A Performance Degradation Tolerable Cache Design by Exploiting Memory Hierarchies

Abstract:

Performance degradation tolerance (PDT) has been shown to be able to effectively improve the yield, reliability, and lifetime of an electronic product. The focus of PDT is on the particular performance degrading faults (pdef) that only incur some performance degradation of a system without inducing any computation errors. The basic idea is that as long as the defective chips containing only the pdef can provide acceptable performance for some applications, they may still be marketable. Critical issues of PDT to be addressed include the portion of the pdef in a faulty chip and their induced performance degradation. For a typical cache design, most of the possible faults are not pdef. In this brief, we propose a cache redesign method, called PDT cache, where all functional faults in the data-storage cells of a cache (major part of the cache) can be transformed into pdef. By transforming this large number of faults into pdef, a faulty cache becomes much more likely to be still marketable. The proposed design exploits the existing hardware resources and the inherent error resilience scheme to reduce the incurred hardware overhead. The logic synthesis results show that the incurred hardware overhead is only 6.29% for a 32-kB cache. We also evaluate the induced performance degradation under various fault densities using the CPU2000 and CPU2006 benchmark programs. The results show that for a 32-kB cache design, when the fault density is

Enhancement of the project:

Existing System:

To support PDT, the issues of: 1) the fraction of pdef in the target design and 2) the induced performance loss by pdef need to be addressed. Although init has been shown that the notion of pdef is applicable to the branch direction predictor, this may not always be the case for other performance-enhancement modules in a processor design. For example, the cache design that occupies a larger area in recent processors is commonly used to accelerate the instructions/data
access process by copying a subset of the instructions/data stored in the slow-speed main memory to a smallsize and high-speed memory. In advanced processors, caches are usually designed in hierarchy, such as level 1 (L1), level 2 (L2), and even level 3 (L3) caches [4]. When the processor is going to load/store certain data from/to the memory, the availability of data in upper level caches (e.g., L1 cache) will be checked. If yes, the data are quickly retrieved/updated. Otherwise, more computation cycles are required to see if the target data exist in a lower level cache (e.g., L2 cache). In order to increase the probability to find the required data, a lower level memory usually has a larger size. For example, the size of L2 cache is usually larger than that of L1 cache. In the worst case, the required data do not exist in any levels of the cache (e.g., when the processor is rebooted). In this case, the slow main memory or hard disk would be accessed, inducing a significant performance impact.

It can be seen that most of the faults in the cache design are quite likely to contaminate the stored data, and thus are functional faults, i.e., those faults that will cause functional errors in a processor. As a result, intuitively there exist almost no pdef in a typical cache design. By increasing the total number of pdef as well as reducing the performance loss, the effective yield of cache designs can be improved. In the literature, there are several available methods that can be used to protect a cache design, including remapping, error correcting code (ECC), and built-in self-repair. However, the fault tolerability of these methods is fixed when the architecture is determined. As a result, the fault-tolerability is limited by the amount of the added redundancy or the amount of the resources that are still functional. That is, the cache may still suffer from the data contamination problem if the actual fault density is larger than the tolerable limit.

Disadvantages:

- fault tolerability is not flexible

Proposed System:

Cache Access Mechanism

To maximize the total number of pdef in a cache design, we modify the access mechanism of a typical cache, explained as follows. Typically when receiving the access request issued by the processor, the cache will check the availability of the requested data. This is done by:
The Master of IEEE Projects

1) Accessing a specific cache word indexed by parts of the memory address;
2) Checking if this word is valid (by examining a particular bit called valid bit);
3) Examining if the specific defined part of the memory address, called tag, is the same as that stored in the accessed cache word.

If yes for both 2) and 3), a cache hit occurs, and the requested data are read/written quickly. Otherwise, a cache miss is said to occur and the processor will access the requested data in lower level of memory (e.g., L2 cache or eventually the main memory).

Cache Access State Diagram

The cache access state diagram of the PDT cache is shown in Fig. 1. When the processor is attempting to access the cache, the Req_P signal is activated. A particular FMOut signal is generated by the faulty map to indicate whether the cache word to be accessed is erroneous or not. When an error-free word is to be accessed (the FMOut signal is deactivated), the cache either sends back/write the required data soon (cache hit) or needs to access lower level cache (cache miss), depending on whether the required data are located in current-level cache.
Hardware Implementation

The architecture of our PDT cache is shown in Fig. 2. As indicated, a memory BIST is used to test the valid and tag bits (the sets of these bits are called valid RAM and tag RAM, respectively) as well as the data RAM. The test results are recorded in the faulty map. In the case that data replacement is required (i.e., when cache miss occurs in normal conditions), the least recently used (LRU) hardware is activated to locate the LRU word for replacement. The controller takes care of all the operations under normal and faulty conditions.

![Architecture of PDT cache](image)

Fault Effect Analysis

We discuss the effects of faults in the components of our PDT cache. Accordingly, we evaluate the distribution of pdef in our cache.
1) Data Storage Part (Tag RAM, Valid RAM, and Data RAM): Once any faults occur in these components, the correct data can still be provided by lower level memory with more computational cycles. The functional correctness of the cache can thus be guaranteed.

2) LRU: Faults in this component would only result in different cache words to be replaced, which may induce more cache misses and thus some performance loss. Functional correctness of the cache is not affected.

3) Faulty Map: When faults occur in this component, our cache protection scheme may become invalidated. For example, for a stuck-at-0 fault, the corresponding cache words may be fetched by the processor, even if the word is actually erroneous. On the other hand, a stuck-at-1 fault may disable fetching of an error-free cache word, resulting in unnecessary cache misses.

4) Controller, Logic Circuitry, and Memory BIST: Clearly, faults in these components are quite likely to induce functional errors in the cache, making wrong or erroneous data to be fetched.

Advantages:

- fault tolerability is flexible

Software implementation:

- Modelsim
- Xilinx ISE