Power Optimization for Conditional Task Graphs in DVS Enabled Multiprocessor Systems

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

Abstract — **In this paper, we focus on power optimization of real-time applications with conditional execution running on a dynamic voltage scaling (DVS) enabled multiprocessor system. The targeted system consists of heterogeneous processing elements with non-negligible inter-processor communication delay and energy. Given a conditional task graph (CTG), we have developed novel online and offline algorithms that perform simultaneous task mapping and ordering followed by task stretching. Both algorithms minimize the mathematical expectation of energy dissipation of non-deterministic applications by considering the probabilistic distribution of branch selection. Compared with existing CTG scheduling algorithms, our online and offline scheduling algorithms reduce energy by 28% and 39% in average, respectively.**

I. INTRODUCTION

Multiprocessor System-on-Chip (MPSoC) is becoming a major system design platform for general purpose and real-time applications, due to its advantages in low design cost and high performance. Minimizing the power consumption is one of the major issues in designing battery operated MPSoC. One of the widely used power reduction technique is *Dynamic Voltage Scaling (DVS)*, which allows the processor to dynamically alter its speed and voltage at run time to trade power for performance.

In a multiprocessor system, the mapping and ordering of tasks changes the task slack time, i.e. the intervals when a processing element (PE) is idle, and hence have a significant impact on the efficiency of DVS. As the system complexity grows, the latency and energy of inter-processor communication increases. A holistic technique must be developed for task mapping, ordering and stretching to reduce both communication and computation energy.

Many of the real-time applications are non-deterministic. The application is divided into several tasks. Some tasks are activated only if certain conditions evaluated by previously executed tasks are true. A conditional task graph (CTG) [4]~[7] captures such relation and hence enables us to model more general application.

Although conditional 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. Techniques that dynamically assign confidence levels [8] or probabilities [9] to the conditional branches have been proposed by previous research works.

In this work, we propose a set of *communication aware and profile-based* (CAP) scheduling algorithms for CTG on a DVS enabled multiprocessor system. The targeted system has a set of heterogeneous PEs, such as DSPs, FPGAs or ASICs, that are connected by interconnect network. The energy and delay for inter-PE communication is not negligible. Each PE has DVS capability. We assume that the branch probabilities are available through static or dynamic branch profiling.

Online and offline CAP scheduling algorithms are presented in this paper. They consider task mapping, task ordering and task stretching altogether to minimize energy dissipation and also satisfy the performance constraint. The task mapping and task ordering are performed simultaneously and their goal is to minimize the inter-processor communication and maximize the task slack. The task stretching algorithm finds the best speed and starting time for each task so that the computing energy is minimized. The algorithms consider the branch probabilities and they minimize the mathematical expectation of energy dissipation.

The offline CAP algorithm formulates and solves the task stretching problem as a *linear programming* (LP) problem which is time consuming. The online CAP algorithm replaces the LP based algorithm with a heuristic algorithm. Experimental results show that compared with the offline algorithm, the online CAP heuristic is 120,000X faster and almost as effective as offline CAP.

Many techniques have been proposed that consider the task mapping and ordering for DVS [1]~[3]. However, these algorithms only consider traditional data-flow graph without conditional execution. One of the major characteristic of CTG is that some tasks are mutually exclusive. These tasks can be mapped to the same PE at the same time. Reference [4] and [5] consider scheduling and mapping for CTG, however, they do not minimize energy dissipation. Wu et al. [6] proposed an algorithm for task ordering and stretching of CTGs running on a DVS enabled system. They search for the optimal task mapping using genetic algorithm (GA). The proposed algorithm provides a complete solution for power optimization of CTGs. However, it does not consider the branch probabilities. An implied assumption of this technique is that all the conditional branches will be selected with equal probability. Furthermore, the GA based task mapping algorithm has high complexity because the inner loop of this algorithm needs to perform the task ordering and stretching of the entire CTG. Shin et al. [7] proposed an algorithm for task ordering and stretching of CTG which considers the run-time behavior. They refer this approach as condition aware scheduling. Under condition aware scheduling, a task has different start time and speed for different combinations of possible branch selections. Therefore, a large table is needed to store the scheduling result. The probabilistic distribution of the branch selections is considered only during task stretching. Another limitation of this algorithm is that it takes task mapping as a fixed input so that the communication overhead cannot be considered.

The characteristics of the proposed CAP algorithms are described as follows.

1. The proposed algorithms consider task mapping, ordering and task stretching altogether for energy reduction.

2. We consider the application with conditional execution as a random procedure. The algorithm explores the fact that the conditional branches will be selected with different probabilities. The algorithm utilizes the probabilistic information that is collected through static or dynamic branch profiling. Its objective is to minimize the mathematical expectation of energy dissipation.

3. Offline and online versions of the CAP algorithm are proposed. The offline algorithm is a compile time scheduling algorithm while the online algorithm has very low complexity so that it can be used with runtime branch prediction for dynamic scheduling.

The experimental results show that, comparing with the scheduling algorithms presented in [7], the offline and online CAP algorithms provide an average of 39% and 28% energy saving respectively.

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 [7]. 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*. 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.

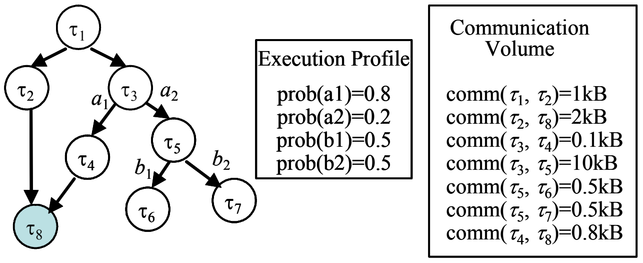
The condition that the task τ is activated is denoted as $X(\tau)$. The condition of an and-node τ_i can be written as $\wedge_{\tau_k}\left(C(\tau_k,\tau_i)\wedge X(\tau_k)\right)$, where τ_k is the predecessor of τ_i. The condition of an or-node τ_j can be written as $\vee_{\tau_k}\left(C(\tau_k,\tau_j)\wedge X(\tau_k)\right)$, where τ_k is the predecessor of τ_j. A *minterm* m is a possible combination of all conditions of the CTG. We use M to denote the set of all possible minterms of a 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 $\mathrm{Comm}(\tau_i, \tau_j)$ 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.

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 the 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.



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.

III. Proposed Scheduling Algorithm

A. Offline CAP algorithm

This section provides an insight into the offline version of the communication aware profile-based (CAP) scheduling algorithm. The algorithm can be divided into two steps. The first step finds task mapping and ordering while the second step finds the task starting time and task speed.

1) Task mapping and ordering

Both online and offline CAP algorithms use the same task mapping and ordering algorithm. It is based on *Dynamic Level based Scheduling* (DLS) proposed by [10]. 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 candidate list, the dynamic level $DL(\tau_i, p_j)$ is calculated using the following formula:

$$DL(\tau_i,p_j)=SL(\tau_i)-\max\left\lfloor DA(\tau_i,p_j),TF(p_j)\right\rfloor, \qquad (1)$$

where p_j is one of the processing elements, $SL(\tau_i)$ is the static level of task τ_i, which is equal to the longest distance from node τ_i to any of the end nodes in the task graph, $DA(\tau_i, p_j)$ is the earliest time that all data required by node τ_i is available at the jth PE with the consideration of both computation and communication delay, and $TF(p_j)$ is the time that the last task assigned to the jth PE finishes its execution. The pair of (τ_i, p_j) which gives the maximum dynamic level will be selected and the mapping is performed accordingly. The task is scheduled to be started at the time $\max\left\lfloor DA(\tau_i,p_j),TF(p_j)\right\rfloor$. After that, the candidate list is updated and the dynamic level of each task in the candidate list is re-calculated.

In this work, we modified the DLS algorithm to consider the mutual exclusiveness among conditional tasks and also consider the probabilistic distribution of branch selection.

The static level $SL(\tau_i)$ is calculated using a dynamic program. The algorithm starts calculating SL of end nodes first and traversing whole graph upwards by updating SL of each node. Since we assume heterogeneous processor environment, we take the average *WCET* (denoted by *$WCET$) for each task to account for variability in execution time on different processors.

The static level of a non-branching node is the maximum static level of its successors plus the *$WCET$ of itself. Let $S(\tau_i)$ be the set of successor nodes of τ_i, equation (2) calculates the static level of a non-branching node.

$$SL(\tau_i) = {}^*WCET(\tau_i) + \max SL(\tau_j), \tau_j \in S(\tau_i) \qquad (2)$$

The static level of a branch fork node is the mean of the static level of all its successors plus the *WCET of itself. Let c_{ij} denote the condition of edge (τ_i, τ_j), equation (3) calculates the static level of a branch fork node.

$$SL(\tau_i) = {}^*WCET(\tau_i) + \textstyle\sum_j prob(c_{ij}) * SL(\tau_j), \quad \tau_j \in S(\tau_i) \qquad (3)$$

The main idea of our mapping and ordering algorithm is to find the most critical path in terms of execution cycles for each node while considering probability of execution for each path. The SL remains constant for each node once calculated. We also modified the calculation of Dynamic Level (DL) to account for the mutual exclusiveness among conditional tasks. The dynamic level of task processor pair (τ_i, p_j) can be calculated as the following.

$$DL(\tau_i, p_j) = SL(\tau_i) - AT(\tau_i, p_j) + \delta(\tau_i, p_j) \qquad (4)$$

The term $\delta(\tau_i, p_j)$ is the difference between *$WCET(\tau_i)$ and $WCET(\tau_i, p_j)$ which accounts for heterogeneous processor architecture. Adding this offset ensures correct evaluation of a task's DL for different processors since SL is computed using average $WCET$. $AT(\tau_i, p_j)$ is the earliest time that task τ_i can start on processor p_j. It must satisfy the following two conditions:

- At time $AT(\tau_i, p_j)$ all the data required by τ_i is available at p_j, i.e. $AT(\tau_i, p_j) \geq DA(\tau_i, p_j)$.
- If task τ_j is scheduled during the interval [$AT(\tau_i, p_j)$, $AT(\tau_i, p_j) + WCET(\tau_i, p_j)$], then τ_j and τ_i are mutually exclusive. This condition allows two mutually exclusive tasks to share the same processor at the same time, and thus making the schedule more efficient.

Computations and communications could be overlapped considering the availability of dedicated communication resource. Multiple data transfers from same node to different nodes are serialized provided the data values are different.

Our mutual exclusion detection procedure for each task is based on branch labeling method discussed in [5]. Considering example CTG of Figure 1 our algorithm detects tasks τ_6 and τ_7 to be mutually exclusive. Some other combinations are not mutually exclusive. For example, when condition a_2 is evaluated to be true, both τ_8 and τ_5 will be executed, and thus are not mutually exclusive. Our algorithm also detects the mutual exclusiveness among data transfers.

Figure 2 shows the flow diagram of our task ordering algorithm. The algorithm begins with the generation of initial ready list which has all start nodes. For each possible (τ_i, p_j), where τ_i is one of the tasks in the ready list, the algorithm calculates the static level and then finds the best pair that has the highest DL given by (4). Task τ_i is then 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.

Our algorithm that searches for $AT(\tau_i, p_j)$ is similar to the Find_AvailableTime() routine in [7]. However, in the Find_AvailableTime() routine, τ_i will be scheduled immediately after the first available time is found and the CTG will be updated. Our algorithm simply returns the first available time without modifying the CTG because not all τ_i will be scheduled to its first available time. We will update the CTG after the best pair of (τ_i, p_j) is selected and scheduled. The term $AT(\tau_i, p_j)$ also captures communication scheduling. For example, in Figure 1 if τ_3 is scheduled before τ_2, comm(τ_1, τ_3) will be scheduled before comm(τ_1, τ_2) as the data transfers from τ_1 are serialized.

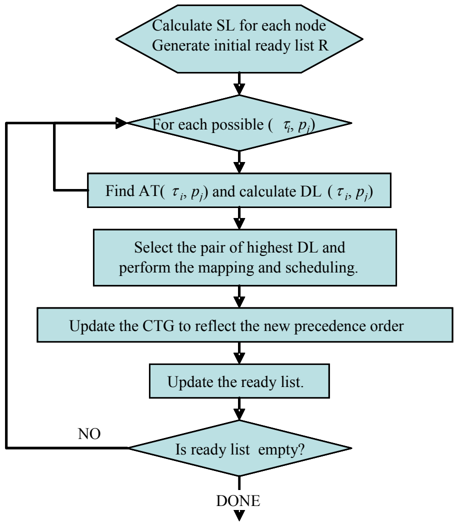

Calculate SL for each node
Generate initial ready list R
For each possible (τ_i, p_j)
Find AT(τ_i, p_j) and calculate DL (τ_i, p_j)
Select the pair of highest DL and perform the mapping and scheduling.
Update the CTG to reflect the new precedence order
Update the ready list.
NO
Is ready list empty?
DONE

Figure 2 Task ordering algorithm flow.

Unlike the algorithm presented in [7], our task mapping and ordering algorithm is not condition aware. The mapping and ordering of each task will not change at runtime even if some conditional branch has been selected. The benefits of using a condition unaware scheduling algorithm are low computation complexity and less storage requirement, both of which are essential for online scheduling. The offline CAP algorithm is overall condition aware because it uses a condition aware task stretching algorithm. However, as we will show later, the online algorithm is condition unaware. Only one speed will be selected for each task and hence the storage of the scheduling results is simplified.

Another major difference between the proposed algorithm and the previous works [6][7] is that the proposed algorithm utilizes the profiled information of the branch probabilities and it reduces the average schedule length instead of the worst case schedule length.

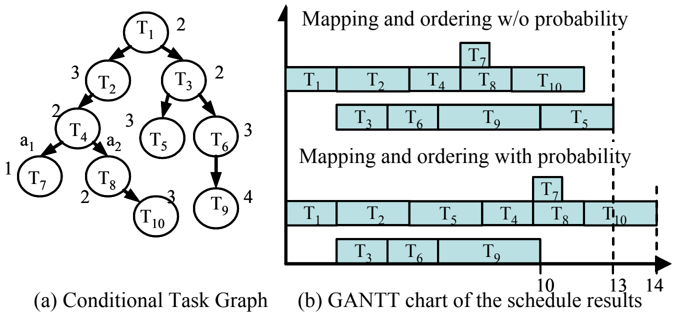

T1 2
3 T2
T3 2
2 T4
a1
a2
3 T5
T6 3
1 T7
2 T8
T10 3
T9 4
Mapping and ordering w/o probability
T7
T1 T2 T4 T8 T10
T3 T6 T9 T5
Mapping and ordering with probability
T7
T1 T2 T5 T4 T8 T10
T3 T6 T9
10
13
14
(a) Conditional Task Graph
(b) GANTT chart of the schedule results

Figure 3 Considering branching probabilities in task ordering

Example 2: Consider the CTG given in Figure 3 (a). The $WCET$ of each task is given beside the circle. The branches a_1 and a_2 are taken with probability 0.9 and 0.1. Assume that the system has 2 PEs and the latency for inter-processor communication is negligible. Without considering the branching probability, T_4 has higher DL than T_5 and it will be mapped to PE1 and T_5 will not be scheduled until the very end. No matter which branch is taken, the overall schedule length is 13. With the branching probability, T_5 has higher DL than T_4. Therefore, it is mapped to PE1 before T4. The schedule length is 10 or 14 depending on whether the branch a_1 or a_2 is taken. Therefore, the average schedule length is 10.4. The reduced schedule length will be transformed to energy saving using task stretching. Table 3 (b) shows the GANTT chart of the schedule results with or without considering the profile information.

2) *Task stretching*

The task stretching routine finds the best starting time and speed of each task that minimizes energy while meeting the performance constraints. The problem is formulated and solved as a constrained linear program. The task stretching algorithm consists of three steps:

Step1: Preprocess the CTG to capture the implied control dependency between the or-node and its related branch fork nodes. As example 1 shows, the precedence requirements of an or-node is unknown until all of its related branch fork nodes have finished. Therefore, for an or-node τ_i and a condition c, if c is one of the literals in $\Gamma(\tau_i)$ and τ_j is the branch fork node of c, then an edge (τ_j, τ_i) must be inserted into the CTG.

Step 2: Duplicate the task graph. For each minterm $m \in M$, a task graph $G_m=(V_m, E_m)$ is created based on the CTG. A task $\tau_i \in V_m$, if τ_i is activated when m is true, i.e. $m \oplus X(\tau_i) \neq 0$. For two nodes τ_i and τ_j in V_m, if there is an edge (τ_i, τ_j) in the original CTG, then the same edge will be added in G_m.

Step 3: Formulate and solve the task stretching problem as a linear program (LP). For each task τ and a minterm $m \in \Gamma(\tau)$, two variables $\sigma_\tau(m)$ and $f_\tau(m)$ are defined. They represent the start time and speed of τ respectively when minterm m is true. The task stretching problem can be formulated into the following linear program:

$$\min \sum_{m \in M, m \neq 1} prob(m) \sum_{\tau \in V_m} \frac{E(\tau, p_\tau)}{F_\tau\ (m)^2} \quad \text{s.t.} \tag{5}$$

$$\sigma_{\tau_i}(m) + \frac{WCET(\tau_i, p_{\tau_i})}{F_{\tau_i}(m)} + \frac{Comm(\tau_i, \tau_j)}{B(p_{\tau_i}, p_{\tau_j})} \leq \sigma_{\tau_j}(m),$$
$$\forall m \in M, m \neq 1, \forall \tau_i, \tau_j \in V_m \text{ and } (\tau_i, \tau_j) \in E_m \tag{6}$$

$$\sigma_{\tau_i}(m) + \frac{WCET(\tau_i, p_{\tau_i})}{F_{\tau_i}(m)} \leq deadline,$$
$$\forall m \in M, m \neq 1, \forall \tau_i \in V_m . \tag{7}$$

In the above equations, p_τ is the processor that task τ is mapped to. $prob(m)$ is the probability that minterm m is true. It can be calculated based on the branch probability. The symbol $F_\tau(m)$ is $f_\tau(m)$, if $m \in \Gamma(\tau)$; otherwise, it is $f_\tau(m')$, where $m' \in \Gamma(\tau)$ and $m' \oplus m \neq 0$.

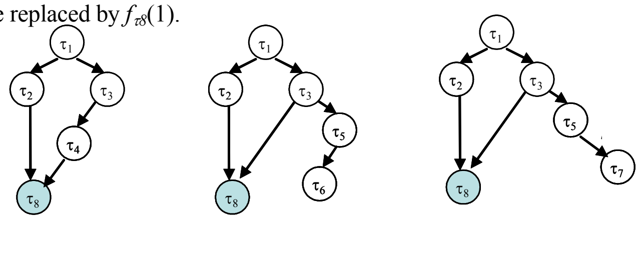
Equation (5) specifies that the objective of the LP is to minimize the mathematical expectation of the energy dissipation. Equation (6) and (7) specify the precedence constraints and deadline constraints of the tasks.

Example 2: Consider the CTG given in Figure 1. Because $\Gamma(\tau_8) = \{1, a_1\}$ and the branch fork node of condition a_1 is τ_3, the edge (τ_3, τ_8) is inserted to represent the control dependency between the or-node τ_8 and the branch fork node τ_3. Then for each minterm in M a task graph is generated, which consists of only the nodes that will be activated in the corresponding minterm. Figure 4 (a)~(c) show the new task graphs. The probabilities of minterms are: $prob(a_1)$=0.8, $prob(a_2b_1)$=0.1, and $prob(a_2b_2)$=0.1.

The LP has 18 variables:

- $\sigma_1(1)$, $f_1(1)$, $\sigma_2(1)$, $f_2(1)$, $\sigma_3(1)$ and $f_3(1)$ are the start time and speed of task $\tau_1 \sim \tau_3$. They do not depend on which condition branch is selected.
- $\sigma_4(a_1)$ and $f_4(a_1)$ are the start time and speed of task τ_4 when branch a_1 is selected.
- $\sigma_5(a_2)$ and $f_5(a_2)$ are the start time and speed of task τ_5 when branch a_2 is selected.
- $\sigma_6(a_2b_1)$ and $f_6(a_2b_1)$ are the start time and speed of task τ_6 when branch a_2 and b_1 are both selected.
- $\sigma_7(a_2b_2)$ and $f_7(a_2b_2)$ are the start time and speed of task τ_7 when branch a_2 and b_2 are both selected.
- $\sigma_8(a_1)$ and $f_8(a_1)$ are the start time and speed of task τ_8 when branch a_1 is selected while $\sigma_8(1)$ and $f_8(1)$ are the start time and speed of task τ_8 when branch a_2 is selected.

Based on the definition of $F_\tau(m)$, the symbol $F_{\tau 8}(a_1)$, in equation (5)~(6) will be replaced by $f_{\tau 8}(a_1)$, while $F_{\tau 8}(a_2b_1)$ and $F_{\tau 8}(a_2b_2)$ will be replaced by $f_{\tau 8}(1)$.



(a) $m=a_1$ (b) $m=a_2b_1$ (c) $m=a_2b_2$

Figure 4 Task graph duplication.

B. *Online CAP algorithm*

Although solving a constrained linear program is a tractable problem, it is still time consuming. As we show in our experimental results, the runtime of the algorithm is significantly high especially for the graphs with many nodes.

As an alternative solution to the stretching problem we suggest a heuristic which performs task stretching with negligible runtime at the reasonable loss of energy saving compared to offline CAP algorithm. The heuristic is based on critical path based slack distribution algorithm considering conditional execution and is similar to the one proposed in [6]. However, there are few noteworthy changes. 1) The DVS techniques suggested in [6] and [7] have multiple speeds for a single task corresponding to different minterms. Therefore, the algorithms have a high complexity and a large schedule table is needed to store the stretching information. 2) The technique in [6] applies same stretching ratio at a time to all PEs, which is suboptimal. 3) The task stretching algorithm in [6] does not consider the branch probabilities. Unlike these techniques, our heuristic algorithm is an online approach in that it does not necessitate a schedule table to store the stretching information and use it at runtime. It is a profile-based approach considering branch probabilities. It calculates only single speed for each task and it can facilitate different scaling ratio for different PEs. Since the heuristic has very low complexity, the whole online CAP algorithm including task ordering and stretching can be called iteratively during runtime when branch probability changes significantly.

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. Associated with each task τ on path p, there is a probability *prob*(*p*, τ), which gives the probability of path *p* given the condition that task τ is started. *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 $\tau_1\text{-}\tau_3\text{-}\tau_5\text{-}\tau_6$ after node τ_5 is b_1.

```
Online CAP task stretching heuristic for CTG G
1. Process initial schedule generated by task ordering algorithm
   shown in Figure 2
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 $slk(\tau_i)>deadline-delay(p)$
10. $slk(\tau_i)=deadline-delay(p)$;
```

Figure 5 Online CAP task stretching heuristic.

For each task, step4 in Figure 5 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 started. One more slack value is calculated as shown in step 7 for critical path with probability equal to 1. It is noteworthy that both slack values are further weighted by the activation probability of node τi. More slack will be allocated to the task that has higher probability 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. Updating these variables dynamically alters the criticality of paths for different nodes and subsequently releasing the tasks, that have been stretched, from consideration. The online CAP algorithm then updates CTG and repeats the above mentioned procedure for another task following the task order generated by offline CAP.

Given a CTG with total nodes |V| and edges |E|, the time complexity of the online stretching heuristic (derivation not shown here due to space limitation) is $O(2|V|^3 + C|V| + |E|)$ where constant C is an upper bound on number of outgoing edges from a node. This low complexity enables the algorithm to be used for dynamic scheduling in a system with the capability of runtime branch prediction.

IV. Experimental results

Simulations have been carried out to evaluate the efficiency of the proposed algorithm. Six scheduling algorithms are evaluated in the experiments. They are: offline CAP, CAP w/o PM (offline CAP without profile-based mapping), CAP w/o PS (offline CAP without profile-based stretching), online CAP w/o PM, online CAP and Reference algorithm. We implemented the scheduling algorithm in [7] according to the best of our ability and denote it as Reference algorithm. The Reference scheduling does not consider profile information in task ordering and it assumes fixed task mapping. Table 1 summarizes the characteristics of the 6 scheduling algorithms. For all the experiments, we consider the execution of the CTG as a random process and we report the mathematical expectation of energy dissipation.

Table 1 Difference of the evaluated scheduling algorithms.

Algorithm	Task ordering		Task stretching		Flexible mapping
	Profile based	Condition aware	Profile based	Condition aware	
Offline CAP	X		X	X	X
CAP w/o PM			X	X	X
CAP w/o PS	X			X	X
Online CAP w/o PM			X		X
Online CAP	X		X		X
Reference		X	X	X	

Firstly, a real life example of a vehicle cruise controller system [12] is experimented for different algorithms. This application is modeled as a CTG with 32 tasks and two branching nodes, each of them forking two conditions. The original cruise control model in [12] did not have probability information, so it is assigned randomly in our experiment. We tested this CTG with fix mapping and no communication between tasks. The application is mapped on five different PEs. The schedule length of 124ms reported by CAP was same as Reference algorithm and the best length reported in [12]. The energy consumption is 102 mJ for both CAP offline and online and same for the Reference algorithm. The results favor the CAP online approach for this application due to its low complexity and equal energy compared to other algorithms.

Next, five test cases are randomly created with different CTGs and different MPSoC architecture. The CTGs are modified from the random task graphs generated by TGFF [11]. The MPSoC architecture consists of either 3 or 4 PEs.

Table 2 Fixed mapping and zero communication cost.

CTG	a/b/c	Offline CAP	CAP w/o PS	Online CAP	Reference
1	25/3/3	596	1414	723	874
2	16/3/1	591	769	641	568
3	15/4/2	223	490	292	215
4	15/4/1	386	630	497	386
5	25/4/3	285	1075	498	501

The first experiment focuses on demonstrating the effectiveness of our task ordering algorithm. The same fixed task mapping is used in both Reference and CAP algorithms. The communication cost is also set to be 0. Table 2 shows the energy dissipation of different scheduling algorithms. Second column of Table 2 displays the characteristics of the CTGs we used. 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. These five

CTG IDs shown in column 1 are used to report results for all experiments in rest of this paper. In average, the CAP has 13% energy reduction over the reference scheduling and 49% energy reduction over the CAP w/o PS. The online CAP has 11% more energy than the Reference algorithm. This is expected since the potential of the DLS based algorithm is known to unfold with flexible mapping.

The second experiment focuses on demonstrating the effectiveness of our task mapping algorithm. In this experiment, the CAP based algorithms perform task mapping together with task ordering. The communication cost is again set to 0. Table 3 shows the energy dissipation of different scheduling algorithms. As we can see, with flexible task mapping, the offline CAP gives more than 43% energy reduction over the Reference algorithm. The improvement is more significant than the first experiment. The difference between offline CAP versus CAP w/o PS is now 46%. The online CAP has an average of 39% of energy reduction over the Reference algorithm. Compared to offline CAP, the online CAP has 8% more energy dissipation. Average energy dissipation of offline CAP and CAP w/o PM is almost same while online CAP has 7% less energy than online CAP w/o PM. This is because offline CAP tries to allocate slack based on critical path while online CAP stretching heuristic calculates slack based on average path length.

Table 3 Flexible mapping and zero communication cost.

CTG	Offline CAP	CAP w/o PS	CAP w/o PM	Online CAP w/o PM	Online CAP	Reference
1	389	1014	382	459	449	874
2	365	498	365	393	393	568
3	158	339	144	187	166	215
4	252	353	260	325	277	386
5	167	422	160	183	173	501

Table 4 shows the comparison of runtime between the offline CAP and online CAP. Note that the unit of the runtime for offline CAP is second while the unit for the online CAP is millisecond. The average speedup of the online CAP is 120,000X. Although the time reported here includes task ordering, mapping and stretching, because both the online and offline algorithms use the same task ordering/mapping routine, the difference in runtime comes only from task stretching step.

Table 4 Comparison of runtime.

CTG	1	2	3	4	5
Offline CAP (sec)	72.69	66.41	37.06	24.11	147.28
Online CAP (msec)	1.26	0.59	0.37	0.31	0.58

Table 5 Flexible mapping and non-zero communication cost.

CTG	Offline CAP	CAP w/o PS	CAP w/o PM	Online CAP w/o PM	Online CAP	Reference
1	422	1146	487	534	508	1396
2	374	522	441	469	404	1206
3	194	592	237	225	211	356
4	413	615	455	546	469	805
5	218	620	254	250	242	777

The last experiment focuses on demonstrating the communication aware capability of our algorithm. In this experiment, we generate the communication volume from a node to its successor based on Computation to Communication ratio (CCR).For this experiment, we chose a value of CCR=1 with some variance (10%) to model data transfer from a node to multiple nodes. Table 5 shows the energy dissipation of different scheduling algorithms. Since the communication cost is non-zero, we can see that the energy dissipation for all test cases increases. However, the energy reduction of the offline CAP scheduling over the Reference scheduling also increases compared with previous two experiments. The average energy reduction is now 60%. The differences between offline CAP versus CAP w/o PS increases to 51% in this case. The online CAP has an average of 56% energy reduction than the Reference algorithm. Compared to offline CAP, the online CAP has 12% more energy dissipation. The offline CAP has now 14% less energy than CAP w/o PM and the improvement of online CAP over online CAP w/o PM increases to 8%.

V. CONCLUSIONS

Online and offline algorithms are proposed that perform simultaneous task mapping and ordering followed by task stretching of a conditional task graph (CTG). The algorithms minimize the mathematical expectation of energy dissipation of non-deterministic applications with random branch selection by utilizing the task execution profile. Both communication and computing energy are reduced in the scheduled result. The experimental results show that, comparing with the previous scheduling algorithm, our offline algorithms give more than 39% energy reduction in average. The online algorithm gives more than 28% energy reduction in average with a speed up of 120,000X over offline algorithm. Our future efforts target a development of an adaptive version of CAP online algorithm that can fit the changing runtime system conditions and utilize the low complexity of CAP. Considering contentions inside communication network would also be an interesting analysis.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.*

[10] 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.

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

[12] Paul Pop, "Scheduling and communication synthesis for distributed real-time systems", *Ph.D. thesis, Linkopings University,2000.*