ApproxMap: On Task Allocation and Scheduling for Resilient Applications

Juan Yi[†], Qian Zhang[‡], Ye Tian[‡], Ting Wang[‡], Weichen Liu[†], Edwin H.-M. Sha[†], Qiang Xu[‡]
[†]College of Computer Science, Chongqing University, Chongqing, China.
[‡]The Chinese University of Hong Kong, Hong Kong.
{jenni1987, wliu, edwinsha}@cqu.edu.cn, {qzhang, tianye, twang, qxu}@cse.cuhk.edu.hk

Abstract— Many emerging applications are inherently error-resilient and hence do not require exact computation. In this paper, we consider the task allocation and scheduling problem for mapping such applications to voltage-scalable multiprocessor systems. The proposed solution, namely ApproxMap, judiciously determines the mapping and execution sequence of resilient tasks to minimize the energy consumption of the application while meeting their target quality requirements and timing constraints. To be specific, ApproxMap generates energy-efficient yet flexible task schedule at design-time, and conducts lightweight online adjustment according to runtime dynamics for further energy-efficiency improvement. Experimental results on various task graphs demonstrate the efficacy of ApproxMap.

I. INTRODUCTION

Error resilience can be broadly defined as the characteristic of an application to produce acceptable outputs despite its constituent computations being performed imperfectly (with error). Many emerging applications (e.g., Recognition, Mining and Synthesis) in the big data era exhibit this intriguing trait [4, 6]. Because these applications are typically compute-intensive, they are usually quite energy-hungry. Therefore, it is desirable to exploit application resilience property and apply approximate computing for energy savings, especially for battery-operated embedded devices.

Numerous approximate computing techniques have been developed to exploit this resilient feature for improved performance and/or energy efficiency gains [6, 11–16]. Techniques at the system level include analysis and characterization of application resilience [6], dynamic modifying operand bit-width [12], design of lightweight quality checker [14, 15], etc. These efforts have established the significant potential of approximate computing, and there is a growing interest in this area.

Recently, Karakonstantis *et al.* [9] propose a software-hardware co-design technique for error-resilient applications, which can identify, schedule and execute the tasks and obtain acceptable quality under given operating conditions. The system identifies critical tasks at runtime based on special directives and schedules these tasks to the appropriate units that can dynamically switch between accurate/approximate operation mode by tuning voltage/frequency. While targeting similar problem for error-resilient application, [9] is quite brief and mainly focuses on software-hardware synergy without giving any systematical approach for task allocation and scheduling.

On the other hand, task allocation and scheduling has been a focused research problem in the parallel processing domain for a long time. In particular, various power-aware task allocation and scheduling techniques have been presented to improve energy-efficiency of multiprocessor systems [3, 8, 17]. These solutions, however, assume that tasks must be executed correctly in the system and do not take error-resilience features of applications into consideration.

To the best of knowledge, resilience-aware mapping and scheduling policy has yet to be explored in the literature, despite significant research efforts have been dedicated to approximate computing. For emerging applications that contain both resilient and error-sensitive tasks running on multi-processor systems, it is possible to over-scale the processor voltage for resilient tasks mapped onto it and use lightweight quality checkers to detect whether it is acceptable. If not, we could re-execute the task with higher supply voltage until the results are with acceptable quality. By judiciously determining the mapping and scaling sequence of resilient tasks, we are able to achieve better energy savings when compared to existing task allocation and scheduling solutions (e.g., [10]) that do not exploit application error-resilience property. The objective of the proposed solution, namely *ApproxMap*, is to investigate how and to what extent energy savings can be achieved via effective and efficient task allocation and scheduling, without compromising the target quality requirements and timing constraints. The main contributions of our work can be summarized as follows:

- We propose a hybrid resilient application mapping and scheduling framework for voltage-scalable multiprocessor system, which integrates a comprehensive design-time analysis methodology with lightweight online adjustment strategy according to runtime dynamics;
- We propose to use a two stage approaches to solve the scheduling problems at design time, where an integer linear programming (ILP) model generates an initial schedule that guarantees the performance requirement in the worst case scenario, and a simulated annealing-based algorithm to refine the schedule that takes potential runtime energy savings into consideration;
- Our run-time scheduler utilizes a novel lightweight run-time heuristic that manages run-time slack reclamation without diminishing the benefits of schedule generated at design time.

The rest of the paper is organized as follows. Section II formulates the resilient application and the error probability model. Section III presents the overall ApproxMap framework. Offline scheduling algorithms and online adjustment strategies are then detailed in Section IV and Section V, respectively. Experimental results are presented in Section VI. Finally, Section VII concludes this paper.

II. PRELIMINARIES

A. System Model

We consider multiprocessor system-on-a-chip (MPSoC) that consists of a set of processor cores, denoted as $\mathcal{M}$. For each processor, the operating voltage can be scaled among a set $\mathcal{V} = \{V_1, V_2, \cdots, V_K\}$, where $V_1 < V_2 < \cdots < V_K$. V_K is the nominal voltage, while the other voltage level could potentially impact the correctness of the computation (i.e., VOS without frequency scaling).

978-1-4673-9569-4/16/$31.00 ©2016 IEEE

We assume that the processors are architecturally identical, and that the only source of heterogeneity is the operational voltage level of processor cores.

B. Application Model

We use the *Data Flow Graph* (DFG) to represent a resilient application to be executed in the system. A DFG $G = \langle \mathcal{T}, \mathcal{E}, \mathcal{R} \rangle$ is a *directed acyclic graph* (DAG), where $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$ is the set of nodes, and $\mathcal{E} \subseteq \mathcal{T} \times \mathcal{T}$ is the set of edges that defines the precedence relations among the nodes in $\mathcal{T}$. Each node $\tau_i \in \mathcal{T}$ represents a task to be executed with worst-case execution time (WCET) et_i. Each edge in the graph ($e_{ij} \in \mathcal{E}$) indicates that task τ_j is dependent on τ_i. A deadline L is associated with DFG G, which means that the application G has to be completed before L. The power consumption of task τ_i under voltage level V_j is denoted as $W_{i,j}$. The energy for executing τ_i at voltage V_j can be expressed as $E_{i,j} = W_{i,j} * t_i$.

Generally speaking, different tasks in an application are not equally susceptible to errors due to different data and control flow properties, internal error masking effects, etc. Therefore, these tasks exhibit distinct error-resilience capabilities. On the other hand, there exist error-sensitive parts (e.g., control flow) that using inexact computations for them may cause fatal errors or even crash the system. Thereby we classify tasks into two categories, namely, resilient tasks and sensitive tasks, and use set $\mathcal{R} = \{Resilt_1, Resilt_2, ..., Resilt_n\}$ to indicate the tasks' fault-tolerant property. For sensitive tasks, we must guarantee its correctness by using nominal voltage without any scaling. For the resilient ones, we can exploit its resilience capability thoroughly and choose the proper operating voltage by using VOS for energy efficiency improvement while still meeting the quality requirements.

C. Error Probability Model

For a given operation, some sets of operands may generate correct outputs, while other sets of operands would result timing error and may lead to unacceptable output quality. In addition, different operations within the ALU may have different critical-path length. To be specific, errors depend upon the voltage selected for execution, the operation and operands. Moreover, as mentioned earlier, for inherently resilient tasks functionality is defined on a continuous scale of output quality. Therefore, incorrect output may be acceptable if it meet the quality requirements. In other words, for a resilient task τ and input I, the result may be acceptable even if it is not the correct output due to the overscaled voltage. In this paper, we model this with the following two assumptions.

Given a task τ and a specific workload as input I, there exists a threshold voltage $V_{th}(\tau, I)$: using any voltage V below the threshold ($V < V_{th}(\tau, I)$) will lead to unacceptable results, while using any voltage above that threshold ($V \geq V_{th}(\tau, I)$) will always lead to a successful execution. Note that different workload as inputs for the same computation may have different threshold voltages.

Given a task τ_i which executes under voltage V_m, the probability that the computation fails to meet quality requirement, denoted as $Pr_{i,m}$, is computed as $Pr_{i,m} = \frac{I'(\tau_i, V_m)}{I(\tau_i)}$, where $I(\tau_i)$ denotes the set of all possible workloads for task τ_i and $I'(\tau_i, V_m)$ denotes the set of inputs for which task τ_i will fail at voltage V_m.

III. System Overview

A. Problem Definition

Based on the above, our resilience-aware task allocation and scheduling on voltage scalable multiprocessors can be formulated as follows. Given a resilient application $G = \langle \mathcal{T}, \mathcal{E}, \mathcal{R} \rangle$ with an associated deadline L, and the voltage scalable system containing a set of cores $\mathcal{M}$, the objective is to allocate and schedule the execution of the resilient application, such that: (i). all tasks are executable; (ii). every task completes by its specified deadline (if any); and (iii). the total energy consumption is minimized.

Note that, for each resilient task with different error probabilities at various operational voltages, we try to execute it with lower voltage whenever possible for energy-efficiency gains. If it cannot meet its quality requirement, we would re-execute the task with higher voltage and this procedure may continue until the task is executed with nominal voltage.

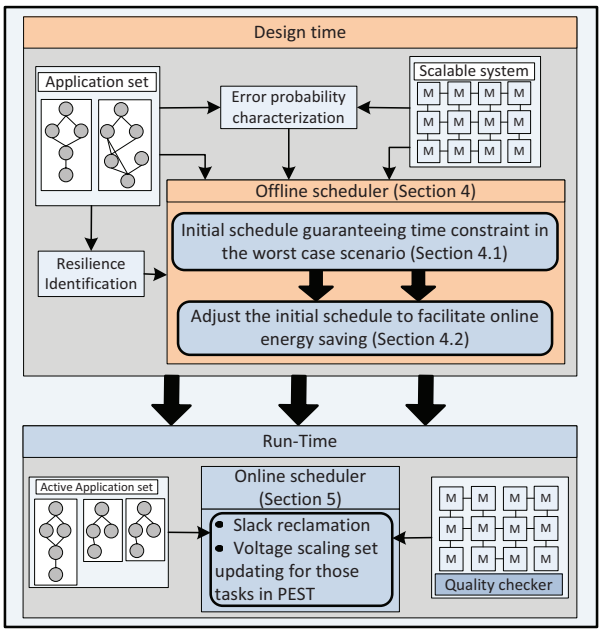

Figure 1. The proposed ApproxMap framework.

B. The Proposed Framework

We propose the so-called *ApproxMap* solution to address the above problem. ApproxMap is performed at two phases, as demonstrated in Figure 1. During the design time, we first provide an optimal initial schedule that minimizes the total expected energy consumption with timing constraint satisfied in the worst case scenario by using an integer linear programming (ILP) model. As the resilient tasks could complete earlier within its scheduled time slot, there might be some online time slacks which could be used to improve the scheduling for the upcoming tasks. However, as shown in Section IV.B, using the schedule from ILP model directly could lead to these slacks are unusable at runtime. To tackle this problem, we then adjust the initial schedule by modifying the task-core assignment, as detailed in Section IV.B. During the run time, we utilize a novel lightweight heuristic that co-manages run-time slack reclamation, voltage scaling set updating and output quality management in a multi-core environment without diminishing the benefits of schedule generated at design time.

IV. Offline Solution

In the offline stage, ApproxMap generates an initial task schedule for runtime execution. Such initialization should have the following features: (i) prevent the application from timing violation; (ii) get as much potential energy savings as possible. To achieve these targets,

we first provide a schedule with minimum expected energy consumption by using an ILP model, and then refine this schedule by a simulated annealing-based algorithm to maximize the potential runtime energy savings.

A. Initialization from ILP Model

In this section, we analyze and formulate the scheduling problem by an ILP model to obtain an optimal initial schedule.

For any resilient task τ_i with input I, it is almost impossible to make exact prediction on which single voltage level can output the result with quality satisfied and energy consumption minimized. In addition, for a single voltage level V_m, it cannot guarantee the output result of τ_i is acceptable for any input. Therefore, we use a voltage scaling set, each of which is ended with the nominal voltage V_K, to represent the execution process of a resilient task.

Definition 1. *For any resilient task $\tau_i \in \mathcal{T}$, the scaling set, denoted as S_i, is defined as the sequence of voltages $S_i = \{V_1, V_2, \cdots, V_K\}$ selected from $\mathcal{V}$, where $V_1 < V_2 <, \cdots, < V_K$ under which task τ_i is going to be executed.*

For instance, $S_i = \{V_2, V_K\}$ is a valid scaling set for task τ_i to execute. With this scaling set, τ_i would first execute with voltage V_2, followed by V_K. Based on error probability model in Section II, the probability that the execution of task τ_i does not meet the quality requirement (i.e., fails) at voltages $V_1, V_2, \cdots, V_{m-1}$ but succeeds at V_m [5] is:

$$P_{succ}(i, V_m) = Pr_{i,m-1} - Pr_{i,m} \qquad \forall i \in \mathcal{T} \tag{1}$$

Suppose a scaling set $S_i = \{V_1, V_2, \cdots, V_m\}$ is scheduled to execute task τ_i, where $V_1 < V_2 <, \cdots, < V_m$, and $V_m = V_K$. The expected energy consumption is:

$$E(\tau_i, S_i) = (W_{i,1} + \sum_{j=2}^{m} Pr_{i,j-1} * W_{i,j}) * et_i \tag{2}$$

Proof. There are m situations that would happen when using this scaling set executing τ_i, namely, executing successfully at V_1 where the probability is $1 - Pr_{i,1}$, executing fails at voltage V_1, but succeeds at V_2. etc. Therefore, according to the definition of expected value, $E(\tau_i, S_i) = (1 - Pr_{i,1}) \cdot W_{i,1} \cdot et_i + P_{succ}(i, V_2) \cdot (W_{i,1} + W_{i,2}) \cdot et_i + \cdots + P_{succ}(i, V_m) \cdot (W_{i,1} + W_{i,2} + \cdots + W_{i,m}) \cdot et_i$, where $Pr_{i,m}$ is 0 and $P_{succ}(i, V_m) = Pr_{i,m-1}$. The value is $(W_{i,1} + \sum_{j=2}^{m} Pr_{i,j-1} * W_{i,j}) * et_i$.

Based on the above, we can get the offline initialization by using an ILP model, and the objective is to find a schedule which satisfies the timing constraint and gives the minimum expected energy consumption $\sum_{i \in \mathcal{T}} E(\tau_i, S_i)$ [1]. For task τ_i, the worst case execution time $|S_i| \cdot et_i$ is considered in the model to prevent the application from timing violation. That is, we assume the output result cannot satisfy the quality requirement using all over-scaled voltage level until it is executed with nominal V_K. Since both the objective function and the constraints are linear, it can be formulated as an ILP problem and solved efficiently. Due to limited space, please refer to the technical report [2] for the detailed formulation.

B. Improved Initialization

Although the schedule generated by ILP model can prevent application from timing violation with minimized expected energy consumption. It may not be the best schedule for online execution. This is because in the offline stage, we assume the worst case execution scenario for each resilient task. However, in actual situation, it is very likely that the output result obtained with over-scaled voltage is acceptable, thus consume less time than the worst case situation. In these cases, the remaining idle time can be utilized at run time to save extra energy according to the execution context.

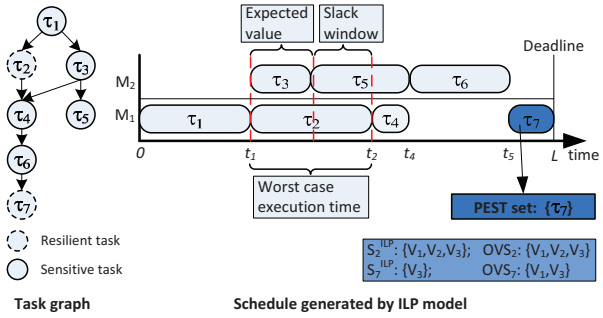

Figure 2. A motivational example.

However, some schedule generated by ILP model may cause the slack time slots unusable at runtime. As illustrated in Figure 2, ILP model generate a schedule for the resilient application (the left side of Figure 2) onto a two core voltage scalable system. Each core can run with any one of the three voltage levels including two overscaled voltage (V_1, V_2) and the nominal voltage (V_3). The application has two fault-tolerant tasks marked as dashed line as shown in the task graph. It can be observed from Figure 2, the voltage set for τ_2 is $\{V_1, V_2, V_3\}$, and worst case execution time of τ_2, that is $3 \cdot et_i$, is assumed. The voltage set for another resilient task τ_7 is $\{V_3\}$, which is not the optimal set with minimum expected energy consumption. Hence, further energy saving can be achieved at runtime if more time budget is available for τ_7, e.g., by updating voltage set to $\{V_1, V_3\}$.

At runtime, task τ_2 would be executed on M_1 following the sequence of its voltage set. In one case where τ_2 is successfully executed with voltage V_1, and τ_4 is executed in advance and finishes before t_4 resulting a time slack. However, this slack cannot be utilized for task τ_7 to update voltage set for further energy saving, since τ_7 has to wait until τ_6 finishes and cannot be executed before time t_5. However, if τ_6 is assigned to M_1, this kind of situation can be avoided and the time slack can be exploited.

For ILP model, the objective value is totally same in either of these cases (scheduling task τ_6 on M_1 or scheduling task τ_6 on M_2). It cannot differentiate those schedules and cannot make better decisions among them. Therefore, as illustrated in this simple example, by fine tuning the schedule generated with ILP model, further energy saving can be achieved at runtime. To address this problem, we propose to use a simulated annealing-based algorithm to refine the schedule generated by the ILP model. In this way, the slack time can be utilized effectively at run time.

B.1 Potential Energy Saving Tasks

To present SA-based algorithm clearly, in this subsection we define the potential energy saving tasks and the $\mathcal{PEST}$ set.

For any resilient task $\tau_i \in \mathcal{T}$, define its optimal scaling set, denoted as OVS_i, to be the scaling set that gives minimum expected energy consumption for task τ_i according to Equation (2).

Definition 2. *A task $\tau_i \in \mathcal{T}$ is a "potential energy saving task", if its scaling set obtained with ILP model, denoted as S_i^{ILP}, is different from OVS_i.*

[1] As the actual energy consumed by a resilient task is unknown until it is successfully executed. Hence, in this offline stage, the expected value is adopted to represent the long-run average energy consumption.

For a "*potential energy saving task*" τ_i, there are two points need to be noted. First, the expected energy with scaling set S_i^{ILP} from ILP is larger than that with OVS_i, according to the optimal scaling set definition. Second, the worst case execution time for τ_i with OVS_i is longer than that with S_i^{ILP}, and ILP abandoned the OVS_i because of the relatively tight time constraint. This gives us a chance on energy saving for task τ_i if there is extra time duration.

Define $\mathcal{PEST}$ to be the task set which contains all the potential energy saving tasks based on the schedule generated from ILP model. The point is to refine the schedule to make sure the runtime slack can be utilized for those tasks belong to $\mathcal{PEST}$.

B.2 Solution representation

For an application $G = \langle \mathcal{T}, \mathcal{E} \rangle$ and voltage scalable system $\mathcal{M}$, the task allocation and schedule is represented as (schedule order sequence; resource assignment sequence), denoted as (SO, RE). Schedule order sequence is the order of all tasks in $\mathcal{T}$ that conforms to the partial order designated by $\mathcal{E}$, and resource assignment sequence is the assignment for each task in the schedule order sequence.

B.3 Cost Function

To find a proper schedule which can facilitate online slack utilization, we assume each resilient task is executed with its expected execution time. By assuming that, multiple slack windows would appear as shown in Figure 2. For those tasks in $\mathcal{PEST}$ set, energy reduction can be achieved if it can utilize those slack window. For example, in Figure 2, the scaling set of task τ_4 can be updated from $\{V_3\}$ to $\{V_2, V_3\}$. The corresponding potential energy saving for τ_4 is $E(\tau_i, \{V_3\}) - E(\tau_i, \{V_2, V_3\})$.

Denote S_i^{SA} as the voltage scaling set of task τ_i under the assumption that tasks are finished with its expected execution time. The proposed simulated annealing-based algorithm assesses the quality of schedule (SO, RE) by measuring its total potential energy saving compared to the solution generated by ILP model. We design our cost function used in the algorithm as follows:

$$cost = \mu \cdot 1_{\{\exists i: f_i > L\}} - \sum_{\tau_i \in PEST} E(\tau_i, S_i^{ILP}) - E(\tau_i, S_i^{SA}) \quad (3)$$

where the first term indicates the deadline violation penalty. To be specific, μ is a sufficient large number, and $1_{\{.\}}$ is the indicator function. This function is equal to 1 if a schedule cannot meet deadline; otherwise, it is equal to 0. Thus, if a schedule violates the deadline constraint, the cost of this solution will be very large and hence be abandoned. Otherwise, the first term disappears and only the second term about potential energy saving remains.

B.4 Simulated annealing process

Given an initial solution from ILP model, the SA-based algorithm starts with a high "temperature". This temperature gradually decreases during the simulated annealing process. At each temperature Ta, a certain amount of iterations is conducted and some neighbor solutions are considered. Once we reach a new solution, its cost (denoted as $Cost_{new}$) is computed using equation 3, and compared to that of the old one (denoted as $Cost_{old}$). If $Cost_{new} < Cost_{old}$, the new solution is accepted; otherwise, the probability that the new solution is accepted is $e^{-(Cost_{new} - Cost_{old})/Ta}$. When Ta meets the predefined ending temperature, the simulated annealing process is terminated and the solution with the lowest cost obtained so far is regarded as the final solution.

V. Online Solution

Utilizing static schedule for run-time workload management shifts the burden associated with the complex task graph scheduling problem to design time. However, systems in the real-world encounter various unpredictable variations at run-time due to the varying resilience capabilities across different tasks and/or datasets. Hence, in this section, we present a lightweight run-time management scheme that provides an integrated solution to address slack reclamation, voltage scaling set updating and output quality management without diminishing the benefits of initialization generated at design time.

Algorithm V.1 Dynamic Adjustment for Slack Reclamation and Scaling Set Updating

Require: Task graph $G = \langle \mathcal{T}, \mathcal{E}, \mathcal{R} \rangle$ to be executed; (SO, RE) where SO is schedule order sequence and RE is the resource assignment sequence; scaling set and start time for each task $\tau_i \in \mathcal{T}$; $\mathcal{PEST}$: potential energy saving task set.
Ensure: Run-time schedule information for task graph G.

```
1:  while there are unscheduled tasks in the task list SO do
2:      τ_i ← pop the first task from SO.
3:      slack ← start_i − T_cu
4:      if τ_i is a sensitive task then
5:          schedule τ_i on RE_i to execute with nominal voltage.
6:      else
7:          if τ_i ∈ PEST and slack ≥ et_i then
8:              slack_times ← floor(slack/et_i).
9:              total_times ← |S_i| + slack_times
10:             S_i ← updateS(τ_i, total_times)
11:         end if
12:         Schedule task τ_i on processor RE_i to execute with scaling
            set S_i. Wake up quality checker to evaluate the output quality.
13:     end if
14: end while
```

Our run-time management scheme, Algorithm V.1, can reclaim the time slacks that become available when a resilient task finishes before its worst case finishing time as predicted in the offline stage. These slacks will be used for $\mathcal{PEST}$ by updating their suboptimal scaling sets to get extra energy saving. As the offline generated schedule includes a designated start time recorded for all task nodes, these information can help us to identify any instances of slack time. Whenever a new task is going to execute, the amount of slack time is calculated by subtracting the node's designated start time by the current time T_{cu} (Line 3). For sensitive task, it is been executed with nominal voltage (Line 5). For resilient task, its voltage scaling set would be updated if it belongs to $\mathcal{PEST}$ set and the time slack is longer than its execution time (Line 7-11), otherwise, the voltage scaling set obtained at offline would be used. Function $updateS(\tau_i, total\_times)$ in Line 10 returns τ_i's optimal scaling set with worst case execution time at most $total\_times \cdot et_i$. As the number of voltage level for the scalable system is limited, it is easy to find out this optimal set. If the time slack is not sufficient for executing task τ_i once, τ_i will start execution earlier than the designated time and thus the slack time can be passed to upcoming task. Resilient task τ_i is then scheduled on RE_i and executed following the voltage scaling set S_i (Line 12).

ApproxMap triggers the quality checker to evaluate if the output result is acceptable every time when task τ_i is executed with a overscaled voltage. The quality checker can be thought as a lightweight calibration unit with small energy consumption overhead [19] which monitors the accuracy and performance dynamically. If the quality requirement under current voltage level is satisfied, we stop the execution process and get the time slack. Otherwise, it will be re-executed with the next higher voltage level in S_i. And hence the probability to produce an unqualified result will become smaller.

TABLE I
ENERGY CONSUMPTION (IN MICROJOULE) OF RANDOM TASK GRAPHS OBTAINED WITH DIFFERENT STRATEGIES.

FR	Application	Ge_Schedule	PriS	ApproxMap			
				Energy	ILP time*	%(GS)	%(PriS)
30%	kbasic_task	2.236	2.153	1.994	0.153	10.82%	7.39%
	kseries_parallel_xover	2.414	2.299	2.128	0.052	11.85%	7.44%
	kseries_parallel	3.627	3.515	3.231	0.087	10.92%	8.08%
	Avg.	-	-	-	-	11.20%	7.63%
50%	kbasic_task	2.236	1.985	1.781	3.842	20.35%	10.28%
	kseries_parallel_xover	2.414	2.140	1.894	1.579	21.54%	11.50%
	kseries_parallel	3.627	3.247	2.871	0.873	20.84%	11.58%
	Avg.	-	-	-	-	20.91%	11.12%
70%	kbasic_task	2.236	1.834	1.567	28.513	29.92%	14.56%
	kseries_parallel_xover	2.414	2.085	1.758	8.734	27.17%	15.68%
	kseries_parallel	3.627	3.039	2.618	4.571	27.82%	13.85%
	Avg.	-	-	-	-	28.30%	14.70%

* The time needed for ILP solver is in seconds.

VI. EXPERIMENTAL RESULTS

We develop a simulator with C++ to evaluate our proposed resilient application mapping and scheduling framework, ApproxMap. The offline ILP model is run under Gurobi 5.60 [1] with CVX 2.1 in the Matlab. Experiments are conducted with a set of pseudo-random task graphs generated by TGFF 3.5 [7] using the sample input files that come with the software package and executed on a voltage scalable platform with 4 processors. Each processor has four operation voltages (1.69 V, 1.46 V, 1.38 V, 1.32 V), in which 1.69V is the environmental-margin point that the minimum voltage required to run without errors at the worst-case operating temperature of 85°C [**?**] and is regarded as the nominal operation voltage. The error rates for each resilient task under those overscaled voltage levels are uniformly distributed between [0.0001 0.1] as demonstrated in [**?**]. The parameters for the simulated annealing approach are set as follows: initial temperature = 100, cooling rate = 0.99, and ending temperature $= 10^{-5}$. We compare ApproxMap with two other strategies. The general scheduling method (*Ge_Schedule*) which executes all the tasks accurately by using nominal voltage is developed as one of the baseline to show that much energy saving can be achieved at system-level scheduling procedure by exploiting error-tolerant property. The other strategy, abbreviated as *PriS*, is a state-of-the-art method proposed in [18]. Since PriS does not consider voltage scaling for resilient tasks, we modify it for fair comparison by choosing a scaling set according to a predefined error probability threshold.

The resilience property is simulated with various task graphs with different proportions of resilient tasks, as shown in Table I. In Table I, Columns 1 and 2 indicate the percentage of resilient tasks and applications; Column "Ge_Schedule" shows the energy consumption (in microjoule, energy model in [20]) of the strategy without considering resilience property, and column "PriS" shows the energy consumption of PriS. In the last four columns, we present energy dissipations by using the proposed solution ApproxMap, the time needed for ILP model (column "ILP time") and the percentage of energy reduction. Columns "%(GS)" and "%(PriS)" show the reduced percentage of energy consumption of the the proposed ApproxMap, compared to *Ge_Schedule* and PriS, respectively.

From the experimental results, we can see that for each random generated task graph in Table I, ApproxMap reduces a significant amount of energy by selectively executing resilient tasks with overscaled voltage. Specifically, it reduces 28.30% energy in the case where 70% tasks are error-resilient, compared to the method executing all tasks precisely (Ge_Schedule). In addition, an average of 11.20% and 20.91% energy reduction is achieved for FR=30% and FR=70%, respectively, which clearly demonstrates that by taking error-resilience feature into consideration at task allocation and scheduling procedure , a substantial amount of energy can be saved effectively. Compared to PriS, ApproxMap reduces 7.63% total energy when 30% tasks are fault-tolerant. As the number of resilient operation increases, it gains more energy savings, and reduces 14.70% energy consumption in the situation where 70% tasks are error-resilient. By comparing these three sets of experiment results (FR=30%, FR=50%, FR=70%), we can find that as the number of resilient tasks increases, ApproxMap gains more energy saving by overscale voltages for those resilient tasks. Note that for random generated task graphs, PriS as well as ApproxMap guarantees the timing requirement in all cases.

The time needed for ILP solver is also presented in Table I (column "ILP time"). It can be observed that it takes less than one seconds to tens of minutes to get the optimal solution for most applications, which is obviously acceptable at offline stage.

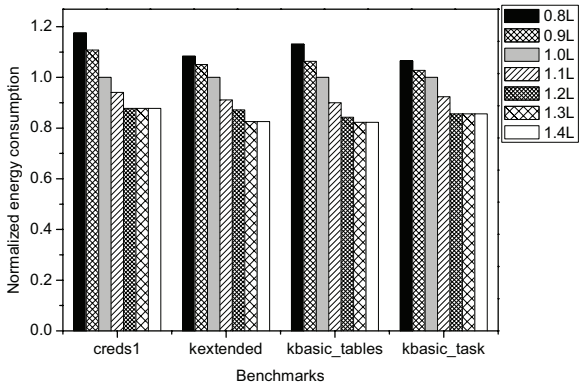

Figure 3. The reduction of energy with the relaxation of deadlines.

The online scheduler is an important component of the system for run-time information gathering and voltage scaling control. To evaluate how much energy saving can be achieved by using the online scheduler, we compare energy consumption of *ApproxMap* with that of using only offline schedule scheme (*Offline*).The experiments are conducted on six task graphs, "creds1", "kbasic_tables", "kbasic_task", "kextended", "kseries_parallel", "kseries_parallel_xover", with relaxed deadline by 0, 10 and 20 percentage, respectively. The results are shown in Table II. As is shown in the chart, when there is no deadline extension, an average of 6.95% more energy saving is

TABLE II
ENERGY CONSUMPTION (IN MICROJOULE) OF OFFLINE STRATEGY AND APPROXMAP FOR RANDOM GENERATED TASK GRAPHS.

Benchmark	L			Deadline relaxed 10%			Deadline relaxed 20%		
	Offline	ApproxMap	Δ%	Offline	ApproxMap	Δ%	Offline	ApproxMap	Δ%
creds1	718.581	685.826	4.56%	649.247	625.647	3.63%	601.349	601.587	-0.04%
kbasic_tables	97.347	91.453	6.05%	87.416	83.258	4.76%	80.169	78.596	1.96%
kbasic_task	1.662	1.567	5.72%	1.439	1.389	3.47%	1.384	1.354	2.17%
kextended	1.274	1.173	7.93%	1.003	0.957	4.59%	0.953	0.926	2.83%
kseries_parallel	2.846	2.618	8.01%	2.264	2.175	3.93%	2.049	2.047	0.10%
kseries_parallel_xover	1.941	1.758	9.43%	1.635	1.546	5.44%	1.503	1.449	3.59%
Avg.			6.95%			4.30%			1.77%

achieved which shows that further energy reduction can be achieved by utilizing time slack and updating voltage set at runtime. The online scheduler reduces by an average of 4.30% and 1.77% energy consumption at relaxed deadline of $1.1 * L$ and $1.2 * L$, respectively. Over the experimental results of the three different deadline requirements, it can be observed that the proportion of energy saving over *Offline* decreases as the process of deadline increasing. This is because, a growing number of resilient tasks can choose scaling set with more time budget, thus more flexible, which leaves small space for online voltage adjustment. It is also observed that the energy consumption obtained with ApproxMap is almost equal to the result of Offline for benchmarks "creds1" and "kseries_parallel", which indicates that the online scheduler does not update any voltage scaling set for resilient tasks, because all the resilient tasks are equipped with optimal scaling set, or the time slack is not big enough to be utilized for any resilient task in $\mathcal{PEST}$. Yet despite all that, developing effective online scheduler is necessary to harnessing the resilience trait for energy saving in all situations.

We are also interested in the trade-off between performance and energy consumption. Thus we conduct experiments for four task graphs ("creds1", "kextended", "kbasic_tables", "kbasic_task") by extending the deadline to different degrees (from $0.8 * L$ to $1.4 * L$). The results are shown in Fig. 3. We can find that the energy consumption generally decreases with the relaxation of deadlines. This is mainly because the flexibility of selecting voltage scaling set increases with respect to the deadline relaxation. Thus, more resilient tasks select better scaling set with lower expected energy consumption. This also increases the possibility of producing timing slack at runtime for even more energy savings. We can also observe that when the deadline constraint is relaxed to a certain point (e.g., deadline relaxation exceeds 120%), energy consumption reduction starts to saturate. This is because each resilient task has been executed with its optimal voltage scaling set.

VII. CONCLUSION

Many emerging applications are inherently error-resilient and hence do not require exact computation. In this work, we present ApproxMap, a hybrid online/offline task allocation and scheduling technique on homogeneous multiprocessor systems, which determines the mapping and scaling sequence of resilient tasks to minimize the energy consumption of the application while meeting its quality requirements and timing constraints. Experimental results show that ApproxMap is able to achieve significant energy savings when compared to existing techniques.

ACKNOWLEDGMENT

This work is partially supported by NSFC 61402060, NSFC 61472052, NSFC 61173014, NSFC 61432017, National 863 Programs 2013AA013202 and 2015AA015304, and Chongqing High-Tech Research Programs cstc2015jcyjA40042 and cstc2014yykfB40007, and the short-term international academic fund of Chongqing University.

REFERENCES

[1] Gurobi optimizer reference manual. http://www.gurobi.com/documentation/6.0/refman.pdf, 2014.
[2] J. Yi, Q. Zhang, Y. Tian, T. Wang, W. Liu, E. Sha, Q. Xu. ApproxMap: On Task Allocation and Scheduling for Resilient Applications. http://cacs.cqu.edu.cn/wp-content/uploads/2015/02/Tech-Rep-yijuan.pdf
[3] D. Bautista, J. Sahuquillo, H. Hassan, S. Petit, and J. Duato. A simple power-aware scheduling for multicore systems when running real-time applications. In *IPDPS* , pages 1–7, 2008.
[4] M. A. Breuer. Multi-media applications and imprecise computation. In *DSD*, pages 2–7, 2005.
[5] A. Cavelan, Y. Robert, H. Sun, and F. Vivien. Voltage Overscaling Algorithms for Energy-Efficient Workflow Computations With Timing Errors. *PhD thesis*, INRIA, 2015.
[6] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *DAC*, page 113, 2013.
[7] R. P. Dick, D. L. Rhodes, and W. Wolf. Tgff: task graphs for free. In *CODES/CASHE*, pages 97–101, 1998.
[8] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *MICRO*, pages 347–358. IEEE/ACM, 2006.
[9] G. Karakonstantis, N. Bellas, C. Antonopoulos, G. Tziantzioulis, V. Gupta, and K. Roy. Significance driven computation on next-generation unreliable platforms. In *DAC*, pages 290–291, 2011.
[10] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*, 31(4):406–471, ACM, 1999.
[11] B. Li, Y. Shan, M. Hu, Y. Wang, Y. Chen, and H. Yang. Memristor-based approximated computation. In *ISLPED*, pages 242–247, 2013.
[12] J. Park, J. H. Choi, and K. Roy. Dynamic bit-width adaptation in dct: An approach to trade off image quality and computation energy. *IEEE Transactions on Very Large Scale Integration Systems (VLSI)*, 18(5):787–793, IEEE, 2010.
[13] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. Enerj: Approximate data types for safe and general low-power computation. In *ACM SIGPLAN Notices*, volume 46, pages 164–174, ACM, 2011.
[14] Q. Zhang, Y. Tian, T. Wang, F. Yuan, Q. Xu. ApproxEigen: An Approximate Computing Technique for Large-Scale Eigen-Decomposition. In *ICCAD*, pages 824–830, 2015.
[15] Q. Zhang, F. Yuan, R. Ye, Q. Xu. Approxit: An approximate computing framework for iterative methods. In *DAC*, pages 1–6, 2014.
[16] Q. Zhang, T. Wang, Y. Tian, F. Yuan, Q. Xu. ApproxANN: an approximate computing framework for artificial neural network. In *DATE*, pages 701–706, 2015.
[17] G. Von Laszewski, L. Wang, A. J. Younge, and X. He. Power-aware scheduling of virtual machines in dvfs-enabled clusters. In *CLUSTER*, pages 1–10, 2009.
[18] Y. Zhang, X. S. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *DAC*, pages 183–188, 2002.
[19] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati and S. Mahlke Sage: Self-tuning approximation for graphics engines In *MICRO*, pages 13–24, 2013.
[20] N. Weste and D. Harris In *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th edition, Addison Wesley, March 2010.
[21] P. Dubey Recognition, mining and synthesis moves computers to the era of tera In *Technology@ Intel Magazine*, pages 1–10, 2005.