Adaptive Scheduling and Voltage Scaling for Multiprocessor Real-time Applications with Non-deterministic Workload

Parth Malani, Prakash Mukre, Qinru Qiu and Qing Wu
Department of Electrical and Computer Engineering, Binghamton University
Binghamton, NY 13902
{parth, pmukre1, qqiu, qwu} @binghamton.edu

Abstract — **The computational workload of some real-time applications varies significantly during runtime, which makes the task scheduling and power management a challenge. One of the major influences to the workload of an application is the selection of conditional branches which may activate or deactivate a large set of operations. Focusing on real-time applications with variable workload which is due to random branch selection, this paper presents a framework of task mapping, scheduling and dynamic voltage and frequency scaling (DVFS) for a multiprocessor system. The proposed framework maintains workload awareness using dynamic profiling of branch probability. The profiled information is utilized by the scheduling and DVFS algorithm that are adopted in this framework to generate statistically optimal solution.**

I. Introduction

Future generation embedded real-time systems will consists of vast number of processing elements (PE). The sharp performance boost achieved by such Multiprocessor System-on-Chip (MPSoC) comes with increase in both computation and communication energy. There have been continuous design efforts that target reducing the energy of battery operated MPSoC using system level power optimization techniques, such as task scheduling or dynamic voltage and frequency scaling (DVFS). However, workload variations due to the non-deterministic nature of application impact the efficiency of these techniques. It is necessary to develop robust techniques which can vigorously target power optimization and adapt to system variation due to input data or environment.

Techniques of scheduling and DVFS for applications with variable workload have been proposed for single processor [1][2] and multiprocessor [3] systems. These works consider a set of preemptive tasks with uncertain execution time. Reference [1] models the voltage scaling process as a filter system and proposes the time-variant voltage scaling algorithm based on the water-filling process in information theory. The authors of [2] propose several workload prediction models which assist the voltage selection. The authors of [3] target at partition a set of tasks with uncertain execution time and map them to a multiprocessor system so that the expected workload is balanced. The cycle demand of each task is divided into several bins and the frequency and voltage of each bin is calculated to minimize the total expected energy consumption.

All of the above works focus on the workload variations caused by uncertain execution time of tasks, which are the atomic scheduling or mapping units in an application. Another major influence to the workload of an application is the selection of conditional branches which may activate or deactivate a large set of operations. In contrast to instruction level branch, here we consider conditional branches that activate or deactivate an entire task and hence generate more visible workload variation. Some examples of such task level branching include branches that either enable or disable IDCT function during MPEG decoding or branches that select different modulation schemes for preamble and payload based on 802.11b physical layer standard. This work focuses on the set of applications which can be decomposed into a set of tasks with relatively constant execution time. The uncertainty of the workload is reflected by the random activation and deactivation of certain tasks during runtime and it is captured by branch selections among tasks. We assume that such branch information is observable to the scheduling software to assist runtime scheduling and power management of the system.

Complex systems running such non-deterministic applications can be modeled by a conditional task graph (CTG) [7]~[10],[17]. The conditional behavior of application is captured by branching tasks and it may affect the execution of other tasks depending on the branch selection. Although branch prediction is a common practice in high performance processors, the prediction will not be perfect. Furthermore, the task graphs that we are working with are high level descriptions of large applications. Their selection of conditional branches depends mostly on the input data, which are random in nature. Hence a branch selection should be considered as a random variable and characterized by its probability distribution. The branch probability can be obtained through online or offline profiling. And it can be predicted based on history. For a technique to provide robust scheduling and energy reduction for an application with non-deterministic workload due to random branch selection, it should be able to

(1) handle mutual exclusiveness among tasks that belong to different branches;

(2) consider the branch selection probabilities and minimize the expected energy dissipation instead of worst case or best case energy dissipation;

(3) predict the distribution of branch probabilities; and

(4) adapt to the changes of probability distribution.

Statistical scheduling and voltage scaling techniques for the CTGs running on multiprocessor systems have been proposed [10][17]. Both works solve the scheduling and DVFS problem in two separate stages. During the first stage, tasks that are mapped to the same processor are ordered for a maximum slack. In the second stage, tasks are stretched to minimize the energy while meeting the deadline constraint. Shin et al. [10] proposed an algorithm for task ordering and stretching of CTG which considers the run-time behavior. The probabilistic distribution of the branch selections is considered during task stretching. The authors of [17] consider task mapping and ordering concurrently and they utilize the branch probability to assist task scheduling as well as task stretching, hence achieve higher energy savings. Both [10] and [17] use NLP based task stretching which has high complexity and they cannot be applied during runtime. Hence, they do not track or adapt to the change of probability distribution. A DVFS algorithm for the CTG on multiprocessor system based on slack distribution is proposed in [9]. However, it does not differentiate tasks with high activation probability from the tasks with low activation probability during slack distribution. Therefore, it will not adapt to the change of the workload.

In this work, we propose a framework of adaptive scheduling and voltage scaling for CTG on a multiprocessor system. The proposed technique predicts the branch probability using a sliding window based approach. When the change of the distribution exceeds certain threshold, an online algorithm is called for re-scheduling and voltage re-selection. Similar as [10] and [17], the online algorithm has a separate task ordering and stretching stages. It employs a modified *Dynamic Level based Scheduling* (DLS) that is adopted from [17] while performs task stretching using a low complexity heuristic. The proposed technique is applied to real-life applications such as MPEG decoder or vehicle cruise control system as well as some random CTG graphs generated by TGFF [14].

The rest of this paper is organized as follows. Section II introduces the application and hardware architecture models. Section III provides detailed introduction of our scheduling algorithm. Sections IV and V present the experimental results and conclusions.

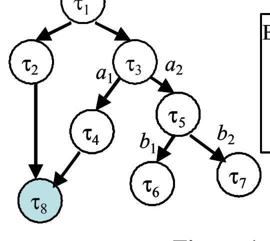
II. Application and Architecture Modeling

The CTG that we are using is similar as the one specified in [10]. A CTG is an acyclic graph <*V*, *E*>. Each vertex $\tau \in V$ represents a task. An edge $e=(\tau_i, \tau_j)$ in the graph represents that the task τ_i must complete before τ_j can start. A conditional edge *e* is associated with a condition *C*(*e*). We use *prob*(*e*) to denote the probability that the condition *C*(*e*) is true. The node with output conditional edge is a *branch fork node*.

A node can be either *and-node* or *or-node* [10]. An and-node is activated when all its predecessor nodes are completed and the conditions of the corresponding edges are satisfied. On the other hand, an or-node is activated when one or more predecessors are completed and the conditions of the corresponding edges are satisfied.

The condition that the task τ is activated is denoted as $X(\tau)$. The condition of an and-node τ_i can be written as $\wedge_{\tau_k}\left(C(\tau_k,\tau_i)\wedge X(\tau_k)\right)$, where τ_k is the predecessor of τ_i. The condition of an or-node τ_j can be written as $\vee_{\tau_k}\left(C(\tau_k,\tau_j)\wedge X(\tau_k)\right)$, where τ_k is the predecessor of τ_j. There also exist data dependency where a task can not begin its execution until the predecessor tasks are finished and the respective data transfers are completed. A *minterm m* is a possible combination of all conditions of CTG. We use *M* to denote the set of all possible minterms of CTG. A task τ is associated with a minterm *m* if $m \subseteq X(\tau)$. In another word, a task τ is associated with a minterm *m* if $X(\tau)$ will be true when *m* is evaluated to be 1. The set of minterms with which τ is associated is denoted as $\Gamma(\tau)$. Two tasks τ_i and τ_j are mutually exclusive if they cannot be activated at the same time, i.e. $X(\tau_i) \oplus X(\tau_j)=0$. To simplify the implementation and discussion, we refer the condition "1" (i.e. always true) as one of the minterms as well.

The volume of data that pass from one task to another is also captured by the CTG. Each edge (τ_i, τ_j) in the CTG associates with a value $\text{Comm}(\tau_i, \tau_j)$ which gives the communication volume in the unit of Kbytes. Finally, we assume a periodic graph and use a common deadline for the entire CTG.



Execution Profile
prob(a1)=0.8
prob(a2)=0.2
prob(b1)=0.5
prob(b2)=0.5

Communication Volume
comm(τ_1, τ_2)=1kB
comm(τ_2, τ_8)=2kB
comm(τ_3, τ_4)=0.1kB
comm(τ_3, τ_5)=10kB
comm(τ_5, τ_6)=0.5kB
comm(τ_5, τ_7)=0.5kB
comm(τ_4, τ_8)=0.8kB

Figure 1 An example of CTG.

Example 1: Figure 1 shows an example of a CTG. All nodes except node τ_8 are and-nodes. The edges coming out from τ_3 and τ_5 are conditional edges. The symbol marked beside a conditional edge gives the condition under which the edge will be activated. For example $C(\tau_3, \tau_4) = a_1$. There are total of 4 minterms in the CTG and $M=\{1, a_1, a_2b_1, a_2b_2\}$. We have $\Gamma(\tau_1)=\Gamma(\tau_2)=\Gamma(\tau_3)=\{1\}$, $\Gamma(\tau_4)=\{a_1\}$, $\Gamma(\tau_5)=\{a_2\}$, $\Gamma(\tau_6)=\{a_2b_1\}$, $\Gamma(\tau_7)=\{a_2b_2\}$ and $\Gamma(\tau_8)=\{1,a_1\}$. The execution profile and communication volume are given beside the CTG. The fact that τ_8 is an or-node indicates that if condition a_1 is true then τ_8 cannot start until τ_2 and τ_4 finish and if condition a_1 is false then τ_8 does not have to wait for τ_4. Note that, in reality, we do not know weather a_1 is true or false until τ_3 finishes. Therefore, in any case, τ_8 must wait until both τ_2 and τ_3 finish. This example shows an implied dependency between an or-node and the branch fork node. More detailed discussion will be provided in the next section.

The following models the architecture of an MPSoC:

- The set of PEs, $P=\{p_1, p_2, ..., p_n\}$
- The energy $E(\tau_i, p_j)$ and worst case execution time $WCET(\tau_i, p_j)$, $\forall \tau_i \in V$ and $\forall p_j \in P$. These values give the energy and delay of each task when it is running on different PEs at the nominal V_{DD}.
- The bandwidth $B(p_i, p_j)$ and transmission energy $E_{tr}(p_i, p_j)$, $\forall p_i, p_j \in P$. These values specify the bandwidth as well as the transmission energy per byte of the communication link between p_i and p_j. We modeled a point-to-point communication link for our interconnect network and dedicated communication resource for each PE. We also assume that the voltage scaling cannot be applied to the communication tasks.

III. Framework of Adaptive Scheduling and DVFS

A. Online task scheduling and streching algorithm

Similar as [10] and [17], the online algorithm has a separate task ordering and stretching stages. It employs a modified *Dynamic Level based Scheduling* (DLS) that is adopted from [17] while performs task stretching using a low complexity heuristic.

The DLS algorithm is a list scheduling algorithm. It considers computation scheduling and communication scheduling altogether. The *ready list* is a list of tasks whose predecessors have been scheduled and mapped. For each task τ_i in the ready list, and each processor p_j, the dynamic level $DL(\tau_i, p_j)$ is calculated. The pair of (τ_i, p_j) which gives the maximum dynamic level will be selected and the mapping is performed accordingly.

The modified DLS algorithm considers the mutual exclusiveness among conditional tasks as well as the probabilistic distribution of branch selection. The dynamic level is calculated using the following:

$$DL(\tau_i, p_j) = SL(\tau_i) - AT(\tau_i, p_j) + \delta(\tau_i, p_j), \qquad (1)$$

where $SL(\tau_i)$ is the static level of task τ_i, . If τ_i is a non-branching node, $SL(\tau_i)$ is calculated as:

$SL(\tau_i) = {}^*WCET(\tau_i) + \max SL(\tau_j), \tau_j \in \{Successors\ of\ \tau_i\}$, and if τ_i is branching node, $SL(\tau_i)$ is calculated as:

$$SL(\tau_i) = {}^*WCET(\tau_i) + \textstyle\sum_j prob(c_{ij}) * SL(\tau_j),\quad \tau_j \in \{Successorsof\ \tau_i\}.$$

$\delta(\tau_i, p_j)$ is the difference between average *worst case execution time* (*WCET*) of τ_i and the *WCET* of τ_i at p_j, and $AT(\tau_i, p_j)$ is the first time that task τ_i can start on processor p_j. The average WCET of a node for each PE is calculated for maximum operating speed/frequency of particular PE. Note that mutual exclusive task may be able to start on the same processor during the same time. For the best pair (τ_i, p_j) that has the highest *DL*, τ_i will be scheduled on p_j at time $AT(\tau_i, p_j)$. Since the schedule of τ_i imposes new precedence order between τ_i and other tasks that are scheduled on the same processor, we also update the CTG to reflect this change. After that, the ready list will be updated and the above mentioned procedure will repeat until the ready list is empty. More information of the modified DLS algorithm and its performance evaluation can be found in [17].

The task stretching algorithm is a profile-based approach considering branch probabilities. It calculates only single speed for each task and it facilitates different scaling ratio for different PEs. Once the CTG is updated, all possible paths in CTG are calculated using Breadth First Search (BFS) algorithm. Also associated with each path *p* is the slack and delay which are denoted as *slk*(*p*) and *delay*(*p*) respectively. A path's delay is the sum of execution time of all nodes along that path with mapping already known through scheduling algorithm. Associated with each task τ on path p, there is a probability *prob*(*p*, τ), which indicates the probability of path *p* given the condition that task τ is activated. *prob*(*p*, τ) is calculated as the joint probability of all the conditional branches lying on the path after node τ. For example, consider the example in Figure 1, the probability $prob(\tau_1\text{-}\tau_3\text{-}\tau_5\text{-}\tau_6,\ \tau_5)=prob(b_1)=0.5$ because the only conditional branch along the path τ_1-τ_3-τ_5-τ_6 after node τ_5 is b_1. For another example, the probability $prob(\tau_1\text{-}\tau_3\text{-}\tau_4\text{-}\tau_8,\ \tau_8)=1$, because there is no conditional branch in this path after node τ_8.

```
Online task stretching heuristic for CTG G
1.   Process initial schedule generated by DLS based task ordering
     algorithm;
2.   Calculate possible paths in CTG using BFS;
3.   For each task $\tau_i$ {
4.       CalculateSlack ($\tau_i$);
5.       Stretch $\tau_i$, lock its schedule and speed;
6.       Update the delay and slack of all paths spanning $\tau_i$ ;
7.       Update the schedule for CTG G;
     }
CalculateSlack ($\tau_i$)
1.   For each minterm $m \in \Gamma(\tau_i)$ {
2.       For all paths of $m \in \Gamma(\tau_i)$ that span node $\tau_i$ {
3.           Find the critical path $p_{worst}$ where $prob(p_{worst}, \tau_i) \neq 1$;
4.           $slk1 += prob(p_{worst}, \tau_i) * wcet(\tau_i) *(slk(p_{worst}) / delay(p_{worst}))$
             $* prob(\tau_i)$;
         }
     }
5.   For each path of $m \in \Gamma(\tau_i)$ where $prob(m) = 1$
6.       Find the critical path $t_{worst}$ ;
7.   $slk2 = wcet(\tau_i) * (slk(t_{worst}) / delay(t_{worst})) * prob(\tau_i)$;
8.   $slk(\tau_i) = min [slk1, slk2]$;
9.   If there is a path p that spans node $\tau_i$ and $slk(\tau_i)$>deadline-
     delay(p) then,
10.  $slk(\tau_i)$=deadline-delay(p);
```

Figure 2 Online task stretching heuristic.

For each task, step4 of main routine shown in Figure 2 determines the available slack by calling CalculateSlack(τ_i) routine. This routine finds the most critical path that has a minimum slack applicable to task τ_i. In case of multiple paths pertaining to different minterms with probabilities less than 1, first the critical path that has the lowest distributable slack ratio (*slk*(*p*)/*delay*(*p*)) and has the probability less than 1 is identified for each minterm. After that the initial slack of τ_i is taken as a probability weighted sum of all these critical path slacks corresponding to each minterm $m \in \Gamma(\tau_i)$ as shown in step 4 of routine. Note that the weight for each path is *prob*(*p*, τ_i), which is the probability of path *p* given the condition that task τ_i is activated. One more slack value is calculated as shown in step 7 for critical paths with probability equal to 1. It is noteworthy that both slack values are further weighted by the activation probability of node τ_i. This allows more slack to be distributed to tasks that are more likely to be activated. The slack of τ_i is now minimum of these two slack values. Because the slack is the average for all possible minterms, at the end of the routine, we need to check for each path that the deadline can be met otherwise, the slack will be adjusted.

Once slack is calculated for a task, the task is stretched and its schedule and speed are locked. Next, all paths that span this task are updated in terms of their respective delay and slack. Effectively the delay and slack of these paths are reduced, reflecting the effect of the already stretched task. Updating these variables dynamically alters the criticality of paths for different nodes and subsequently releasing the tasks that are being stretched from consideration. The online task stretching heuristic then updates CTG and repeats the above mentioned procedure for another task following the task order generated by ordering algorithm.

Being simple to implement but energy efficient, online stretching heuristic also displays low complexity. Given a CTG with total nodes $|V|$ and edges $|E|$, the time complexity of step-1 and step-2 in

Figure 2 is $O(|V|^3 + |E|)$. Step-4 has a complexity of $O(|V|^3)$ while the complexity of step-5 and step-6 is O(1). Assume that the number of outgoing edges from each node can be bounded by a constant *C*, the time complexity of step-7 is $O(C|V|)$. The total time complexity of the online stretching heuristic is $O(2|V|^3 + C|V| + |E|)$. This low complexity enables the algorithm to be used for dynamic scheduling in a system with the capability of runtime branch prediction. The next subsection describes the algorithm that performs adaptive scheduling and DVFS.

B. Adaptive scheduling and DVFS

One of the limitations of the on-line algorithm is that its efficiency relies heavily on the accuracy of the probabilistic distribution of the branch selections. Such distribution is not fixed during the runtime. For example, the vehicle cruise controller selects to increase or decrease the reference speed based on the road condition. In other real-life application of MPEG video decoder, the decoding process keeps varying according to the contents of the visual scene. Each video frame in the encoded video stream is composed of various macroblocks that represents 16x16 pixel area of the image. The macroblock decoding is the core of the decoding process of the software MPEG player [16] which is repeated for the entire video stream. The macroblocks are encoded differently for changing visual scene and this difference impacts the workload of decoding process. Such a selective behavior can be easily depicted by a conditional task graph. Figure 3 shows the CTG for decoding one MPEG macroblock. In the figure, the light and dark dotted lines forked from the same node indicate mutually exclusive branches.

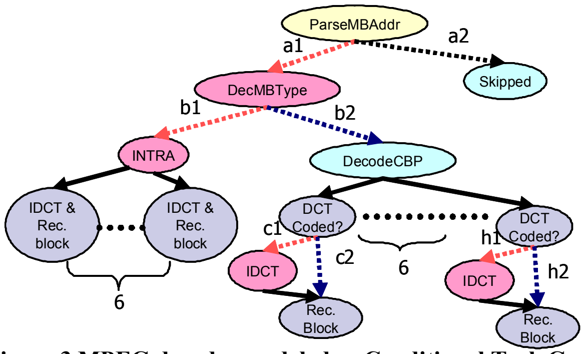


Figure 3 MPEG decoder modeled as Conditional Task Graph

Due to space limitations we can not show the whole graph which is little more complex than the one shown in Figure 3. The original

CTG consists of 40 tasks including 9 branching nodes. The task labeled as *Skipped* is a branch fork node with two branches. The branches in this CTG model of MPEG are marked as a~h with their respective numbers. Note that branches d, e, f and g are not shown in the graph, however, they are identical as branches c and h. The software first determines if a macroblock is a skipped block. If it is true, then branch a2 is taken otherwise branch a1 is taken and the software further determines if the macroblock belongs to type I. Again, if the answer is true then branch b1 is taken and otherwise branch b2 is taken. If the macroblock is an Intra block (type I block) then the IDCT function will be performed. Otherwise, there are 6 blocks that belong to a macroblock and each block may require or not require the IDCT function. This is represented by branches (c~h). To decode a macroblock in an I frame, branch a1 and b1 will be selected with probability 1. However, the major portion of a video stream belongs to B or P frames. To decode a macroblock in a B or P frame, all branches in the CTG have the chance to be selected.

We applied the software MPEG decoder to decode a sequence of 1000 macroblocks inside a video stream. This is equivalent as invoke the CTG in Figure 3 1000 times. The actual branch selection during this period is extracted and plotted in Figure 4 as the first data series (labeled as *Selection*.) A "1" ("0") indicates that branch b1 is selected (not selected). The branch probability within a window of 50 iterations is plotted as the second data series (labeled as *prob.*) The figure shows that the branch selection is a random variable and is very difficult to be predicted accurately. On the other hand, its probability distribution has relatively less variation during a longer period, and hence may be predictable. This is because nearby macroblocks tend to be encoded in a similar way due to the locality of images. However, local fluctuations and slow drifting of branch probability exist which motivates us to develop an adaptive algorithm which captures such changes.

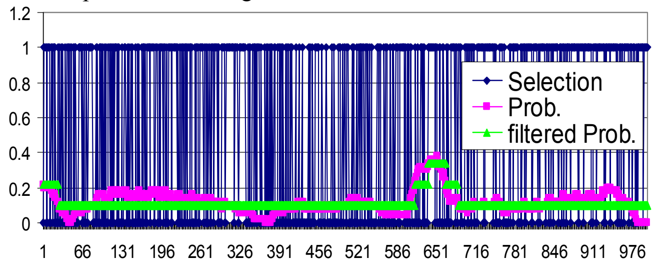


Figure 4 Dynamic branch selection in MPEG decoder

We propose a window-based adaptive algorithm for the best energy saving in a system with slowly varying branch selection probabilities. For each branch fork task, a fixed length buffer/window is maintained that stores the most recent *L* branch decisions pertaining to *L* instances of the CTG. Each time after a branch fork task is executed; a new branch decision is shifted into the buffer. The branch probabilities are then recalculated. If the difference between the new distribution and the old distribution is greater than a threshold value, the on-line scheduling and DVFS will be triggered. All the tasks will be executed with their newly evaluated speed until the next threshold crossing occurs. The third series of data in Figure 4 shows how our algorithm adapts to such dynamic behavior of the application. Each time the difference between the newly computed probabilities crosses the threshold, which is set to 0.1 in this example, the branch probability is updated with this new value. This update also results in invocation of scheduling and DVFS. Because the procedure is similar as a low-pass filter, the third data series in Figure 4 is labeled as *filtered Prob.* As we can see, the window size and the threshold determine how frequently the online scheduling and DVFS is called and they also impact how well the algorithm adapts.

IV. EXPERIMENTAL RESULTS

We begin by comparing the proposed online algorithm against the works suggested in [10] and [17]. We assume all algorithms have the accurate information about average branch probability and we do not consider the adaptive behavior of the online algorithm for a fair comparison. Also, for simplicity, we assume unit load capacitance to calculate energy and the only variable is speed/frequency of the PE. We do not consider switching overhead for DVFS. Table 1 shows the normalized energy dissipation of the 5 randomly generated CTGs under these three scheduling and DVFS algorithms, where the reference algorithm 1 and 2 represent the algorithms presented in [10] and [17] respectively. The energy results are normalized by taking energy given by online algorithm as a base of 100 in each case. The 5 CTGs are modified from the random task graphs by TGFF [14]. The branching probabilities for all branching nodes were randomly generated. The first column indicates the test CTG index while the second column displays the characteristics of the graph. We use a triplet (*a*/*b*/*c*) to characterize a test case where *a* represents the number of nodes in the CTG, *b* represents the number of PEs in the MPSoC and *c* represents the number of conditional branching nodes in the CTG. Online algorithm provides an average of 39% energy reduction over the reference algorithm 1. It results 8% more energy in average compared to the reference algorithm 2. However, the average runtime of reference algorithm 2 was 70 seconds while the online algorithm took merely 0.6 ms in average for each CTG, which represents about 120,000X average speedup. The speed up mainly comes from replacing the NLP based DVFS algorithm with a slack distribution based heuristic. As a matter of fact, the complexity of the NLP based algorithm is so high that we cannot apply the reference algorithm 2 to the MPEG problem.

Table 1 Energy consumption of online algorithm

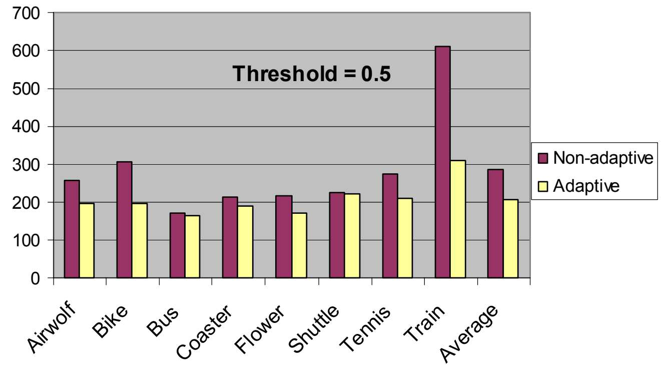
CTG	a/b/c	Reference Algorithm 1	Reference Algorithm 2	Online Algorithm
1	25/3/3	195	87	100
2	16/3/1	145	93	100
3	15/4/2	130	95	100
4	15/4/2	139	91	100
5	25/4/3	290	97	100

The next set of experiments compares the effectiveness of the online algorithm when being applied adaptively or non-adaptively. We first report the results achieved by applying the proposed algorithms on a software MPEG decoder [16]. The modeled CTG is shown in Figure 3. The multiprocessor system consists of 3 PEs. We inserted monitors in decoder code to record the branch selection by running real movie clips. The decisions of braches a~h are encoded as avector $<x_1, x_2, \ldots, x_n>$. The *i*th position of such vector indicates the branch decision for the ith branching node in the graph.

A sequence of 2000 vectors is generated from a movie chip. The first 1000 vectors are considered as a training sequence while the second 1000 vectors are considered as testing sequence. The non-adaptive online algorithm uses the profiled branch probability from the training sequence. From this point onwards we will use the terms *online* and *non-adaptive* interchangeably to refer to non-adaptive online algorithm. The adaptive algorithm uses a sliding window of size 20. The average energy for 1000 testing vectors is compared. All the movies except *Shuttle* are SIF resolution and the series of 1000 vectors constitutes little more than 3 video frames. The movie *Shuttle* is of lower resolution (QCIF) comprising of roughly 10 frames. We tested for a threshold of 0.5 and 0.1 for adaptive algorithm.

Figure 5 shows the average energy dissipation under the adaptive and online algorithm for eight different movie clips. The average energy savings of the adaptive algorithm over online algorithm for the threshold value of 0.5 is 21% while for threshold value of 0.1 is 23%.

Table 2 shows the number of times the online scheduling and DVFS was called for each movie. The average re-scheduling number is 9 (less than 3 per frame) for threshold T=0.5 and 162 for threshold T=0.1 (54 per frame). The results show that the appropriate threshold selection minimizes the overhead at negligible loss in energy savings (2% in this case).



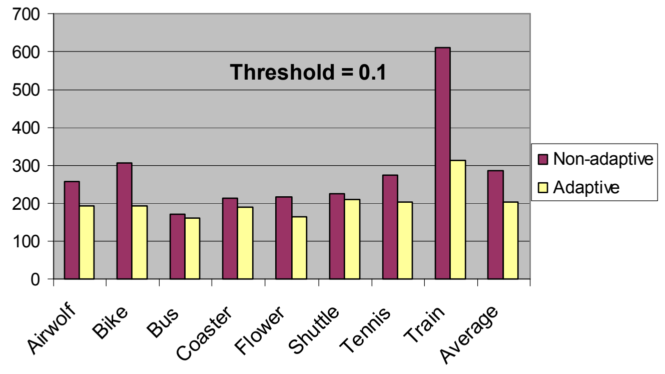


Figure 5 MPEG energy consumption with varying thresholds

Table 2 Algorithm call count for MPEG movies

Movie	Airwolf	Bike	Bus	Coaster	Flower	Shuttle	Tennis	Train
T=0.5	7	7	14	9	9	32	10	5
T=0.1	164	164	238	153	198	276	198	157

Table 3 Energy consumption of vehicle cruise controller system

Vector sequence	1	2	3
Non-adaptive	155	206	147
Adaptive	148	196	139

In our second experiment, we analyzed the energy consumption of a vehicle cruise controller system modeled as a conditional task graph [15]. The application is mapped on to a system with 5 PEs and consists of 32 tasks including two branching nodes. We generated three sets of 1000 vectors simulating a branch selection pertaining to real run of vehicle that encounters uphill, downhill, straight and bumpy road condition. Again, the first set of vectors is the training sequence which provides the profiled average branch probabilities for non-adaptive online algorithm. We then tested the adaptive and non-adaptive methods using all of the 3 sequences. We used threshold value of 0.1 for first two sets and 0.5 for the third vector set.

Table 3 shows the energy values. The energy savings over online algorithm in all three cases were almost identical hovering around 5%. The low energy savings can be contributed to the fact that there are only three minterms in the CTG model of the cruise control system. Also the deadline we used was double of the optimum schedule length. The combined effect results in less room for adaptive algorithm to extract its potential. The CTG typically has two minterms resulting from a same branching node that are almost equal in energy and thus change in probability least affects the energy. Although the non-adaptive method has perfect information on the long term average of the branch probability for sequence 1, it still provides less optimal scheduling than the adaptive algorithm because it does not consider the local fluctuation of the probability distribution. The call count to adaptive algorithm (re-scheduling) was 150 in average for 0.1 threshold value and 9 for threshold of 0.5.

Finally, we tested our algorithm on some random conditional task graphs which are modified from the task graphs generated by TGFF [14]. The MPSoC architecture consists of either 3 or 4 PEs. We tested 10 different graphs with two different graph structures. Graphs 1~5 are fork-join task graphs and they contain nested conditional branches. They will be referred as Category 1 CTG. Graphs 6~10 do not have fork-join structure or nested conditional branch. And they will be referred as Category 2 CTG. Both MPEG and cruise controller CTGs belong to category 1. Our experimental results show that the adaptive algorithm favors the application in the first category.

Observed from the MPEG decoding application, the average probability fluctuation per branch was 0.4~0.5 during runtime. We generated testing vectors for random CTGs with similar fluctuation in branch probabilities. The test vectors are generated in a way so that the average probabilities of all branches of any branching node for the entire set of vectors were equal. However there was considerable fluctuation. Three scenarios are considered for the non-adaptive algorithm

1. The profiled average branch probability favors the minterm with the lowest energy.
2. The profiled average branch probability favors the minterm with the highest energy.
3. The profiled average branch probability is accurate.

For the adaptive algorithm, a window size of 20 and threshold of 0.1 and 0.5 are used.

Table 4 shows energy results for online algorithm profiled for lowest energy minterm bias. The overall average energy savings of adaptive algorithm over online in this case is approximately 22% for 0.5 threshold and 23% for 0.1. Because of the inaccurate profiled information, the non-adaptive algorithm provides efficient scheduling for low energy minterms. Any occurrence of higher energy minterms imposes severe penalty. Adaptive algorithm on the other hand does not depend on profile information and also sustain the local fluctuation in probabilities inside the vector set. There is an exception in case of CTG4 though. Energy given by adaptive algorithm for both threshold values is higher compared to online algorithm. We analyzed the case and observed that CTG4 has only three minterms that are almost equal in energy and thus online algorithm performs very well in this case. We also noticed that average improvement in case of Category 1 CTGs is 8% higher than the Category 2 CTGs. The results favors adaptive algorithm for nested CTGs. We would be interested in future to verify this trend. The number of calls to online scheduling and DVFS is also listed in each case and the trend is similar to MPEG experiment.

Table 5 shows the results with online algorithm profiled for highest energy minterm bias. The energy savings in this case is now 3% and 5% for threshold values of 0.5 and 0.1 respectively. The online energy reduces considerably as the misprediction penalty only occurs for lowest energy minterm. The average energy savings of Category 1 CTGs is 7% compared to 3% in case of Category 2 CTGs. Although the vector set used to evaluate the energy is same, the energy results for adaptive algorithm are slightly different for same threshold values in both tables. This is because the initial branch probabilities of algorithm are taken same as the profiled probabilities of online algorithm, which is different in both cases.

Table 4 Energy savings with online algorithm profiled for lowest energy minterm bias vector set

CTG	a/b/c	Online	Adaptive			
			Threshold = 0.5		Threshold = 0.1	
			Energy	# of calls	Energy	# of calls
1	25/3/3	329	148	10	132	251
2	16/3/1	578	532	3	530	164
3	15/4/2	263	193	8	183	203
4	15/4/1	471	557	3	544	187
5	25/4/3	381	165	10	166	240
6	25/3/3	877	535	8	529	223
7	16/3/1	494	453	4	482	104
8	15/4/2	332	268	5	254	227
9	15/4/1	488	452	3	453	174
10	25/4/3	299	240	9	237	231

Table 5 Energy savings with online algorithm profiled for highest energy minterm bias vector set

CTG	a/b/c	Online	Adaptive			
			Threshold = 0.5		Threshold = 0.1	
			Energy	# of calls	Energy	# of calls
1	25/3/3	150	147	10	131	251
2	16/3/1	538	532	3	530	164
3	15/4/2	190	192	8	182	203
4	15/4/1	575	558	3	544	187
5	25/4/3	186	165	10	165	240
6	25/3/3	522	530	8	525	223
7	16/3/1	521	454	4	482	104
8	15/4/2	237	266	5	252	227
9	15/4/1	531	451	3	452	174
10	25/4/3	240	240	9	236	231

Figure 6 shows comparison of the energy consumption of the non-adaptive algorithm with ideal profiling information versus the adaptive algorithm. The threshold value of 0.5 was used. The resulting energy graph is shown in Figure 6. The overall energy savings of adaptive algorithm over non-adaptive algorithm in this case is 10%. The average improvement is 16% for Category 1 CTGs and 5% for Category 2 CTGs.

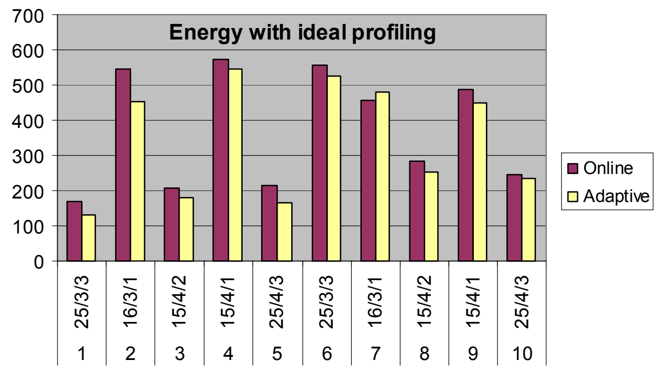


Figure 6 Energy consumption with ideal profiling

V. CONCLUSIONS

A framework for task mapping, scheduling and DVFS is proposed for real-time applications with non-deterministic workload running on multiprocessor platform. The proposed algorithm adapts to rapidly changing system and input conditions, which affect the entire execution flow, to achieve better energy savings by utilizing the profile information. Experimental results show the effectiveness and applicability of proposed approach for variety of real-life applications. Our future efforts target the development of mathematical model to verify the capability of proposed approach for wide range of applications.

REFERENCES

[1] X. Zhong, C.Z. Xu, "Energy-aware Modeling and Scheduling of Real-time Tasks for Dynamic Voltage Scaling," *Proceedings of Real-Time Systems Symposium, December 2005.*

[2] A. Sinha, A.P. Chandrakasan, "Dynamic Voltage Scheduling using Adaptive Filtering of Workload Traces," *Proceedings of Fourteenth International Conference on VLSI Design, January, 2001.*

[3] C. Xian, Y.H. Lu, Z. Li, "Energy-Aware Scheduling for Real-Time Multiprocessor Systems with Uncertain Task Execution Time," *Proceedings of Design Automation Conference, June, 2007.*

[4] J. Luo and N. K. Jha, "Static and Dynamic Variable Voltage Scheduling Algorithms for Real-time Heterogeneous Distributed Embedded Systems," *Proceeding Of International Conference on VLSI Design*, pp.719-726, 2002.

[5] Y. Zhang, X. Hu, and D. Z. Chen, "Task Scheduling and Voltage Selection for Energy Minimization," *In Proc. Of Design Automation Conference*, pp.183-188, 2002.

[6] J. Hu and R. Marculescu, "Energy-Aware Communication and Task Scheduling for Network-on-Chip Architectures under Real-Time Constraints," *Proceeding of Conference and Exhibition on Design, Automation and Test in Europe*, 2004.

[7] P. Eles, K. Kuchcinski, Z. Peng, A. Doboli, and P. Pop, "Scheduling of Conditional Process Graphs for the Synthesis of Embedded Systems," *Proceedings of Design, Automation and Test in Europe*, 1998.

[8] Y. Xie and W. Wolf, "Allocation and Scheduling of Conditional Task Graph in Hardware/Software Co-synthesis," *Proceedings of Conference and Exhibition on Design, Automation and Test in Europe*, 2001.

[9] D. Wu, B.M. Al-Hashimi and P. Eles, "Scheduling and Mapping of Conditional Task Graph for the Synthesis of Low Power embedded Systems," *IEE Proceedings of Computers and Digital Techniques*, Volume 150, Issue 5, pp. 262-273, Sept. 2003.

[10] D. Shin and J. Kim, "Power-Aware Scheduling of Conditional Task Graphs in Real-Time Multiprocessor Systems," *Proceedings of International Symposium on Low Power Electronics and Design*, 2003.

[11] E. Jacobsen, E. Rotenberg, and J.E. Smith, "Assigning Confidence to Conditional Branch Predictions," *Proceedings of the 29th Annual International Symposium on Microarchitecture*, Nov. 1996.

[12] A. K. Uht and V. Sindagi, ''Disjoint Eager Execution: An Optimal Form of Speculative Execution,'' *Proceedings of the 28th Annual International Symposium on Microarchitecture, Nov. 1995.*

[13] G.C. Sih and E.A. Lee. "A Compile Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architecture," *IEEE Transactions on Parallel and Distributed Systems*, Volume 4, Issue 2, Page(s):175 – 187, Feb. 1993.

[14] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task graphs for free," *Proc. of Int. Workshop Hardware/Software Codesign*, Mar. 1998.

[15] Paul Pop, "Scheduling and Communication Synthesis for Distributed Real-time Systems", *Ph.D. thesis, Linkopings University,2000.*

[16] http://bmrc.berkeley.edu/frame/research/mpeg

[17] P. Malani, P. Mukre, Q. Qiu, "Profile-Based Low Power Scheduling for Conditional Task Graph: A Communication Aware Approach," *Proceedings of IEEE International Symposium on Circuits and Systems*, May 2007.