on CAD of Integrated Circuits and Systems, vol. 27, no. 8, pp. 1467–1478, 2008.

[21] Xscale, "Intel xscale microarchitecture," 2007, http://developer.intel.com/design/intelxscale/.

[22] Xscalepower, "Intel xscale microarchitecture:benchmarks," 2005, http://web.archive.org/web/20050326232506/http://developer.intel.com/de%sign/intelxscale/benchmarks.htm.

[23] R. Xu, R. G. Melhem, and D. Mossé, "Energy-aware scheduling for streaming applications on chip multiprocessors," in *RTSS*, 2007, pp. 25–38.

[24] R. Xu, D. Mossé, and R. G. Melhem, "Minimizing expected energy consumption in real-time systems through dynamic voltage scaling," *ACM Trans. Comput. Syst.*, vol. 25, no. 4, 2007.

[25] M.-T. Yang, R. Kasturi, and A. Sivasubramaniam, "A pipeline-based approach for scheduling video processing algorithms on now," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 2, pp. 119–130, 2003.

[26] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time cpu scheduling for mobile multimedia systems," in *SOSP*, 2003, pp. 149–163.

[27] Y. Zhang, X. Hu, and D. Z. Chen, "Task scheduling and voltage selection for energy minimization," in *DAC*, 2002, pp. 183–188.

[28] D. Zhu, R. G. Melhem, and B. R. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems," in *IEEE Real-Time Systems Symposium*, 2001, pp. 84–94.