EnSuRe: Energy & Accuracy Aware Fault-tolerant Scheduling on Real-time Heterogeneous Systems

Sangeet Saha[1], Adewale Adetomi[2], Xiaojun Zhai[1], Server Kasap[3], Shoaib Ehsan[1], Tughrul Arslan[2],[1]Klaus McDonald-Maier
[1]*Embedded and Intelligent Systems Laboratory, University of Essex, UK*
[2]*Ewireless Research Group, School of Engineering, University of Edinburgh, UK*
[3]*School of Computing, Electronics and Maths, Coventry University, UK*
{[1]sangeet.saha, [1]xzhai, [1]sehsan, [1]kdm }@essex.ac.uk, {[2]Adewale.Adetomi, [2]T.Arslan}@ed.ac.uk , [3]server.kasap@coventry.ac.uk

***Abstract*—This paper proposes an energy efficient real-time scheduling strategy called *EnSuRe*, which (i) executes real-time tasks on low power consuming primary processors to enhance the system accuracy by maintaining the deadline and (ii) provides reliability against a fixed number of transient faults by selectively executing backup tasks on high power consuming backup processor. Simulation results reveal that *EnSuRe* consumes nearly 25% less energy, compared to existing techniques, while satisfying the fault tolerance requirements. *EnSuRe* is also able to achieve 75% system accuracy with 50% system utilisation. Further, the obtained simulation outcomes are validated on benchmark tasks via a fault injection framework on Xilinx ZYNQ APSoC heterogeneous dual core platform.**

***Index Terms*—Heterogeneous processors, Real-time systems, Fault-tolerant scheduling, Energy efficiency**

I. Introduction

In real-time computing, correctness does not only depend on the precision of the results, but also on time at which these are produced. For such critical systems, approximated results obtained within the deadline are preferable over the accurate results generated after this deadline. In UAVs, initially an inaccurate, but acceptable quality image is generated from the received data. Then, based on the available resources and energy, the obtained image may be further refined [1]. Utilising approximate computation approaches, a real-time task consists of a mandatory part, followed by an optional part [2]. The mandatory part must be executed entirely in order to produce an acceptable result within a deadline, while the optional part will be executed for further refinement of the generated result and to provide a higher accuracy of the applications executed. However, as the mandatory parts have timing constraint, provisions must be made against faults. In order to handle these faults, typically tasks are re-executed on a backup processor to deliver the correct result [3].

In real-time scheduling, recently the authors in [4], [5], [6], have studied the combined problem of minimizing energy consumption while providing fault tolerance guarantees. However, these studies are limited to either uniprocessor systems or homogeneous multiprocessors. For heterogeneous systems, the authors in [3], [7], [8], have employed standby sparing and primary/backup techniques to provide energy aware fault tolerant solutions. However these works consider hard real-time tasks, not emerging approximation based real-time tasks. Moreover, all of these studies employ standard scheduling scheme like Earliest-Deadline- First (EDF) and Earliest-Deadline-Late (EDL) scheduling policies. The authors also made a strict assumption that all tasks share a fixed and common deadline. In modern safety critical systems, such assumptions are no longer valid, because based upon their respective criticality, individual tasks must have unique deadlines. Thus, the proposed techniques may perform poorly on multiprocessor system, where multiple tasks require to complete execution requirements within multiple deadlines.

In this paper, given a set of real-time tasks to be executed on a heterogeneous multiprocessor system, we propose *EnSuRe* which i) tolerates total k number of faults within the scheduling window ii) finishes task's execution within the deadlines and iii) enhances system accuracy while maintaining energy efficiency.

This work is supported by the UK Engineering and Physical Sciences Research Council through grants EP/R02572X/1, EP/P017487/1 EP/V000462/1 and EP/V034111/1.

II. Proposed Approach: *EnSuRe*

We consider a real-time application ($\mathcal{A}$), which consists of a set of n real-time tasks $T = \{T_1, T_2, ..., T_n\}$. Each task T_i ($1 \leq i \leq n$) has a mandatory part, with an execution requirement of M_i to be finished within the deadline, d_i and an optional part with an execution requirement of O_i.

A. Schedule generation phase

EnSuRe employs *a time-partitioning based* [9] scheduling approach for a set of n real-time tasks $\mathcal{A} = \{T_1, T_2, \ldots, T_n\}$ on the multiprocessor system. Let us consider the difference between any two consecutive deadlines (i.e. χ^{th} and $(\chi-1)^{th}$ task deadline) termed as "time window" and it can be found as:

$$TWL_\chi = d_\chi - d_{\chi-1} \tag{1}$$

For each task, the execution rate demand is denoted by its weight and can be found as: $wt_i = \frac{M_i}{d_i}$, where M_i denotes the mandatory execution time and d_i denotes the deadline. For any time-window each task will be allowed to be executed for a certain time span proportional to its weight and these spans for each individual task will be termed as "workload-quota" and can be calculated as:

$$WQu_i^{\chi} = (\lceil wt_i \times TWL_{\chi} \rceil) \quad \forall T_i \in \mathcal{A} \tag{2}$$

Within a time-window, total system-wide capacity can be calculated as: $TWL_{\chi} \times m_{pri}$, where m_{pri} is the number of available primary core. For a feasible schedule, this capacity must be equal or greater than the total workload-quota for all running tasks, i.e. $(\sum_{i=1}^{n} WQu_i^{\chi})$. Thus, the following condition must be satisfied to obtain a feasible schedule.

$$\sum_{i=1}^{n} WQu_i^{\chi} \leq TWL_{\chi} \times m_{pri} \tag{3}$$

EnSuRe will initiate the task allocation from the first primary core, as per their workload-quota. However, if the core is not completely occupied then the available slack AS_j^{χ} of the j^{th} primary core for the χ^{th} time-window is:

$$AS_j^{\chi} = TWL_{\chi} - \sum_{i=1}^{n} WQu_i^{\chi} \times \theta_{ij} \tag{4}$$

where θ_{ij} equals 1 if T_i has been assigned to j^{th} core; otherwise $\theta_{ij} = 0$.

According to our strategy, this available slack will be used for the execution of optional portion of tasks so that the system accuracy can be enhanced. In order to allocate the optional portion of tasks within a time-window, we have defined a factor called "Urgency Factor (UF)", the urgency factor (UF_i) of task T_i can thus be defined as:

$$UF_i = d_i - t^{slack} \tag{5}$$

where t^{slack} denotes the time instant where the slack time starts within a time-window.

B. Fault handling phase

After scheduling, *EnSuRe* creates a list called *"backup"* in non-increasing order of M_i^{HP}, where M_i^{HP} denotes the execution length of mandatory part of T_i on high performance backup core. As *EnSuRe* needs to handle only k number of faults, it reserves an execution slot on HP for possible backup task execution. We termed this slot as *"BES (Backup Execution Slot)"*. Then *EnSuRe* decides when to activate this *"BES"* slot inside a time-window. Thus, the *"BST (Backup Start Time)"* is calculated. If a the task is executed with zero error, then the result is committed. This in turn, removes the task from the *backup* list. Hence, as soon as a primary task completes successfully, the size of the*"BES"* slots on the HP core reduces dynamically. Algorithm 1 shows the pseudocode of *EnSuRe* .

Algorithm 1: *EnSuRe*

```
Input: Temporal parameters of tasks ∈ $\mathcal{A}$ and
       time-windows;
Output: Generate fault-tolerant schedule
for each time-window do
    /***** For primary core(s), Schedule generation
      *********/
    For each task, calculate workload-quota using
      Equation 2;
    if Equation 3 is satisfied then
        while $\mathcal{A} \neq NULL$ do
            Execute task $T_i$ in the primary core(s) as per
              workload-quota;
            Remove $T_i$ from $\mathcal{A}$ if workload-quota
              completes;
        Calculate available slack (AS) using Equation 4 for
          each core;
        Calculate $UF_j$ for each task $T_i$ using Equation 5;
        Store the $UF$ values in ascending order in set $\mathcal{U}$;
        while $AS^{\chi} \neq NULL \ OR \ \mathcal{U} \neq NULL$ do
            Execute optional portion of $T_i \in \mathcal{U}$;
        /***** For backup core, fault handling *********/
        If Tasks are schedulable then
        Create backup list in non-increasing order of $M_i^{HP}$;
        for first k tasks in backup do do
            BES = BES + $M_i^{HP}$;
        BST= $TWL_{\eta}$ - BES;
        Reserve BES unit of slots on HP from BST instant;
```

III. Experiments an Analysis

A. Performance Metrics

Normalized Energy Consumption (NEC) and *Normalized Achieved Accuracy (NAA)* have been used for evaluation. NAA is the ratio between *total executed optional portion* and *total available optional portions for all tasks*.

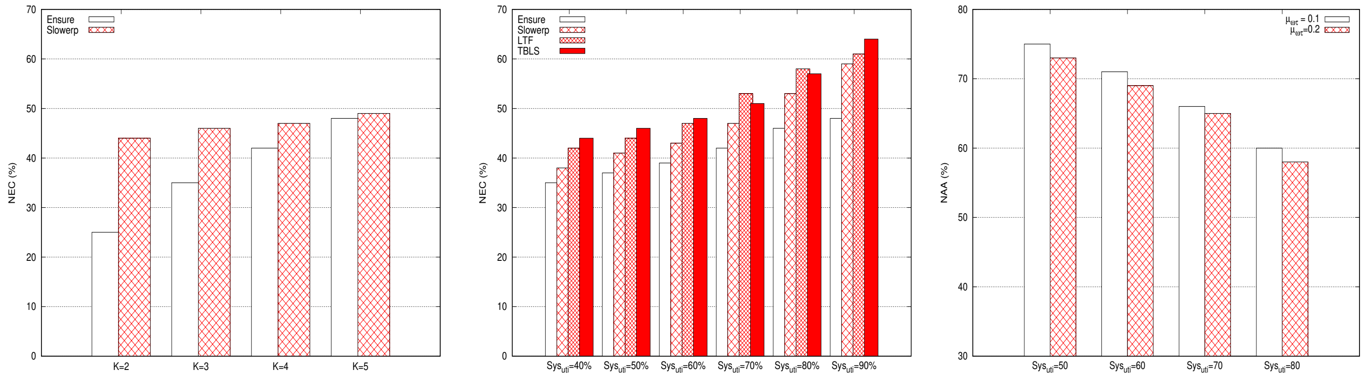
The ranges of the mandatory portion M_i and the optional portion O_i are obtained from [2]. The weights ($wt_i = \frac{M_i}{d_i}$) of the tasks are generated from normal distribution with standard deviation $\sigma_{wt} = 0.1$ and the mean values i.e. $\mu_{wt} = 0.1$, $\mu_{wt} = 0.2$. Having the tasks weights, the total workload of the system (Sys_{WL}) can be obtained by adding the weights of all tasks. Hence, the system utilisation (Sys_{uti}) can be defined as:

$$Sys_{uti} = \frac{Sys_{WL}}{m_{pri}} \times 100\% \tag{6}$$

B. Results and Analysis

1) Evaluating the impact of k: Figure 1(a) exhibits how energy consumption varies with increasing number of faults. As per the trends in Figure 1(a), it can be concluded that the higher the number of faults, the higher is the energy consumption for *EnSuRe*. However, *SlowerP* [7] consumes a fixed energy consumption. This behavior of *SlowerP* can be argued by the fact that irrespective number of faults, this strategy keeps a backup space for all tasks. In contrast, for *EnSuRe* as k increases the *BES* also increases which in turn increases overall power consumption.

2) Evaluating the impact of utilisation: Figure 1(b) shows how the energy consumption varies with respect to varying system utilisation. The number of faults set as $k = 4$. It may be observed from Figure 1(b) that with the increasing system utilisation, the energy consumption also increases for both *EnSuRe* and *SlowerP*. We have further compared *EnSuRe* with two existing strategies "LTF" and "TBLS" as proposed in [10]. "LTF" and "TBLS" gives higher priority to the tasks with higher execution length thus, in order to maintain deadline, the HP core is also used for primary execution which leads



(a) Impact of number of faults (b) Impact of system-utilisation (c) NAA (%) varying μ_{wt}

Fig. 1: Performance of EnSuRe

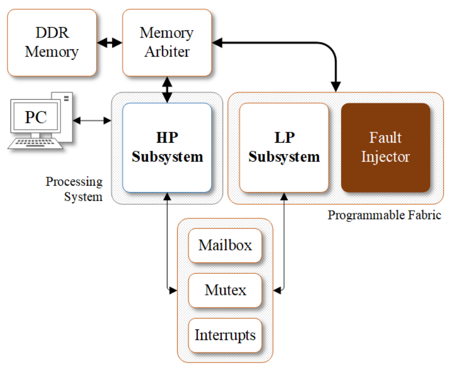


Fig. 2: The ZYNQ test-bed

to high energy consumption. It can be observed that in case of highest system utilisation (Sys_{uti}=90%), *EnSuRe* consume 25% less energy than "TBLS".

From Figure 1(c), *EnSuRe* is able to achieve 75% accuracy when Sys_{uti} is 50%. However, as the utilisation increases the slack in primary core(s) decreases and thus, NAA decreases with the increase in Sys_{uti}. It has to be noted that for a Sys_{uti}, if the μ_{wt} varies from 0.1 to 0.2, the NAA remains comparable.

IV. Hardware Implementation

A. Architectural Setup

We have implemented *EnSuRe* on a heterogeneous system on a Xilinx Zynq-7000 All-Programmable SoC [11], with Arm Cortex-A9 CPU in the Processing System (PS) side, which serves as the HP core; and FPGA fabric in the Programmable Logic (PL) side, which is used to implement the LP core and other system components. Figure 2 represents the proposed architecture. The host PC executes the *EnSuRe* algorithm.

B. Fault Injection and Detection Framework

The fault injection framework needed to confirm the integrity of the TMR MicroBlaze Subsystem relies on the *TMR Inject* IP core. Fault injection is actually carried out by injecting a different instruction at a certain instruction address of one of the three processors. This causes a mismatch among the processors and such mismatch is detected by a TMR comparator. The framework is shown in Figure 3.

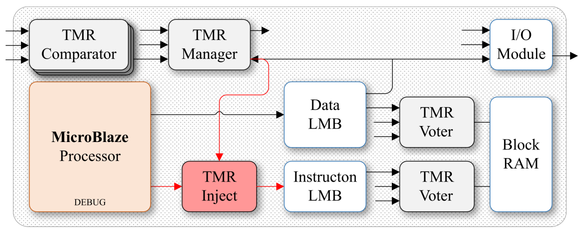


Fig. 3: Fault injection and detection

C. Energy consumption

We have created synthetic tasks from MiBench benchmark [2]. The execution times for HP core and LP core are measured for ARM core (freq: 650 MHz) and MicroBlaze core (freq: 100 MHz). We have evaluated the *EnSuRe* by injecting ($k = 3$) faults.The average scheduling length is taken as 30000 ms and we executed the simulations 5 times by injecting the faults at arbitrary positions in the scheduling length. The final value is calculated from the average of these obtained values. Table I shows the energy consumption of *EnSuRe* and SlowerP for the entire scheduling length.

TABLE I: Enrgy Consumption in Joule

Avg. number of tasks	EnSuRe	SlowerP
8	7.83	11.26
12	9.68	14.57
16	13.58	17.84

V. Conclusion

EnSuRe schedule tasks on primary core such that tasks could meet their deadlines and accuracy can be enhanced. However, the backup core is intelligently used for energy efficiency and fault tolerance. As per the obtained simulation behavior, it can be argued that *EnSuRe* can be employed for energy efficient operation and the simulation outcomes are further validated on ZYNQ APSoC heterogeneous systems with benchmark tasks.

References

[1] S. Mittal, “A survey of techniques for approximate computing,” *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, p. 62, 2016.

[2] L. Mo, A. Kritikakou, and O. Sentieys, “Approximation-aware task deployment on asymmetric multicore processors,” in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1513–1518.

[3] Y. Guo, D. Zhu, H. Aydin, J.-J. Han, and L. T. Yang, “Exploiting primary/backup mechanism for energy efficiency in dependable real-time systems,” *Journal of Systems Architecture*, vol. 78, pp. 68–80, 2017.

[4] M. A. Haque, H. Aydin, and D. Zhu, “On reliability management of energy-aware real-time systems through task replication,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 813–825, 2016.

[5] M. Fan, Q. Han, and X. Yang, “Energy minimization for on-line real-time scheduling with reliability awareness,” *Journal of Systems and Software*, vol. 127, pp. 168–176, 2017.

[6] B. Zhao, H. Aydin, and D. Zhu, “Energy management under general task-level reliability constraints,” in *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*. IEEE, 2012, pp. 285–294.

[7] A. Roy, H. Aydin, and D. Zhu, “Energy-aware standby-sparing on heterogeneous multicore systems,” in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2017, pp. 1–6.

[8] P. P. Nair, R. Devaraj, and A. Sarkar, “Fest: Fault-tolerant energy-aware scheduling on two-core heterogeneous platform,” in *2018 8th International Symposium on Embedded Computing and System Design (ISED)*. IEEE, 2018, pp. 63–68.

[9] S. Saha, X. Zhai, S. Ehsan, S. Majeed, and K. McDonald-Maier, “Rasa: Reliability-aware scheduling approach for fpga-based resilient embedded systems in extreme environments,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2021.

[10] A. Roy, H. Aydin, and D. Zhu, “Energy-efficient fault tolerance for real-time tasks with precedence constraints on heterogeneous multicore systems,” in *2019 Tenth International Green and Sustainable Computing Conference (IGSC)*. IEEE, 2019, pp. 1–8.

[11] L. Crockett, D. Northcote, C. Ramsay, F. Robinson, and R. Stewart, *Exploring Zynq MPSoC: With PYNQ and Machine Learning Applications*, 2019.