Journal of
Low Power Electronics
Vol. 8, 1–16, 2012

Low Power Task Scheduling and Mapping for Applications with Conditional Branches on Heterogeneous Multi-Processor System

Yang Ge[1,*], Yukan Zhang[1], Parth Malani[2], Qing Wu[3], and Qinru Qiu[1]

[1] *Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, 13210, USA*
[2] *Intel Corporation, Santa Clara, CA, 95054, USA*
[3] *Air Force Research Laboratory, Rome, NY, 13441, USA*

Many real time applications demonstrate uncertain workload that varies during runtime. One of the major influences of the workload fluctuation is the selection of conditional branches which activate or deactivate a set of instructions belonging to a task. In this work, we capture the workload dynamics using Conditional Task Graph (CTG) and propose a set of algorithms for the optimization of task mapping, task ordering and dynamic voltage/frequency scaling of CTGs running on a heterogeneous multi-core system. Our mapping algorithm balances the latency and the energy dissipation among heterogeneous cores in order to maximize the slack time without significant energy increase. Our scheduling algorithm considers the statistical information of the workload to minimize the mean power consumption of the application while maintaining a hard deadline constraint. The proposed DVFS algorithm has pseudo linear complexity and achieves comparable energy reduction as the solutions found by mathematical programming. Due to its capability of slack reclaiming, our DVFS technique is less sensitive to the slight change in hardware or workload and works more robustly than the conventional DVFS techniques.

Keywords: Dynamic Voltage and Frequency Scaling, Multiprocessor System-on-Chip, Processor Scheduling, Workload Variation.

1. INTRODUCTION

The Multiprocessor System-on-Chip (MPSoC) has become a major system design platform for general purpose and real-time applications, due to its low design cost and high performance. Power consumption, which has already been a roadblock for current single core systems, will continue to be a major design concern for MPSoC. There have been continuous design efforts targeting at energy reduction of MPSoC. Dynamic Voltage and Frequency Scaling (DVFS), which allows the processor to dynamically alter its speed and voltage at run time, is one of the most popular energy reduction techniques at system level.

Many techniques have been proposed that perform power aware task mapping, ordering and DVFS for applications with fixed workload.[1~3] However, they may not achieve their best performance with real life applications that demonstrate workload variations.

This paper considers task scheduling and voltage selection of applications with variable workload on a heterogeneous multiprocessor system-on-chip. We focus on the set of applications which can be decomposed into repeated 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. Examples of such task level branching include branches that 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. We model an application with dynamic branch selections using a conditional task graph (CTG).[4~8]

Although instruction level branch prediction is a common practice in high performance processors, the prediction is not perfect. The task graphs that we are working with are high level abstraction of large applications. The selection of conditional branches depends mostly on the input data, which are random variables. This makes accurate branch prediction even more difficult. Techniques that

*Author to whom correspondence should be addressed.
Email: yage@syr.edu

dynamically assign confidence levels[9] or probabilities[10] to the conditional branches have been proposed by previous research works. For many applications, the branch probability exhibits strong temporal correlations and can be predicted based on history. Such information will be used by the proposed scheduling algorithm for expected energy minimization.

We study the power aware task mapping and scheduling techniques for the CTGs with a particular interest in their performance on the heterogeneous multi-core system. It is important to investigate the low power techniques for the heterogeneous multi-core system, not only because it is the architecture of many embedded computing systems such as the Pasta sensor node,[15] the LEAP system,[16] and the mPlatform,[17] but also because, due to the within-die variation and transistor wear-out, a homogenous multi-core system will exhibit heterogeneity over time.[18]

The proposed framework optimizes the task mapping, task ordering and dynamic voltage/frequency scaling of CTGs running on a heterogeneous multi-core system. Due to the probabilistic branching, the execution of CTG is a random process. Our goal is to minimize the mean power consumption instead of the worst case power consumption of the application while meeting the hard real-time constraint. Our mapping algorithm balances the latency and the energy dissipation among heterogeneous cores in order to maximize the slack time without significant energy increase. Our scheduling algorithm minimizes the mean power consumption of the application while maintaining a hard deadline constraint. Our DVFS algorithm dynamically distributes slacks during runtime, and achieves comparable energy reduction as the solutions found by non-linear mathematical programming. Due to its capability of slack reclaiming, our DVFS technique is less sensitive to the slight change in hardware or task execution and works more robustly than the conventional DVFS techniques. Please note that to minimize the power consumption of an application with hard real-time constraint is equivalent to minimize the energy consumption, because the energy is the product of power and execution time.

To the best of our knowledge, this is the first paper that addresses low power scheduling of conditional task graphs on heterogeneous multi-core platform. Compared to the existing works, the uniqueness of our approach can be summarized as the following.

The proposed framework optimizes task mapping and scheduling simultaneously. We modify the *Dynamic Level based Scheduling* (*DLS*) algorithm[11] to find mapping and scheduling of CTGs on a heterogeneous platform. The complexity of task mapping and ordering is $O(MN)$ where M is the number of *processing elements* (*PE*s) in the system and N is the number of tasks in the task graph.

We consider the application with conditional execution as a random procedure in which the branch selection probabilities can be profiled (or measured). Our algorithm utilizes such statistical information in each optimization step to reduce the mean power consumption.

The proposed slack reclaiming based DVFS algorithm has pseudo-linear complexity with the respect to the number of tasks in the CTG. It also performs more robustly than the DVFS solution found by mathematical programming.

More generalized hardware and software models are targeted in this work. The hardware platform consists of heterogeneous processing elements with different power-performance tradeoff characteristics. The only assumption that is required by our DVFS algorithm is a convex relation between the power and performance for each PE, which is usually true for many devices. The software application consists of a set of tasks with data and control dependencies and hard deadline constraint.

The rest of the paper is organized as follows: Section 2 introduces the existing works in related area. Section 3 introduces the application and hardware architecture models; Sections 4 and 5 present in detail the underlying scheduling, mapping and DVFS algorithms. Section 6 gives the experimental results. Finally, Section 7 concludes the paper.

2. RELATED WORKS

Task mapping, ordering, and DVFS for CTGs have been studied in many literatures. In Ref. [4], Dolif et al. propose an exact analytic formulation of the stochastic objective function for conditional task mapping and scheduling based on task graph analysis. Their approach considers stochastic branching probabilities and improves the overall performance by optimizing the bus activity. In Ref. [5], Eles et al. present an approach to minimize the worst case delay, which considers the communication workload and branch control. A recursive algorithm is proposed to generate the schedule table, which stores the start time of each task at different execution scenarios. In Ref. [6], Xie et al. propose a task mapping and scheduling algorithm which is sensitive to mutually exclusive tasks in the CTG. Their algorithm exploits the processor sharing for mutually exclusive tasks to minimize the worst case delay of the CTG. All these works do not minimize energy dissipations.

Wu et al.[7] utilize a scheduling table for runtime DVFS. The slack time of each task can be derived from the table. A genetic algorithm based task mapping algorithm is proposed for further energy savings. An implicit assumption of this technique is that all the conditional branches will be selected with equal probability, which might not be realistic for a real system. Furthermore, the GA based task mapping algorithm has very high complexity because the inner loop of this algorithm needs to perform the task ordering and stretching for the entire CTG.

Shin et al.[8] proposed an algorithm for task scheduling and DVFS of CTG which considers the run-time behavior. The algorithm is based on a modified list scheduling and

non-linear programming (NLP) based voltage/frequency optimization. Because the algorithm schedules a task with different start time and speed under different branch selection scenarios, it is referred as condition aware scheduling. However, their task mapping is assumed to be fixed.

In Refs. [13 and 14] we proposed a Communication Aware Probability-based (CAP) scheduling algorithms for CTGs on a DVFS enabled multiprocessor platform. However, the algorithm assumes that all the PEs have the same energy-performance characteristics. Furthermore, its DVFS algorithm relies on evaluating all the paths, which has exponential complexity. In this paper, we completely revised the CAP algorithm and consider a heterogeneous system where each PE has unique power-performance characteristics. The complexity of the DVFS algorithm is also significantly reduced.

Power aware task mapping, scheduling and DVFS techniques for heterogeneous multiprocessor system has been studied in Refs. [19∼25]. In Ref. [19], a two-phase algorithm is proposed for distributed real-time embedded system. A static scheduling algorithm first generates the schedule for all tasks in the CTG, and then a dynamic scheduling algorithm determines the speed ratio for mutual exclusive tasks during the runtime. In Ref. [20], the authors focus on the critical paths and distribute the slack time over the tasks on the critical path to achieve energy savings. Both works assume that only processors have DVFS capability and their power-performance characteristics are identical. References [21 and 22] study the energy-efficient software partition in heterogeneous multiprocessor system. In Ref. [21], Goraczko et al. formulate the software partitioning problem as an integer linear programming (ILP) constrained by the deadline. The solution of the ILP gives the number of processors needed by the system and task mapping information. In Ref. [22], Yang et al. present an approximate algorithm for task partition to find near-optimal task mapping and scheduling with minimum energy consumption. In both these works, tasks are assumed to be independent to each other. A leakage aware DVFS algorithm is proposed in Ref. [23]. Unlike our work, it only considers heterogeneous system with only 2 types of processing elements (PE), one with high performance and high power consumption while the other with low performance and low power consumption. The objective is to maximize the number of tasks mapped on the PEs with low performance while satisfy the deadline constraint. Deadline Monotonic (DM) rule is used in the task scheduling while Random Algorithm (RA), First-Fit Decreasing Density (FFDD), Genetic Algorithm (GA) and Simulated Annealing (SA) are used for task mapping. In Ref. [24], Azevedo et al. propose to do intra task DVS based on the program checkpoint, while in Ref. [25], Yang et al. rely the Pareto curve profiling for a task graph to do online task mapping and scheduling. Both of these works do not consider energy and performance heterogeneity of the multiprocessor platform.

3. APPLICATION AND ARCHITECTURE MODELING

The CTG that we are using is similar to the one specified in Ref. [8]. A CTG is a directed acyclic graph $\langle V, E \rangle$. 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 task τ_j can start. A conditional edge e is associated with a condition $C(e)$. The tail node of a conditional edge is called *branch fork node*. We use $prob(e)$ to denote the probability that the condition $C(e)$ is true; and we use $succ(\tau)$ and $pred(\tau)$ to denote the set of successors and predecessors of a task τ. We assume that all tasks have the same deadline.

A node is activated when all its predecessors are completed and the conditions of the corresponding edges are satisfied. The condition that the task τ_i is activated is called the *activation condition* and denoted as $X(\tau_i)$. It can be written as $\vee_{\tau_k \in pred(\tau_i)}(C(\tau_k, \tau_i) \wedge X(\tau_k))$, where "$\wedge$" and "$\vee$" represent logic operations "AND" and "OR".

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 $X(\tau)$ is true when m is evaluated to be 1. The set of minterms with which τ is associated is referred as *activation set* of τ and 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., $\Gamma(\tau_i) \cap \Gamma(\tau_j) = \phi$. The set of all unique $\Gamma(\tau)$, $\forall \tau \in V$, is called *activation space* and denoted as T.

The amount 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 $Comm(\tau_i, \tau_j)$ which gives the communication volume. 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. The edges coming out from τ_3 and τ_5 are conditional edges therefore nodes τ_3 and τ_5 are branch fork nodes. The symbol marked beside a conditional edge denotes 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 $M = \{a_1b_1, a_1b_2, a_2b_1, a_2b_2\}$. We have $\Gamma(\tau_1) = \Gamma(\tau_2) = \Gamma(\tau_3) = \{a_1, b_1, a_1, b_2, a_2, b_1, a_2, b_2\}$, $\Gamma(\tau_4) = \Gamma(\tau_8) = \{a_1, b_1, a_1, b_2\}$, $\Gamma(\tau_5) = \{a_2, b_1, a_2, b_2\}$, $\Gamma(\tau_6) = \{a_2, b_1\}$, $\Gamma(\tau_7) = \{a_2, b_2\}$. The execution profile and communication volume are given beside the CTG.

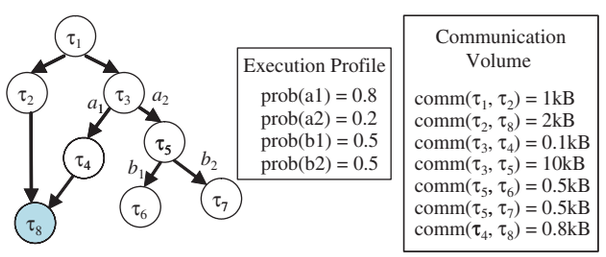


Fig. 1. An example of a CTG.

The activation space is:

$$T = \big\{\{a_1b_1, a_1b_2, a_2b_1, a_2b_2\}, \{a_1b_1, a_1b_2\}, \{a_2b_1, a_2b_2\}, \{a_2b_1\}, \{a_2b_2\}\big\} \quad \square$$

The hardware platform consists of a set of N PEs $PE = \{p_1, p_2, \ldots, p_N$. We use $WCET(\tau_i, p_j)$, $\tau_i \in V$, $p_j \in PE$, to denote the *worst case execution time* of task τ_i on processor p_j. $WCET(\tau_i, p_j)$ is expressed in the number of clock cycles. We use $BW(p_ip_j)$ to denote the transmission bandwidth of the communication links between p_i and p_j.

A PE p_i in the hardware platform is DVFS enabled. Without loss of generality, we assume that its *dynamic cycle energy* $E_{i,\,dyn}$, which is the accumulated dynamic power consumption during a clock cycle, is a convex and monotonically decreasing function of clock period T, and $-\infty < dE_{i,\,dyn}(T)/dT < 0$, $T_{\min} \le T \le T_{\max}$. Its *overall cycle energy*, $E_{i,\,\text{total}}$ is calculated as $E_{i,\,\text{total}}(T) = E_{i,\,\text{leak}}(T) + E_{i,\,dyn}(T)$. From Ref. [30], leakage current I_{leak} has an exponential relation (convex) against V_{dd}, while V_{dd} is inversely proportional (convex) to T, i.e., $V_{dd} \propto f = 1/T$. Therefore, I_{leak} has a convex relation against T. Furthermore, because $E_{i,\,\text{leak}} = I_{\text{leak}}V_{dd}T \propto I_{\text{leak}}$, $E_{i,\,\text{total}}(T)$ is still a convex function of T

THEOREM 1. *There is an optimal clock period T^* that minimizes the $E_{i,\,\text{total}}(T)$ and $E_{i,\,\text{total}}(T)$ is a convex and monotonically decreasing function for $T_{\min} \le T \le T^*$.*

PROOF. Because $E_{i,\,\text{total}}(T)$ is a convex function of T, due to the property of convex function, there is a $T^* \in [T_{\min}, T_{\max}]$ such that T^* minimizes $E_{\text{total}}(T)$ and $E_{\text{total}}(T)$ is a decreasing function for $T_{\min} \le T \le T^*$. $\square$

We refer T^* and $f^*_{clk} = 1/T^*$ as *break even clock period* and *break even clock frequency*. Our DVFS algorithm explores the convex relation between E_{total} and the clock period. We limit our DVFS algorithms to choose clock frequencies in the range $[f^*_{clk}, f_{\max}]$, where f^*_{clk} and $f_{\max}$ are break even and maximum clock frequency of the system. In this range the total cycle energy will monotonically decrease as the clock period increases. Note that the proof of Theorem 1 assumes that the $E_{dyn}(T)$ and $E_{\text{total}}(T)$ are continuous functions. In Section 5, we will discuss how to generalize the proposed DVFS algorithm to discrete voltage and frequency scaling.

4. TASK MAPPING AND SCHEDULING

Our task mapping and scheduling is based on a modified *Dynamic Level based Scheduling* (*DLS*)[11] algorithm that finds task mapping and ordering simultaneously. Our goal is to implement an on-line scheduling, mapping DVFS algorithm that adaptively change with the branch probability. That's why we choose the DL-based approach. The DLS algorithm is a list scheduling algorithm. It maintains a *ready list* that stores tasks whose predecessors have been scheduled and mapped. For each task τ_i in the ready list, the dynamic level $DL(\tau_i,\ p_j)$ is calculated using the following equation:

$$DL(\tau_i, p_j) = SL(\tau_i) - AT(\tau_i, p_j) \tag{1}$$

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 (i.e., the longest remaining execution time), $AT(\tau_i, p_j)$ is the earliest time that task τ_i can start at processor p_j, it is calculated as $AT(\tau_i, p_j) = \max\lfloor DA(\tau_i, p_j), TF(p_j)\rfloor$, where $DA(\tau_i,\ p_j)$ 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 $(\tau_i,\ p_j)$ 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 $\max\lfloor DA(\tau_i, p_j), TF(p_j)\rfloor$. After that, the ready list is updated and the dynamic level of each task in the ready list is re-calculated. The general flow of the DLS algorithm is given in Algorithm 0. Because we need to calculate the DL for each task processor pair in the system, the complexity of the DLS algorithm is $O(|V| * N)$, where $|V|$ is the number of tasks and N is the number of PEs in the system.

The original DLS algorithm does not handle mutual exclusive tasks. Its goal is to minimize the makespan of the application. Although minimal makespan scheduling usually enables more aggressive DVFS and hence lower power consumption in a homogenous multi-core system, this is not always true in a heterogeneous system. In this work we adopt the general flow of the DLS algorithm but modify the way that the dynamic level (*DL*) is calculated to consider the mutual exclusiveness among conditional tasks, the probability of branch selection and the energy performance heterogeneity among processors.

Algorithm 0. General flow of DLS algorithm
Alg. 0: DSL

```
1.  Calculate SL for each task
2.  ready_list = ∅;
3.  While (not all tasks have been scheduled) {
4.      For each pair (τ, p), τ is a ready task and
          p ∈ PE, calculate DL(τ, p);
5.      Select the pair (τ_i, p_j) with the maximum DL;
6.      Map the task τ_i to processor p_j and set its start
          time to AT(τ_i, p_j);
7.      ready_list = ready_list ∪ τ_i
8.  }
```

4.1. Minimum Average Makespan CTG Scheduling

Our first step is to modify the *DL* function to explore the mutual exclusiveness among conditional tasks and consider the branch selection probability. The goal of such

modification is to minimize the average makespan instead of the worst case makespan of the application. In the modified algorithm, the static level (*SL*) of a task τ is used to represent the average remaining execution time when τ has just started.

The static level of a non-branching node is the maximum static level of its successors plus the average *WCET* (denoted as $WCET_{\text{avg}}$) of itself. Let $Succ(\tau_i)$ be the set of successors of τ_i, Eq. (2) calculates the *SL* of a non-branching node.

$$SL(\tau_i) = WCET_{\text{avg}}(\tau_i) + \max_{\tau_j} SL(\tau_j), \quad \tau_j \in Succ(\tau_i) \quad (2)$$

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

$$SL(\tau_i) = WCET_{\text{avg}}(\tau_i) + \sum_j prob(c_{ij}) SL(\tau_j), \quad \tau_j \in Succ(\tau_i) \quad (3)$$

where $prob(c_{ij})$ is the probability that condition c_{ij} is true.

For a task τ_i without successors, its static level equals to its average worst case execution time: $SL(\tau_i) = WCET_{\text{avg}}(\tau_i)$.

The dynamic level of task-processor pair (τ_i, p_j) is calculated as the following equation.

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

The term $\delta(\tau_i, p_j)$ is the difference between $WCET_{\text{avg}}(\tau_i)$ and $WCET(\tau_i, p_j)$ which accounts for the performance heterogeneity. Adding this offset ensures correct evaluation of a task's *DL* for different processors since *SL* is computed using the 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:

1. At time $AT(\tau_i, p_i)$ all data required by τ_i is available at p_i, i.e., $AT(\tau_i, p_i) \geq DA(\tau_i, p_i)$.
2. If a task τ_j is scheduled during the interval $[AT(\tau_i, p_i), AT(\tau_i, p_i) + WCET(\tau_i, p_i)]$, 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.

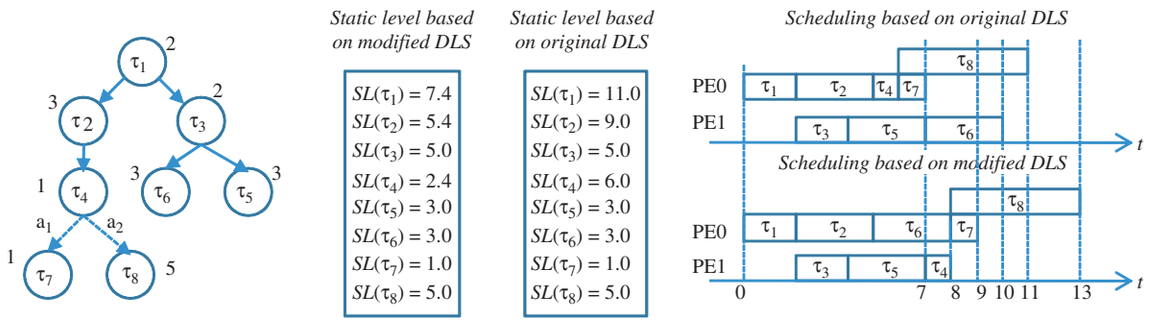
The data available time of a task τ_i on processor p_j is calculated as the latest time when its predecessor completed execution plus the time to transfer the data, i.e., $DA(\tau_i, p_j) = \max_{\tau'}[ET(\tau') + comm(\tau', \tau_i)/BW(p', p_j)]$, $\forall \tau' \in pred(\tau_i)$, where p' is the processor where τ' is mapped to and $ET(\tau')$ is the end time of task τ'. Here we assume point to point communication between two processors. Please note that we can extend the framework to systems with shared communication links by considering all data communications as separate tasks and all communication links as separate resources and schedule communication tasks in the similar way as the computation tasks. Please note that our DLS algorithm could also handle the overhead of reconfiguring voltage and frequency by adding the overhead to the variable $AT(\tau_i, p_j)$.

EXAMPLE 2. Consider the CTG given in Figure 2(a). The *WCET* of each task is marked beside the node. The branches a_1 and a_2 are taken with probability 0.9 and 0.1. The system has 2 PEs. In order to simplify the discussion, we assume that both PEs are identical and the interprocessor communication is negligible. Figure 2(b) gives the static level of each task calculated using our modified DLS and the original DLS. As we can see, $SL(\tau_4)$ is greater than $SL(\tau_5)$ and $SL(\tau_6)$ in the original DLS while their relations are reversed in the modified DLS. This is because the original DLS considers the worst case performance, which happens along the path τ_4–τ_5, disregarding the fact that task τ_5 is rarely activated. Figure 2(c) gives the schedule found by these two DLS algorithms. In both cases, τ_7 and τ_8 are scheduled to start on the same processor at the same time because these two tasks are mutually exclusive. Using the original DLS, τ_4 is mapped and scheduled before τ_5 and τ_6 because the $SL(\tau_4)$ is greater than $SL(\tau_5)$ and $SL(\tau_6)$. This order is reversed in the modified DLS. The worst case makespan of application scheduled using the original and the modified DLS are 11 and 13 respectively. However the average makespan is $11 \times 0.1 + 10 \times 0.9 = 10.1$ for the original DLS and $13 \times 0.1 + 9 \times 0.9 = 9.4$ for the modified DLS. □

4.2. Balancing Energy and Performance in a Heterogeneous Platform

In a homogenous system, because all PEs are identical in energy performance characteristics, the total energy dissipation is determined solely by the slowdown ratio. Minimizing the average makespan will generally produce more slacks for each task and enable a lower execution speed for more energy reduction. Such monotonic relation does not exist in heterogeneous system, where the application energy dissipation does not only depend on the slowdown ratio, but also on which core the application is mapped to.

There is a fundamental tradeoff between energy and performance in a heterogeneous system. Mapping a task τ to a faster PE could result better performance and shorter execution time but will incur higher energy consumption. To compensate the extra energy incurred by task τ, later tasks have to choose slower PEs to run and result longer latency. The performance gain by running task τ at faster speed will diminish. From this perspective, higher energy consumption is equivalent to longer execution latency. Our modified DLS algorithm is already capable of handling performance heterogeneity among PEs (by introducing $\delta(\tau_i, p_j)$ in Eq. (4)), to further consider energy heterogeneity in the algorithm, we propose a heuristic method to convert energy heterogeneity into performance heterogeneity, and handle them in a unified way.



(a) CTG (b) Static level calculation (c) GANTT chart of scheduled results

Fig. 2. Compare modified DLS with original DLS.

In the rest of this sub-section, we use $WCET_{\text{avg}}(\tau)$ and $WCET(\tau, p)$ to denote the average WCET of task τ and the WCET of task τ on processor p. $E_{\text{total}}(p, T)$ and $E_{\text{avg}}(T)$ to denote the overall cycle energy of processor p and the average cycle energy of all PEs when their clock period is T. We use $prob(\tau)$ to denote the probability that task τ is activated. Finally, we use $\Delta E(\tau, p)$ to denote the difference between the energy dissipation of task τ running on p and the mean energy dissipation of τ running on the heterogeneous system:

$$\Delta E(\tau, p) = [E_{\text{total}}(p, T_{\min}) \cdot WCET(\tau, p) - E_{\text{avg}}(T_{\min}) \cdot WCET_{\text{avg}}(\tau)] * prob(\tau)$$

Here $T_{\min} = 1/f_{\max}$ is the period of the fastest clock. Note that $\Delta E(\tau, p)$ represents the extra energy of running task τ on p, and it could be positive or negative. In order to make up for this extra energy, we must run the entire application at a lower clock frequency and lower supply voltage (or a higher clock frequency and higher supply voltage if $\Delta E(\tau, p)$ is negative.) Assume that setting the clock period to T' can balance the energy. Then we have,

$$\Delta E(\tau, p) = \sum_{\tau' \in V} WCET_{\text{avg}}(\tau') * [E_{\text{avg}}(T_{\min}) - E_{\text{avg}}(T')] \quad (5)$$

Here we use $\sum_{\tau \in V} WCET_{\text{avg}}(\tau) * E_{avg}(T)$ to approximate the total energy of the application when all processors are running at clock frequency $1/T$. This is only a rough estimation of the true energy dissipation of the application because not all tasks have been scheduled and mapped yet. Solving Eq. (5) we get the T':

$$T' = E_{\text{avg}}^{-1}\left[E_{\text{avg}}(T_{\min}) - \frac{\Delta E(\tau, p)}{\sum_{\tau' \in V} WCET_{\text{avg}}(\tau')}\right]$$

To make up for the extra energy incurred by mapping τ to processor p, the total execution time of the application must be extended by $(T' - T_{\min}) \sum_{\tau} WCET_{\text{avg}}(\tau)$. Therefore, we adjust the $WCET$ of τ and calculate its effective $WCET$ as:

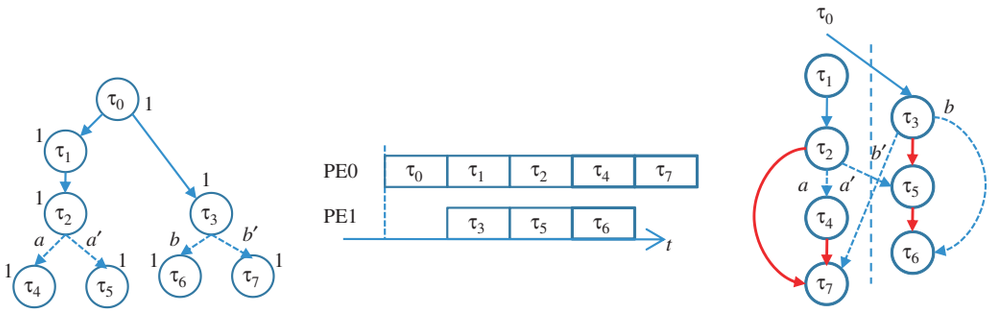
$$WCET_{\text{eff}}(\tau, p) = WCET(\tau, p) + \lambda(T' - T_{\min}) \sum_{\tau} WCET_{\text{avg}}(\tau) \quad (6)$$

The parameter λ is introduced to provide a tradeoff between performance and energy. It will be referred as the *balance factor* in the rest of the paper. When λ is 0, the algorithm is reduced to the minimum average makespan scheduling as we introduced in Section 4.1 regardless of energy heterogeneity among processors. When λ is very large, the algorithm maps a task to the minimum energy processor, without making effort to reserve slacks for the DVFS algorithm which will be applied later.

4.3. Adding Control Edges to the CTG

After task mapping and ordering, the original CTG must be modified to reflect the new precedence constraints. Edges representing control dependencies will be inserted. These control edges will be used to find the execution paths for DVFS control.

Consider the example in Figure 3. The original CTG is given in Figure 3(a). It has 2 sets of conditional branches selected based on conditions a, a', b and b'. Assume two PEs are available in the system. The mapping and ordering of tasks are given in Figure 3(b). Some of the execution orders among tasks have already been guaranteed by the data dependencies specified in the original CTG. For those tasks that have no data dependencies but are mapped to the same PEs, new edges must be inserted to specify their execution order. For example, directed edges between (τ_4, τ_7), (τ_3, τ_5), and (τ_5, τ_6) need to be added. For a conventional task graph, it is not necessary to add the edge (τ_2, τ_7), because their precedence relation is guaranteed by the edges (τ_2, τ_4) and (τ_4, τ_7). However, in a CTG, node τ_4 only exists when condition a is true. When a is false, both edges (τ_2, τ_4) and (τ_4, τ_7) do not exist and the precedence information between τ_2 and τ_7 is lost. Therefore, it is necessary to add an edge from τ_2 to τ_7 to specify their order. Note that it is not necessary to add edges (τ_1, τ_7) or (τ_0, τ_7), because if τ_1 or τ_0 are activated then τ_2 will be activated. Knowing that τ_1 and τ_0 are executed before τ_2 and that τ_2 is executed before τ_7 is enough to infer that τ_1 and τ_0 are executed before τ_7. Figure 3(c) gives the modified CTG. The red edges in the graph are inserted control edges.



(a) Original CTG (b) Task ordering and mapping (c) Modified CTG

Fig. 3. Example of modified CTG with control edges.

The above example shows that a control edge (τ_i, τ_j) must have the following properties:

1. Both τ_i and τ_j are mapped to the same PE.
2. Task τ_i is executed before task τ_j.
3. If there is another control edge (τ_k, τ_j), then τ_k and τ_i do not have identical activation set, i.e., $\Gamma(\tau_k) \neq \Gamma(\tau_i)$.
4. If a task τ_k is mapped to the same PE as τ_i and τ_j, and $\Gamma(\tau_k) = \Gamma(\tau_i)$, then τ_k is executed either after τ_j or before τ_i.

Algorithm 1 gives the algorithm that adds control edges to the CTG. For each task τ and each unique activation set Γ, it searches for the latest task whose activation set is Γ and is executed before τ. Let $\mathscr{C}$ denote the number of unique activation sets in the CTG, the worst case complexity of Algorithm 1 is $O(\mathscr{C} * |V|^2)$. The modified CTG will be used by the DVFS algorithm introduced in the next section.

Algorithm 1. Algorithm for adding control edges to CTG

Alg. 1: Relink CTG

```
1.  For each task τ ∈ V {
2.      For each activation set Γ ∈ T {
3.          τ_link = NULL;
4.          For each task τ' executed before τ on the
              same PE {
5.              If Γ(τ') = Γ and τ' is executed before
                  τ_link then τ_link = τ';
6.          }
7.          If τ_link NULL then add control edge from
              τ_link to τ;
8.      }
9.  }
```

5. DVFS BASED ON SLACK RELCAIMING

Although it gives the optimal solution, solving the above mentioned NLP is time consuming. In the next, we will introduce a heuristic DVFS algorithm based on *slack reclaiming* (*SR*). In the rest of the paper, it is referred as *SR_DVFS*. In order to apply the slack reclaiming algorithm during the runtime, the following information is needed for each task: *overall average remaining execution time* ($ARET_\tau$), the *maximum remaining execution time* ($MxRET_\tau$), the *average remaining execution time along the most critical path on different PEs* ($ARET\_PE_\tau[i]$, $1 \leq i \leq N$), and a *look-up-table of slack distribution rules* (LUT_τ). We divide the total available slack into multiple discrete levels. Given the available slack l, the element $LUT_\tau[l]$ specifies the amount of slacks (out of l) that should be distributed to the PE that τ is mapped to.

Algorithm 2. Algorithm for ARET, MxRET calculation

Alg. 2: Calc. ARET, MxRET

```
1.  Topological sort the modified CTG;
2.  For each task τ starting from the lowest topological
      order {
3.      ARET_τ = 0;
4.      For each minterm m ∈ Γ(τ){
5.          max_aret = 0;
6.          For each successor τ_i of τ with m ∈ Γ(τ_i) {
7.              if(max_aret < ARET_τi + comm(τ, τ_i)/
                  BW(τ, τ_i))
8.                  max_aret = ARET_τi + comm(τ, τ_i)/
                      BW(τ, τ_i);
9.          }
10.         ARET_τ = pr(m)/pr(τ) * max_aret;
11.     }
12.     ARET_τ = ARET_τ + WCET(τ, p_τ);
13.     MxRET_τ = 0;
14.     For each successor τ_i of τ {
15.         if(MxRET_τ < MxRET_τi + comm(τ, τ_i)/
              BW(τ, τ_i) + WCET(τ, p_τ))
16.             MxRET_τ = MxRET_τi + comm(τ, τ_i)/
                  BW(τ, τ_i) + WCET(τ, p_τ);
17.     }
18. }
```

Algorithm 2 gives the algorithm that calculates the values of *ARET* and *MxRET* of each task. The algorithm processes the tasks based on the reverse topological order of the modified CTG. Steps 3 through 10 in the algorithm calculates the ARET of a task τ. For each minterm m, the

ARET of τ is the maximum value of its successor's ARET plus the data communication time. The overall ARET of τ is the sum of the ARETs under each minterm m weighted by the condition probability that minterm m is true given that task τ is activated (i.e., $pr(m)/pr(\tau)$). Finally, the ARET of task τ should include the execution time of τ itself as specified in Step 10. Steps 12 through 17 in the algorithm calculate the *MxRET*. It is equal to the largest *MxRET* of τ's successors plus the data communication time. Let $\mathcal{S}$ denote the maximum number of immediate successors of a task, the worst case complexity of the algorithm is $O(\mathcal{MS}|V|)$.

EXAMPLE 4. We use the CTG given in Figure 4 to demonstrate the *ARET* and *MxRET* calculation. To simply the discussion, we assume that the *WCET* of a task is 1 no matter which PE it is mapped to. Also assume that $pr(a) = 0.1$, $pr(a') = 0.9$, $pr(b) = 0.9$, $pr(b') = 0.1$, and the communication bandwidth is infinite. We start calculating *ARET* from tasks τ_2 and τ_7 because they have the lowest topological order. $ARET_{\tau_2} = WCET(\tau_2) = 1$, $ARET_{\tau_7} = WCET(\tau_7) = 1$.

$$ARET_{\tau_6} = \left(\frac{pr(ab)}{pr(\tau_6)}\right) ARET_{\tau_2} + \left(\frac{pr(a'b)}{pr(\tau_6)}\right) ARET_{\tau_2} + WCET(\tau_6) = 2;$$

$$ARET_{\tau_3} = (pr(ab) + pr(a'b)) ARET_{\tau_6} + (pr(ab') + pr(a'b')) ARET_{\tau_7} + WCET(\tau_3) = 2.9;$$

$$ARET_{\tau} = pr(ab)/pr(a)\max(ARET_{\tau_6} ARET_{\tau_2}) + pr(ab')/pr(a) ARET_{\tau_2} + WCET(\tau_4) = 2.9$$

The ARET for the rest of tasks and their MxRET are given in the following table. □

The values of ARET and MxRET are used to determine the remaining slack that will be distributed to the current task and the following tasks. In a heterogeneous multi-core system, where each PE has different power-performance tradeoff characteristics, the slacks will be distributed to different PEs non-uniformly. Therefore, we need to know not only the overall average remaining execution time, but also the remaining execution time on different PEs.

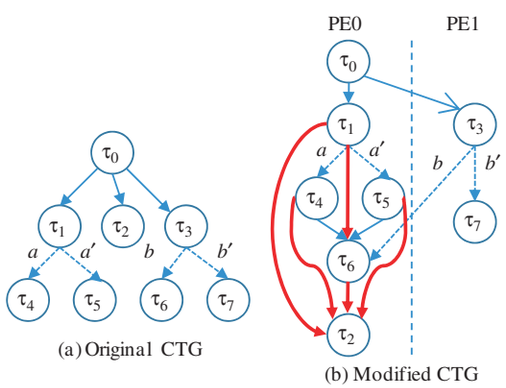


Fig. 4. An example of impact set and speed selection.

Table I. ARET and MxRET of each task in Example 4.

τ	τ_0	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
MxRET	5	4	1	3	3	3	2	1
ARET	4.9	3.9	1	2.9	2.9	2.9	2	1

Based on Algorithm 2, the ARET of a task reflects the longest path in the modified CTG. For example, the ARET of τ_0 in Example 4 reflects the average length of path $\tau_0 \rightarrow \tau_1 \rightarrow \tau_4(\tau_5) \rightarrow \tau_6 \rightarrow \tau_2$, which is the longest path starting from τ_0 in the modified CTG. Considering only the longest path while ignoring the power-performance tradeoff efficiency of each PE will not result the best slack distribution policy. For example, if PE_1 in Example 4 has much higher power consumption than PE_0 and is much more effective in power-performance tradeoff, then we may want to run τ_0 a little faster and reserve more slacks for tasks running on PE_1. Let $f_{EPT}(p)$ denote the *energy-performance tradeoff* (*EPT*) factor of processor p. It is calculated as $f_{EPT}(p) = dE_{\text{total}}(T)/dT|_{T=1/f_{\max}}$. We choose the path that has the highest EPT as the most critical path and collect the average remaining execution time on different PEs along this critical path.

Algorithm 3 gives the algorithm that calculates the variables ARET_PE[i] for each PE i. For each task τ, a variable $AR_{EPT}(\tau)$ is maintained. It records the average total EPT of the remaining tasks along the most critical path (i.e., the path with the highest EPT efficiency). The ARET_PE of a task τ under minterm m is determined by the ARET_PE of its successor that is active under the same minterm and has the largest remaining EPT. The overall ARET_PE of a task is the sum of its ARET_PEs under different minterms weighted by the conditional probability that the minterm m is active given the condition that the task τ is going to be executed. Similar to Algorithm 2, the worst case complexity of Algorithm 3 is also $O(\mathcal{MS}|V|)$.

Algorithm 3. Algorithm for ARET_PE calculation

Alg. 3: Calc. ARET_PE

1. For each task τ starting from the lowest topological order {
2. $ARET\_PE_\tau[i] = 0$, $0 \le i \le N$;
3. For each minterm $m \in \Gamma(\tau)$ {
4. $eng = 0$;
5. Find τ_i with the largest eng_{τ_I}, $\tau_I \in succ(\tau)$ and $m \in \Gamma(\tau_i)$
6. $eng_\tau = pr(m)/pr(\tau) * eng_{\tau_i}$;
7. $ARET\_PE_\tau[i] = ARET\_PE_\tau[i] + pr(m)/pr(\tau)$ $ARET\_PE_{\tau_i}[i]$, $0 \le i \le N$;
8. }
9. $ARET\_PE_\tau[p_\tau] = ARET\_PE_\tau\ [p_\tau] +$ $WCET(\tau, p_\tau)$;
10. $eng_\tau = eng_\tau + WCET(\tau, p_\tau) * E_{\text{total}}(p_\tau)$
11. }

EXAMPLE 5. Consider the CTG in Example 4. Assume that $f_{EPT}(PE_1)$ is 10 and $f_{\text{EPT}}(PE_0)$ is 1, then we have:

$$ARET\_PE_{\tau_3}[PE_0] = ARET\_PE_{\tau_6}[PE_0] * pr(b)/1 = 1.8,$$

$$ARET\_PE_{\tau_3}[PE_1] = ARET\_PE_{\tau_7}[PE_1] * \frac{pr(b')}{1} + WCET(\tau_3, PE_1) = 1.1,$$

$$ARET\_PE_{\tau_0}[PE_0] = ARET\_PE_{\tau_3}[0] + WCET(\tau_0 PE_1) = 2.8,$$

$$ARET\_PE_{\tau_0}[PE_1] = ARET\_PE_{\tau_3}[1] = 1.1.$$

As we can see, the ARET_PE shows the computing time distribution along the path with the highest EPT efficiency. For example, for τ_0, this path is τ_0–τ_3–τ_7(τ_6–τ_2). This information determines how the average available slack will be distributed. □

Let $E_{p_i}(T_{clk} \cdot L)$ denote the energy dissipation over L clock cycles on processor p_i when the clock period is set to T_{clk}. We know that $E_{p_i}(T_{clk} \cdot L) = LE_{\text{total}}(p_{iT_{clk}})$. Replacing $T_{clk} \cdot L$ with a new variable D, $E_{p_i}(D)$ gives the energy dissipation of p_i running for a duration D at clock speed $1/T_{clk}$. D is a linear function of T_{clk} and $E_{p_i}(D)$ is a convex function of D. Assume that, scaling the voltage and frequency extends task execution time from D_1 to D_2. Let $ES_i(D_1, D_2)$ denote the energy saving, i.e., $ES_i(D_1, D_2) = E_{p_i}(D_1) - E_{p_i}(D_2)$.

THEOREM 2. *The energy savings of the optimal continuous DVFS is incremental, i.e.,* $ES_i(D, D+Y) = ES_i(D, D+X) + ES_i(D+X, D+Y)$, $\forall X \in (0, Y)$. *Furthermore, the energy saving* $ES_i(D, D+X), X > 0$ *is a decreasing function of D.*

PROOF. The first part of the theorem can be proved from the definition of the function *ES*( ). An optimal DVFS with continuous voltage and frequency levels will set the T_{clk} to be D/L. The left side of the equation is:

$$ES_i(D, D+Y) = CE_{\text{total}}(D/L) - CE_{\text{total}}((D+Y)/L)$$

The right side of the equation is:

$$ES_i(D, D+X) + ES_i(D+X, D+Y) = CE_{\text{total}}(D/L) - E_{\text{total}}((D+X)/L) + CE_{\text{total}}((D+X)/L) - CE_{\text{total}}((D+Y)/L)$$

The left and right sides equal to each other.

To prove the second part of the theorem, we only need to prove that $ES_i(D, D+X) - ES_i(D+Y, D+Y+X) > 0$, $\forall X, Y > 0$.

The left part of the inequality is:

$$ES_i(D, D+X) - ES_i(D+Y, D+Y+X) = E_{p_i}(D) - E_{p_i}(D+X) - E_{p_i}(D+Y) + E_{p_i}(D+Y+X)$$

Because $E_{p_i}( )$ is a convex function, we know that:

$$E_{p_i}(D) + E_{p_i}(D+Y+X) \geq E_{p_i}(D+X) + E_{p_i}(D+Y).$$

The second part of the theorem is also proved. □

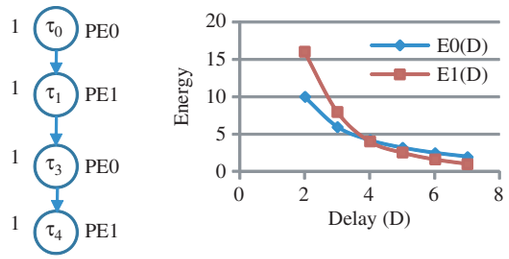
Based on these properties, we design the algorithm that generates the slack distribution table shown in Algorithm 4. The input of the algorithm is the average remaining execution time (i.e., $ARET_\tau$), the remaining execution time on different PEs (i.e., ARET_PE$_\tau[i]$, $1 \leq i \leq N$), the energy saving characteristics of each PE, $ES_i( )$, and the maximum slow down ratio S_i of each PE$_i$. The algorithm increases the slack allocation to each PE with uniform time step and records the corresponding energy saving in an array $ES_i[]$ (Steps 2∼4). This procedure ends when the maximum slow down ratio is reached. The value of $ES_i[j]$ tell us the energy saving of extending the total execution time of all tasks on PE_i from $D+j-1$ to $D+j$, where D is the minimum execution time. Based on Theorem 2, the array $ES_i[]$ is decreasing. Using the information in $ES_i[]$, in Steps 7∼11, we distributed slacks one unit by one unit to the PEs that has the highest energy saving. The amount of slacks distributed to p_τ is stored in array $LUT_\tau[idx]$. The ith element of $LUT_\tau[]$ gives the amount of slacks p_τ receiving when the total available slack is i. Let $\mathcal{T}$ denote the minimum time steps in slack distribution. The maximum length of each LUT array is bounded by $deadline \cdot S_{\max}/\mathcal{T}$. The worst case complexity of Algorithm 4 is $O(N \cdot deadline \cdot S_{\max}/\mathcal{T})$.

Algorithm 4 Slack distribution LUT generation

Alg. 4: Generate slack distribution table for task

1. Input: $ARET_\tau$, $ARET\_PE_\tau$ [i], $ESi( )$, $S_{\max}$(maximum slow down ratio)
2. For each PE i with non-zero $ARET\_PE_\tau[i]$ {
3. $ES_i[l] = ES_i(ARET\_PE_\tau[i]+l, ARET\_PE_\tau[i]+l+1)$, $1 \leq l <$ min(deadline, $S_{\max} * ARET\_PE_\tau[i]$);}
4. *Clear* LUT_τ [], $idx = 0$, $slack = 0$;
5. While (not all $Es_i[]$ arrays are empty) {
6. Compare the leading elements in $ES_i[]$ arrays, $1 \leq i \leq N$;
7. Assume the largest value located in the pth array (i.e., $ES_p[0]$);
8. If $p = p_\tau$ then $LUT_\tau[idx{+}{+}] = {+}{+}$slack;
9. else $LUT_\tau[idx{+}{+}] =$ slack;
10. delete the leading element in $ES_p[]$ and point to the next one;
11. }

EXAMPLE 6. Consider the critical path given in Figure 5(a). Assume tasks τ_0 and τ_2 are mapped to PE$_0$ while tasks τ_1 and τ_3 are mapped to PE$_1$. Also assume the WCET of each task is 1 under maximum clock frequency. The energy delay characteristic is given in Figure 5(b). The maximum slow down ratio for both processors is 3. Obviously PE$_1$ is



(a) Critical Path (b) Energy delay characteristics

	ES(2,3)	*ES*(3,4)	*ES*(4,5)	*ES*(5,6)
ES_0	4	1.8	0.7	0.5
ES_1	8	4	1.5	0.9

(c) Energy saving arrays

Fig. 5. An example of LUT generation.

more efficient in DVFS based energy performance tradeoff. The energy saving arrays are presented in Figure 5(c) and they are calculated directly from the energy delay characteristics. If the execution time of τ_0 and τ_2 is 2, then allocating one unit of slack to PE_1 will give 8 units of energy savings. However, if the execution time of τ_0 and τ_2 has already been slowed down to 4, then allocating one more unit of slack to PE_1 will only generate 1.5 units of energy savings. The slack distribution table for task τ_0 is: $LUT_{\tau_0}[1\sim8] = \{0, 0, 1, 1, 2, 2, 3, 4\}$. Which means that if the overall slack is less than 3, no slack should be given to PE_0 (i.e., the PE where τ_0 is mapped to); if the overall slack is greater than 2 and less than 5, then 1 slack should be allocated to PE_0; if the overall slack is greater than 4 and less than 7, then 2 slacks should be allocated to PE_0. If there are 7 and 8 overall slacks, then PE_0 should receive 3 and 4 slacks respectively. □

Algorithm 5 gives the SR_DVFS algorithm. It is executed during runtime. Before executing a task τ, the SR_DVFS algorithm calculates the available slack (Step 1), then it uses the look-up-table to determine how many slacks should be distributed to the current PE where task τ is mapped to (Step 2). The execution speed (i.e., s_τ) is calculated in Step 3 of the algorithm. We then check the longest path to see if this solution violates the deadline constraint (Step 4). If the answer is false, the algorithm returns, otherwise we recalculate the available slack by considering only the longest path (Step 5) and repeat the previous steps. Note that Algorithm 5 has a constant complexity; however, it must be executed each time before a task is executed.

Algorithm 5 SR_DVFS algorithm
Alg. 5: *SR_DVFS* (τ, t): τ—current task id, t—current time

1. Calculate the available slack $t_{sl} = deadline - t - ARET_\tau$;
2. The slack distributed to p_τ is $t_{sl}(\tau) = LUT(t_{sl})$;
3. $s_\tau = (t_{sl}(\tau) + ARET\_PE_\tau)/ARET\_PE_\tau$;
4. if $(WCET(\tau, p_\tau) * (s_\tau - 1) + MxRET_\tau + t > deadline)$ {
5. $t_{sl} = deadline - t - MxRET_\tau$;
6. $t_{sl}(\tau) = LUT(t_{sl})$;
7. $s_\tau = (t_{sl}(\tau) + ARET\_PE_\tau)/ARET\_PE_\tau$;
8. }

If the application consists of single path across different PEs, then the SR_DVFS gives the optimal solution. However, parallel execution paths usually exist in a multi-core system and there are synchronization events among these execution paths. Therefore, the SR_DVFS is only a greedy heuristic. Our experimental results show that compared to the NLP based DVFS, the system using SR_DVFS algorithm consumes 4.1% more energy in average.

The effectiveness of the SR_DVFS relies on two important conditions: (1) the power consumption of a PE must be a convex function of its clock period; (2) the PEs must support continuous voltage and frequency scaling. The first condition can be satisfied by most existing microprocessors with support of DVFS. The second assumption is not quite realistic, because many real life microprocessors only support discrete DVFS. However, it has been pointed out in Ref. [26] that using intra-task DVFS, we can approximate any voltage and frequency setting using 2 discrete voltage and frequency levels. Even if the intra-task DVFS is not available, the SR-DVFS algorithm works fine under discrete voltage–frequency because it reclaims the slack that cannot be utilized by previous tasks due to the voltage (frequency) round up. Note that the NLP based approach cannot handle discrete voltage (frequency) level due to the high complexity. As we will show in Section 6, compared to a DVFS choice that simply rounds up the voltage and frequency solution of the NLP to the nearest level, the slack reclaiming algorithm reduces the energy by 41.37%.

6. EXPERIMENTAL RESULTS

We assume that there are three different kinds of PEs in the system. They are XScale 80500, XScale PXA255 and PowerPC processors. These processors operate at different voltage and frequency, and they have different power/performance characteristics. We obtained their cycle energies under different voltage and frequencies from Refs. [27, 28 and 29] respectively. We use curve fitting to approximate the cycle energy as a continuous function of the cycle period and use this model to predict the cycle energy for any cycle period which is between the maximum and minimum supported frequency. We summarize the processors' parameters in Table II and plot the cycle energy curves in Figure 6.

To account for different graph structures and complexities which resemble numerous real applications, we carried experiments on CTGs modified from random task graphs generated by TGFF.[12] The MPSoC consists of either 3 PEs

Table II. Processor parameters.

Type	Max_V (Volt)	Max_f (MHz)	Min_V (Volt)	Min_f (MHz)
PPC	1.9	333	1	33
Xscale	1.49	733	0.91	333
PXA255	1.3	400	0.85	100

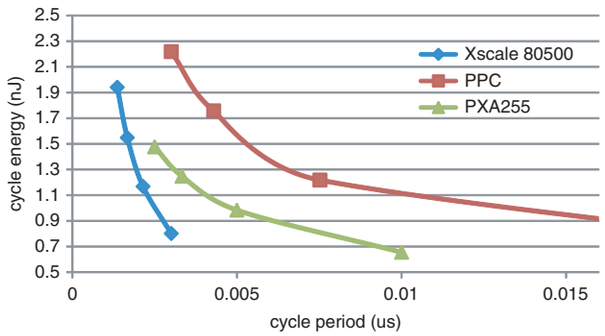


Fig. 6. Processor cycle energy curves.

(one of each aforementioned mentioned type) or 4 PEs (with an additional Xscale 80500 processor). Table III gives the summary of some statistics of the 6 test cases including the number of tasks, the number of PEs, and the number of branch fork nodes in the CTG. The first row of the table gives the ID of each test case, which will be used in the rest of the paper.

In addition to these 6 cases, and in order to test the scalability of the proposed algorithms and its performance on large applications and systems, we also construct a large CTG with 86 tasks and allocate them on a 16-PE MPSoC system. We call the test case CTG-large.

Furthermore, we apply our algorithm on a real-world application, i.e., the MPEG decoder. The MPEG 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 16×16 pixel area of the image. The macroblocks are further classified into I, P and B blocks. Different types of macroblock require different decoding procedure. For example, the I-blocks will perform the IDCT function while the B-blocks may skip the IDCT. The CTG for MPEG decoding process have 46 tasks. Among these tasks, 9 of them are branch nodes. Due to space limitations we cannot show the whole graph of the decoding process. We map the MPEG decoding CTG to a system consists of 6 PEs. We call the test case CTG-mpeg.

We define the *system utilization* as $U = \sum_{\tau_i \in V} WCET(\tau_i)/(D \cdot N)$ where D is the application deadline and N is the number of PEs. Obviously, the effectiveness of the low power scheduling algorithm should change as the system utilization varies.

We refer to our algorithm as Balanced Energy Slack Scheduling (BESS) as it considers both the energy and performance in a heterogeneous system. For each CTG, the algorithm generates a task mapping, a schedule and a voltage/frequency selection. We call the combination a *Solution*.

Sometime, the mapping between tasks and PEs are predetermined. We can extend BESS to find the task execution order for the CTGs with fixed mapping by considering only those given task-PE pairs in DL calculation. We refer to this degenerated version of BESS as BESS-FM (BESS with Fixed Mapping). For comparison purpose, we implemented NLP based DVFS similar to the algorithm described in Ref. [8]. By default, the BESS algorithm uses NLP based DVFS. We use BESS-SR to refer to the BESS algorithm utilizing the slack reclamation based DVFS.

One of the major questions of BESS is how to determine the value of λ in Eq. (6). When λ is 0, the algorithm degenerates into the minimum average make span scheduling; and when λ is very large, the algorithm maps a task to the minimum energy processor without considering the slacks for the DVFS algorithm. Strictly speaking, the optimum value of λ is a function of many parameters, including the topology of the CTG, the branch probability, the performance and power characteristics of the multicore system, and the system utilization, etc. For those 8 test cases, we vary their branch probability and deadline (which consequently affects the system utilization) and generate a large number of different scenarios. For each scenario, we sweep λ to find its optimal value. We found that (1) the best value of λ is always within a small range for majority of the scenarios; (2) some values of λ consistently outperform other values for all system utilization and branch probability settings, and the difference between these local maximum points is not significant.

As an example, Figure 7 shows the energy as a function of λ for CTG0 under different system utilization from 20% to 70%. Please note that not all λ have feasible solutions for all system utilizations, for example, there is feasible

Table III. Summary of generated test cases.

Test case ID	0	1	2	3	4	5
Number of nodes in CTG	12	25	16	15	15	25
Number of PEs	3	3	3	4	4	4
Number of branch fork nodes	1	3	1	2	1	3
Probability of each branch fork nodes	(0.9, 0.1)	(0.9, 0.1) (0.9, 0.1) (0.9, 0.1)	(0.9, 0.1)	(0.9, 0.1) (0.9, 0.1)	(0.8, 0.1, 0.1) (0.8, 0.2)	(0.8, 0.2) (0.8, 0.2)

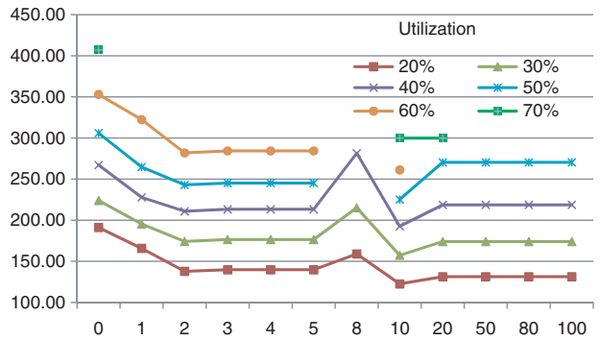


Fig. 7. The effect of λ on energy consumption.

schedule when the system utilization is at 70% and λ is-between 1 and 8. As shown in the figure, most of the energy curves have two local minima. The first one occurs at $\lambda \in$ Refs. [2, 5] while the second occurs at $\lambda = 10$. The first local maximum has slightly higher energy than the second local maximum. However, overall, both local minima give much lower energy than other data points in the energy curve. Similar trend is observed for other test cases. Based on this observation, we use a fixed λ for each CTG in our experiments. It is set to be the best value that minimizes the energy of the CTG at 50% utilization and equal branch selection probability. As we will show later, the task mapping obtained by BESS algorithm outperforms the best random mapping in 10,000 samples. Given the limited range of the optimal value of λ, searching for the best λ is much easier than searching for the best mapping directly.

6.1. Comparison with Random Mapping

In the first experiment, we evaluate the quality of the task mapping function in BESS. For each test case, large numbers of random mappings are generated and scheduled using BESS-FM. The best result is compared to the result found by the original BESS, which optimizes both task mapping, ordering and DVFS.

Table IV compares the system energy dissipation of running the solution generated by BESS, the best results and average results of 10,000 random mappings scheduled using BESS-FM and the improvement of BESS over the best and average random mapping. As we can see, even with 10,000 random samples, we cannot find a task mapping that is as energy efficient as the one found by BESS.

Table IV. Comparison to random mapping.

		Best of 10,000 random mapping (mJ)		Average of 10,000 random mapping (mJ)	
CTG#	BESS (mJ)	BESS-FM	Impr. (%)	BESS-FM	Impr. (%)
CTG0	157.38	163.97	4.02	291.54	46.02
CTG1	101.01	120.88	16.44	158.72	36.36
CTG2	196.99	378.24	47.92	763.07	74.18
CTG3	298.25	320.77	7.02	506.21	41.08
CTG4	512.08	558.99	8.39	829.40	38.26
CTG5	918.38	1042.85	11.94	1250.61	26.57
CTG-large	1114.44	2604.5	57.21	3686.68	69.77
CTG-mpeg	92.99	110.29	15.69	162.92	42.92

6.2. Energy Consumption Under Different System Utilization

In the following experiments, we will demonstrate the necessity of considering delay and energy jointly during every step of the resource management algorithm, and will compare the solution of our algorithm with four other scheduling algorithms: the *Minimum Make Span* (*MMS*), the *Minimum Energy* (*MNRG*), the *Earliest Start Time* (*EST*), and one of the previous works in Ref. [2]. These 4 algorithms are very similar to BESS. The only deviation is how the dynamic level (i.e., DL) is calculated. The MMS algorithm uses the original WCET instead of the effective WCET that converts energy heterogeneity to performance heterogeneity. Therefore, it maps tasks to minimize the average make span as stated in Section 4.1. Please note that the MMS algorithm is equivalent to the CAP algorithm.[13, 14] The MNRG algorithm calculates the dynamic level as: $DL(\tau_i, p_j) = -WCET(\tau_i, p_j) \times E_{\text{total}}(p_j, f_{\max})$; therefore, it maps a task to the PE that is the most energy efficient for this task. Finally the EST algorithm calculates the dynamic level as: $DL(\tau_i, p_j) = -AT(\tau_i, p_j)$, therefore, it maps tasks to where they can start the earliest.

To the best of our knowledge, there is no existing work that considers the exact same problem as this paper (i.e., task mapping, scheduling and DVFS for CTG in a heterogeneous multiprocessor platform.) In order to compare with existing work, we modified the algorithm proposed by Zhang et al. in Ref. [2] to consider conditional branches. Their algorithm is modified so that the slack is replaced by average slack and mutually exclusive tasks can share the same PE at the same time.

Figure 8 shows the energy consumption of the 8 test cases under those 5 scheduling algorithms. In those figures X-axis represents the percentage system utilization and Y-axis represents the energy consumption. Please note that not all scheduling algorithms have feasible solutions for all system utilizations. As we expected, when the system utilization increases, the energy consumptions also increase because tasks have to run at higher speed to guarantee the deadline.

As shown in the figure, in general, MNRG is ineffective as it cannot find feasible solution for most of the test cases. The BESS algorithm is able to find better solutions than other four algorithms in all cases because it is not only able to leave enough slack for DVFS but also able to explore the energy heterogeneity. Because Ref. [2] approach does not consider power/performance heterogeneity, it could not achieve the same energy savings as BESS. Overall,

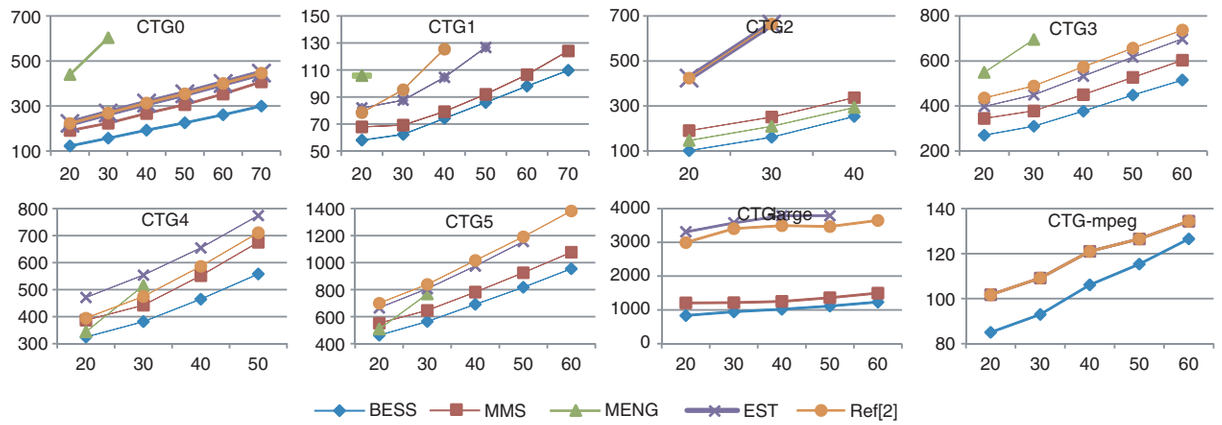


Fig. 8. Energy consumption under different system utilization.

comparing to BESS, MMS, MNRG, EST and Ref. [2] consume 25.3%, 85.7%, 69.8% and 70.3% more energy for test case 0–5 respectively.

Please note that because the search spaces of test case CTG-large and CTG-mpeg are too large, the NLP based DVFS is not able to find a solution within a reasonable time, while our SR algorithm only takes several seconds. Therefore, we integrate the SR based DVFS to all the mapping and scheduling algorithms in CTG-large and CTG-mpeg test case. As shown in this figure, in CTG-large case, BESS algorithm outperforms MMS, EST and Ref. [2] by 21.6%, 73.0%, 69.9%. In CTG-mpeg case, MMS, EST and Ref. [2] algorithm have very similar energy consumption, while the BESS algorithm outperforms them by 11.64%.

6.3. The Effectiveness of Slack Reclaiming Algorithm

In order to evaluate the performance of our slack reclamation (SR) based DVFS heuristic, we developed a fast performance evaluation framework based on OMNeT++.[31] OMNeT++ is a discrete event simulation environment, and its simulation kernel is written in C++. Users can define the behavior of their own components in the network and describe the network topology and communication overhead using a high level language provided by the OMNeT++.

We again vary the system utilization from 20% to 70% and compare the quality of DVFS solutions found by the SR and NLP algorithm. The results are shown in Figure 9. The same task mapping and ordering found by BESS are used for both DVFS algorithms.

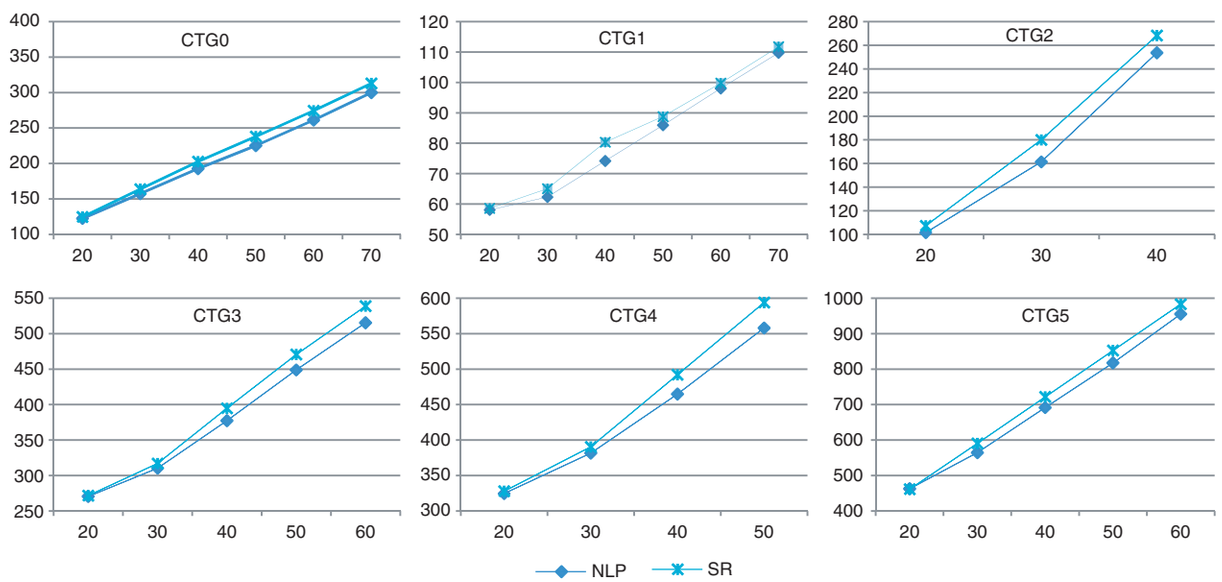


Fig. 9. Comparison of NLP and SR based DVFS.

Table V. Comparison of continuous and discrete DVFS.

	Continuous DVFS (mJ)		Discrete DVFS (mJ)	
CTGs	NLP	SR	NLP	SR
CTG 0	192.64	202.66	415.29	218.92
CTG 1	73.19	80.34	183.86	92.82
CTG 2	253.59	268.27	321.29	270.44
CTG 3	377.38	394.75	614.2	418.48
CTG 4	464.83	492.13	859.52	491.86
CTG 5	697.23	721.79	1985.3	775.60

As shown in these figures, the SR algorithm tracks the NPL based DVFS algorithm very well and gives close energy consumptions. In average, systems using SR based DVFS method only consume 4.1% more energy than the system using NLP based DVFS.

Table V shows the comparison between the continuous and discrete DVFS. As shown in the table, the performance of NLP based DVFS is severely degraded when only discrete voltage and frequency level is supported by the system, while the SR algorithm dynamically adjusts the slowdown ratio based on current remaining slack. On average, the discretized slack reclaiming algorithm reduces the energy dissipation by 41.37% comparing to discretized NLP algorithm. Please note that for test case CTG-large CTG-mpeg, the only feasible DVFS algorithm is Slack Reclaiming based algorithm, thus in this subsection, we do not have results for them.

In the next experiment we restrict the PE's voltage and frequency to those discrete levels specified in Table II. With NLP based DVFS, in order to guarantee deadline, the voltage and frequency are always round up to the nearest level. With SR based DVFS, the extra slack generated by rounding up of current task will be reclaimed by future tasks.

6.4. Sensitivity to Branch Probability Change

In previous experiments, we assume that the branch probabilities are known and fixed during run time. In the next set of experiments, we examine the sensitivity of BESS to the change of branch probability. The results will help us to understand the performance of BESS under inaccurate software model or insufficient profiling information.

The solution of BESS consists of 3 parts: task mapping, task ordering and voltage/frequency selection. When the branch probability of an application changes from $\mathscr{A}$ to $\mathscr{B}$ during the runtime, we have 4 options: (1) Continue using the initial scheduling found based on branch probability $\mathscr{A}$. This option will be referred as "No update" as it updates nothing. (2) Continue using the original task mapping and ordering, but update the voltage and frequency selection using the new branch probability $\mathscr{B}$. This option will be referred as "DVFS only." (3) Continue using the original task mapping, but update the task ordering and voltage/frequency selection. It will be referred as "DVFS + SCH." (4) Update everything in solution including task mapping, ordering and voltage/frequency selection. We refer to this option as "All update." In the rest of the paper, we refer to the original branch probability $\mathscr{A}$ as the "*biased branch probability*" and the updated branch probability $\mathscr{B}$ as the "*actual branch probability*." We refer to the first 3 scheduling options that update none or only part of the solution as *biased scheduling* and the last scheduling option as the *unbiased scheduling*.

We use CTG0 to illustrate the impact of branch probability on the quality of solution. We set the actual branch probability $\mathscr{B}$ to $(0.1, 0.9)$. Figure 10 shows the energy dissipation of the system under 3 biased scheduling algorithms with the biased probability $\mathscr{A}$ varying from (0.1, 0.9) to (0.9, 0.1). All 3 curves give the same best result when the biased probability $\mathscr{A}$ equals to the actual branch probability $\mathscr{B}$, which stands for the unbiased scheduling option (i.e., "ALL") that updates everything in the solution.

From this figure, we can see that the more accurate probability information we use in scheduling, the more energy savings we can achieve. We also observe that when the biased probability is close to the actual probability, the performance of the biased scheduling and the unbiased scheduling are close to each other. For example, when the biased probability goes from $(0.1, 0.9)$ to $(0.6, 0.4)$, the energy difference between the biased and the unbiased solution is less than 4%. On the other hand, when the biased probability is far from the real probability, the energy difference between the biased and the unbiased solution can be as large as 25%. The above observation implies that if the branch probability changes during run time, as long as the difference between the actual probability and the biased probability is small, the energy consumption will not significantly deviate from the optimum value. Therefore, we do not need to rerun the entire scheduling process (which can be very time consuming for systems with large number of tasks and PEs) to adapt the probability change.

The similar trend is found for other CTGs. First, we define the *Bias Distance* to quantify the difference between the actual branch probability and the biased

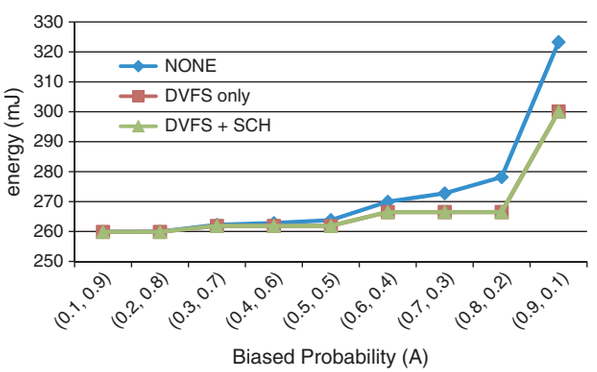


Fig. 10. Energy consumption under inaccurate branch probability.

Table VI. Solution quality degradation as the bias distance increases.

Bias distance	0.2 (%)	0.4 (%)	0.6 (%)	0.8 (%)	1 (%)	1.2 (%)	1.4 (%)	1.6 (%)
DVFS + SCH	0.14	0.30	0.69	1.12	1.71	2.33	3.16	5.12
DVFS only	0.22	0.34	0.74	1.19	1.79	2.44	3.32	5.12
No update	0.27	0.54	1.06	1.72	2.57	3.54	4.85	7.47

probability. Assume a CTG has n conditional edges and the actual branch probabilities for these branches are $B = (b_1, b_2, \ldots, b_n)$, the biased branch probabilities are $A = (a_1, a_2, \ldots, a_n)$. Then the distance between A and B is defined as the L_1-norm of the vector $A - B$, i.e., $D(A, B) = \sum_{i=1}^{n} |a_i - b_i|$. A large bias distance means that the biased probability used for optimization is far from the actual probability during runtime, which, intuitively, could induce low quality solution. This trend is shown in Table VI. For each CTGs, we select one branch fork node and vary its real branch probabilities to generate test cases whose biased distance ranging from 0.2 to 1.6. For each test case, the energy dissipation of the system under 3 biased scheduling algorithms is recorded. Table VI gives the percentage energy increase of biased solutions over the unbiased solution. As expected, when the bias distance increase, the solution quality degrades and energy consumption increases. Furthermore, performing "DVFS only" and "DVFS + SCH" give very similar results in terms of energy savings. This indicates that without correct task mapping, updating the task ordering does not make much difference.

7. CONCLUSION

In this paper, we proposed a framework for simultaneous task mapping and ordering followed by DVFS of control intensive real-time applications modeled as probabilistic conditional task graph. The goal of our scheduling algorithm is to minimize the mathematical expectation of the energy by utilizing the branch selection probability. The proposed mapping and scheduling algorithm balances the energy and performance of tasks running on a heterogeneous multi-core platform. The proposed slack reclaiming DVFS algorithm effectively distributes the slack based on current branch selection at run time and achieves similar energy reduction as mathematical programming based DVFS with much lower complexity.

We compared our proposed mapping and ordering algorithm with the minimum makespan, the minimum energy, the earliest start time and the mapping and ordering algorithm proposed in Ref. [2] on 8 CTGs, and the proposed algorithm can reduce 25.3%, 85.7%, 69.8% and 70.3% energy consumption respectively. The proposed Slack Reclaiming DVFS algorithm could find good solution which only consumes 4% more energy compared to the solution found by Nonlinear Mathematical Programming based DVFS when the operating voltage can be adjusted continuously. When the operating voltage is discrete, our Slack Reclaiming DVFS algorithm reduces the energy dissipation by 41.37% comparing to discretized NLP algorithm.

References

1. J. Luo and N. K. Jha, Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. *Proc. International Conference on VLSI Design*, January **(2002)**, pp. 719–726.
2. Y. Zhang, X. Hu, and D. Z. Chen, Task scheduling and voltage selection for energy minimization. *Proc. Design Automation Conference*, June **(2002)**, pp. 183–188.
3. J. Hu and R. Marculescu, Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. *Proc. Conference and Exhibition on Design, Automation and Test in Europe*, February **(2004)**, pp. 234–239.
4. E. Dolif, M. Lombardi, M. Ruggiero, M. Milano, and L. Benini, Communication-aware stochastic scheduling framework for conditional task graphs in multi-processor systems-on-chip. *Proc. International Conference on Embedded Software*, September **(2007)**, pp. 47–56.
5. P. Eles, K. Kuchcinski, Z. Peng, A. Doboli, and P. Pop, Scheduling of conditional process graphs for the synthesis of embedded systems. *Proc. Conference and Exhibition on Design, Automation and Test in Europe*, February **(1998)**, pp. 132–139.
6. Y. Xie and W. Wolf, Allocation and scheduling of conditional task graph in hardware/software co-synthesis. *Proc. Conference and Exhibition on Design, Automation and Test in Europe*, March **(2001)**, pp. 620–625.
7. D. Wu, B. M. Al-Hashimi, and P. Eles, Scheduling and mapping of conditional task graph for the synthesis of low power embedded systems. *IEEE Proceedings of Computers and Digital Techniques*, September **(2003)**, Vol. 150, pp. 262–273.
8. D. Shin and J. Kim, Power-aware scheduling of conditional task graphs in real-time multiprocessor systems. *Proc. International Symposium on Low Power Electronics and Design*, August **(2003)**, pp. 408–413.
9. E. Jacobsen, E. Rotenberg, and J. E. Smith, Assigning confidence to conditional branch predictions. *Annual International Symposium on Microarchitecture*, November **(1996)**, pp. 142–152.
10. A. K. Uht and V. Sindagi, Disjoint eager execution: An optimal form of speculative execution. *Proc. International Symposium on Microarchitecture,* November **(1995)**, pp. 313–325.
11. 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*, February **(1993)**, Vol. 4, pp. 175–187.
12. R. P. Dick, D. L. Rhodes, and W. Wolf, TGFF: Task graphs for free. *Proc. International Workshop on Hardware/Software Codesign* March **(1998)**, pp. 15–18.
13. P. Malani, P. Mukre, Q. Qiu, and Q. Wu, Adaptive scheduling and voltage scaling for multiprocessor real-time applications with non-deterministic workload. *Proc. Design Automation and Test in Europe*, March **(2008)**.
14. P. Malani, P. Mukre, and Q. Qiu, Power optimization for conditional task graphs in DVS enabled multiprocessor systems. *Proc. International Conference on VLSI-SoC*, October **(2007)**.
15. B. Schott, M. Bajura, J. Czarnaski, J. Flidr, T. Tho, and L. Wang, A modular power-aware microsensor with $>1000\times$ dynamic power range. *Proc. Information Processing in Sensor Networks*, April **(2005)**.
16. D. McIntire, K. Ho, B. Yip, A. Singh, W. Wu, and W. J. Kaiser, The low power energy aware processing (leap) embedded networked

sensor system. *Proc. International Conference on Information Processing in Sensor Networks* **(2006)**.
17. D. Lymberopoulos, B. Priyantha, and F. Zhao, Mplatform, A reconfigurable architecture and efficient data sharing mechanism for modular sensor nodes. *Proc. Information Processing in Sensor Networks* **(2007)**.
18. D. Roberts, R. G. Dreslinski, E. Karl, T. Mudge, D. Sylvester, and D. Blaauw, When homogeneous becomes heterogeneous—Wearout aware task scheduling for streaming applications. *Proc. Workshop on Operationg System Support for Heterogeneous Multicore Architectures*, September **(2007)**.
19. J. Luo and N. Jha, Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. *Proc. Asia and South Pacific Design Automation Conference* **(2002)**.
20. Y. Liu, B. Veeravalli, and S. Viswanathan, Novel critical-path based low-energy scheduling algorithms for heterogeneous multiprocessor real-time embedded systems. *Proc.* 13*th International Conference on Parallel and Distributed Systems* **(2007)**.
21. M. Goraczko, J. Liu, D. Lymberopoulos, S. Matic, B. Priyantha, and F. Zhao, Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems. *Proc. Design Automation Conference* **(2008)**.
22. C. Y. Yang, J. J. Chen, T. W. Kuo, and L. Thiele, An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems. *Proc. Design, Automation and Test in Europe Conference and Exhibition* **(2009)**.
23. J. Goossens, D. Milojevic, and V. Nelis, Power-aware real-time scheduling upon dual CPU type multiprocessor platforms. *Proc.* 12*th International Conference on Principles of Distributed Systems* **(2008)**.
24. A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau, Profile-based dynamic voltage scheduling using program checkpoints. *Proc. DATE* **(2002)**.
25. P. Yang and F. Catthoor, Pareto-optimization-based run-time task scheduling for embedded systems. *Proc. CODES + ISSS* **(2003)**.
26. T. Ishihara and H. Yasurra, Voltage scheduling problem for dynamically variable voltage processors. *Proc. International Symposium on Low Power Electronics Design*, August **(1998)**.
27. K. Choi, W.-C. Cheng, and M. Pedram, Frame-based dynamic voltage and frequency scaling for an MPEG player. *J. Low Power Electron.* 1, 27 **(2005)**.
28. G. Dhiman and T. Rosing, Dynamic voltage frequency scaling for multi-tasking systems using online learning. *Proc. Intern. Symp. on Low Power Electronics Design*, August **(2007)**.
29. J. Lu and Q. Qiu, Scheduling and mapping of periodic tasks on multi-core embedded systems with energy harvesting. *Proc. International Green Computing Conference*, July **(2011)**.
30. Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan, Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects. University of Virginia, Department of Computer Science Technical Report **(2003)**.
31. OMNeT++: http://www.omnetpp.org/.

Yang Ge

Yang Ge *received his B.S. degree in telecommunication engineering from Zhejiang University, China in 2007, and M.S. degree from the department of Electrical and Computer Engineering of Binghamton University, USA in 2009. He is currently working on his Ph.D. degree in Department of Electrical Engineering and Computer Science in Syracuse University, USA. His research interests include power and thermal analysis and optimization for multi and many-core system.*

Yukan Zhang

Yukan Zhang *received her B.S. degree in electrical engineering from Nankai University, China in 2006, and M.S. degree from the department of Electrical and Computer Engineering of Binghamton University, USA in 2009. She is currently working on her Ph.D. degree in Department of Electrical Engineering and Computer Science in Syracuse University, USA. Her research interests include energy harvesting and management for embedded systems.*

Qinru Qiu

Qinru Qiu *received her M.S. and Ph.D. degrees from the department of Electrical Engineering at University of Southern California in 1998 and 2001 respectively. She received her B.S. degree from the department of Information Science and Electronic Engineering at Zhejiang University, China in 1994. Dr. Qiu is currently an associate professor at the Department of Electrical Engineering and Computer Science in Syracuse University. Before joining Syracuse University, she has been an assistant professor and then an associate professor at the Department of Electrical and Computer Engineering in State University of New York, Binghamton. Her research areas are energy efficient computing systems, energy harvesting real-time embedded systems, and neuromorphic computing. She has published more than 50 research papers in referred journals and conferences. Her works are supported by NSF, DoD and Air Force Research Laboratory.*

Qing Wu

Qing Wu *received his Ph.D. degree from the department f Electrical Engineering at University of Southern California in 2002. He received his B.S. and M.S. degrees from the department of Information Science and Electronic Engineering at Zhejiang University (Hangzhou, China) in 1993 and 1995, respectively. Dr. Wu is currently a Senior Electronics Engineer at the United States Air Force Research Laboratory (AFRL), Information Directorate (RI). Before joining AFRL, he was an Assistant Professor in the Department of Electrical and Computer Engineering at State University of New York, Binghamton. His research interests include large-scale computational intelligence models, high-performance computing architectures, circuits and systems for energy-efficient computing. He has published more than forty research papers in international journals and conferences.*