Flexible ECC Management for Low-Cost Transient Error Protection of Last-Level Caches

Abstract:

The conventional error correcting code (ECC) schemes for caches are based on a fixed mapping between cache data words and ECC check bits, and fixed ECC word granularity. This leads to inefficient usage of the ECC check bits. We propose to manage the check bits flexibly for low-cost error protection of last-level caches. The proposed ECC schemes work at the word level, whereas the conventional ECC schemes work at the cache line or set level. The proposed schemes protect only dirty words with ECC check bits using a flexible mapping. Moreover, the proposed schemes utilize variable ECC word granularities. Dirty (modified) words that are unlikely to be modified further before being evicted are collectively protected with a larger ECC word granularity. The proposed schemes reduce DRAM and data bus energy overheads by 28% and 45%, respectively, with the same area overhead as previously proposed competitive schemes. Our schemes show more energy reduction results for multicore systems without noticeable performance degradation. The proposed architecture of this paper analysis the logic size, area and power consumption using Xilinx 14.2.

Enhancement of the project:

Existing System:

To utilize ECC check bits more efficiently, we propose flexible ECC management schemes for last-level caches (LLCs), which consume a sizable area for the ECC check bits. Unlike the conventional error protection schemes that maintain SEC-DED bits for all the cache words of all the cache lines, the proposed schemes protect only dirty (modified) cache words by using a flexible mapping between dirty words and ECC check bits. The proposed schemes basically utilize parity codes as error detecting codes (EDCs), which are more beneficial than the conventional Hamming SEC-DED codes in terms of latency and energy overheads, and utilize SEC-DED codes only for error correction. Though parity codes have lower error detection
capability than SEC-DED codes, the proposed schemes have almost the same error protection capability as the conventional scheme due to low probability of multibit errors.

From our experiments using the SPEC CPU2000 benchmarks, many cache lines in LLC are clean (not modified) and many cache words of dirty cache lines are also clean. To effectively utilize this observation, we encode the locations of dirty cache lines and their dirty words, and store them in our unique error protection structure. In addition, we also observed that most LLC lines are unlikely to be modified further after a small number of modifications. Thus, if certain cache lines are predicted not to be modified further, the proposed schemes protect two or four cache words belonging to these cache lines together with larger SEC-DED word granularities. Consequentially, the proposed schemes can significantly reduce the area overhead with relatively lower overheads than the previously proposed competitive scheme.

Disadvantages:

- error protection is less

Proposed System:

To protect against the transient errors, current memory systems commonly adopt SEC-DED codes and protect all cache lines equally, assuming that all data bits have the same vulnerability to transient errors. As a result, error protection codes consume 12.5% area overhead of the entire cache size when 8-bit SEC-DED codes are used. To reduce the area overhead, we devise a flexible ECC management architecture employing a word-level data protection based on the following three important observations.

First, not all cache lines are dirty. Because the number of write operations is smaller than that of read operations in most applications and owing to natural eviction of dirty cache lines, the fraction of dirty cache lines at a specific time is not large. Second, not all cache words in a dirty cache line are dirty. Although a cache line becomes dirty as a result of a write operation, not all cache words of the cache line are modified. Finally, not all cache lines are repeatedly modified. Most LLC lines are not frequently written during their lifetimes because most memory requests are satisfied by higher-level caches.

Flexible ECC Check Bits Mapping:
We propose a word-level data protection technique that provides different error protection mechanisms for clean words and dirty words. Because clean data have their duplicate in a lower level of the memory hierarchy, they can be recovered from a transient error by refetching the duplicate. Therefore, EDC is sufficient to protect clean data; dirty data also require EDC and ECC as well. The proposed scheme functions at the word level with a flexible ECC check bit mapping, which will be explained in the following discussion, whereas the scheme proposed in works at the line level with no flexible mapping of ECC check bits.

<table>
<thead>
<tr>
<th>Cache Data Array</th>
<th>ECC Check bit Area</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Way 0</strong></td>
<td><strong>Way 1</strong></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><img src="image1.png" alt="Image" /></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 1. Conceptual comparison of (a) baseline architecture, (b) DLn, and (c) our flexible ECC management. Gray rectangles: dirty words.

Fig. 1 shows the conceptual views of the conventional baseline architecture, DLn (n dirty line), and the proposed architecture. In Fig. 1, we assume a two-way set associative cache and each cache line is composed of four cache words. Each number indicates a cache word and its corresponding ECC check bits. In other words, a cache word is protected by an ECC check bit whose number is the same with that of the cache word. Gray rectangles indicate dirty cache words, whereas clean cache words are colored white.

On a read operation from a cache line, the parity codes are accessed in parallel with data to check any transient error [Fig. 2(a)]. If an error is detected in a clean cache line, the erroneous data are replaced with the correct data stored in the main memory. If the cache line is dirty, the error is corrected using the ECC check bits stored in the ECC string.
In contrast, for a write access, the proposed mechanism checks whether the cache line being accessed is clean [Fig. 2(b)]. On a clean line write, the number of existing dirty cache lines must be verified to determine if a newly generated dirty cache line violates the first restriction; at most, n dirty lines can simultaneously exist in a cache set. If this does not violate the first restriction, ECC words for both existing dirty cache words and newly written cache words are counted. On a clean line write, the number of existing dirty cache lines must be verified to determine if a newly generated dirty cache line violates the first restriction; at most, n dirty lines.
The Master of IEEE Projects

can simultaneously exist in a cache set. If this does not violate the first restriction, ECC words for both existing dirty cache words and newly written cache words are counted.

Variable ECC Word Granularity

We also propose a second scheme that adopts variable ECC word granularities. This scheme is designed to alleviate the additional cleaning write backs caused by the restrictions of the ECC string. The principal concept of this scheme is to exploit the benefit from large granularity ECC words. For frequently modified cache lines, ECC words of small granularity are used; those of large granularity are used for stable cache lines.

ECC that utilizes Hamming SEC-DED codes requires eight check bits to protect 64 bits of data, nine check bits for 128 data bits, and ten check bits for 256 data bits. That is, protecting data with a larger ECC word granularity is beneficial in terms of area overhead; one additional check bit can manage double the number of data bits. Consequently, we can protect additional dirty cache words with the same numbers of ECC words by exploiting the large granularity of the ECC word, thereby reducing the number of cleaning write backs incurred by REA.

Advantages:

- Increasing the size of on-chip memories
- Optimal error protection

Software implementation:

- Modelsim
- Xilinx ISE