Soft Error-Aware Architectural Exploration for Designing Reliability Adaptive Cache Hierarchies in Multi-Cores

Arun Subramaniyan[3], Semeen Rehman[2], Muhammad Shafique[1], Akash Kumar[2], Jörg Henkel[4]
[1]Institute of Computer Engineering, Vienna University of Technology (TU Wien), Austria
[2]Chair for Processor Design, TU Dresden, Germany
[2]University of Michigan-Ann Arbor, USA
[2]Chair for Embedded Systems, Karlsruhe Institute of Technology, Germany
Corresponding Authors: arunsub@umich.edu, muhammad.shafique@tuwien.ac.at, semeen.rehman@tu-dresden.de

Abstract— Mainstream multi-core processors employ large multi-level on-chip caches making them highly susceptible to soft errors. We demonstrate that designing a reliable cache hierarchy requires understanding the *vulnerability interdependencies across different cache levels*. This involves vulnerability analyses depending upon the parameters of different cache levels (partition size, line size, etc.) and the corresponding cache access patterns for different applications. This paper presents a novel soft error-aware cache architectural space exploration methodology and vulnerability analysis of multi-level caches considering their vulnerability interdependencies. Our technique significantly reduces exploration time while providing reliability-efficient cache configurations. We also show applicability/benefits for ECC-protected caches under multi-bit fault scenarios.

I. Introduction and Related Works

Soft errors are transient bit-flip faults in hardware that may lead to program failures and output data errors [1]. In particular, due to their transistor sizing, their ability to latch these bit-flips and their large footprint, multi-level caches are one of the most vulnerable components in multi-cores [3]. Besides the increasing system soft error rate (SER) trends due to high transistor integration [2], recent studies have also demonstrated the occurrence of frequent multi-bit errors in memory cells [5].

Soft errors in the private L1 caches of different cores may quickly propagate to different CPU components due to their close proximity and can corrupt the program state [4]. In contrast, errors in the shared L2 or last-level caches may exhibit more error masking potential but can have a severe effect across multiple applications, especially in case of shared data. Several offline analytical models have been developed to estimate the last-level cache vulnerability to both single and multi-bit errors [24][25]. Many efforts have also focused on soft error analysis, modeling and mitigation techniques for L1 or L2 caches *individually* [3]-[5], [18]. Furthermore, parameters of different cache levels (partition size, line size, and associativity) can also significantly impact the hit/miss ratio for different applications depending on their cache access patterns.

In this paper, we demonstrate that these varying cache access patterns also lead to varying vulnerability in different cache levels and their vulnerability interdependencies. Therefore, to design a vulnerability-minimizing and reliability-adaptive cache hierarchy, a soft error-aware cache architectural space exploration methodology is required. Such a methodology needs to find a good set of reliability-wise efficient cache configurations to facilitate runtime adaptation. However, the proliferation of on-chip cores (and private L1 caches), increasing last-level cache sizes and competing multi-threaded applications with data sharing, have also increased the complexity of the architectural space exploration. In such scenarios, *there exists a need for strategies to quickly explore the design space and identify reliability-wise suitable cache configurations*.

A. State-of-the-Art Cache Space Exploration and Limitations

Cache architectural design space exploration strategies typically employ configurable caches as their baseline. Several exploration strategies have been proposed with the aim of maximizing performance *or* minimizing energy consumption [6][7][8], but ignore the reliability aspects. For instance, the work in [9] identifies the sensitivity of energy consumption to different L1 cache design parameters and the work in [10] proposes an analytical model to estimate cache hit/miss rates for multiple L1 cache configurations in a single simulation pass. However, besides ignoring the reliability optimization, these prior works have primarily focused on a single cache level [9] [10] and have not analyzed the interaction between concurrently executing applications in case of multi-cores [11]. Besides that, state-of-the-art (even including the two-level cache exploration techniques [8]) mainly considered single-threaded applications without data sharing *or* did not fully explore all parameters of the cache hierarchy when core interactions were considered [12]. However, the *vulnerability of applications to soft errors is also highly dependent on parameters of the cache hierarchy as we will demonstrate in this work.* Although reliability-aware resizing of L1 cache has been explored in [13], this work does not extensively explore the reliability impact and optimization of other cache parameters, L2 cache, and inter-dependencies across different cache levels.

In summary, *in order to design reliable cache hierarchies, vulnerability interdependencies across different cache levels and synergistic reliability optimization for the complete cache hierarchy (i.e., considering multiple cache levels) need to be explored.*

B. Novel Contributions

In order to explore the potential for reliability optimization in a multi-core cache hierarchy and exploit this – so far – unused optimization potential, we propose the following novel contributions:

1) **A Comprehensive Analysis of Vulnerability Interdependencies across Different Cache Levels** for different applications considering single- and multi-threaded workloads and data sharing. Our analysis in Section III illustrates that L1 and L2 cache parameters can have a significant impact on the vulnerability of different applications. Furthermore, the analysis of vulnerability interdependencies in a multi-level cache hierarchy illustrates that changing the L1 cache configuration for an application affects the vulnerability and access patterns to the L2 cache. This analysis is leveraged for designing a reliable cache hierarchy and its architectural exploration.
2) **A Soft Error-Aware Cache Architectural Space Exploration Methodology** that employs an efficient heuristic for selecting an appropriate set of reliability-wise beneficial configurations for the complete cache hierarchy in multi-cores. It prunes the design space of cache configurations while considering the vulnerability effects of different levels of caches on each other. The pruned design space enables an identification of vulnerability minimizing cache configurations, while considering their performance/energy properties.

To show the benefits of the proposed approach, we perform an extensive design space exploration and comparison of the cache configurations selected by our methodology with a baseline non-reconfigurable cache augmented with the vulnerability minimizing *Early Writeback* [22] technique and state-of-the-art reliability-aware reconfigurable last-level cache [18]. *When applied to SECDED-ECC protected caches, the proposed approach reduces multi-bit failures by 46%.*

II. Cache Architecture and Vulnerability Models

A. Cache Architecture and Reconfiguration Model

We adopt the configurable cache substrate based on the well-established architectures like [9][14], Motorola M*Core [20], and Intel Sandy Bridge prototype [21]. The following architectural extensions for

cache reconfiguration are employed: (1) *way-concatenation* to reduce cache associativity, without affecting its size; (2) *way-shutdown* to reduce cache size; and (3) *line resizing* by fetching multiple cache lines on an access. The reconfiguration mechanism does not lie on the critical path and requires minimal hardware extensions. For the shared L2 cache, we adopt a way-based cache partitioning approach (like [15][16]) and extended the traditional LRU replacement policy to support selection of eviction candidates from only within a designated partition. Without the loss of generality, in this paper, we adopt two levels of caches in a multi-core processor. Each core has a private L1 instruction and data cache. There exists a shared L2 cache partitioned among different applications. Fig. 1 presents the architectural overview of a multi-level cache architecture with the support for parameter reconfiguration and cache partitioning.

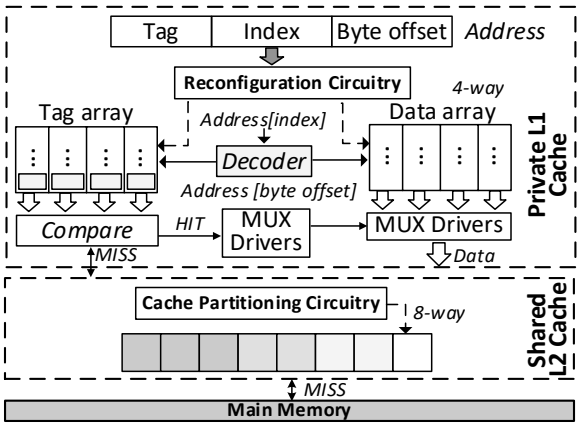

Fig. 1 Architectural overview of an adaptive multi-level cache.

B. Cache Vulnerability Modeling

Cache vulnerability estimation models have been proposed in [3][4] that primarily target temporal vulnerability (i.e., error probability due to the time a cache line is vulnerable to soft errors) but *do not (precisely) capture the spatial vulnerability* (i.e., error probability due to active cache regions and vulnerable bits in a cache line) for different applications. In this paper, we employ the advanced cache vulnerability model of [18], which jointly accounts for both spatial and temporal vulnerabilities. An extensive comparison of the model to existing models and its accuracy is described in [28]. In the following, we briefly explain this cache vulnerability model to understand the novel contributions of this paper and for reproducibility of the results.

Cache Vulnerability Components: Fig. 2 illustrates different vulnerability components in cache accesses, as defined below.

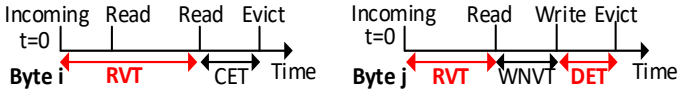

Fig. 2 Vulnerability components in cache accesses.

1) *Read Vulnerable Time (RVT)* denotes the interval between a read and previous access to the same cache byte.
2) *Write Non-Vulnerable Time (WNVT)* denotes the interval between a write and previous access to the same cache byte.
3) *Clean Eviction Time (CET)* denotes the interval between eviction and previous read (or incoming, if not accessed) of the same cache block. It is non-vulnerable.
4) *Dirty Eviction Time (DET)* denotes the interval between eviction and previous access of a dirty cache block. It is vulnerable.

Cache Vulnerability Models: In order to quantitatively estimate the vulnerability of a cache level in configuration C_i, running phase P_j of an application (i.e., parts of an application having the same behavior), two metrics are utilized: *Cache Vulnerability Index CVI(P_j,C_i)* (see Eq. 1) and *Cache Criticality Factor CCF(P_j,C_i)* (see Eq. 3). *CVI* includes *VulPeriod*$_l$*(P_j,C_i)* to capture the temporal vulnerability of the cache line w.r.t. the program state, and *VulBits*$_l$ (i.e., the bits required for architecturally correct execution), to capture its spatial vulnerability.

$$CVI(P_j, C_i) = \frac{\sum_{\forall l \in L} VulBits_l \times VulPeriod_l(P_j, C_i)}{\sum_{\forall l \in L} Bits_l \times ExecTime(P_j)} \quad (1)$$

CCF(P_j,C_i) of an application in phase P_j with cache configuration C_i is defined as the product of *CVI(P_j,C_i)* and the fraction of the total cache area used by phase P_j of the application weighted with the error probability $P_{error}(fr)$. L is the set of all cache lines.

$$cA_{(P_j,C_i)} = LA \times N_{AL}/N_L \quad (2)$$

$$CCF(P_j, C_i) = CVI(P_j, C_i) \times cA_{(P_j,C_i)} \times P_{error}(fr) \quad (3)$$

cA$_{(Pj,Ci)}$ refers to the fraction of the total cache area used by the application in phase P_j. It considers the number of cache lines N_{AL} used in phase P_j. *Bits*$_l$ and N_L refer to the total number of bits in a cache line and the total number of cache lines, respectively. *ExecTime(P_j)* refers to the execution time of the phase P_j. *LA* is the cache line area. *CCF* jointly accounts for both the spatial and temporal vulnerability of the cache. For more details on computing *VulPeriod*$_l$*(P_j,C_i)* and *cA*, refer to [18][28].

Fault Modeling and Injection: The raw SRAM soft error rate (*fr*) is determined based on the technology node, particle flux rate at a given altitude, operating voltage, and cache circuit design. We inject a number of single- and multi-bit faults at random bit positions of cache lines at different cache levels, and observe the fault propagation effects. Only spatial multi-bit faults are considered in this work, since the failure rates due to spatial multi-bit faults are shown to be up to 8 orders of magnitude greater than temporal multi-bit faults for a conservative cache lifetime of 100 years [5]. We assume the N x 1 multi-bit fault mode geometry which affects bits stored on a word line. This mode has been shown to be the most dominant and requires the greatest degree of ECC protection [2]. A conventional block-level interleaved SECDED scheme is used to demonstrate the applicability of the proposed approach to ECC protected caches. The reliability gains provided by word level SECDED schemes do not justify their area costs [24]. Let N_{FI} denote the total number of fault injection experiments, $N_{failure}$ be the number of program failures and p_{flip} be the probability that a particle strike leads to a bit-flip. Using this information, the probability of failure at a given fault rate *fr*, $P_{error}(fr)$ can be calculated as in Eq 4. Note that $P_{error}(fr)$ is also phase and configuration dependent. *CCF* directly relates to the mean-time-to-failure (*MTTF*) of the cache to soft errors as described in [17]. Optimizations that reduce *CCF* contribute to improving *MTTF*.

$$P_{error}(fr) = p_{flip} \times \frac{N_{failure}(fr)}{N_{FI}(fr)} \quad (4)$$

III. Vulnerability Analysis for Multi-Level Caches

For this motivational analysis, we used different applications from the *PARSEC* and *MiBench* benchmark suites (details in Section V). We perform two different types of analysis, as discussed below.

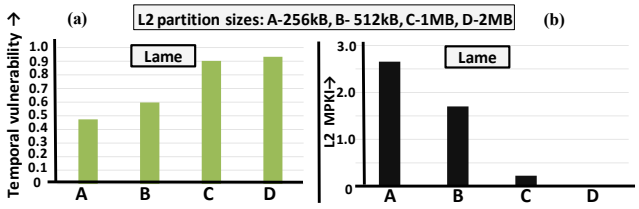

Fig. 3 (a) L2 cache vulnerability across different partition sizes-Lame; (b) L2 cache MPKI across different partition sizes-Lame.

Analysis 1 – Impact of Different L1 and L2 Cache Parameters on Vulnerability: Fig. 3 shows that there exists a tradeoff between L2 cache vulnerability and cache performance, measured in terms of Misses-Per-Kilo-Instructions (MPKI) for the *Lame* application using different L2 partition configurations. This can be attributed to the greater number of read hits for *Lame* in larger L2 cache partitions. This increases the likelihood of error propagation towards the CPU. Similar observations for L1 cache can also be made for *LU_nc* and *Lame* in Fig. 4 .

Observation-1: Parameter adaptation for L1 and L2 cache and L2 partitioning may be leveraged to increase the reliability of the cache

hierarchy to soft errors. We refer to this as reliability-aware cache hierarchy adaptation.

Analysis 2 – Vulnerability Interdependencies for Different L1 and L2 Cache Configurations: Fig. 4 illustrates that, since different applications have diverse access patterns and working set requirements, there exists a significant variation between their utilization of different cache levels that leads to different temporal vulnerabilities.

Observation-2: For a given configuration, different applications show varying vulnerability for L1 and L2 caches.

For the **L1 instruction cache** which is read-only, increase in size results in greater number of read hits but greater likelihood of error propagation for all the chosen applications. Increase in associativity and line sizes, also has a similar effect. For the **L1 data cache**, two distinct access patterns are observed: applications for which increasing cache sizes results in: a) increase in vulnerability (e.g., *LU_nc*) and b) decrease in vulnerability (e.g., *Lame*). For *LU_nc*, the cache stores are distributed across multiple cache lines, lowering the likelihood of overwriting faulty data. *Lame* on the other hand, shows high reuse of cache blocks and many cache stores that overwrite faulty data. These effects make large data caches (e.g., configuration B) have lesser temporal vulnerability.

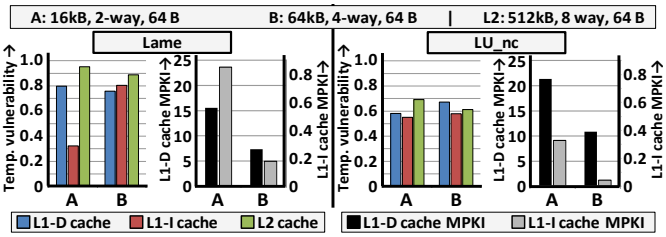

Fig. 4 L1 data and instruction cache vulnerability and MPKI across different configurations for two applications and their impact on the L2 cache vulnerability.

Moreover, Fig. 4 shows that the L1 instruction and data cache configurations can also impact the vulnerability of the L2 cache. For instance, increasing L1 cache sizes results in reduced vulnerability for both *LU_nc* and *Lame*. In particular, *LU_nc* shows close to 12% reduction in L2 cache vulnerability when larger L1 caches are used. These can be attributed to the increased number of hits in the upper cache levels.

Observation-3: In order to realize a reliable cache hierarchy, a selection of an appropriate set of configurations for different cache levels needs to account for their vulnerability effects on each other.

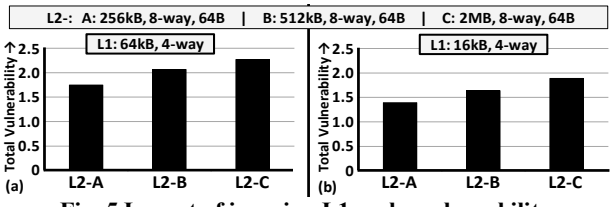

Fig. 5 Impact of ignoring L1 cache vulnerability.

Fig. 5 shows that neglecting the L1 cache can lead to sub-optimal configurations. In [18], the L2 cache is partitioned to maximize reliability, while the L1 cache has a fixed configuration as shown in Fig. 5(a). When the L1 cache is adapted as shown in Fig. 5(b), close to 21% lower cache hierarchy vulnerability is achieved. This shows that a fixed L1 configuration and ignoring interdependencies between multiple cache levels can lead to substantially higher overall vulnerability.

IV. Soft Error-Aware Architectural Exploration For Designing Reliable Cache Hierarchies

A. Architectural Methodology Overview

Fig. 6 shows the overview of our soft error-aware cache exploration methodology. Based on the processor configuration (i.e., number of cores, cache levels) and the possible cache configurations (i.e., partition/line size and associativity), the applications are analyzed for their performance and vulnerability. This is afterwards used to select the final cache configuration using the exploration heuristic (Section IV.B).

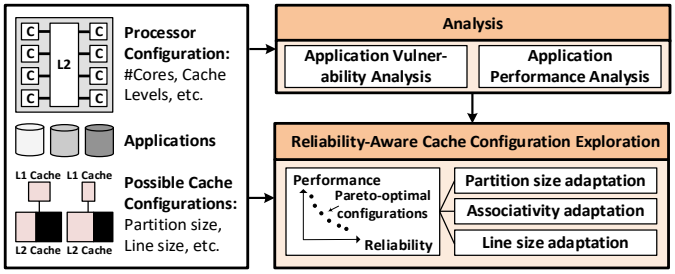

Fig. 6 Cache architectural space exploration methodology.

There are two main challenges to be addressed by our methodology: (1) large cache architectural design spaces that increase exponentially with the number of cores in the system; and (2) multi-threaded applications with data sharing that require coordinated tuning between cores and cache levels. These challenges are explained in detail below:

Challenge 1 – Large Architectural Design Space: Consider a 4-core system with private L1 instruction and data caches and shared L2 cache and a reconfigurable multi-level cache substrate with three possible size configurations, three associativity configurations and two common line size configurations (32B and 64B). There are 18 possible configurations for the private instruction and data cache of each core, leading to a total of 162 cache configurations per core. Note, that the number of configurations is not 324 = (18×18) since we consider a common line size for both instruction and data caches. For our 4-core system, this leads to 162^4 (> 600 million) configurations for the L1 cache alone. When the L2 cache is also considered, the configuration space grows to more than 5 billion points. Exhaustive exploration and finding an optimal solution of such a large design space is infeasible, especially for large applications and many cores. *Therefore, a fast exploration heuristic is required.*

Challenge 2 – Consideration of Multi-Threaded Applications: Let us consider another scenario where each core executes an independent application task. In that case, reliability-aware parameter adaptation/selection can be performed separately for the private L1 cache and L2 partition of the application. However, in case of multi-threaded workloads with dependent tasks and data sharing (via the shared L2 cache), it is important to explore vulnerability interdependencies between multiple cache levels to jointly identify the configurations of all active cores. For data-sharing applications it is also important to note that performance degrades when only private L2 partitions are available. For such applications, a shared L2 partition may be designated after identifying the sharing behavior at design-time. Coherence misses are used to determine the degree of data sharing between threads (similar to [12]). A higher number of coherence misses indicates greater degree of data sharing between threads. *Therefore, an optimization that considers vulnerability interdependencies and shared partitions for multi-threaded workloads is required.*

To address the above challenges, we propose a novel heuristic that quickly explores the architectural configuration space and identifies a reliability-wise efficient cache hierarchy configuration. Our heuristic prunes the configuration space at design time and finds the *Pareto-frontier* (i.e., Pareto-optimal configuration points w.r.t. vulnerability and performance). These Pareto-optimal configurations can enable a run-time system to perform reliability-aware cache reconfiguration for different user-provided performance constraints; the run-time aspects are out of the scope of this paper, and left as a future work. Vulnerability of the cache configuration is also dependent on the data set chosen. For both profiling and performance analysis, the design time step can be configured to consider the average of different types and sizes of input. The history based online vulnerability prediction approach proposed in [18] can be employed to account for run-time variations.

Algorithm 1: Reliability-Aware Cache Hierarchy Adaptation

1. **INPUT:** Set of 'N' concurrently executing application threads T = {T_1, $T_2 \ldots T_N$}, set of instruction, data and L2 cache configurations C_{cache} = {S_{cache}, A_{cache}, L_{cache}} where *cache* ∈ *{ICache, DCache, L2Cache}*
2. **OUTPUT:** Mapping function of application set T to minimum vulnerability cache hierarchy configuration TC: T→ {C_{opt} ={$C_{ICache,min}$, $C_{Dcache,min}$, $C_{L2Cache,min}$}}
BEGIN
Step 1: Application profiling
1. $Group\ T_i, i \in [1, N]$ *based on cache access pattern*
2. ***if*** $num\_coherence\_misses > data\_sharing\_threshold$ ***then***
3. $T.isdataSharing \leftarrow true$ *//coordinate tuning between cores*
4. ***end if***
5. ***for*** t ∈ T **do**
6. $CV_{min}^t \leftarrow \infty$ *//initialize minimum cache vulnerability*
Step 2: Cache (partition) size adaptation
7. $C_{ICache}.S_{min}^t, CV_{min}^t \leftarrow adaptParameter(T_i, size, ICache, CV_{min})$
8. $C_{DCache}.S_{min}^t, CV_{min}^t \leftarrow adaptParameter(T_i, size, DCache, CV_{min})$
9. $C_{L2Cache}.S_{min}^t, CV_{min}^t \leftarrow adaptParameter(T_i, size, L2Cache, CV_{min})$
Step 3: Line size adaptation
10. $C_{ICache}.L_{min}^t, CV_{min}^t \leftarrow adaptParameter(T_i, line\_s, ICache, CV_{min})$
11. $C_{DCache}.L_{min}^t, CV_{min}^t \leftarrow adaptParameter(T_i, line\_s, DCache, CV_{min})$
12. $C_{L2Cache}.L_{min}^t, CV_{min}^t \leftarrow adaptParameter(T_i, line\_s, L2Cache, CV_{min})$
Associativity adaptation
14. $C_{ICache}.A_{min}^t, CV_{min}^t \leftarrow adaptParameter(T_i, assoc, ICache, CV_{min})$
15. $C_{DCache}.A_{min}^t, CV_{min}^t \leftarrow adaptParameter(T_i, assoc, DCache, CV_{min})$
16. $C_{L2Cache}.A_{min}^t, CV_{min}^t \leftarrow adaptParameter(T_i, assoc, L2Cache, CV_{min})$
17. ***end for***
END

B. Heuristic for Configuration Exploration

Our heuristic identifies a set of Pareto-optimal configurations to curtail the optimization space. Configurations that do not lie on the Pareto-frontier (e.g., the ones which are both performance-wise and reliability-wise worse) are pruned at this step. Our analysis in Fig. 7 demonstrates that the vulnerability of applications shows greater sensitivity to changes in cache (partition) size, when compared to line size and associativity for all cache levels. For example, it can be seen from Fig. 7 that the instruction cache vulnerability changes by 87%, while the change is less than 25% for line size and associativity for *Lame*. Although line size can also impact the vulnerability of I-cache and D-cache for *Bodytrack* and *Cholesky*, its effect is comparable to that of size adaptation. For the L2 cache it can be seen that size adaptation shows the greatest change in vulnerability for all applications. Therefore, the heuristic aims at first identifying reliability-wise optimal cache (partition) sizes before proceeding to adapt the other cache parameters. Algorithm 1 shows the operation of our heuristic.

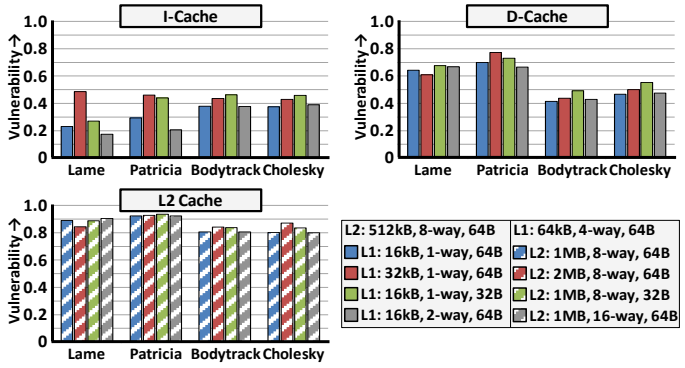

Fig. 7 Vulnerability sensitivity to different parameters.

The inputs to Algorithm 1 include the set of concurrently executing application threads, the set of multi-level cache configurations with each cache configuration $C_{cache,i}$={$S_{cache,i}$, $A_{cache,i}$, $L_{cache,i}$} and $cache \in \{Icache, Dcache, L2cache\}$. Here $S_{cache,i}$, $A_{cache,i}$ and $L_{cache,i}$ denote the size, associativity and line size of the i[th] cache configuration at the respective level. **The goal is** to allocate multi-level cache configurations to individual applications, i.e., *TC*: $T \rightarrow \{C_{Imin}, C_{Dmin}, C_{L2min}\}$ such that

Algorithm 2: Parameter Adaptation

Function *adaptParameter(t, param, level, CCF_{min}, P_τ, E_τ)*
// param – size (S), associativity (A), line size (L)
1. ***for*** $i \in param_{level}$ ***do*** *// loop over all parameter values*
2. $C_{cache}^{(i)} \leftarrow$ *cache configuration with changed parameter i*
3. $CCF(t, C_{cache}^{(i)}) \leftarrow computeCCF(t, C_{cache}^{(i)})$
4. $CCF_{other} \leftarrow 0$ *// to find vulnerability impact on other caches*
5. ***for*** $j \in cache - \{level\}$ ***do***
6. $CCF(t, C_j^{(i)}) \leftarrow computeCCF(t, C_j^{(i)})$
7. $CCF_{other} \leftarrow CCF_{other} + CCF(t, C_j^{(i)})$
8. $P_{loss} \leftarrow computePerfOverhead(\ )$
9. $E_{loss} \leftarrow computeEnergyOverhead(\ )$
10. ***end for***
11. $CCF_{total} \leftarrow CCF(t, C_{cache}^{(i)}) + CCF_{other}$
12. ***if*** $CCF_{total} \leq CCF_{min}$ && $P_{loss} \leq P_\tau$ && $E_{loss} \leq E_\tau$ ***then***
13. $CCF_{min} \leftarrow CCF_{total}$; $param_{level}^{optimum} \leftarrow i$; $C_{cache}^{(optimum)} \leftarrow C_{cache}^{(i)}$
14. ***end if***
15. ***end for***
16. *return* $param_{level}^{optimum}, CCF_{min}$

the total vulnerability to soft errors (measured in terms of *CCF*) is minimized under user tolerated performance and energy constraints, P_τ and E_τ respectively.

Objective: $minimize \sum_{\forall t \in T} \sum_{\substack{\forall i \in Icache, \forall j \in Dcache \\ \forall k \in L2cache}} CCF(t, \{i, j, k\})$

such that: $P_{loss} < P_\tau$ and $E_{loss} < E_\tau$ where:P_{loss} and E_{loss} are the performance and energy overheads incurred in identifying reliability-wise optimal cache hierarchy configurations. Note that P_τ and E_τ are user-configurable parameters.

Algorithm 1 employs the *parameter adaptation function* as shown in Algorithm 2, where the different parameter values are assigned and the best combination is selected. Following the flow of Algorithm 1, the vulnerability of the multi-level cache hierarchy is minimized in a three-step process.

1) Application Profiling: In this step, multi-threaded application threads are categorized based on their cache access patterns. For this, we monitor different metrics like *hit/miss rates* and *coherence misses* for the baseline cache configuration. Hit/miss rates capture cache usage while coherence misses indicate inter-thread data sharing. Application threads which show similar cache behavior may be grouped together (Algorithm 1, Line 1) and cache tuning maybe performed for a single thread of each group. The obtained configuration is communicated to the rest of the threads of the same group, thereby curtailing the exploration cost/time. Similarly, if the application shares data, every tuning step is required to complete on all active cores before processing to explore a new configuration. Data sharing applications therefore require a coordinated tuning effort, potentially leading to an increase in exploration time (Algorithm 1, Line 3). Once different applications are grouped and their cache behaviors are determined, the following exploration steps are carried out in sequence.

2) Cache (Partition) Size Adaptation: Keeping the L1 instruction cache in the smallest configuration (e.g., 16kB, 1-way, 32B) vary its size and identify its Pareto-optimal configurations. They are identified by iterating over all configuration points and selecting the vulnerability minimizing configuration out of a subset with same performance properties, while discarding all the non Pareto-optimal points. The same procedure is repeated for the data cache. Using the minimum vulnerability instruction and data cache configurations from the Pareto-set, Pareto-optimal configurations of the L2 partition sizes for the application are identified using a greedy search.

3) Line Size and Associativity Adaptation: In this step, we fix the L1 cache sizes and L2 partition sizes to those identified in the previous steps. Vary the line sizes of the instruction cache and identify the

most reliable configuration. A similar procedure is repeated for the L1 data and L2 caches. Associativity adaptation is performed in a similar manner to line size adaptation for each cache level.

It must be kept in mind that during L2 partition adaptation, each application thread must be guaranteed to execute on a minimum sized L2 partition. If reconfiguration at the L1 level is found to result in significant performance overheads, our parameter adaptation unit limits reconfiguration to the L2 cache level.

V. Experimental Setup

The experimental setup and parameters are summarized in Table I. We extended the Gem5 [19] cycle-accurate full system simulator with the following additional features: (1) vulnerable period estimation for applications (2) identifying the actively used cache lines; (3) supporting different private L1 cache configurations for different cores and adapting shared L2 partitions (also at run time) for different applications; (4) tracking LRU cache blocks within an L2 partition; and (5) analyzing and characterizing the impact of single- and multi-bit errors for different applications. The details on the fault model and error characterization are presented in Section II. As adopted by the cache and architecture communities [7], we report statistics only for the parallel and thread synchronization phases of applications although the initialization and thread creation phases are also simulated. To account for a wide range of run-time scenarios, we consider multi-threaded applications from the *PARSEC* and *SPLASH-2* benchmark suites and single-threaded embedded applications from the *MiBench* suite for our evaluations. Note that the effect of the operating system scheduling and I/O are taken into account in the results. Overall execution time is used as the performance metric. This takes into account the possible increase in DRAM access rates due to increased L2 misses/write-backs as a result of reconfiguration. In order to compute the energy of the cache hierarchy and the hit/response latencies of different cache configurations, we use Cacti 6.5 [29]. We consider both the leakage energy and the dynamic energy due to cache accesses which includes the energy spent in writing dirty cache lines to lower cache levels or main memory during reconfiguration. The additional energy and latency overheads imposed by the reconfiguration logic are negligible, but are taken into account in the results.

Table I. Experimental setup and system parameters

Core parameters	Alpha 21264 cores; Linux 2.6; Number of cores= 1,2,4; Core frequency=2GHz; Number of Data TLB entries=64; Number of Instruction TLB entries=48
Private L1 Cache parameters (Configurable)	Data and Instruction Cache: 16kB-64kB, 1-4 way, 32B-64B line size, Hit Latency=2-3 cycles ; Number of MSHR's=4; Baseline: 64kB,4 way, 64B
Shared L2 Cache Parameters (Configurable)	256kB-2MB,4-16 way, 32-64B line size; Hit Latency=19-21 cycles; Number of MSHR's=20; Coherence=MOESI Snooping based; Replacement policy=LRU based; Baseline: 512kB, 8 way, 64B

VI. Results and Discussion

A. Reliability-Adaptive Cache Design Space

Fig. 8 shows the reliability-performance-energy design space for different applications. All applications show pareto-optimal cache hierarchy configurations that trade-off between vulnerability and performance. The configurations that deliver highest performance also show high vulnerability to soft errors. Another observable trend is that applications also differ in terms of the range of reliability-adaptive cache configurations they provide. *Lame* and *Patricia* show significant variations in vulnerability (up to 54.8% and 35.2% respectively) across cache hierarchy configurations and performance constraints. The vulnerability of *Lame* and *Patricia* is highly sensitive to L1 cache resizing. Both these applications show high reuse of cache lines and large L1 caches with greater number of store hits are less vulnerable. *Cholesky* and *Basicmath* show distinct performance based on whether their working sets fit in the cache. L2 size adaptation significantly changes their vulnerability. Although the performance range is restricted, vulnerability variations are observed on account of changing cache access patterns with parameter adaptation. Both *Bodytrack* and *Streamcluster* show plenty of reads to the data cache. Increasing data cache size leads to large increases in energy consumption. However, vulnerability variations are limited since these applications show less reuse of cache lines and distribute their cache accesses. Table II provides selected configuration comparison with state-of-the-art performance maximizing [16] and energy minimizing [17] heuristics for two example applications.

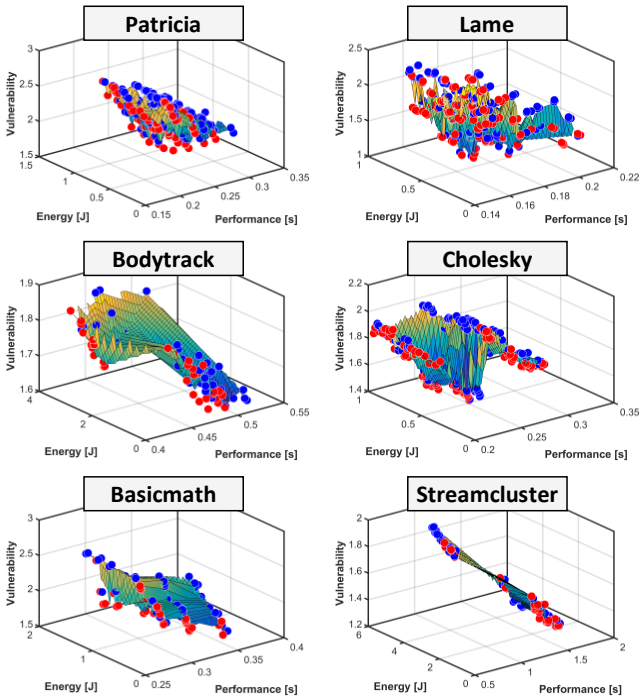

Fig. 8 Vulnerability, performance and energy for different cache configurations. Pareto-optimal configurations are shown in red.

Table II. Comparing configurations selected by different methodologies. Legend: (cacheType_size_associativity_lineSize)

Application	Performance optimizing [11]	Energy optimizing [12]	Reliability optimizing
Patricia	D_64kB_4_I_64kB_4 L2_2MB_8_64B	D_16kB_1_I_32kB_1 L2_256kB_8_32B	D_16kB_2_I_16kB_4 L2_512kB_8_64B
Cholesky	D_64kB_4_I_64kB_4 L2_2MB_16_64B	D_16kB_1_I_16kB_1 L2_256kB_4_32B	D_32kB_1_I_16kB_1 L2_256kB_8_64B

B. Vulnerability Comparison to State-of-the-Art

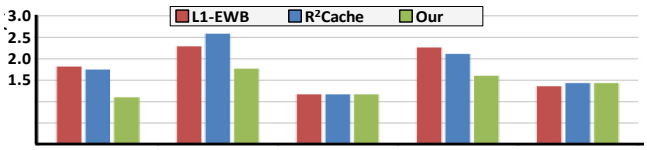

Fig. 9 Vulnerability comparison of the proposed approach with different state-of-the-art techniques

Fig. 9 shows the vulnerability savings of the proposed approach when compared to state-of-the-art non-reconfigurable baseline cache with L1 Early WriteBack (EWB) [22] and Reliability-Aware-Last-Level Cache Partitioning (R^2Cache) [18] schemes. It can be seen that >50% vulnerability savings can be obtained for *Lame*, *Patricia* and *Basicmath*. For *Lame* and *Basicmath*, the instruction cache data has large vulnerable time and shows good reuse. Smaller instruction cache sizes are less vulnerable for these applications. EWB does not operate on read-only instruction caches and cannot provide reliability benefits. Although R^2Cache finds reliability-wise optimal L2 cache configurations, it does not reconfigure L1 caches and provides limited benefits. *Streamcluster* and *Fluidanimate* benefit from EWB that reduces the

high DET resulting from dirty data in the data cache. Although our approach cannot completely eliminate DET, we obtain comparable reliability savings to EWB. Averaged across applications, the proposed approach provides 17% and 39.4% reliability savings when compared to EWB and R^{2}Cache respectively under 10% performance overhead.

C. Exploration Time Savings

Fig. 10 shows the time savings achieved by the proposed heuristic when compared to (1) exhaustive exploration of the design space, (2) a multi-level cache tuning approach that tunes different cache levels independently (MCT-1) [23], and (3) a multi-level cache tuning heuristic that coordinates tuning amongst cache levels and traverses the design space in the order size, line size and associativity but ignores application data sharing (MCT-2) [6]. While data sharing applications like *Bodytrack* require more design effort to coordinate tuning between multiple cores, the proposed heuristic still identifies reliability-wise efficient cache configurations while exploring less than 2% of the entire exhaustive cache configuration design space.

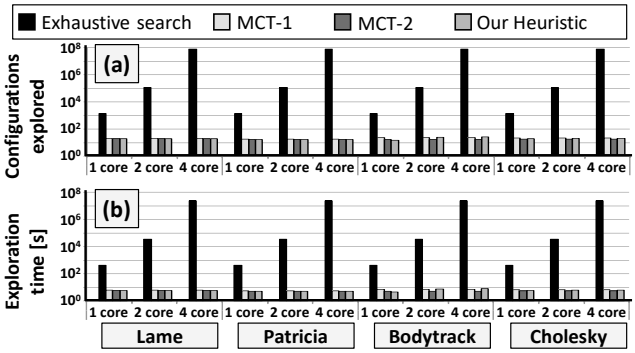

Fig. 10 Exploration time savings for different number of cores.

D. Applicability in ECC Protected Caches

Fig. 11 (a) shows the reduction in the number of multi-bit failure scenarios of the reliability-wise efficient cache hierarchy configuration found by our technique in comparison with a non-reconfigurable SECDED (Single Error Correction Double Error Detection) protected L1 and L2 cache [26][27] (L1 caches: 32kB, 4 way, 64B and L2 cache: 512kB, 8-way, 64B). The parameters and interleave assumed for ECC are discussed in Section II.B. Cache hierarchy parameter adaptation can reduce both the spatial and temporal vulnerability of different cache levels and an average 46% reduction in multi-bit failure scenarios is observed for applications of the *MiBench* suite. Fig. 11 (b) shows the distribution of multi-bit failure scenarios and the error masking properties of *MiBench* applications for an ECC-protected non-reconfigurable cache hierarchy. Amongst the chosen applications, *Bitcount* shows the least number of failures, because bit flips in its cached input array (0's and 1's) lead to incorrect outputs. *Patricia* on the other hand deals with a tree data structure and bit flips often lead to pointer corruption and subsequently application failures.

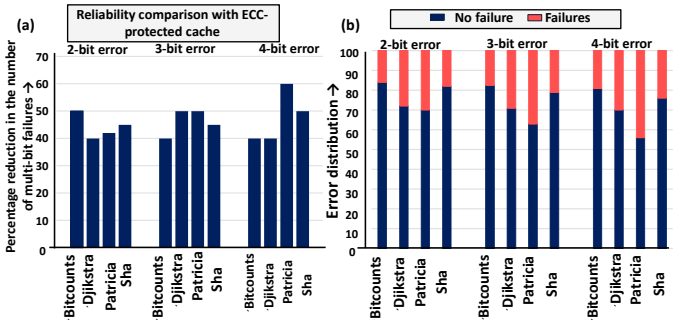

Fig. 11 (a) Reduction in number of multi-bit failures in comparison to a fixed configuration multi-level ECC protected cache; (b) Number of failure scenarios for a non-reconfigurable ECC protected multi-level cache.

VII. Conclusion

In this paper, an analysis of the vulnerability interdependencies across different cache levels in multi-core processors is presented, which shows the impact of different access patterns and vulnerabilities of different applications for diverse cache configurations. This knowledge is leveraged to propose a soft error-aware cache architectural space exploration methodology, which determines a set of reliability-wise beneficial configurations for the complete cache hierarchy while significantly reducing the exploration time. The selected configurations will enable a run-time system to perform reliability-aware cache reconfiguration in performance/energy-constrained scenarios, which is planned as a future work. Since the vulnerability of applications also varies in different execution phases, reliable caches could further benefit from extending the approach in this direction.

References

[1] R. Baumann, "*Radiation-induced soft errors in advanced semiconductor technologies*", IEEE TDMR, vol. 5, no. 3, pp. 305-316, 2005.

[2] E. Ibe et al., "*Impact of Scaling on Neutron-Induced Soft Error in SRAMs From a 250 nm to a 22 nm Design Rule*", IEEE TED, vol.57, no.7, pp.1527-1538, 2010.

[3] S. Wang, J. S. Hu, S. G. Ziavras, "*On the characterization and optimization of on-chip cache reliability against soft errors*", IEEE TC, 58(9):1171-1184, 2009.

[4] W. Zhang, "*Computing cache vulnerability to transient errors and its implication*", DFT Symposium, pp. 427-435, 2005.

[5] M. Wilkening et al., "*Calculating Architectural Vulnerability Factors for Spatial Multi-Bit Transient Faults*", IEEE MICRO-47, pages 293-305, 2014.

[6] W. Wang, P. Mishra, S. Rank, "*Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems*", DAC, pp.948-953, 2011.

[7] K. T. Sundararajan, T. M. Jones, N. P. Topham, "*RECAP: region-aware cache partitioning*", ICCD, pages 294-301, 2013.

[8] A. Gordon-Ross, F. Vahid, N. Dutt, "*Automatic tuning of two-level caches to embedded applications*", DATE, pp. 208-213, 2004.

[9] C. Zhang, F. Vahid, W. A. Najjar, "*A highly-configurable cache architecture for embedded systems*" ISCA, pages 136-146, 2003.

[10] A. Janapsatya, A. Ignjatovic, S. Parameswaran, "*Finding optimal L1 cache configuration for embedded systems*", ASP-DAC, pages 796-801, 2006.

[11] I. Nawinne, J. Schneider, H. Javaid, S. Parameswaran, "*Hardware-based fast exploration of cache hierarchies in application specific MPSoCs*", DATE, 2014.

[12] M. Rawlins, A. Gordon-Ross. "*A cache tuning heuristic for multicore architectures*", IEEE Trans. Computers, 62(8):1570-1583, 2013.

[13] Y. Cai, M. T. Schmitz, A. Ejlali, B. M. Al-Hashimi, S. M. Reddy, "*Cache size selection for performance, energy and reliability of time-constrained systems*", ASP-DAC, pp.923-928, 2006.

[14] C. Zhang, F. Vahid, R. L. Lysecky, "*A self-tuning cache architecture for embedded systems*", ACM Trans. Embedded Comput. Syst., 3(2):407–425, 2004.

[15] A. Settle, D. Connors, E. Gibert, A. Gonzalez, "*A dynamically reconfigurable cache for multithreaded processors*", J. of Embedded Computing 2.2, pp. 221-233, 2006.

[16] M.K. Qureshi, Y.N. Patt, "*Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches*", MICRO-39, pp 423-432, 2006.

[17] S. S. Mukherjee et al., "*A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor*", MICRO, pp. 29-40, 2003.

[18] F. Kriebel, A. Subramaniyan, S. Rehman, S. J. B. Ahandagbe, M. Shafique, J. Henkel, "*R^{2}Cache: reliability-aware reconfigurable last-level cache architecture for multi-cores*", CODES+ISSS, 2015

[19] N. Binkert et al. "*The gem5 simulator*", SIGARCH Comput. Archit. News, pages 1-7, August 2011.

[20] A. Malik, B. Moyer, D. Cermak, "*A low power unified cache architecture providing power and performance flexibility*", ISLPED, 2000.

[21] H. Cook et al. "*A hardware evaluation of cache partitioning to improve utilization and energy-efficiency while preserving responsiveness*", ISCA, 2013.

[22] R. Jeyapaul, A. Shrivastava, "*Enabling energy efficient reliability in embedded systems through smart cache cleaning*", ACM TODAES, 18(4):53, 2013.

[23] W. Wang, P. Mishra, "*Dynamic Reconfiguration of Two-Level Cache Hierarchy in Real-Time Embedded Systems*", J. Low Power Electronics 7(1): 17-28, 2011

[24] J. Suh, M. Manoochehri, M. Annavaram, M. Dubois, "*Soft error benchmarking of L2 caches with PARMA*", SIGMETRICS, 2011

[25] H. Asadi, V. Sridharan, M. B. Tahoori, D. R. Kaeli, "*Vulnerability analysis of L2 cache elements to single event upsets*", DATE, 2006

[26] M. K. Qureshi, Z. Chishti, "*Operating SECDED-based caches at ultra-low voltage with FLAIR*", DSN, pp. 1–11, 2013

[27] A. R. Alameldeen et al., "*Energy-efficient cache design using variable-strength error-correcting codes*", ISCA, pp. 461-472, 2011

[28] F. Kriebel et al. "*Reliability-Aware Adaptations for Shared Last-Level Caches in Multi-Cores*", TECS, vol 15, no. 4, 2016

[29] "Cacti 6.5", http://www.hpl.hp.com/research/cacti