# Reliability and Energy-aware Cache Reconfiguration for Embedded Systems

Yuanwen Huang and Prabhat Mishra
Department of Computer and Information Science and Engineering
University of Florida, Gainesville FL 32611-6120, USA
{*yuanwen, prabhat*}*@cise.ufl.edu*

***Abstract*—Cache vulnerability due to soft errors is one of the reliability concerns in embedded systems. Dynamic reconfiguration techniques are widely studied for improving cache energy without considering the implications of cache vulnerability. Maintaining a useful data longer in the cache can be beneficial for energy improvement due to reduction in miss rates, however, longer data retention negatively impacts the vulnerability due to soft errors. This paper studies the trade-off between energy efficiency improvement and reduction in cache vulnerability during cache reconfiguration. We propose two heuristic approaches for reliability- and energy-aware dynamic cache reconfiguration. Experimental results demonstrate that our proposed approaches can provide drastic reduction in cache vulnerability with minor impact on energy and performance.**

## I. Introduction

Soft errors are transient faults in CMOS circuits, which are caused by energy carrying particles (cosmic rays or substrate alpha particles). These transient faults flip bits in storage cells or change the logic values in functional units. Soft error rate per chip is expected to grow due to the growing density of transistors on chip [1]. Previous studies have concluded that unprotected memory elements are the most vulnerable components to soft errors [2]. The cache in embedded microprocessors is most susceptible to soft errors for several reasons: (i) cache occupies the majority of chip area, (ii) cache has an extremely high density of transistors, and (iii) cache cell size scales down, which reduces the critical charge needed to flip a bit in stored data. Due to widespread use of embedded systems in safety-critical devices, it is necessary to protect embedded caches from soft errors.

Dynamic Cache Reconfiguration (**DCR**) is a widely studied method for optimizing energy and performance in embedded systems [3] [4]. The basic idea of cache reconfiguration is that different programs have varying data and instruction access characteristics during execution (runtime) and DCR tries to find the optimal cache configuration for a given application (program). For example, we can improve performance by increasing cache size when a program needs a lot of data accesses. Similarly, we can save energy by shutting down a part of the cache if the program is not so data-intensive. However, cache reconfiguration will also affect the vulnerability due to soft errors. A large cache size for a data-intensive program might have fewer cache misses and thus improve energy and performance efficiency, but it is also likely to increase the vulnerability of cache data because of longer data retention in the cache. This interesting trade-off between performance, energy and vulnerability is the motivation for this work.

It is a major challenge to improve the reliability of real-time embedded systems with special design considerations of real-time constrains. Hard real-time systems require that all tasks must complete execution before their deadlines to ensure correct execution. Due to stringent timing constraints, scheduling for hard real-time systems must perform task schedulability analysis based on task attributes (such as deadlines, priorities, and periods). For soft real-time systems, minor deadline misses may result in temporary service degradation, but will not lead to incorrect behavior. An efficient cache reconfiguration framework is proposed for energy optimization in soft real-time systems in [4]. They exploit the flexibility of soft real-time systems and manage to achieve considerable energy savings with minor impacts on user experiences. However their method does not consider the vulnerability of cache due to soft errors.

To the best of our knowledge, there are no prior efforts in analyzing the cache vulnerability during cache reconfiguration. We propose a methodology for using cache reconfiguration in soft real-time systems. Our approach provides an efficient cache tuning strategy based on static profiling and dynamic scheduling of tasks. Our proposed research is able to balance performance, energy consumption and vulnerability, so that tasks can meet their deadlines and energy savings while vulnerability reduction can also be achieved.

The rest of the paper is organized as follows. Section II presents related work on DCR and cache vulnerability. Section III motivates the reader by illustrating the effect of DCR on performance, energy consumption and vulnerability. Section IV presents our cache reconfiguration methodology. Section V presents the experimental results. Finally, Section VI concludes the paper.

## II. Background and Related Work

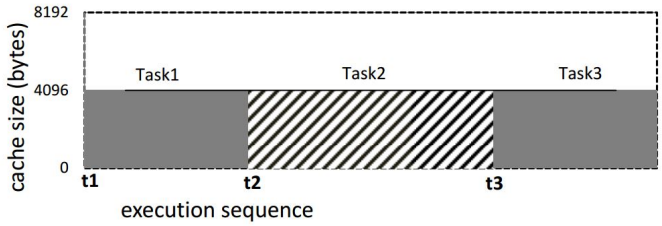
This section surveys existing works in two related domains: cache reconfiguration and cache vulnerability.

### A. Cache Reconfiguration

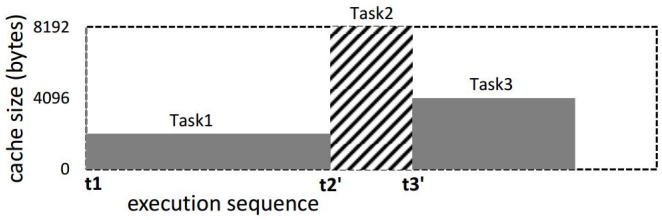
Applications have varied instruction and data access patterns, which means that they require different cache requirements in terms of cache size, line size, and associativity. In inter-task (application-based) cache reconfiguration, DCR tunes the cache when a new task starts its execution. Fig. 1

This work was partially supported by the NSF grants (CNS-1526687 and CNS-1441667) and SRC grant (2014-TS-2554).

illustrates that DCR can improve overall performance by tuning cache size for a system with three tasks. We assume that cache size is the only tunable parameter of cache for the ease of illustration (line size and associativity remain the same). Fig. 1(a) shows a traditional system using a fixed *base cache*, whereas in Fig. 1(b) each task uses its favorable cache configuration and the overall execution time is improved.




(a) A traditional system




(b) A system with reconfigurable cache

Fig. 1: DCR in a system with three tasks.

DCR has been extensively studied by previous works [3] [4] [6]. The underlying cache architecture used in our work contains four banks which can operate as four separate ways. The cache ways can be configured to shut down so as to vary the cache size. Similarly, line size can be adjusted by configuring the fetch unit to different lengths and alter way associativity by concatenating ways. There are many prior efforts in developing energy- and performance-aware cache reconfiguration techniques. Wang et al. [3] studied scheduling-aware cache reconfiguration for energy saving in real-time systems. Cai et al. [7] showed that cache size could impact performance, energy and reliability. However, none of the previous works has considered cache vulnerability improvement during DCR.

### B. Cache Vulnerability

In order to facilitate reliability analysis of cache, a measurement method is needed for the quantification of cache vulnerability due to soft errors. Mukherjee et al. [8] introduced the concept of Architectural Vulnerability Factor (AVF). Vulnerability analysis divides a bit's lifetime into vulnerable and un-vulnerable intervals. A bit is vulnerable for an interval, if soft errors that happen in this interval will cause the program to get contaminated data. Similar to [8] and [11], we measure the vulnerability of cache on a per-byte basis. Activities during the lifetime of a byte includes "*idle*", "*fill*", "*read*", "*write*" and "*eviction*". As shown in Fig. 2, the vulnerable intervals are marked by two black rectangles: the data is vulnerable between the first *write* and the second *read* as well as between the second *write* and the third *read*. During these two intervals, the data needs to be read for reuse, while a flipped bit can corrupt the data, causing the program to use corrupted cache data. While the interval between the second *read* and the second *write* is un-vulnerable, the data will be updated by the *write* operation even if soft errors corrupt it.

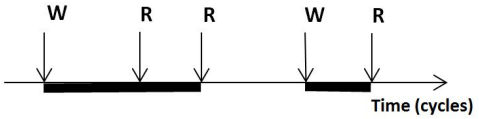



Fig. 2: Vulnerable intervals of a data element in cache (where W=Write Access, R=Read Access).

*Byte Cycles* is an widely used term for measuring cache vulnerability. We measure the vulnerability of cache as the summation of vulnerable intervals of all bytes. It can be defined as follows:

$$Vulnerability = \sum_{all\ bytes} vulnerable\ time\ of\ byte_i$$

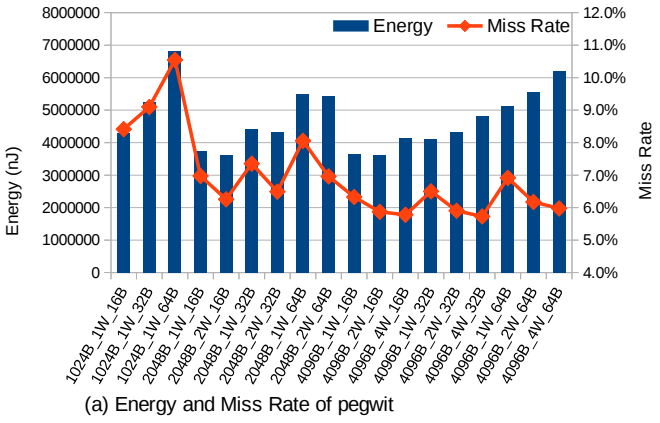
Major reliability improvement techniques include error detection and error prevention [1] [9]. Error detection techniques, such as parity caching and error-correcting codes, use spatial redundancy to detect errors. Error prevention techniques [10], such as periodic flushing and early write-back, are introduced. These hardware techniques need extra hardware support in cache, and are not sensitive to the data access pattern of the applications. In this paper, we assume no error prevention, as we aim to reduce vulnerability with the given reconfigurable cache architecture during DCR. Our goal is to take advantage of the reconfigurable cache and the data access pattern of applications to reduce vulnerability while still save energy and meet timing constraints.

## III. Motivation: Illustrative Example

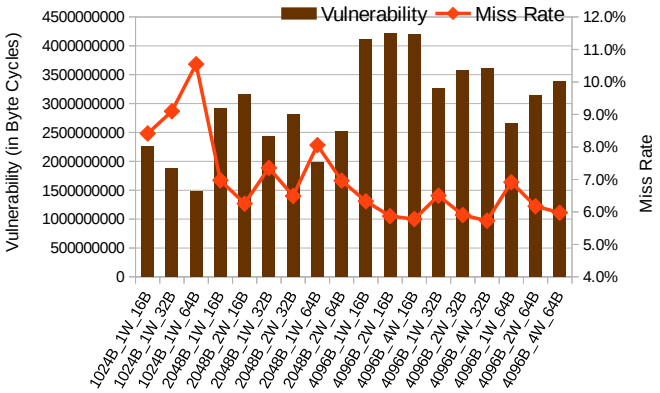
Existing techniques for cache reconfiguration do not consider cache vulnerability due to soft errors. Fig. 3 illustrates the interesting behaviors of vulnerability and energy consumption under different cache configurations. We run the program *pegwit* (a benchmark from MediaBench [12]) for 18 times, and each run uses a different configuration for L1 data cache. Each configuration consists of three parameters: cache size, associativity and line size. For example, 1024B_1W_64B implies a cache configuration with cache size of 1024 bytes, one way with 64 bytes line size.

Fig. 3 shows that the energy consumption, vulnerability and miss rate change drastically as we tune cache configurations. Both energy and vulnerability relate to cache miss rates and cache configurations. However, the correlation behaviors are quite different and even conflicting in certain scenarios. In Fig. 3(a), energy consumption decreases when miss rate decreases (the first 9 cache configurations), but keeps increasing for the last 9 cache configurations even though miss rates are fairly low. The reason is that total energy consumption is the sum of dynamic and static energy. For the first 9 cache configurations, the total energy is dominated by dynamic energy consumption, thus the total energy decreases when

miss rate (dynamic energy consumption) decreases. However, for the last 9 cache configurations with large cache size, the total energy is dominated by static energy consumption even though miss rates are low. In Fig. 3(b), the relation between vulnerability and miss rate is a little more complex. Cache size has a significant influence on vulnerability. Configurations with cache size of 1024B is much less vulnerable than configurations with cache size of 2048B and 4096B. For configurations with the same cache size, vulnerability decreases when miss rate increases and vice versa. For the same cache size, lower miss rate means that more dirty data is staying in cache for longer time, which contributes to vulnerability.




(a) Energy and Miss Rate of pegwit




(b) Vulnerability and Miss Rate of pegwit

Fig. 3: Energy (a) and vulnerability (b) values of *pegwit* benchmark using different cache configurations.

There are two interesting observations here: (i) small cache size might have high energy consumption but less vulnerable; (ii) low miss rate might be energy friendly but leads to higher vulnerability. These observations motivate us to investigate the trade-off between vulnerability, energy and performance during DCR. In this paper, we develop a cache reconfiguration framework that considers both energy and cache vulnerability. Since both vulnerability and energy depend on program characteristics and cache configurations, we statically analyze various cache configurations for each application. Such an approach is suitable for embedded systems since applications are known a priori. Based on static analysis, we propose two heuristic approaches for inter-task dynamic cache tuning that can select suitable configurations during runtime.

## IV. DCR for Energy and Reliability

### A. System Model

Let us define the reliability-aware DCR problem with consideration of both energy and cache vulnerability. The system we consider can be modeled as:

- A processor with a reconfigurable cache which supports $m$ possible cache configurations $C = \{c_1, c_2, c_3, ..., c_m\}$.
- A set of $n$ independent tasks $T = \{t_1, t_2, t_3, ..., t_n\}$.
- Each task $t_i \in T$ has attributes including arrival time, period and deadline. Non-preemptive execution is employed, which means, a task will continue execution until completion once it starts to execute.

Let $e_{t_i}^{c_j}$, $p_{t_i}^{c_j}$ and $v_{t_i}^{c_j}$ denote the energy, execution time (performance) and vulnerability of task $t_i$ when it is run on cache configuration $c_j$. The reliability-aware DCR problem is to find a cache assignment for the task set such that energy consumption and vulnerability are minimized with each of the tasks satisfying its deadline. One common practice for dealing with multi-objective optimization problem is to optimize one objective at a time while transforming other objectives into constraints. We introduce the *Vulnerability-aware Energy Optimization* (**VAEO**) problem, which aims at minimizing the total energy consumption, while adding vulnerability of tasks as constraints. A heuristic algorithm based on run-time task scheduling is proposed for solving the VAEO problem. We also introduce the *Energy-aware Vulnerability Optimization* (**EAVO**) problem, which aims at minimizing the total vulnerability while adding energy consumption constraints.

### B. Vulnerability-aware Energy Optimization (VAEO)

The VAEO DCR problem can be defined as the following:

$$\text{minimize} \sum_{i=1}^{n} e_{t_i}^{c_j} \tag{1}$$

subject to

$$v_{t_i}^{c_j} \leq V_{t_i}, \ \forall i \in [1, n] \tag{2}$$

$$a_{t_i} + w_{t_i} + p_{t_i}^{c_j} \leq D_{t_i}, \ \forall i \in [1, n] \tag{3}$$

Let $n$ represent the total number of task arrivals within the least common multiple (hyper-period[1]) of all task periods. $\sum_{i=1}^{n} e_{t_i}^{c_j}$ is the total energy consumption of $n$ tasks[2]. Equation 2 and 3 contain the vulnerability and timing constraints. $V_{t_i}$ is the upper bound for vulnerability of task $t_i$. Here $a_{t_i}$, $w_{t_i}$, $p_{t_i}^{c_j}$, $D_{t_i}$ denote the arrival time, queuing time, execution time, and deadline of task $t_i$. The optimization goal is to find a set of cache configuration assignments for all tasks so that

[1]A hyper-period is the Least Common Multiple (LCM) of all the periods in the task set. The basic idea of using hyper-period is that once we find a profitable (for energy or vulnerability) schedule for one hyper-period, the exactly same schedule can be applied to subsequent hyper-periods.

[2]It will be precise to call $n$ as the total number of "jobs" as in real-time system terminology. However, for ease of discussion, we do not distinguish between tasks and jobs.

the total energy consumption is minimized with vulnerability and timing constraints. We choose $V_{t_i}$ as the vulnerability of task $t_i$ when it is executed with the *base cache*, the most profitable cache configuration decided during design time. In other words, we set the vulnerability as a constraint to ensure that it is always at least as reliable as the *base cache*.

In Equation 3, arrival time $a_{t_i}$ and deadline $D_{t_i}$ are known upon the arrival of the task, while queuing time $w_{t_i}$ and execution time $p_{t_i}^{c_j}$ depend on the scheduling and cache reconfiguration algorithms. Queuing time $w_{t_i}$ depends on the scheduler and is determined by the priority of this task and the other tasks currently in the queue. Execution time $p_{t_i}^{c_j}$ is determined by the cache configuration $c_j$ which will be assigned to this task by the cache reconfiguration algorithm.

### C. Heuristic Approach for VAEO Problem

Tasks arrive periodically and each task is inserted into a list of ready tasks upon arrival. We propose a heuristic approach, which employs Earliest Deadline First (EDF) as our underlying scheduling algorithm. EDF fetches the task with the highest priority (earliest deadline) to execute. The cache configuration selection algorithm will pick a configuration for this task and try to satisfy Equation 2 and 3 if possible. Our heuristic approach chooses between the VAEO cache configuration and performance optimal (PO) cache configuration for this task.

- ***VAEO cache configuration*** of a task is the configuration which satisfies Equation 2 and consumes the least energy (i.e. the VAEO configuration) among all possible configurations.
- ***PO cache configuration*** of a task is the configuration which has the shortest execution time (i.e. the PO configuration), but PO configuration might not satisfy Equation 2.

The intuition behind our approach of choosing between PO and VAEO configuration are as follows:

(1) The VAEO configuration satisfies the vulnerability constraint in Equation 2 and it is most beneficial for energy savings, although it might have long execution time. We would like to always choose the VAEO configuration for energy optimization, as long as this choice would not cause the task itself or any of the subsequent tasks to violate their deadlines.

(2) The PO configuration is aimed on Equation 3 for satisfying timing constraints. If the VAEO configuration of a task causes deadline violations, we would conservatively choose the PO configuration instead. With this task running under the PO configuration, the subsequent tasks will have more slack time for scheduling and possibly save energy.

Algorithm 1 illustrates the runtime cache selection algorithm for VAEO approach. Let us assume that our system uses non-preemptive EDF scheduling for the task set. Tasks arrive periodically and currently available tasks will be put into the list of ready tasks (LRT), which is maintained as a priority queue based on the deadlines of tasks. Algorithm 1 is called when the processor is ready to execute a new task. The term $p_{t_i}^{PO}$ stands for the execution time of task $t_i$ using its PO configuration, and $p_{t_i}^{VAEO}$ stands for the execution time using its VAEO configuration.

**Algorithm 1** Cache Configuration Selection for VAEO

**Input**: List of ready tasks (LRT) and task profile table.
**Output**: VAEO or PO cache configuration.

**Step 1**: Sort all tasks in LRT by priority and fetch the task $t_c$ with highest priority.

**Step 2**: $t_1$ to $t_m$ are tasks left in LRT, from highest to lowest priority. $\tau$ represents the current time.
//***check the schedulability of each task in LRT***//
**for** $j = 1$ to $m$ do **do**
  **if** $\tau + p_{t_c}^{PO} + \sum_{i=1}^{j} p_{t_i}^{PO} > D_{t_j}$ **then**
    Discard task $t_j$
  **end if**
**end for**

**Step 3**: Select cache configuration for current task $t_c$. Let $m'$ be the number of tasks in LRT left after Step 2.
//***test the feasibility of using VAEO config for $t_c$***//
**if** $\tau + p_{t_c}^{VAEO} > D_{t_c}$ **then**
  $OK_{VAEO}$=false;
**else**
  $OK_{VAEO}$=true;
  **for** $j = 1$ to $m'$ **do**
    **if** $\tau + p_{t_c}^{VAEO} + \sum_{i=1}^{j} p_{t_i}^{PO} > D_{t_j}$ **then**
      $OK_{VAEO}$=false;
    **end if**
  **end for**
**end if**
**if** $OK_{VAEO}$==true **then**
  **return** VAEO configuration for task $t_c$
**else**
  **return** PO configuration for task $t_c$
**end if**

Step 1 fetches the current task $t_c$ to be executed, which is the highest priority task from LRT. Step 2 checks the schedulability of the tasks left in LRT, when the current task $t_c$ is executed with PO cache configuration. The schedulability of each task $t_j$ left in the LRT is checked by $\tau + p_{t_c}^{PO} + \sum_{i=1}^{j} p_{t_i}^{PO} > D_{t_j}$, which tests whether its deadline can be met with the assumption that all preceding tasks (and itself) use PO cache configurations. If $t_j$ cannot satisfy its deadline even with this conservative assumption, $t_j$ should be discarded. The discarding process is done from highest priority to lowest priority, so as to achieve fewest discarded tasks. This step ensures that all tasks in LRT will satisfy their deadlines with their PO configurations, when the current task $t_c$ is executed with its PO configuration. This step will be skipped if LRT is empty. In Step 3, we try to test the feasibility of using its VAEO configuration for the current task $t_c$, which will help improve vulnerability and energy consumption. The appropriate cache configuration for the current task $t_c$ is selected by checking whether it is safe to use its VAEO configuration. VAEO configuration is safe, only

if no tasks in the LRT will fail to meet their deadlines with their PO configurations. If the VAEO configuration is not safe for $t_c$, we will conservatively execute the current task $t_c$ with its PO configuration, which can ensure all tasks left in the LRT to satisfy their deadlines with their PO configurations (otherwise they would have already been discarded in Step 2). This algorithm runs in time of $O(m)$ where *m* is the total number of tasks in LRT.

### D. Energy-aware Vulnerability Optimization (EAVO)

Now we are ready to develop a similar heuristic approach to solve the EAVO problem, which has the total vulnerability of task set as the optimization objective and energy consumption of tasks as constraints. The EAVO DCR problem can be defined as the following:

$$\text{minimize} \sum_{i=1}^{n} v_{t_i}^{c_j} \tag{4}$$

subject to

$$e_{t_i}^{c_j} \leq E_{t_i},\ \forall i \in [1,n] \tag{5}$$

$$a_{t_i} + w_{t_i} + p_{t_i}^{c_j} \leq D_{t_i},\ \forall i \in [1,n] \tag{6}$$

$\sum_{i=1}^{n} v_{t_i}^{c_j}$ is the total vulnerability of $n$ tasks. Equation 5 contains the energy constraints and $E_{t_i}$ is the upper bound for energy consumption of task $t_i$. Equation 6 contains the timing constraints, which is the same as Equation 3. The optimization objective is to find a set of cache configuration assignments for all tasks so that the total vulnerability is minimized without violating energy and timing constraints. We choose $E_{t_i}$ as the energy consumption of task $t_i$ when it is executed with the *base cache*. In other words, we aim to minimize vulnerability of task set while ensuring that tasks can meet their deadlines and consume no more energy than the *base cache*.

Similar to VAEO, our EAVO approach will choose between the EAVO configuration and PO configuration. The cache configuration selection algorithm of our EAVO approach is similar to Algorithm 1 except that all "VAEO" phrases in Algorithm 1 need to be replaced with the phrase "EAVO". Due to space constraints, we will not duplicate the algorithm for our EAVO approach.

## V. Experiments

### A. Experimental Setup

The configurable caches used in our work are from the cache architecture introduced in [4]. The underlying cache architecture contains a configurable cache with a four-bank cache with sizes of 1 KB, 2 KB and 4 KB, line sizes of 16 bytes, 32 bytes and 64 bytes, and associativity of 1-way, 2-way and 4-way. In order to quantify reliability-aware DCR trade-off, we selected benchmarks from MediaBench [12] and EEMBC Automotive [13] benchmark suites. Table 1 shows our four task sets with three selected benchmarks in each set. All of the tasks are executed with the default input parameters provided with the benchmark suites. The *Base Cache*[3] is chosen as a 4KB, 2-way set-associative cache with line size of 32 bytes and this base configuration meets the need of tested benchmarks.

| | **Task 1** | **Task 2** | **Task 3** |
|---|---|---|---|
| **Task Set 1** | epic* | pegwit* | cjpeg* |
| **Task Set 2** | toast* | mpeg2* | dijkstra* |
| **Task Set 3** | AIFFTR01** | AIFIRF01** | A2TIME01** |
| **Task Set 4** | RSPEED01** | BITMNP01** | IDCTRN01** |

TABLE I: Task set benchmarks from **MediaBench* [12] and ***EEMBC* [13]

We modified the SimpleScalar simulator [14] for cache vulnerability analysis and energy consumption estimation. We performed the vulnerability analysis during cache accesses for each byte in instruction and data cache. The vulnerability estimation function collects all the vulnerable intervals for each valid byte in cache. We applied the same energy model as in [4] to calculate both dynamic and static energy consumption, and the energy consumption was estimated using CACTI 4.2 [15] with a 0.18 $\mu m$ technology. For static profiling of each task to find the PO, VAEO, and EAVO cache configurations, we developed Perl scripts to exhaustively search the design space of all possible cache configurations. Since we only consider systems with one level of reconfigurable cache architecture, the space of possible cache configurations is small. The statistics for all possible cache configurations for a task can be collected in a reasonable time (a few hours). Once we have the profile tables for all the tasks, we use an EDF scheduler to simulate the system for a hyper-period. The cache selection algorithms are integrated in the scheduler to make decisions to reconfigure the cache during simulation. The optimization for instruction cache and data cache are independent.

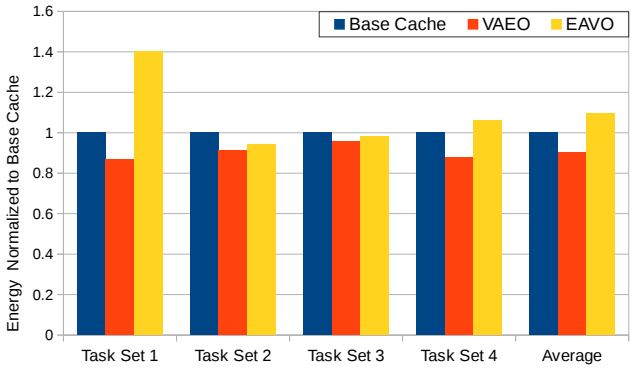
### B. Results and Analysis

Using the methodology described in Section IV, we apply our VAEO and EAVO approaches on each task set. Fig. 4 and Fig. 5 show results for instruction cache and data cache, respectively. VAEO approach can improve both energy and vulnerability while EAVO approach can significantly reduce vulnerability with minor impact on energy consumption. There are a few interesting observations on our results:

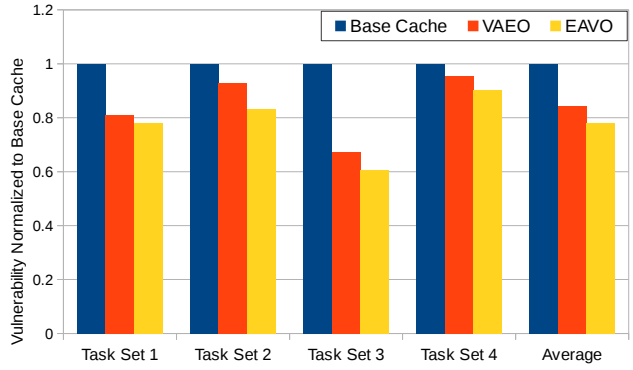
(1) As expected, VAEO approach always consumes less energy, but has higher vulnerability than EAVO approach. This is aligned with our optimization goals: VAEO is for energy optimization and EAVO is for vulnerability optimization.

(2) VAEO approach improves vulnerability as well as energy consumption. As we mentioned earlier, the *Base Cache* is selected as 4096B_2W_32B because this configuration will ensure all tasks to meet their deadlines. A large cache size of 4096B can have high performance but also cause high vulnerability. So *Base Cache* is among the cache configurations with

[3]***Base Cache*** refers to the cache used in typical real-time systems, which is chosen to ensure durable task schedules. Typically, *base cache* is the globally optimal cache configuration determined during design time for a set of tasks.

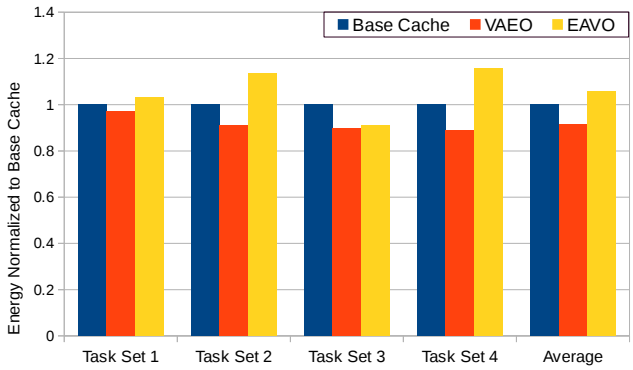



(a) Energy Consumption of Instruction Cache

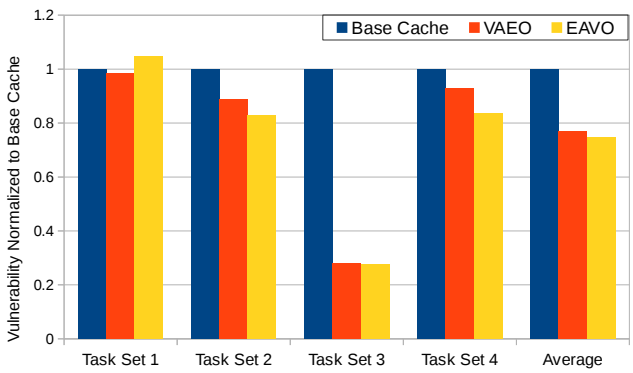



(b) Vulnerability of Instruciton Cache

Fig. 4: Instruction cache energy and vulnerability. *VAEO can improve both energy (9.6%) and vulnerability (15.8%). EAVO can significantly reduce vulnerability (21.9%) with minor impact on energy consumption (9.7%).*




(a) Energy Consumption of Data Cache




(b) Vulnerability of Data Cache

Fig. 5: Data cache energy and vulnerability. *VAEO can improve both energy (8%) and vulnerability (23%). EAVO can significantly reduce vulnerability (25.4%) with minor impact on energy consumption (5.9%)*

high vulnerability and leaves plenty of room for vulnerability reduction.

(3) But EAVO approach may increase the energy consumption. As shown in Section III, small cache size and large miss rate tend to have low vulnerability, while it usually comes with a cost of inferior performance and high energy consumption. Moreover, when a task uses its EAVO cache configuration has extremely bad performance, it may force the subsequent tasks to choose PO cache configurations in order to meet their deadlines and consume a lot more energy for these tasks.

(4) For data cache optimization of Task Set 1, EAVO approach gains neither energy nor vulnerability improvement. That is because EAVO cache configurations for some tasks have extremely bad performance, which forces too many subsequent tasks to choose PO cache configurations and increase the vulnerability for the task set.

## VI. Conclusions

Dynamic cache reconfiguration is widely used for improving energy and performance in embedded systems. While cache vulnerability is a well studied area, previous research efforts did not explore cache vulnerability in the context of cache reconfiguration. In this paper, we developed algorithms to reduce cache vulnerability due to soft errors with energy and performance considerations. Our experimental results demonstrated that our approach can significantly improve the reliability of both instruction and data caches.

## References

[1] V. Sridharan and D. Liberty. A Study of DRAM Failures in the Field. HPCNSA(SC), 2012.
[2] R. Jeyapaul and A. Shrivastava. Smart Cache Cleaning: Energy efficient vulnerability reduction in embedded processors. CASES, 2011.
[3] W. Wang et al. Dynamic Cache Reconfiguration for Soft Real-Time Systems. TECS, 2012.
[4] W. Wang et al. Dynamic Reconfiguration in Real-Time Systems - Energy, Performance, Reliability and Thermal Perspectives. Springer, 2012.
[5] C. Ekelin. Clairvoyant Non-Preemptive EDF Scheduling. ECRTS, 2006.
[6] P. Hsu and T. Hwang. Thread-criticality aware dynamic cache reconfiguration in multi-core system, ICCAD, 2013.
[7] Y. Cai et al. Cache size selection for performance, energy and reliability of time-constrained systems. ASP-DAC, 2006.
[8] S. Mukherjee et al. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. MICRO, 2003.
[9] V. Sridharan et al. Reducing data cache susceptibility to soft errors. TDSC, 2006.
[10] G. H. Asadi et al. Balancing performance and reliability in the memory hierarchy. ISPASS, 2005.
[11] A. Biswas et al. Computing architectural vulnerability factors for address-based structures. ISCA, 2005.
[12] C. Lee et al. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. MICRO, 1997.
[13] http://www.eembc.org. EEMBC, The Embedded Microprocessor Benchmark Consortium.
[14] http://www.simplescalar.com. The SimpleScalar Simulator.
[15] http://www.hpl.hp.com/research/cacti/. CACTI.