Vulnerability-aware Energy Optimization using Reconfigurable Caches in Multicore Systems

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

***Abstract*—Dynamic cache reconfiguration has been widely explored for energy optimization and performance improvement for single-core systems. Cache partitioning techniques are introduced for the shared cache in multicore systems to alleviate inter-core interference. While these techniques focus only on performance and energy, they ignore vulnerability due to soft errors. In this paper, we present a static profiling based algorithm to enable vulnerability-aware energy-optimization for real-time multicore systems. Our approach can efficiently search the space of cache configurations and partitioning schemes for energy optimization while task deadlines and vulnerability constraints are satisfied. Our experimental results demonstrate that our approach can achieve 19.2% average energy savings compared with the base configuration, while drastically reduce the vulnerability (49.3% on average) compared to state-of-the-art techniques.**

I. Introduction

Multicore architectures consist of multiple processor cores to improve execution performance of application programs. Multicore processor usually has on-chip caches to resolve the performance bottleneck caused by the increasing gap between processor and memory speed. In a typical multicore system, each core maintains its private L1 caches while all cores share the same L2 cache. There are many optimization techniques for multi-level on-chip caches to improve performance and energy consumption of the overall system [5], [6], [11]. With the increasing demand for high reliability and availability, vulnerability of caches due to soft errors is gaining increasing importance. Data corruption caused by soft errors can change the behavior of applications and may eventually result in a system failure. As for performance and energy improvement, it is beneficial to maintain a useful data longer in the cache. However, longer data retention can negatively impact the vulnerability or probability of data corruption due to soft errors. It is a great challenge to keep vulnerability under control while we optimize the cache subsystem for improvement in performance and energy consumption.

Application-based techniques on cache optimization have been very effective in improving performance and energy consumption. One of the most successful techniques for cache energy optimization is dynamic cache reconfiguration (DCR). The basic idea of DCR is to select a suitable cache configuration to satisfy the specific data access behavior of the application. By tuning the cache configuration (cache size, associativity, and line size) at runtime, it is possible to optimize the energy consumption and improve performance of different applications. DCR has been well studied for energy savings in both uniprocessor [3] and multicore systems [11]. Recent work by Huang et al. [15] studies the impact of DCR on vulnerability in the L1 caches for a uniprocessor. However, there are no existing efforts in vulnerability-aware cache reconfiguration for multicore systems.

As for a shared L2 cache, it may cause performance degradation because of data contentions among different cores. Cache partitioning (CP) techniques are introduced to alleviate this problem by judiciously dividing the shared cache and mapping a dedicated partition of the cache to each core. CP can improve the performance of independent tasks running on different cores, by eliminating inter-task interference on the shared cache. DCR and CP are both cache optimization techniques to properly tune the cache subsystem based on the data access pattern of applications. Previous work by Wang et al. [11] explores the idea of combining DCR and CP for energy optimization in real-time multicore systems. However, their work does not consider vulnerability.

In this paper, we propose a vulnerability-aware energy optimization technique which integrates cache reconfiguration (DCR) of private L1 caches and cache partitioning (CP) of the shared L2 cache. This paper makes four important contributions: (i) We explore the inter-dependence of L1 DCR and L2 CP for performance, energy consumption as well as vulnerability; (ii) We are able to minimize energy consumption without violating both vulnerability and real-time constraints; (iii) Our fast and scalable static profiling algorithm can efficiently search the design space of L1 configurations and L2 partitions, making it feasible to find the optimal result using dynamic programming; and (iv) Our results demonstrate that our approach can provide significant energy savings compared with the base configuration as well as drastic reduction in vulnerability compared to the state-of-the-art techniques.

The remainder of the paper is organized as follows. Related approaches are discussed in Section II. The architecture model and an motivational example are presented in Section III. Section IV presents our approach for vulnerability-aware optimization. Section V presents the experimental results. Section VI concludes the paper.

II. Related Approaches

Reconfigurable cache architectures are extensively studied in [1], [16], [17]. Gordon-Ross et al. [2] utilizes DCR to

improve performance by online feedback and dynamic self-tuning of the cache. An energy-efficient approach using DCR is proposed in [3] for soft real-time systems by using static profiling and dynamic reconfiguration. DCR in two-level cache hierarchy in uniprocessor is studied in [4]. DCR for multicore systems is studied in [5] for thread-fairness and performance improvement. Wang et al. proposes an energy-efficient approach for multicore systems in [11] by using DCR on private L1 caches and cache partitioning (CP) on the shared L2 cache. CP is a special case of reconfiguration on the shared cache among multiple cores [6], [7]. Initial works of CP aim at improving the performance of multicore systems [7], [8]. Reddy et al. investigates energy-efficient CP for multitasking embedded systems in [9]. However, none of the above approaches takes vulnerability into consideration.

To combat the data vulnerability due to soft errors, error correction codes (ECC) are used in lower levels of the memory hierarchy [10]. However, ECC might not be suitable for caches because of short access time constraints [12]. Cai et al. [13] is the first to consider cache configuration (only cache size selection) for energy and vulnerability in time-constrained systems. Huang et al. [15] proposes a DCR approach for performance, energy and vulnerability trade-offs in uniprocessor-based systems. To the best of our knowledge, the proposed work is the first attempt in studying vulnerability-aware optimizations in multicore systems in the presence of reconfigurable caches.

III. MODELING SYSTEMS WITH RECONFIGURABLE CACHES

In this section, we describe the modeling of multicore systems with reconfigurable caches. First, we describe the underlying multicore architecture. Next, we present the energy and vulnerability models. Then, we provide an illustrative example to motivate the need for the proposed exploration framework. Finally, we present the problem formulation.

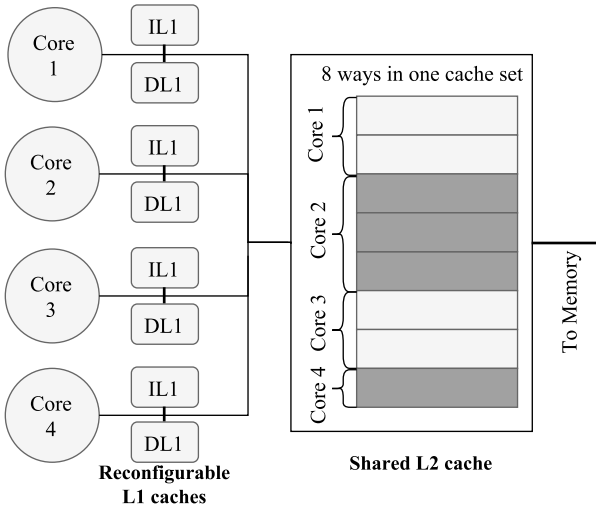


Fig. 1: **A multicore system with reconfigurable L1 caches and a partition-enabled shared L2 cache.**

A. Multicore Architecture Model

Figure 1 shows a typical multicore system with a shared on-chip L2 cache and private L1 caches for each core. In this paper, we assume that the private L1 caches (both IL1 and DL1) are reconfigurable, and the shared L2 cache is equipped with way-based partitioning. The L1 caches can reconfigure its cache size, associativity, and line size. The reconfigurable cache architecture is the same as [2], [3]. The cache size is tuned by selectively shutting down the banks with gated-V_{dd} techniques. The associativity is reconfigured by logically concatenating ways. The line size can be changed by fetching multiple unit-length blocks in one access. The reconfigurable architecture is lightweight, which introduces negligible overhead [3].

The shared L2 cache with way-based partitioning [7] is illustrated in Figure 1. Each L2 cache set (8-way associativity as in this example) is partitioned into four parts, each of which will be assigned to one core. Each core will access only the assigned portion of the cache sets and enforce the LRU replacement policy among its individual group of ways. The number of ways assigned to a core is referred to as its *partition factor*. As shown in Figure 1, Core 1 has a L2 partition factor of 2. In this paper, we use dynamic reconfiguration of the L1 caches and static partitioning of the shared L2 cache. In other words, L1 cache configurations can be tuned for each application on each core during runtime. While L2 partition factors are pre-determined for each core and remain unchanged during runtime, all applications running on that core have the same L2 partition factor.

B. Energy and Vulnerability Models

The *Energy Model* is adopted from the one used in [3]. The cache energy consumption consists of static and dynamic energy: $E = E_{sta} + E_{dyn}$. The static energy dissipation E_{sta} is computed as $E_{sta} = P_{sta} \times t$, where P_{sta} is the static power of cache. Dynamic energy dissipation E_{dyn} comes from both cache accesses and cache misses.

$$E_{dyn} = Accesses \times E_{access} + Misses \times E_{miss} \quad (1)$$

$$E_{miss} = E_{offchip\_access} + E_{block\_fill} \quad (2)$$

where E_{access} and E_{miss} are the energy required per cache access and per cache miss, respectively. E_{access} and E_{miss} are constant values for one specific configuration. $E_{offchip\_access}$ is the energy for accessing the lower level of the memory hierarchy, and E_{block_fill} is the energy for filling the cache block with fetched data.

The *Vulnerability Model* is based on per-byte analysis of cache data with respect to the sequence of operations during its lifetime in the cache. Operations on a byte include "fill", "read", "write" and "evict". Similar to [15], the vulnerability analysis divides the lifetime of a byte into vulnerable and un-vulnerable intervals. The *vulnerable intervals* are of four types: "fill-to-read", "read-to-read", "write-to-read", "write-to-evict". We measure the vulnerability of cache as the summation of vulnerable intervals of all bytes in all cache blocks.

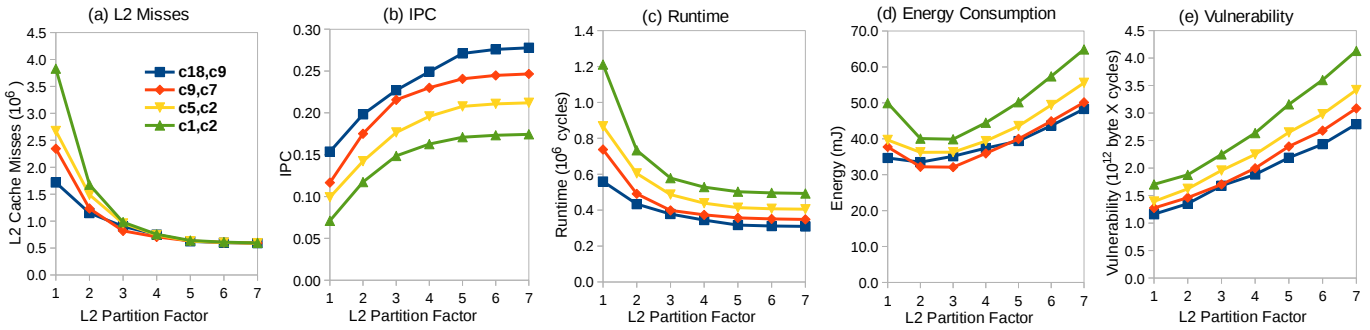


Fig. 2: **Inter-dependence of L1 DCR and L2 CP on (a) L2 Misses, (b) IPC, (c) Runtime, (d) Energy and (e) Vulnerability.**

C. Illustrative Example

Figure 2 shows the impact of L1 DCR and L2 CP for benchmark $qsort$ from MiBench [20]. The L2 partition factor can change from 1 to 7 in a 8-way associative L2 cache. Four pairs of cache configurations[1] for IL1 and DL1 are randomly chosen. We observe that different L1 configurations will lead to different L2 cache misses (Figure 2a) and pipeline throughput (i.e. IPC in Figure 2b). This is expected since L1 configuration determines the number accesses to the L2 cache, as well as the pipeline throughput. Secondly, as L2 partition factor w increases, L2 cache misses will decrease and eventually converge (for $w \geq 4$) for different L1 configurations. However, the IPC shows great diversity even when L2 partition factor is large.

Figure 2c-2e show the runtime, energy consumption and cache vulnerability of the benchmark, respectively. It is interesting to see that they have different patterns as L2 partition factor w increases. Runtime will decrease drastically as w increases, which is accordant with the pattern of IPC. Energy consumption will decrease to a minimal point (for $w = 3$), but it will increase when w becomes larger. This is because dynamic energy (caused by a lot of cache misses) dominates the total energy consumption when w is small, while static energy dominates when w is too large. However, vulnerability will increase with w. This is expected for two reasons: (1) a large w means that L2 cache has more valid area and is holding more data, which remains vulnerable to soft errors; (2) the decrease in cache misses (data replacement) also indicates that data are residing in the cache for longer time, which means data will have longer vulnerable intervals. While a large L2 partition facilitates performance, it might jeopardize energy consumption and vulnerability. This shows that performance, energy and vulnerability have very different (often conflicting) cache requirements.

Given the above observations, both L1 DCR and L2 CP have major impact on performance, energy consumption and vulnerability. The interesting trade-offs between them is the motivation of this paper to explore for optimization. We exploit L1 DCR and L2 CP simultaneously for vulnerability-aware energy optimization for real-time multi-core systems.

[1]Here $c18$ and $c9$, for example, stands for the IL1 and Dl1 using the 18^{th} and 9^{th} configuration, respectively.

D. Problem Formulation

We model our multicore system as follows:

- The multicore processor has m cores $\mathbb{P}$ $\{p_1, p_2, ..., p_m\}$.
- Each core has private IL1 and DL1, both of which can be reconfigured to r configurations $\mathbb{C}$ $\{c_1, c_2, ..., c_r\}$.
- The shared L2 cache is ω-way associative, which supports way-based partitioning.
- A set of n independent tasks $\mathbb{T}$ $\{\tau_1, \tau_2, ..., \tau_n\}$ with a common deadline D.

Our optimization goal is to find a reconfiguration scheme $\mathbf{R}$ for the private L1 caches and a partitioning scheme $\mathbf{P}$ for the shared L2 cache such that the overall energy consumption E is minimized without violating vulnerability constraints and task deadlines. Assume that we are given the following:

- A task mapping scheme $\mathbf{M}$: $\mathbb{T} \rightarrow \mathbb{P}$, which assigns tasks to each core. In this paper, we assume that the task mapper $\mathbf{M}$ is given, which can ensure that the total runtime on each core is comparable. ρ_k is the number of tasks mapped to core k.
- A reconfiguration scheme $\mathbf{R}$ for L1 caches: $C_I, C_D \rightarrow \mathbb{T}$, which assigns one IL1 and DL1 configuration to each task.
- A partitioning scheme $\mathbf{P}$ for L2 cache: $\mathbf{P}$ = $\{w_1, w_2, ..., w_m\}$, which allocates w_k ways to core k.

For task $\tau_{k,i} \in \mathbb{T}$ (the i_{th} task on core k), $e_{k,i}(c_I, c_D, w_k)$ denotes the energy consumption of the cache subsystem when the task is executed with L1 configurations (c_I, c_D) and L2 partition factor w_k. Similarly, let $t_{k,i}(c_I, c_D, w_k)$ and $v_{k,i}(c_I, c_D, w_k)$ denote the execution time and the total vulnerability. Our minimization problem can be formulated as follows:

$$E = \sum_{k=1}^{m} \sum_{i=1}^{\rho_k} e_{k,i}(c_I, c_D, w_k) \tag{3}$$

is minimized subject to:

$$\max_{k=1..m}(\sum_{i=1}^{\rho_k} t_{k,i}(c_I, c_D, w_k)) \leq D \quad (4)$$

$$\sum_{i=1}^{\rho_k} v_{k,i}(c_I, c_D, w_k) \leq V_k, \forall k \in [1, m] \quad (5)$$

$$\sum_{k=1}^{m} w_k = \omega; w_k \geq 1, \forall k \in [1, m] \quad (6)$$

Equation (4) guarantees that all tasks will meet the deadline D. Equation (5) guarantees that the total vulnerability of the tasks on each core is constrained by the threshold V_k, which is chosen as the base case vulnerability. Equation (6) verifies that the partitioning scheme is valid.

IV. Vulnerability-aware DCR+CP

In this section, we present our approach which utilizes the static profiles of tasks to efficiently search the design space for the optimal energy solution. Our three-step optimization approach is illustrated in Figure 3, with the **first step** to profile each task, the **second step** to use a dynamic programming algorithm to optimize for all cache configurations on each core, and the **third step** to combine the optimal solutions on each core by trying out all feasible L2 partition schemes.

A. Task Profiling

Theoretically, we can do static profiling for the whole task set $\mathbb{T}$ for all possible L1 reconfiguration schemes $\mathbf{R}$ and all possible L2 partition schemes $\mathbf{P}$. However, it is not feasible to do this exhaustive exploration because of excessive simulation time. Assume that we have a four-core processor with an 8-way associative L2 cache. Each core is assigned with three tasks and the IL1 and DL1 cache each has 18 configurations [3]. The total number of architectural simulations would be $((18^2)^3)^4 * 35$. To be specific, $((18^2)^3)^4$ would be all L1 configurations (both IL1 and DL1) for the tasks (three on each core) across four cores. This needs to be multiplied by 35, which is the total number of valid L2 partition schemes according to Equation (6) with m=4 and ω=8. If each simulation takes only 1 minute, the total simulation time is longer than the age of the universe.

Fortunately, we can drastically reduce the complexity of static profiling by exploiting the inherent independence in our system. Tasks running on different cores are independent (with no inter-task data sharing). After introducing L2 partitioning, each task is essentially isolated on a separate core with private L1 caches and a dedicated L2 partition. Therefore, we can profile each task as if it is executed independently on a uniprocessor with a w_i-way associative L2 cache (with capacity equal to w_i/ω of the original L2). The total number of simulations required for the entire task set would be $r^2 \cdot (\omega - 1) \cdot n$, where r^2 is the number of IL1 and DL1 combinations, $(\omega - 1)$ is the number of possible L2 partition factors, and n is the total number of tasks. Using the same example above, it takes $18^2 \times 7 \times 12$ simulations with 12 tasks. For benchmarks used in our experiments, the static profiling can finish within three days. We simulate each task with all possible IL1 and DL1 cache configurations, along with all possible L2 partition factors. After static profiling, each task has a profile table with $r^2 \cdot (\omega - 1)$ entries, each of which contains the runtime, energy consumption, vulnerability for the specified L1 configurations and L2 partition factor.

Note that the profiling can be done off-line for one specific input pattern for a program. In this work, we assume that the input size remains the same but content can vary. This is a reasonable assumption for real-time embedded systems. We performed our offline analysis by varying input patterns (data values) for all the benchmarks and observed that it has minor impact on the footprint of data access. Since profile of vulnerability and energy estimation for data pages depends on the data access pattern, our static profiling will still remain effective for different input patterns. Our observations are consistent with the ones made by existing literature [3].

B. Optimization on Each Core

In order to find the optimal solution under deadline and vulnerability constraints, we first optimize on each core (find profitable L1 configurations), and then optimize across all cores (find the best L2 partition scheme). In this subsection, we explain our approach for optimization on each core. Since static partitioning of L2 is used, tasks on the same core share the same L2 partition factor w_k. This fact enables us to treat each core as a subproblem, which optimizes the energy consumption for a given core under different L2 partition factors. In other words, we find cache assignment $\mathbf{R}$ to minimize $E_k(w_k) = \sum_{i=1}^{\rho_k} e_{k,i}(c_I, c_D, w_k)$ constrained by $\sum_{i=1}^{\rho_k} t_{k,i}(c_I, c_D, w_k) \leq D$ and $\sum_{i=1}^{\rho_k} v_{k,i}(c_I, c_D, w_k) \leq V_k$, with k and w_k fixed for $\forall k \in [1, m]$ and $\forall w_k \in [1, \omega - 1]$.

This subproblem is to choose L1 configurations for each task so that the total energy is optimized with constraints. The optimization goal is to minimize energy, which can be discretized to simplify the problem. We can use a dynamic programming algorithm to search for the optimal solution. Let $e_k^{min}(w_k)$ and $e_k^{max}(w_k)$ denote the minimum possible energy ($\sum_{i=1}^{\rho_k} min\{e_{k,i}(c_I, c_D, w_k)\}$) and the maximum possible energy ($\sum_{i=1}^{\rho_k} max\{e_{k,i}(c_I, c_D, w_k)\}$) on core k, respectively. The energy consumption $E_k(w_k)$ of core k using partition factor w_k is bounded by $[e_k^{min}(w_k), e_k^{max}(w_k)]$. Let S_i^E denote the current solution found for the first i tasks. It has a cumulative energy consumption of E while the execution time and vulnerability are minimized. The execution time $T[i][E]$ for S_i^E is stored in a two-dimensional table T. The vulnerability for S_i^E is stored in another two-dimensional table V. As we try out all possible (c_I, c_D) configurations, we update the solution for S_i^E whenever runtime or vulnerability can be improved. The dynamic programming process uses the recursive formula shown in Figure 4 to update the two tables. The solutions for the first i tasks (the i^{th} row in the two tables) are built upon the previous step, i.e., the $(i-1)^{th}$ row. All entries in T and V are initialized to some very large value. Based on the above recursive formula, we update the tables one row at a time for all energy values

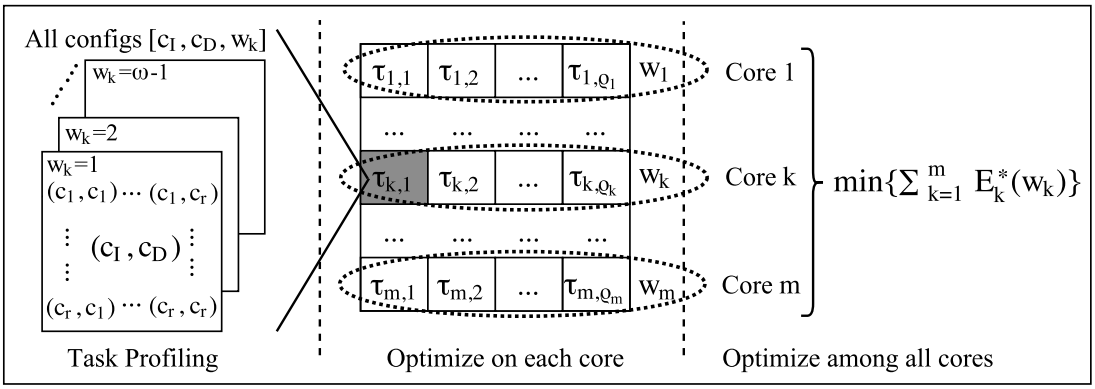


Fig. 3: Three-step optimization: the first step statically profiles each task, the second step optimizes for each partition factor on each core to find the best L1 cache configurations, the third step combines the optimal solution on all cores to find the best L2 partition scheme.

If $(T[i][E] > T[i-1][E - e_{k,i}(c_I, c_D, w_k)] + t_{k,i}(c_I, c_D, w_k)$ **&&** $V[i][E] > V[i-1][E - e_{k,i}(c_I, c_D, w_k)] + v_{k,i}(c_I, c_D, w_k))$
{
$T[i][E] = T[i-1][E - e_{k,i}(c_I, c_D, w_k)] + t_{k,i}(c_I, c_D, w_k)$
$V[i][E] = V[i-1][E - e_{k,i}(c_I, c_D, w_k)] + v_{k,i}(c_I, c_D, w_k)$
}

Fig. 4: **Recursive formula for dynamic programming**

in $[e_k^{min}(w_k), e_k^{max}(w_k)]$. When the i^{th} row is calculated, all previous $(i-1)$ rows are already computed. The final optimal energy consumption $E_k^*(w_k)$ can be found by:

$$E_k^*(w_k) = \min\{E_k \mid T[\rho_k][E_k] \leq D \;\&\&\; V[\rho_k][E_k] \leq V_k\} \tag{7}$$

Equation 7 provides the solution for core k with partition factor w_k, which has minimum energy consumption with deadline and vulnerability constraints satisfied.

C. Optimization Across All Cores

In this step, we combine the solutions found on each core and search for the minimum total energy consumption E^* of all cores within all L2 partition schemes $\mathbf{P}$. For a given partition factor w_k on core k, the optimal energy $E_k^*(w_k)$ is already calculated in the first step. A valid partition scheme $\{w_1, w_2, ..., w_m\}$ is one that complies with Equation (6). The final total energy E^* can be found by:

$$E^* = min\{\sum_{k=1}^{m} E_k^*(w_k)\}, \quad \forall\{w_1, w_2, ..., w_m\} \in \mathbf{P} \tag{8}$$

Since the number of valid partition schemes is small (35 for 4-core processor with an 8-way associative L2 cache), an exhaustive search on all partition schemes is feasible. In our experiment, we assume that after the tasks on a core finish execution the core along with its private L1 caches and the designated L2 partition is turned off. Thus, E^* will be the final energy consumption for all cores running with the optimal configuration and partitioning scheme.

Algorithm 1 shows the major steps of our cache reconfiguration and partitioning approach. In the **first step** (line 1-6), for each task $\tau_{k,i}$, we simulate the task with all possible configurations $[c_I, c_D, w_k]$. We collect the energy, vulnerability and runtime numbers of the task using the configurations and save them in its profile table. In the **second step**, our algorithm iterates to find the best L1 configurations for all tasks in core k with partition factor w_k. During each iteration (line 9 to 23), all discretized energy values (e) and all L1 cache configurations (1 to r^2) for current task $\tau_{k,i}$ are examined. The dynamic programming process of the first task on a core is shown in line 9 to 14, and that of task 2 to ρ_k is in line 15 to 22. Line 23 gets the optimal solution $E_k^*(w_k)$ for core k with partition factor w_k. In the **third step** (line

Algorithm 1: Vulnerability-aware DCR+CP

```
   /* 1st step: Task profiling (Section IV-A)                          */
1  for k = 1 to m do
2      for i = 1 to ρ_k do
3          for w_k = 1 to ω − 1 do
4              for c_I, c_D ∈ ℂ do
5                  Simulate task τ_{k,i} with config=[c_I, c_D, w_k]
6                  Collect t_{k,i}(config) e_{k,i}(config) v_{k,i}(config)
   /* 2nd step: Optimize on each core (Section IV-B)                   */
7  for k = 1 to m do
8      for w_k = 1 to ω − 1 do
9          for e = e_k^min(w_k) to e_k^max(w_k) do
10             for c_I, c_D ∈ ℂ do
11                 if e_{k,1}(c_I, c_D, w_k) == e then
12                     if t_{k,1}(c_I, c_D, w_k) < T[1][e] &&
                       v_{k,1}(c_I, c_D, w_k) < V[1][e] then
13                         T[1][e] = t_{k,1}(c_I, c_D, w_k)
14                         V[1][e] = v_{k,1}(c_I, c_D, w_k)
15         for i = 2 to ρ_k do
16             for e = e_k^min(w_k) to e_k^max(w_k) do
17                 for c_I, c_D ∈ ℂ do
18                     e' = e − e_{k,i}(c_I, c_D, w_k)
19                     if T[i − 1][e'] + t_{k,i}(c_I, c_D, w_k) ¡ T[i][e]
                       &&V[i − 1][e'] + v_{k,i}(c_I, c_D, w_k) ¡ V[i][e]
20                     then
21                         T[i][e]=T[i − 1][e'] + t_{k,i}(c_I, c_D, w_k)
22                         V[i][e]=V[i − 1][e'] + v_{k,i}(c_I, c_D, w_k)
23         E_k^*(w_k) = min{e_k | T[ρ_k][e_k] ≤ D & V[ρ_k][e_k] ≤ V_k}
   /* 3rd step: Optimize across cores (Section IV-C)                   */
24 for all P_j = {w_1, w_2, ..., w_m} ∈ P do
25     E_j^* = ∑_{k=1}^{m} E_k^*(w_k)
26     E^* = min(E^*, E_j^*)
27 return E^*
```

24 to 26), our algorithm iterates over all valid partitioning schemes to find the global optimal energy consumption. Line 25 gets the energy consumption for partition scheme P_j, and line 26 updates the final solution E^* with the minimal energy consumption. The time complexity for the first step is $O(m{\cdot}\rho_k{\cdot}\omega{\cdot}r^2)$, where m is the number of cores, ρ_k is number of tasks on each core, ω is the number of ways in L2 cache, r^2 is the number of L1 configurations. The time complexity for the second step is $O(m \cdot \omega \cdot \rho_k \cdot r^2 \cdot (e^{max} - e^{min}))$, where $e^{max} - e^{min}$ is the energy range. The time complexity for the third step is $O(m \cdot |\mathbf{P}|)$, where m is the number of cores and $|\mathbf{P}|$ is the number of partition schemes. In our experiments, our proposed approach can find the optimal solution in less than three days, which is mostly the time of the first step for profiling. Since our approach is based on static (offline) analysis and one-time effort, this is a reasonable time.

V. Experiments

In order to evaluate the effectiveness of our approach, we use the architectural simulator gem5 [18] in system emulation (SE) mode to simulate the multicore system as shown in Figure 1. We enhanced the simulator to support reconfiguration of L1 caches and way-based partitioning of the shared L2 cache. We also embedded our measurement for vulnerability of caches in the simulator, while the energy estimation of the cache subsystem is calculated with a script after simulation. We configured our system with a four-core processor running at 500MHz on each core with the TimingSimpleCPU model in gem5. The shared L2 cache supports 32KB, 8-way associative with 32-byte lines. There are 35 valid schemes to partition the L2 ways among the four cores. The L1 caches have a base configuration as 4KB, 2-way associative with 32-byte lines, which offers effective size of 1KB, 2KB, and 4KB, and associativity of 1-way, 2-way, and 4-way, and line size of 16-byte, 32-byte and 64-byte. There are 18 configurations in total for the L1 caches[2]. We used 20 applications from the MiBench [20] and SPEC CPU2000 [21] benchmark suites as our tasks for evaluation. Table I shows the task sets used in our experiments. We choose 4 task sets which contain 2 tasks running on each core, 3 task sets which contain 3 tasks on each core, and 2 task sets which contain 4 tasks on each core. The task assignment on cores is based on the rule that each core will have comparable execution time and vulnerability.

In our results, we will compare the following three approaches:

- **CP Only**: the base configuration, which has L1 in base configurations and uniform L2 cache partitioning among cores.
- **DCP+CP[11]**: the energy-aware approach in [11] using DCR on L1 and CP on L2.
- **Our Approach**: our vulnerability-aware energy optimization approach using DCR on L1 and CP on L2.

Here **CP Only** refers to the base configuration of the system, which has uniform L2 cache partitioning among the four cores with all the L1 caches in base configuration. For our vulnerability-aware approach, the vulnerability threshold on each core is set as that of the base system (**CP Only**). We want to minimize the energy consumption while ensure that the vulnerability be at least better than the base system.

[2]It is fewer than 3^3 since not all combinations are valid [3].

A. Deadline and Vulnerability Threshold

It is meaningful to see how deadline and vulnerability threshold affect the optimization process. Figure 5 shows the optimal energy consumption (i.e. $E_1^*(w_1)$ as in Equation 7) of core 1 using partition factor ($w_1 = 2$) for task set 9, under different deadline and vulnerability constraints. In Figure 5a, as we gradually vary the deadline from 4600 ms to 3600 ms, the optimal energy found by the dynamic programming algorithm will become worse. When the deadline is shorter than 3690 ms, there is no feasible solution. In Figure 5b, as we gradually reduce the vulnerability threshold from 8.4×10^{12} to 7.2×10^{12} bytes-cycles, the optimal energy solution will also become worse. There is no solution when vulnerability threshold is set smaller than 7.3×10^{12} bytes-cycles. In this example, we can get a converged optimal energy solution (2753 mJ) with a deadline larger than 4300 ms and a vulnerability threshold larger than 8.0×10^{12} bytes-cycles. Note that in Figure 5a we removed the vulnerability constraint (i.e. set vulnerability threshold as infinity) to solely investigate the effect of deadline and vice versa for Figure 5b.

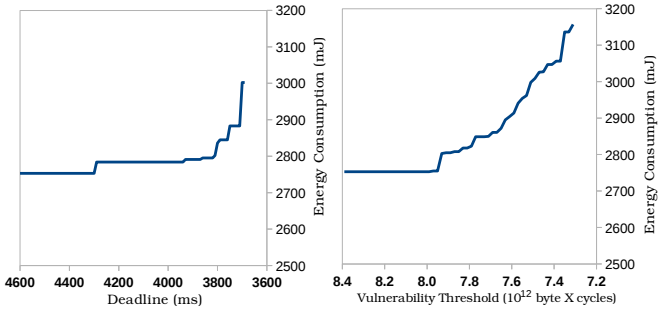


Fig. 5: **Effects of Deadline and Vulnerability Threshold.**

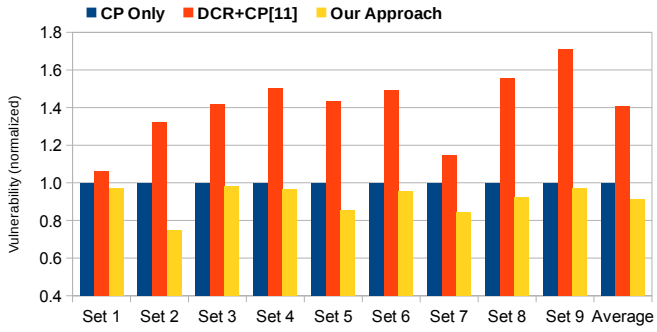
This example suggests that the choice of deadline and vulnerability threshold can affect the optimal energy solution. In our experiments, the deadline is chosen in a way so that each core can reach the converged minimum energy under the base configuration setting. The vulnerability threshold on each core is also same as the base system which runs with uniform L2 partition and the base configuration for L1s. These settings are performed under the assumption that our approach should not be more vulnerable than the base system while improving the energy profile. This assumes that our system should be at least less vulnerable than the base system. In other words, we want our energy optimization process to be vulnerability-aware.

B. Vulnerability-aware Energy Reduction

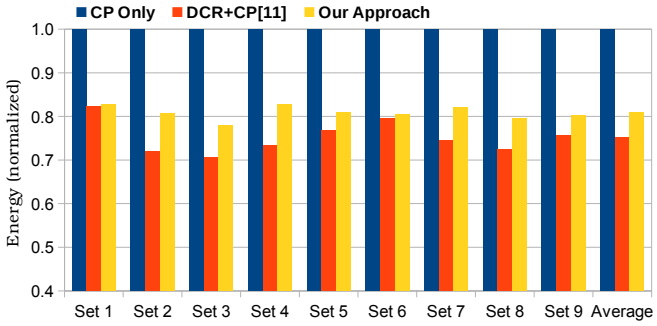
Figure 6 illustrates the comparison of vulnerability and energy consumption of the nine task sets in Table I. Here the

TABLE I: TASK SETS FROM THE MIBENCH [20] AND SPEC CPU2000 [21] BENCHMARKS

Task set	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Set 8	Set 9
Core 1	qsort vpr	mcf sha	applu lucas	mgrid FFT	mcf toast sha	mgrid parser gcc	vpr sha FFT	sha mcf untoast toast	gcc stringsearch parser dijkstra
Core 2	parser toast	gcc bitcount	dijkstra swim	dijkstra parser	gcc parser stringsearch	toast FFT mcf	CRC32 lucas untoast	applu gcc bitcount ammp	untoast mcf ammp bitcount
Core 3	untoast swim	patricia lucas	ammp FFT	CRC32 swim	patricia qsort vpr	bitcount ammp applu	mgrid bitcount qsort	lucas FFT CRC32 patricia	lucas patricia qsort vpr
Core 4	dijkstra sha	basicmath swim	basicmath stringsearch	applu bitcount	basicmath CRC32 ammp	qsort dijkstra patricia	applu parser stringsearch	vpr basicmath mgrid swim	basicmath toast applu CRC32



(a) Vulnerability



(b) Energy Consumption

Fig. 6: **Comparison of vulnerability and energy consumption for the cache hierarchy.**

vulnerability is the maximum vulnerability among four cores while energy consumption is the total energy consumption of all L1 caches and L2 partitions. The maximum vulnerability provides an indication of the overall reliability of the cache subsystem since all the cores are independent with its private L1 caches and designated L2 partition.

Figure 6a shows the results for vulnerability reduction. Compared with **CP Only**, our approach reduces vulnerability by up to 25.2% and on average 8.8%. Compared with [11], our approach achieves up to 73.9% reduction in vulnerability and 49.3% on average. Figure 6b shows the energy savings. Compared with **CP Only**, our approach reduces energy consumption by up to 22.2% and 19.2% on average. Compared with [11], our approach consumes on average 5.6% and up to 9.5% more energy. In summary, our vulnerability-aware energy optimization can significantly reduce energy (on average 19.2%) compared with the base system. Compared with the state-of-art approach for energy optimization, we gain significant vulnerability reduction (on average 49.3%) with minor energy overhead (on average 5.6%).

In order to understand the rationale of above improvement, we would like to analyze the optimal solutions returned by Algorithm 1 for two different tasks sets. Table II and Table III show the results of L2 partition factors and [IL1, DL1] cache configurations found by our approach for task set 1 and task set 9, respectively. Task set 1 has two tasks on each core, with a partition scheme of [2,2,1,3] ways dedicated for each core. Task set 9 has four tasks on each core, with a partition scheme of [2,2,2,2]. We can see that different tasks have very different L1 configurations, which shows the necessity of DCR to suit the unique needs of a task. For a certain task, the best [IL1, DL1]configurations depend not only on the task itself (i.e. its data access patterns), but also the L2 partition factor as well as the deadline and vulnerability threshold. There are a few tasks appearing in both Set 1 and Set 9. For benchmarks *qsort*, *vpr*, *parser*, and *toast*, they have the exact same L2 partition factor and L1 configurations for the two sets. For benchmark *untoast*, Set 1 and Set 9 have chosen different L1 configurations when Set 1 (Core 3) uses a partition factor of 1 and Set 9 (Core 2) uses a partition factor of 2. Because Set 9 assigns a larger partition factor, *untoast* can execute with smaller L1 cache sizes ([1KB, 1KB]) for reducing energy under the deadline and vulnerability constraints.

Vulnerability-constrained systems can tolerate up to certain vulnerability level due to its implemented mitigation solution. Therefore, existing energy-optimization techniques (such as [11]) are not applicable on them. For example, if a system can tolerate up to 20% more vulnerability compared to the base configuration, most of the energy savings (except for Set 1 and Set 7) are meaningless since they crossed the vulnerability threshold. In other words, apparent energy benefit of [11]

is not useful in practice. Therefore, our vulnerability-aware energy optimization approach is vital for multicore systems with vulnerability constraints.

TABLE II: TASK SET 1: CACHE CONFIG ($[c_I, c_D, w_k]$)

Set 1	**Core 1** $w_1 = 2$	**Core 2** $w_2 = 2$	**Core 3** $w_3 = 1$	**Core 4** $w_4 = 3$
Task 1	[4KB_4W_16B, 2KB_2W_32B] **qsort**	[2KB_2W_64B, 4KB_4W_16B] **parser**	[2KB_2W_32B, 2KB_2W_16B] **untoast**	[2KB_2W_64B, 2KB_2W_16B] **dijkstra**
Task 2	[1KB_1W_64B, 4KB_4W_16B] **vpr**	[4KB_1W_64B, 1KB_1W_16B] **toast**	[4KB_4W_32B, 2KB_2W_32B] **swim**	[1KB_1W_64B, 1KB_1W_32B] **sha**

TABLE III: TASK SET 9: CACHE CONFIG ($[c_I, c_D, w_k]$)

Set 9	**Core 1** $w_1 = 2$	**Core 2** $w_2 = 2$	**Core 3** $w_3 = 2$	**Core 4** $w_4 = 2$
Task 1	[1KB_1W_64B, 2KB_2W_16B] **gcc**	[1KB_1W_64B, 1KB_1W_16B] **untoast**	[4KB_4W_16B, 2KB_2W_32B] **lucas**	[1KB_1W_64B, 4KB_4W_16B] **basicmath**
Task 2	[4KB_1W_32B, 4KB_4W_16B] **stringsearch**	[1KB_1W_32B, 1KB_1W_16B] **mcf**	[1KB_1W_64B, 1KB_1W_16B] **patricia**	[4KB_1W_64B, 1KB_1W_16B] **toast**
Task 3	[2KB_2W_64B, 4KB_4W_16B] **parser**	[1KB_1W_64B, 1KB_1W_16B] **ammp**	[4KB_4W_16B, 2KB_2W_32B] **qsort**	[1KB_1W_64B, 1KB_1W_16B] **applu**
Task 4	[2KB_2W_64B, 2KB_2W_16B] **dijkstra**	[1KB_1W_32B, 1KB_1W_32B] **bitcount**	[1KB_1W_64B, 4KB_4W_16B] **vpr**	[2KB_1W_32B, 2KB_2W_16B] **CRC32**

VI. CONCLUSION

Cache vulnerability is a major concern in embedded systems design due to increasing cache size and soft errors. While both vulnerability and energy optimization have received considerable attention in recent years, there are no existing works on vulnerability-aware energy optimization for multicore systems. In this paper, we presented a vulnerability-aware energy optimization technique for real-time multicore systems. Our approach integrates dynamic cache reconfiguration (DCR) of private L1 caches and cache partitioning (CP) of the shared L2 cache. L2 CP is effective in reducing inter-core interference, while applying L1 DCR can further reduce the energy consumption under the performance and vulnerability constraints. Our task profiling technique based on the independence between tasks can drastically reduce the complexity of design space exploration. Our proposed algorithm uses dynamic programming by discretizing the energy values, which can efficiently search the space to find optimal L1 cache configurations for each task and L2 cache partition factors for each core. Experimental results demonstrated that we can achieve 19.2% average energy savings compared with the base system, while drastically reduce the vulnerability (49.3% on average) compared to the existing approaches.

VII. ACKNOWLEDGEMENTS

This work was partially supported by the National Science Foundation under grant CNS-1526687. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agency.

REFERENCES

[1] C. Zhang, F. Vahid, and W. Najjar. A highly configurable cache for low energy embedded systems. ACM Trans. Embed. Comput. Syst. 4, 2 (May 2005), 363-387.

[2] A. Gordon-Ross and F. Vahid. A Self-Tuning Configurable Cache. ACM/IEEE Design Automation Conference, San Diego, CA, 2007, pp. 234-237.

[3] W. Wang, P. Mishra, and A. Gordon-Ross. Dynamic Cache Reconfiguration for Soft Real-Time Systems. ACM Trans. Embed. Comput. Syst. 11, 2, Article 28 (July 2012), 31 pages.

[4] W. Wang and P. Mishra. Dynamic Reconfiguration of Two-Level Caches in Soft Real-Time Embedded Systems. IEEE Computer Society Annual Symposium on VLSI, Tampa, FL, 2009, pp. 145-150.

[5] P. Y. Hsu and T. Hwang. Thread-criticality aware dynamic cache reconfiguration in multi-core system. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Jose, CA, 2013, pp. 413-420.

[6] S. Mittal, Y. Cao and Z. Zhang. MASTER: A Multicore Cache Energy-Saving Technique Using Dynamic Cache Reconfiguration. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 22, no. 8, pp. 1653-1665, Aug. 2014.

[7] A. Settle, D. Connors, E. Gibert, and A. Gonzlez. A dynamically reconfigurable cache for multithreaded processors. J. Embedded Comput. 2, 2 (April 2006), 221-233.

[8] D. Kaseridis, J. Stuecheli and L. K. John. Bank-aware Dynamic Cache Partitioning for Multicore Architectures. International Conference on Parallel Processing, Vienna, 2009, pp. 18-25.

[9] R. Reddy and P. Petrov. Cache partitioning for energy-efficient and interference-free embedded multitasking. ACM Trans. Embed. Comput. Syst. 9, 3, Article 16 (March 2010), 35 pages.

[10] N. N. Sadler and D. J. Sorin. Choosing an Error Protection Scheme for a Microprocessor's L1 Data Cache. International Conference on Computer Design, San Jose, CA, 2006, pp. 499-505.

[11] W. Wang, P. Mishra and S. Ranka. Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems. ACM/EDAC/IEEE Design Automation Conference (DAC), New York, NY, 2011, pp. 948-953.

[12] G.-H. Asadi, V. S. Mehdi, B. Tahoori, and D. Kaeli. Balancing Performance and Reliability in the Memory Hierarchy. IEEE International Symp. on Performance Analysis of Systems and Software, 2005 (ISPASS), pp. 269-279.

[13] Y. Cai, M. T. Schmitz, A. Ejlali, B. M. Al-Hashimi and S. M. Reddy. Cache size selection for performance, energy and reliability of time-constrained systems. Asia and South Pacific Conference on Design Automation, 2006., Yokohama, 2006, pp. 6 pp.-.

[14] M. Maghsoudloo and H. Zarandi. Cache vulnerability mitigation using an adaptive cache coherence protocol. The Journal of Supercomputing, June 2014, Volume 68, Issue 3, pp 10481067.

[15] Y. Huang and P. Mishra. Reliability and energy-aware cache reconfiguration for embedded systems. International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, 2016, pp. 313-318.

[16] W. Wang, P. Mishra and S. Ranka, Dynamic Reconfiguration in Real-Time Systems - Energy, Performance, Reliability and Thermal Perspectives, ISBN: 978-1-4614-0277-0, Springer, July 2012.

[17] Yuanwen Huang, System-on-Chip Vulnerability Analysis and Mitigation Techniques, Ph.D. Dissertation, University of Florida, June 2017.

[18] N. Binkert et al. The gem5 simulator. SIGARCH Comput. Archit. News 39, 2, August 2011.

[19] T. David et al. CACTI 4.0. Technical Report HPL-2006-86, HP Laboratories Palo Alto, 2006.

[20] Guthaus, Matthew R., et al. MiBench: A free, commercially representative embedded benchmark suite. WWC, 2001.

[21] CPU2000. SPEC CPU2000. http://www.spec.org/cpu2000.