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, judicious-
ly 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 com-
putations 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 develope-
d to exploit this resilient feature for improved performance and/or
ergy efficiency gains [6, 11–16]. Techniques at the system level
include analysis and characterization of application resilience [6], dy-
namic modifying operand bit-width [12], design of lightweight quali-
ty 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 iden-
fy, 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 accu-
rate/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 giv-
ing any systematic approach for task allocation and scheduling.

On the other hand, task allocation and scheduling has been a fo-
cused research problem in the parallel processing domain for a long
time. In particular, various power-aware task allocation and schedul-
ing 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 allo-
cation and scheduling solutions (e.g., [10]) that do not exploit appli-
cation error-resilience property. The objective of the proposed solu-
tion, namely ApproxMap, is to investigate how and to what extent
energy savings can be achieved via effective and efficient task allo-
cation and scheduling, without compromising the target quality re-
quirements and timing constraints. The main contributions of our work can
be summarized as follows:

- We propose a hybrid resilient application mapping and schedul-
ing framework for voltage-scalable multiprocessor system,
  which integrates a comprehensive design-time analysis method-
ology with lightweight online adjustment strategy according to
  runtime dynamics;

- We propose to use a two stage approaches to solve the schedul-
ing problems at design time, where an integer linear program-
ming (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 di-
  minishing 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 algo-
rithms 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 con-
stitutes of a set of processor cores, denoted as $\mathcal{M}$. For each core,
the operating voltage can be scaled among a set $\mathcal{V} = \{V_1, V_2, \ldots, V_K\}$, where $V_1 < V_2 < \cdots < V_K$. $V_K$ is the nom-
inical voltage, while the other voltage level could potentially impact
the correctness of the computation (i.e., VOS without frequency scaling).
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 $G = (T, E, R)$ is a directed acyclic graph (DAG), where $T = \{t_1, t_2, ..., t_n\}$ is the set of nodes, and $E \subseteq T \times T$ is the set of edges that defines the precedence relations among the nodes in $T$. Each node $t_i \in T$ represents a task to be executed with worst-case execution time (WCET) $e_{t_i}$. Each edge $(t_i, t_j) \in E$ indicates that task $t_j$ is dependent on $t_i$. 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 $t_i$ under voltage level $V_j$ is denoted as $P_{i,j}$. The energy for executing $t_i$ at voltage $V_j$ can be expressed as $E_{i,j} = P_{i,j} \times t_i$.

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 $R = \{Result_1, Result_2, ..., Result_m\}$ 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 core correct outputs, while other sets of operands would result in 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 $t$ 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 $t$ and a specific workload as input $I$, there exists a threshold voltage $V_{th}(t, I)$: using any voltage $V$ below the threshold ($V < V_{th}(t, I)$) will lead to unacceptable results, while using any voltage above that threshold ($V \geq V_{th}(t, I)$) will always lead to a successful execution. Note that different workload as inputs for the same computation may have different threshold.

Given a task $t$, which executes under voltage $V_m$, the probability that the computation fails to meet quality requirement, denoted as $Pr_{t,m}$, is computed as $Pr_{t,m} = I'(\tau_t, V_m) \times I(\tau_t)$, where $I(\tau_t)$ denotes the set of all possible workloads for task $t$ and $I'(\tau_t, V_m)$ denotes the set of inputs for which task $t$ 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 $G = (T, E, R)$ with an associated deadline $L$, and the voltage scalable system containing a set of cores $M$, 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 lower voltage and this procedure may continue until the task is executed with nominal voltage.

![Figure 1. The proposed ApproxMap framework.](image-url)
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 \( \tau_i \) 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 \( \tau_i \) 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 \( S_i = \{V_1, V_2, \ldots, V_K\} \) selected from \( V \), where \( V_1 < V_2 < \cdots < V_m < V_K \) under which task \( \tau_i \) is going to be executed.

For instance, \( S_i = \{V_2, V_K\} \) is a valid scaling set for task \( \tau_i \) to execute. With this scaling set, \( \tau_i \) 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 \( \tau_i \) does not meet the quality requirement (i.e., fails) at voltages \( V_1, V_2, \ldots, V_{m-1} \) but succeeds at \( V_m \) [5] is:

\[
P_{\text{succ}}(i, V_m) = P_{r_i,m-1} - P_{r_i,m}, \quad \forall i \in \mathcal{T} \tag{1}
\]

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

\[
E(\tau_i, S_i) = (W_{i,1} + \sum_{j=2}^{m} P_{r_i,j-1} \cdot W_{i,j}) \cdot e \cdot t_i \tag{2}
\]

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

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 \( \tau_i \), the worst case execution time \( |S_i| \cdot e \cdot t_i \) 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.](image)

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 (\( V_K \)). 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 \( \tau_2 \) is \( \{V_1, V_2, V_3\} \), and worst case execution time of \( \tau_2 \), that is \( 3 \cdot e \cdot t_i \), is assumed. The voltage set for another resilient task \( \tau_1 \) 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 \( \tau_1 \), e.g., by updating voltage set to \( \{V_1, V_3\} \).

At runtime, task \( \tau_2 \) would be executed on \( M_1 \) following the sequence of its voltage set. In one case where \( \tau_2 \) is successfully executed with voltage \( V_1 \), and \( t_i \) is executed in advance and finishes before \( t_i \) resulting a time slack. However, this slack cannot be utilized for task \( \tau_1 \) to update voltage set for further energy saving, since \( \tau_1 \) has to wait until \( t_i \) finishes and cannot be executed before time \( t_5 \). However, if \( \tau_6 \) 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 \( \tau_6 \) on \( M_1 \) or scheduling task \( \tau_6 \) 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 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 \( \tau_i \) 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 \( \{S_{i,LP}\} \), 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” \( \tau_i \), 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 \( \tau_i \) 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 \( \tau_i \) if there is extra time duration.

Define \( PEST \) 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 \( PEST \).

### B.2 Solution representation

For an application \( G = (T, E) \) and voltage scalable system \( M \), 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 \( T \) that conforms to the partial order designated by \( E \), 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 \( PEST \) set, energy reduction can be achieved if it can utilize those slack window. For example, in Figure 2, the scaling set of task \( \tau_i \) can be updated from \( \{V_3, V_5\} \) to \( \{V_2, V_5\} \). The corresponding potential energy saving for \( \tau_i \) is \( E(\tau_i, \{V_3\}) - E(\tau_i, \{V_2\}) \).

Denote \( S_i^{SA} \) as the voltage scaling set of task \( \tau_i \) 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:

\[
\text{cost} = \mu \cdot 1_{(\text{if } f_i > L)} - \sum_{\tau_i \in PEST} E(\tau_i, S_i^{ILP}) - E(\tau_i, S_i^{SA})
\]

where the first term indicates the deadline violation penalty. To be specific, \( \mu \) is a sufficient large number, and \( 1_{(\text{if } f_i > L)} \) 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 \( T_a \), a certain amount of iterations is conducted and some neighbor solutions are considered. Once we reach a new solution, its cost (denoted as \( Cost_{\text{new}} \)) is computed using equation 3, and compared to that of the old one (denoted as \( Cost_{\text{old}} \)). If \( Cost_{\text{new}} < Cost_{\text{old}} \), the new solution is accepted; otherwise, the probability that the new solution is accepted is \( e^{-\frac{Cost_{\text{new}} - Cost_{\text{old}}}{T_a}} \). When \( T_a \) 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 \( G = (T, E, R) \) 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 \( \tau_i \in T; PEST \): potential energy saving task set.

**Ensure:** Run-time schedule information for task graph \( G \).

```plaintext
1: while there are unscheduled tasks in the task list \( SO \) do
2: \( \tau_i \leftarrow \) pop the first task from \( SO \).
3: \( \text{slack} \leftarrow \text{start}_i - T_{cu} \)
4: if \( \tau_i \) is a sensitive task then
5: \( \text{schedule } \tau_i \) on \( RE \), to execute with nominal voltage.
6: else
7: if \( \tau_i \in PEST \) and slack \( \geq e_t \), then
8: \( \text{slack times} \leftarrow \text{floor}(\text{slack}/e_t) \).
9: total_times \leftarrow |S_i| + slack_times
10: \( S_i \leftarrow \text{update} (S_i, \text{total_times}) \)
11: end if
12: end if
13: Schedule task \( \tau_i \) on processor \( RE \), to execute with scaling set \( S_i \). Wake up quality checker to evaluate the output quality.
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 \( PEST \) 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 after a designated start time recorded by the current task \( T_{cu} \) (Line 3). For sensitive task, it has been executed with nominal voltage (Line 5). For resilient task, its voltage scaling set would be updated if it belongs to \( PEST \) 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 \( \text{update} (S_i, \text{total_times}) \) in Line 10 returns \( \tau_i \)’s optimal scaling set with worst case execution time at most \( \text{total_times} \cdot e_t \). 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 \( \tau_i \) once, \( \tau_i \) will start execution earlier than the designated time and thus the slack time can be passed to upcoming task. Resilient task \( \tau_i \) is then scheduled on \( RE \), 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 \( \tau_i \) is executed with a over-scaled 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.

<table>
<thead>
<tr>
<th>FR</th>
<th>Application</th>
<th>Ge_Schedule</th>
<th>PriS</th>
<th>ApproxMap</th>
<th>Energy</th>
<th>ILP time*</th>
<th>%(GS)</th>
<th>%(PriS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30%</td>
<td>kbasic_task</td>
<td>2.236</td>
<td>2.153</td>
<td>1.994</td>
<td>0.153</td>
<td>0.087</td>
<td>10.92%</td>
<td>8.08%</td>
</tr>
<tr>
<td></td>
<td>kseries_parallel_xover</td>
<td>2.214</td>
<td>2.299</td>
<td>2.128</td>
<td>0.052</td>
<td>11.85%</td>
<td>11.44%</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>3.627</td>
<td>3.515</td>
<td>3.234</td>
<td>0.087</td>
<td>10.92%</td>
<td>8.08%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>kseries_parallel</td>
<td>2.214</td>
<td>2.140</td>
<td>1.894</td>
<td>0.759</td>
<td>21.54%</td>
<td>11.50%</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>50%</td>
<td>kbasic_task</td>
<td>2.236</td>
<td>1.985</td>
<td>1.781</td>
<td>0.842</td>
<td>20.35%</td>
<td>11.28%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>kseries_parallel_xover</td>
<td>2.214</td>
<td>2.140</td>
<td>1.894</td>
<td>0.759</td>
<td>21.54%</td>
<td>11.50%</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>3.627</td>
<td>3.247</td>
<td>2.871</td>
<td>0.875</td>
<td>20.84%</td>
<td>11.58%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>kseries_parallel</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>70%</td>
<td>kbasic_task</td>
<td>2.236</td>
<td>1.844</td>
<td>1.567</td>
<td>0.713</td>
<td>28.51%</td>
<td>14.56%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>kseries_parallel_xover</td>
<td>2.214</td>
<td>2.085</td>
<td>1.758</td>
<td>0.713</td>
<td>27.17%</td>
<td>15.58%</td>
<td></td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>3.627</td>
<td>3.039</td>
<td>2.618</td>
<td>0.713</td>
<td>27.82%</td>
<td>13.85%</td>
<td></td>
</tr>
</tbody>
</table>

* 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 5.6.0 [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 1.69 V is the environmental-margin point that the minimum voltage required to run without errors at the worst-case operating temperature of 85°C [7] 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 [7]. The parameters for the simulated annealing procedure 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 over-scaled 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 task 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
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 $1.1 \times L$ and $1.2 \times L$, 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 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 $0.8 \times L$ to $1.4 \times L$). 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 cstc2014yykjB40007, and the short-term international academic fund of Chongqing University.

REFERENCES


