Profile-Based Low Power Scheduling for Conditional Task Graph: A Communication Aware Approach

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

***Abstract* — This work focuses on power optimization of real-time applications with conditional execution running on a dynamic voltage scaling (DVS) enabled multiprocessor system. A novel algorithm is proposed that performs simultaneous task mapping and ordering followed by task stretching of a conditional task graph (CTG). The algorithm minimizes the mathematical expectation of energy dissipation of non-deterministic applications with random branch selection by utilizing the task execution profile. Compared with existing scheduling algorithm, the experimental results show that our algorithm has 32% energy reduction in average.**

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 has 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.

In this work, we consider off-line task scheduling for CTG on a multiprocessor system with non-negligible communication cost. The system has a set of heterogeneous PEs, such as DSPs, FPGAs or ASICs, that are connected by an interconnect network. Each PE is DVS enabled. The application is distributed to different PEs. A set of algorithms are proposed that provide a complete solution for task mapping, task ordering and task stretching. 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. Because the execution flow is unknown at the time when the scheduling is performed, we consider the application as a random procedure with probabilistic branch selection. The algorithm utilizes the task execution-profile to minimize the expected energy dissipation and also satisfy the performance constraint.

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 for the scheduling of CTGs. However, it assumes that all the conditional branches are equally important. 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. The profile information of the CTG is considered during task stretching. But this algorithm takes task mapping as a fixed input so that the communication overhead cannot be considered.

In this work, we propose a communication aware profile-based scheduling (CAP) algorithm. The characteristics of algorithm are described as follows.

1. The proposed algorithm provides a complete solution which includes task mapping, ordering and task stretching. The scheduling procedure can be divided into two steps. The first step performs simultaneous task mapping and ordering and the second step performs task stretching.

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 information from task

execution profile. Its objective is to minimize the mathematical expectation of energy dissipation.

3. The algorithm for simultaneous task mapping and ordering has very low complexity. The algorithm is based on the *dynamic level scheduling* (DLS) algorithm given in reference [8]. It has the potential to be streamlined and be used for on-line scheduling.

4. The task stretching problem is solved using integer linear programming. Only computational tasks are stretched.

The experimental results show that, comparing with the scheduling algorithms presented in [7], our algorithm provides an average of 32% energy reduction.

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 (τ_i, τ_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 $C(e)$ and a probability $Prob(e)$. A node can be either *and-node* or *or-node*. An and-node is activated when all its predecessors are completed. On the other hand, an or-node is activated as soon as one of its predecessors is completed.

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. A task τ is associated with a minterm m if m is one of the minterms of $X(\tau)$. 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 $\Gamma(\tau)$.

The volume of data that passes from one task to another is also captured by the CTG. Each edge (τ_i, τ_j) in the CTG associates with a value Comm(τ_i, τ_j) which gives the communication volume in the unit of Kbytes.

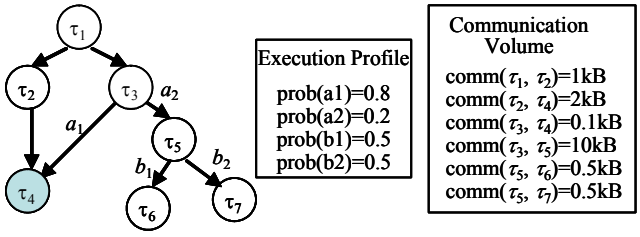

Figure 1 An example of CTG

Figure 1 shows an example of a CTG. All nodes except node τ_4 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 3 minterms in the CTG. They are $\{a_1, a_2b_1, a_2b_2\}$. We have $\Gamma(\tau_4) = \{a_1, a_2b_1, a_2b_2\}$ and $\Gamma(\tau_6) = \{a_2b_1\}$. The execution profile and communication volume are given beside the CTG.

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 latency $D(\tau_i, p_j)$, $\forall \tau_i \in V$ 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 $B(p_i, p_j)$, $\forall p_i, p_j \in P$. These values specify the bandwidth of the communication link between p_i and p_j.

III. Proposed Scheduling Algorithm

This section provides an insight into our communication aware profile-based (CAP) scheduling algorithm. The algorithm is based on Dynamic Level based Scheduling (DLS) proposed by [8]. The DLS algorithm is a list scheduling algorithm. The candidate 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 $DL(\tau_i, p_j)$ between the task τ_i and a processing element p_j is calculated using the following formula:

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

where $SL(\tau_i)$ is the static level of task τ_i, it 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 task and PE pair which gives the maximum dynamic level will be selected and the mapping is performed accordingly. After that, the candidate list is updated and the dynamic level of each task in the candidate list is re-calculated.

We used a dynamic program to calculate the static level of each task processor pair. 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 processor. Let $S(\tau_i)$ be the set of successor nodes of τ_i, i.e. for any node $\tau_j \in S(\tau_i)$, there exist an edge (τ_i, τ_j). Let c_{ij} denote the condition of edge (τ_i, τ_j) if it is a conditional edge. Then the SL in our algorithm is defined as,

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

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

Equation (2) formulates the SL for non-branching node while equation (3) provides the SL for branching node. Incorporating the branch probabilities in the calculation of SL for branching nodes reflects our consideration of the probabilistic behavior of entire graph. The algorithm starts calculating SL of end nodes first and traversing whole graph upwards by updating SL of each node. The probability of condition selection changes the task static level. Our

algorithm considers various condition probabilities to provide more efficient task schedule with lower expected energy.

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. Table 1 shows the *SL* calculation for example CTG given in Figure 1. Here a system consisting of 3 processors is assumed and *WCET for each task indicates the average of all WCETs on each feasible processor that can run it.

Table 1 Static level calculation for CTG in Figure 1

Task	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
**WCET*	7	8	6	10	5	12	8
SL	25	18	17	10	15	12	8

Many factors affect task scheduling on heterogeneous processor environment. Apart from varying execution times, there is a communication overhead between a node and its successor node if they are mapped on different processor. We assume zero communication between nodes on same processor. It is always not advantageous to map the task on a processor that could run it fastest. Also there can be a processor that does not have resources required to run the task. The algorithm proposed here intelligently makes mapping decisions considering both computation and communication. The goal is to achieve minimum length for whole schedule to provide maximum slack for task stretching.

The task can not start execution until the processor on which it is mapped is busy executing some other task (computation). Each task is activated to be ready to execute once all its predecessor tasks are finished and the data passed from these tasks has been received (communication). Both these constraints are modeled by Dynamic Level (*DL*) defined for each task-processor pair shown below.

$$DL(\tau_i, p_j) = SL(\tau_i) - \max\left[DA(\tau_i, p_j), TF(pj) \right] + \delta(\tau_i, p_j) \quad (4)$$

The term $\delta(\tau_i, p_j)$ models the difference between *WCET(τi) and WCET(τ_i, p_j) accounting for heterogeneous processor architecture. This term is necessary to offset the average WCET considered in *SL* calculation.

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. Each node could be executed on a set of processors. For each such possible node-processor pairs, the algorithm then finds the best pair (τ_i, p_j) that has the highest amount of *DL* given by (4). Task τ_i is then scheduled on p_j using FindAvailableTime() function from [7].

If two tasks in a graph belong to different conditional execution paths, they will be never executed at the same time. If we consider a periodic graph, these tasks will never execute together in a given period. Such tasks are called mutually exclusive tasks. They can be scheduled at the same time on same processor since at run time only one of them will execute, thus making the schedule more efficient. 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. All other combinations are not mutually exclusive. For example, even if condition a_2 is evaluated to be true, τ_4 and τ_5 can start executing at the same time, and thus are not mutually exclusive.

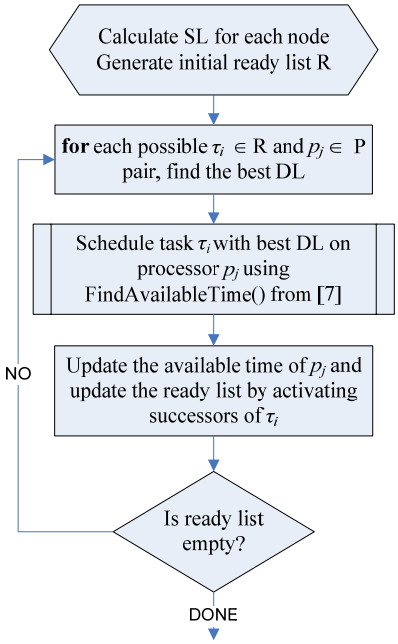

Figure 2 Task ordering algorithm flow

The FindAvailableTime() routine finds the best schedule for the node-processor pair selected having best *DL*. The variable TF(p_j) in (3) relies on the latest task τ_k scheduled on p_j to finish and ignores the mutual exclusiveness among different tasks scheduled before τ_k. The FindAvailableTime() further searches the scheduled task queue of processor to find the timeslot which could be of best fit to current task. It can end up scheduling task in a time slot overlapping with some other task in case they are mutually exclusive. Thus it builds on DLS algorithm to further improve the schedule.

Once the ready node list is empty, the algorithm reports the schedule to task stretching routine. Our task stretching routine consists of numerical method based model discussed in [7] which work towards the objective of minimizing the total energy consumption by stretching each task and maintaining performance requirements in terms of task deadline. We also added communication model to account for communication energy. We assume that communication tasks can not be stretched and treat them as fix overhead. To report the total energy, the model takes different graphs as an input corresponding to different minterms described in section II. The task ordering algorithm generates these different graphs pertaining to minterms by removing invalid nodes and associated edges and inserting additional edges to maintain valid execution and performance requirements. For example, if minterm a_2b_1 in CTG of Figure 1 is true, τ_4 can not start until τ_3 is finished and thus the graph for this minterm preserves this edge. Same way node τ_7 and edge (τ_5 , τ_7) are removed from this graph. The stretching algorithm processes all minterms simultaneously and reports the final energy based on execution probability of each minterm.

IV. EXPERIMENTAL RESULTS

Simulations have been carried out to evaluate the efficiency of the proposed algorithm. 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 [9]. The MPSoC architecture consists of either 3 or 4 PEs. In the rest of the paper, we use a triplet (*a*/*b*/*c*) to characterize a test case where *a* represents the number 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.

Besides the CAP algorithm that is presented in section III, three other scheduling algorithms are evaluated in the experiments. The first and second algorithms are similar as the CAP algorithm. However, one of them does not consider the profile information in task mapping and the other does not consider the profile information in task stretching. They are denoted as CAP w/o PM (CAP without profile-based mapping) and CAP w/o PS (CAP without profile-based stretching). We also implemented the ordering and stretching algorithm presented in [7], which is denoted as Reference scheduling in the rest of the paper. The Reference scheduling does not consider profile information in task ordering and it assumes fixed task mapping. For all the experiments, we consider the execution of the CTG as a random process and we report the mathematical expectation of energy dissipation.

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 5 test cases under different scheduling algorithms. In average, the CAP has 5% energy reduction over the reference scheduling.

Table 2 Energy dissipation for test cases with fixed mapping and zero communication cost

ID	*a/b/c*	Reference	CAP w/o PM	CAP w/o PS	CAP
1	25/3/3	507	483	647	483
2	16/3/1	756	774	1490	774
3	15/4/2	579	579	917	579
4	15/4/1	958	809	954	809
5	25/4/3	678	635	826	635

Table 3 Energy dissipation for test cases with flexible mapping and zero communication cost

ID	*a/b/c*	Reference	CAP w/o PM	CAP w/o PS	CAP
1	25/3/3	507	436	526	374
2	16/3/1	756	469	983	538
3	15/4/2	579	354	463	317
4	15/4/1	958	484	485	413
5	25/4/3	678	201	187	147

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. As we can see, with flexible task mapping, the CAP based algorithms give more energy reduction than the Reference algorithm. The average energy reduction is 47%. We can also see that the CAP algorithm outperforms both CAP w/o PM and CAP w/o PS. This shows that it is necessary to consider the profile information in both task mapping and task stretching steps.

The third experiment focuses on demonstrating the communication aware capability of our algorithm. In this experiment, we randomly generate the communication delay and energy between PEs, and run the scheduling algorithm. Since the communication cost is non-zero, we can see that the energy dissipation for all test cases increases. The average energy reduction is now 46%.

Table 4 Energy dissipation for test cases with flexible mapping and non-zero communication cost

ID	*a/b/c*	Reference	CAP w/o PM	CAP w/o PS	CAP
1	25/3/3	521	603	639	465
2	16/3/1	843	487	1004	552
3	15/4/2	663	378	472	320
4	15/4/1	1085	541	499	427
5	25/4/3	765	251	229	177

V. CONCLUSION

We propose a novel algorithm that performs simultaneous task mapping and ordering followed by task stretching of a conditional task graph (CTG). The algorithm minimizes the mathematical expectation of energy dissipation of non-deterministic applications with random branch selection by utilizing the task execution profile. Both communication and computation energy are reduced in the scheduled result. The experimental results show that, comparing with the previous scheduling algorithm, our algorithm gives more than 32% energy reduction in average.

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] 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.

[9] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task graphs for free," *Proc. of Int. Workshop Hardware/Software Codesign*, pp. 97-101, Mar. 1998.