Microelectronics Reliability 102 (2019) 113494

Contents lists available at ScienceDirect

Microelectronics Reliability

journal homepage: www.elsevier.com/locate/microrel

Reducing false positives due to double adjacent errors in instruction TLBs

A. Sánchez-Macián[a,*], L.A. Aranda[a], P. Reviriego[b], J.A. Maestro[a]

ARIES Research Center, Universidad Antonio de Nebrija, C/ Pirineos 55, 28040 Madrid, Spain
Department of Telematics Engineering, Universidad Carlos III de Madrid, Avda. Universidad 30, 28911 Leganés, Spain

ARTICLE INFO

Keywords:
Error detection
Fault tolerance
Reliability
Translation lookaside buffer

ABSTRACT

Translation lookaside buffers (TLBs) are cache structures used to make the translation process between virtual pages and physical pages faster. Instruction TLBs store the virtual page number of the last accessed instruction memory pages and the corresponding physical page numbers. Single and double adjacent errors, which are common in harsh environments, may affect TLBs. This paper presents a technique to provide instruction TLB resilience to these single and double adjacent errors without additional storage requirements, by taking advantage of the spatial locality principle that is present in program execution.

1. Introduction

Operating systems allocate programs in the virtual memory space, which is divided into virtual pages. These pages are then mapped to physical memory in a dynamic way, using page tables located in main memory. To speed up the translation process, a cache is used to store the last accessed virtual memory page numbers and the related physical page numbers (PPNs). This cache is called Translation Lookaside Buffer (TLBs) [1]. Data and instructions are typically separated in different TLBs. TLBs can be implemented using a content-addressable memory (CAM) [2] where the virtual page numbers (VPNs) are stored, and a Random Access Memory (RAM) that keeps the information of the corresponding physical page numbers.

Soft errors (temporary errors on one or more bits) can affect entries of the TLB CAM. This can result in false negatives, which may impact the performance of the system, and false positives, where a wrong translation is performed and wrong physical page numbers (and related instructions) are retrieved. The consequences of this include hard faults, a system freeze or silent data corruption (SDC).

Bit flips may be produced by different sources. Radiation in harsh environments can produce a Single Event Upset (SEU), where a single bit is corrupted. Multiple Cell Upsets (MCUs) can also occur, affecting several bits. MCUs are, in many cases, adjacent to each other [3]. When those bits correspond to the same word, it is considered a Multiple Bit Upset (MBU). Adjacent errors are a typical scenario of events produced by radiation. When a single particle strike produces several bit flips, these are usually adjacent, since the particle affects a limited region. Another scenario is when several independent particles produce different bit flips: in this case, there is no spatial correlation and errors do not have to be adjacent. Since error events are seldom, it is usual to focus on the single event failure case. Downsizing in process technology node and spacing between cells is a key factor in susceptibility to Multiple Bit Upsets. Reducing supply voltage to limit power consumption also increases the probability of a MBU by decreasing the charge needed by the different particles to cause this effect [4, 5].

Different solutions to protect the TLB against errors have been proposed, including the use of parity, error correcting codes (ECCs) or spatial redundancy (e.g. [6-8]).

This paper presents a solution to provide instruction TLB resilience to single and double adjacent errors occurring in the CAM without additional storage requirements, by taking advantage of the spatial locality principle that is present in program execution.

2. Fault tolerance and instruction TLBs

As previously pointed out, a common implementation of a TLB includes a CAM that stores the virtual page numbers and a RAM for the physical page numbers. Other information such as valid bits or permission bits are also stored in the TLB. In this scenario, when an instruction address needs to be accessed, the virtual page has to be mapped to a physical page. Thus, the queried virtual page number is checked against the entries of the TLB CAM to verify if the mapping is cached from a previous request. An error in an entry of the CAM can produce the following consequences:

- False negatives (Fig. 1): when the error affects the entry that holds

* Corresponding author.
E-mail addresses: asanchep@nebrija.es (A. Sánchez-Macián), laranda@nebrija.es (L.A. Aranda), revirieg@it.uc3m.es (P. Reviriego), jmaestro@nebrija.es (J.A. Maestro).

https://doi.org/10.1016/j.microrel.2019.113494
Received 11 April 2019; Received in revised form 8 July 2019; Accepted 13 August 2019

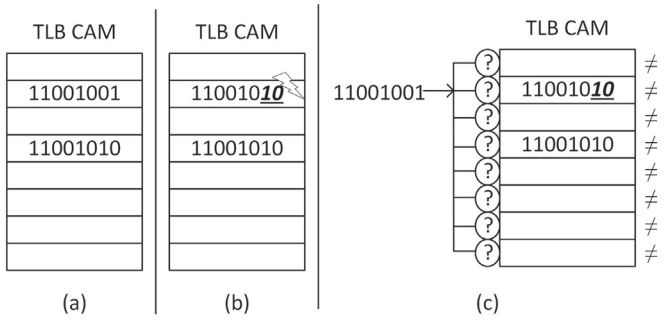

Fig. 1. Double adjacent bit flip producing a false negative. (a) Original content. (b) Content after error. (c) False negative in the second entry.

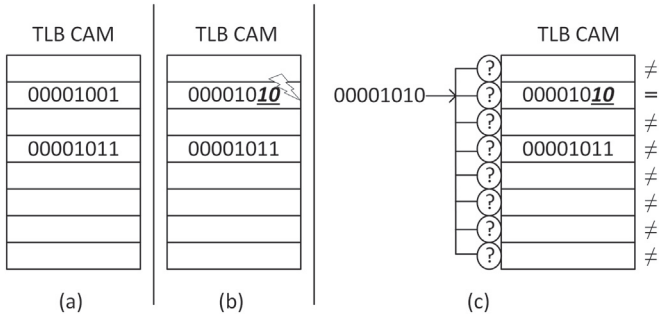

Fig. 2. Double adjacent bit flip producing a false positive. (a) Original content. (b) Content after error. (c) False positive in the second entry.

the requested virtual page number. The value no longer matches against the entry, a miss is produced and the information has to be retrieved again from the page table (or an intermediate cache structure). It may affect performance.

- False positives (Fig. 2): when the error converts an existing entry from the CAM into a different virtual page number that is requested in a later access. Thus, a match is found for a wrong entry, and the associated physical page number is retrieved, executing an instruction which is different from the one expected. A hard fault, system freeze or silent data corruption are possible effects of this false positive.

Notice that an error not always produces a false positive or a false negative as the corrupted entry may be evicted from the CAM before the key is queried, or the resulting virtual page number may not be used by the program after the error. To provide protection against soft errors in TLBs, different schemes have been proposed. A common solution uses a parity bit to detect single errors and then invalidate the entry of the TLB [6]. When more advanced protection is required, error correction can be performed by using ECCs, which require additional storage and encoding/decoding capabilities. Some of these solutions use extra backup structures (e.g. [7]) or special matchlines (e.g. [8]) to reduce the delay introduced by the decoding.

An alternative is to ignore false negatives, as their performance effect in the system is, in general, neglectable (it depends on the fluence of errors) and focus on avoiding false positives. There is, typically, a spatial locality effect in the execution of the instructions in a program that can be used for the protection of the instruction TLB. The spatial locality principle implies that given a memory access, the following references will have addresses that are likely close to the first one. That is to say, if a word is referenced, then words whose addresses are nearby will be referenced soon. Based on this principle, the authors proposed a scheme [9] to enhance the resilience of an instruction TLB CAM against single errors without extra storage requirements. This was extended to double adjacent errors when an extra parity bit was already available for each entry of the memory.

The spatial locality principle has already been used with different approaches to protect caches, by taking advantage of the replication of tag information (e.g. [10-12]). The solution in [10] checks if adjacent cache lines have the same tag bits and includes extra information to be able to use those adjacent tags if an error is detected in a line. Authors in [11] use a small Tag Replication Buffer to duplicate most recently accessed tag entries, and a pointer from the original tag array to point to its replica. The scheme proposed in [12] uses replication as well, however it also modifies the replacement algorithm and keeps track of the incoming and evicting lines to minimize the probability of an MBU affecting tag and replica simultaneously. Notice that these solutions require additional overhead in terms of replica bits and/or pointers, apart from specialized hardware resources.

Next, a technique based on the scheme presented in [9] is developed to reduce the number of false positives in the TLB when single or double adjacent errors occur, without requiring any parity bit.

3. Proposed scheme

The proposed scheme takes advantage of the spatial locality principle of instruction execution to reduce the number of false positives. To do so, the virtual page numbers are encoded in a special way before being stored into the TLB CAM. In the encoding process, the most significant bit (MSB) and the bit next to the MSB (MSB-1) are recomputed, while the rest of the bits do not change. The encoded version of the virtual page number (VPN′) is calculated as follows when the number of bits of the VPN is odd:

$$VPN'[i] = VPN[i] \quad \forall\, i \neq (MSB, MSB-1)$$
$$VPN'[MSB] = VPN[MSB] \oplus VPN[MSB-2] \oplus \ldots \oplus VPN[0]$$
$$VPN'[MSB-1] = VPN[MSB-1] \oplus VPN[MSB-3] \oplus \ldots \oplus VPN[1]$$

If the number of bits of the VPN is even, then:

$$VPN'[i] = VPN[i] \quad \forall\, i \neq (MSB, MSB-1)$$
$$VPN'[MSB] = VPN[MSB] \oplus VPN[MSB-2] \oplus \ldots \oplus VPN[1]$$
$$VPN'[MSB-1] = VPN[MSB-1] \oplus VPN[MSB-3] \oplus \ldots \oplus VPN[0]$$

As commented, all the bits except the MSB and MSB-1 keep their original values. This is similar to the scheme presented in [9], but now two bits are modified in the encoding process. The parity for the odd bits and the one for the even bits are calculated separately and stored in the two most significant bit positions in a way that each of these positions also contributes to the parity stored in itself. This encoding process increments the Hamming distance between close addresses, while reducing the distance between remote addresses. An example of the encoding process is shown next for a set of consecutive addresses. Notice that the MSB and MSB-1 bit positions are already holding valid values, i.e., they are not supposed to be empty and, consequently, they are not equivalent to traditional parity bits.

$$01111110 \Rightarrow MSB(0 \oplus 1 \oplus 1 \oplus 1),\ MSB-1(1 \oplus 1 \oplus 1 \oplus 0)$$
$$\Rightarrow 11111110$$
$$01111111 \Rightarrow MSB(0 \oplus 1 \oplus 1 \oplus 1),\ MSB-1(1 \oplus 1 \oplus 1 \oplus 1)$$
$$\Rightarrow 10111111$$
$$10000000 \Rightarrow MSB(1 \oplus 0 \oplus 0 \oplus 0),\ MSB-1(0 \oplus 0 \oplus 0 \oplus 0)$$
$$\Rightarrow 10000000$$
$$10000001 \Rightarrow MSB(1 \oplus 0 \oplus 0 \oplus 0),\ MSB-1(0 \oplus 0 \oplus 0 \oplus 1)$$
$$\Rightarrow 11000001$$

Fig. 3 shows a TLB CAM filled with the encoded values of virtual page numbers 0 to 3 and 5 to 8 (VPN 00000000 ⇒ VPN′ 00000000, VPN 00000001 ⇒ VPN′ 01000001, etc.). A query is performed for VPN 4 (VPN 00000100 ⇒ VPN′ 01000100). Since this value is not in the table, no match is produced. However, notice that no match would have been produced either in the case that the TLB had been affected by a

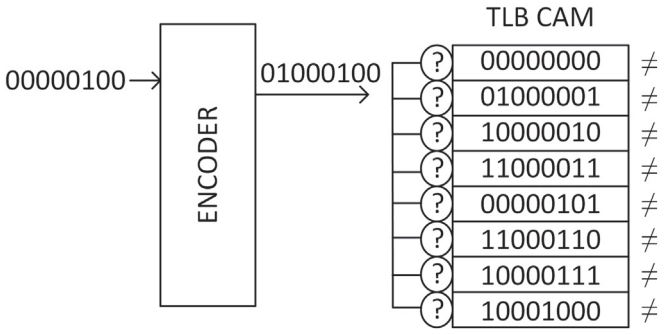

Fig. 3. Example of a TLB CAM with the proposed scheme.

single-bit error, because all the encoded VPNs are, at least, at a Hamming distance of 2. Therefore, a single error would never produce a false positive in this example (neighbour addresses). Due to the distribution of bits that results from the even and odd parity encoding, double adjacent errors would not produce a false positive either. It is also important to point out that it is not necessary to perform decoding of the encoded VPNs. The query is performed by encoding the incoming request and comparing it to the encoded entries of the TLB CAM.

4. Evaluation

To verify the solution, a fault injection campaign was performed. To do so, a dynamic instrumentation tool, Pin [13], was used to record the VPNs requested by a subset of programs from the CPU2006 SPEC benchmark [14], when executing them in an Intel x64 architecture running Windows 7. Notice that, for this particular version of the operating system, only 44 bits are assigned to the program virtual memory address. The Virtual Page Number has 32 bits. The page offset has 12 bits (i.e. page size of 4 KB) that are dropped in the injection experiments, focusing only in the 32-bit VPNs, since only the latter are actually stored in the TLB CAM.

A simulation was performed using this information to feed the TLB CAM and to compare a non-protected scenario and the proposed solution. Statistical fault injection was performed to select when (specific moment in the program execution) and where (which entry of the CAM and which bit(s) of the entry) to inject the errors. A total of 27,000 injection experiments were performed per benchmark and scenario on a TLB CAM with 8-entries (such as the Level 0 TLB in the AMD Zen microarchitecture [15]) using an LRU (Least Recently Used) replacement algorithm. Each entry stores a 32-bit VPN, matching the width of the ones for the architecture where Pin was executed. The simulation was performed step by step, injecting one or more bit errors in a TLB CAM entry when the specific trace line was reached. Then, execution continued until a false positive was observed or the faulty entry was evicted. Results were logged, TLB was emptied and the simulation restarted from the beginning for a different injection configuration (formed by the trace line, the entry number and the bit to be flipped).

Five integer (429.mcf, 445.gobmk, 456.hmmer, 464.h264ref and 471.omnetpp) and five floating point benchmarks (433.milc, 444.namd, 453.povray, 470.lbm and 483.xalancbmk) were selected. Results for single injection, double adjacent error injection, double error injection and triple error injection are presented in Table 1. For clarification purposes, it should be mentioned that double and triple error injections have been performed in random non-adjacent bits of the same VPN entry. The experiments show that, when no protection is used, there is a percentage of single errors that can produce a false positive (0.3 to 2.2% approximately) that depends on the benchmark. Double-adjacent errors have also a similar effect on false positives (up to 2.2%). In both cases, the proposed solution is able to avoid the false positives that are produced in the scenario without protection. This is validated for each of the ten benchmarks. When comparing the double (random) and triple error injection experiments for the unprotected scenario and the proposed scheme, the results are better in all cases, although the number of false positives are reduced for the unprotected scenario. This is because the probability of obtaining a remote address that does not follow the locality principle is much higher as more bits of the same VPN are modified. For double errors, the proposed scheme will work whenever the errors affect an even and an odd bit which statistically should be approximately 50% of the cases. For triple errors, either the MSB or the MSB-1 bits will change, behaving in a similar way to the single error scenario.

Table 1
Percentage of false positives in the non-protected scenario and the proposed scheme. SE = single error injection; DAE = double-adjacent error injection; DE = double error injection in the same VPN entry; TE = triple error injection in the same VPN entry.

Benchmark	Technique	SE	DAE	DE	TE
429.mcf	Non-protected	0.66%	1.06%	0.11%	0%
	Proposed	0%	0%	0.10%	0%
455.gobmk	Non-protected	0.30%	0.25%	0.07%	0%
	Proposed	0%	0%	0.02%	0%
456.hmmer	Non-protected	0.98%	0.78%	0.16%	0.04%
	Proposed	0%	0%	0.08%	0%
464.h264ref	Non-protected	0.96%	1.32%	0.19%	0.02%
	Proposed	0%	0%	0.10%	0%
471.omnetpp	Non-protected	0.88%	0.75%	0.09%	0.02%
	Proposed	0%	0%	0.06%	0%
433.milc	Non-protected	1.49%	2.08%	0.13%	0.02%
	Proposed	0%	0%	0.05%	0%
444.namd	Non-protected	2.20%	2.23%	0.22%	0.01%
	Proposed	0%	0%	0.11%	0%
453.povray	Non-protected	0.45%	0.68%	0.13%	0.02%
	Proposed	0%	0%	0.09%	0%
470.lbm	Non-protected	0.27%	0.04%	0.10%	0%
	Proposed	0%	0%	0.04%	0%
483.xalancbmk	Non-protected	0.37%	0.58%	0.09%	0.01%
	Proposed	0%	0%	0.04%	0%

The proposed technique works as long as the spatial locality holds. Even if this is not the case, the proposed solution should still behave better (with some false positives) as the probability of simultaneously having two encoded remote addresses within the restricted Hamming distance in the TLB is still very low. These false positives could happen, for example, if an error is produced in a VPN entry (e.g. the encoded version of 01111111, represented as 10111111, is converted into 1011110<u>1</u>) and a procedure call is performed that moves to a remote address in the proximity of that particular VPN in error before it is evicted (e.g. it jumps to VPN 11111101 which is encoded as 10111101).

Notice that errors may produce a double match when a bit flip turns one of the stored VPNs into a value that it is already present in the CAM. In those cases, as the injections are performed randomly, the entry in error can be any of the two matching values. To address this possibility, we have considered a priority encoder where the first match in the table is selected. This has been done for the two solutions being compared so the effect is the same for both of them.

The solution has been implemented (Fig. 4) in a Xilinx FPGA. In particular, a Nexys 4 DDR board has been selected to compare the proposed solution to the single error mitigation technique presented in [9], for a 32-bit VPN. Both techniques work with CAMs that do not include parity storage by default, and therefore they are comparable. The results are shown in Table 2. The proposed solution provides better protection, slightly reduces area utilization (in terms of used LUTs) and has the same delay (in LUT levels traversed) as the reference solution. However, there is a drawback to this approach when compared to the single error technique in [9]. Overloading the MSB-1 bit reduces the distance between remote addresses when the Hamming distance is at least 2. Thus, there is a higher probability of false positives as the distance between VPNs in the CAM increases. Consequently, if a large

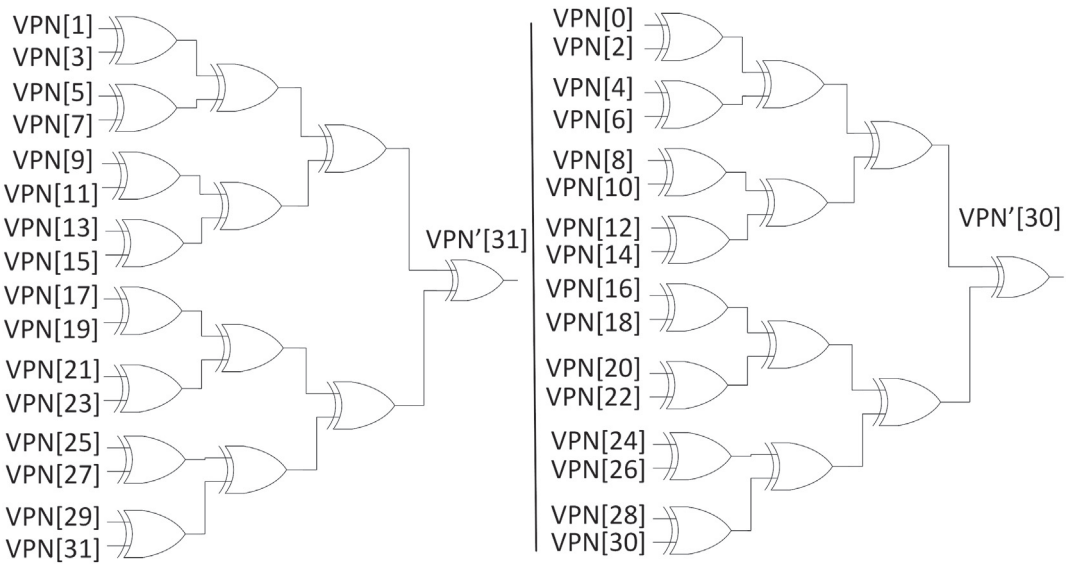

Fig. 4. XOR parity tree for 32-bit VPN encoding.

number of double errors is expected, then the technique proposed in this paper is an interesting approach since it provides double error mitigation with very limited area and delay overheads. On the other hand, if most of the expected events are single errors, then the approach presented in [9] would be the simplest approach to mitigate them.

An extended Hamming Single Error Correction, Double Error Detection (SEC-DED) code is also included in Table 2 as a reference. Notice that SEC-DED codes would remove all the false positives due to double and single errors but need extra storage space (7 parity bits instead of no extra bits needed by the proposed technique) and have a significant penalty in terms of area (more than 3×) and delay (2×) overhead.

Notice that TLB lookups are in the critical path of every memory access. Thus, minimizing the impact of a solution in terms of delay is important. To provide the required protection, the proposed solution introduces a delay when compared to an unprotected version of the TLB, but this delay is smaller than that of a SEC-DED code. This solution also behaves better in terms of delay that adding a parity bit as the number of bits used to compute the parity information in the presented scheme is half of those used by that standard solution. In the proposed scheme, the MSB is overloaded with the parity calculated by odd bits, and the MSB-1 with the even bits, while the single parity bit solution uses all the bits to generate the new parity.

5. Applicability

The purpose of the proposed scheme is to increase the "intrinsic" reliability of the system, meaning the reliability achieved without incurring storage overhead (as using parity bits or ECCs does). Therefore, with the proposed solution, the system provides a larger "base" reliability than in the standard case with no solution at all. But this does not mean that this proposal is a "final" solution, i.e. intended to strictly work on its own. Other solutions can be implemented on top of it in order to achieve higher reliability levels.

As an example, adding a parity bit to the scheme is straightforward. While the parity bit protects against single errors (and an odd number of errors), the proposed scheme adds protection for false positives due to double-adjacent errors and also reduces the number of those produced by double random errors. Moreover, it comes without any extra cost in terms of area or performance when added to the parity bit just by rearranging the way parity is calculated. For example, for 32 bits, the proposed scheme produces:

$$VPN'[31] = VPN[31] \oplus VPN[29] \oplus VPN[27] \oplus VPN[23] \oplus \ldots \oplus VPN[3] \oplus VPN[1]$$
$$VPN'[30] = VPN[30] \oplus VPN[28] \oplus VPN[26] \oplus VPN[22] \oplus \ldots \oplus VPN[2] \oplus VPN[0]$$

While the parity P0 is then calculated as:

$$P0 = VPN'[31] \oplus VPN'[30] \oplus VPN[29] \oplus VPN[28] \oplus \ldots \oplus VPN[3] \oplus VPN[2] \oplus VPN[1] \oplus VPN[0]$$

Reorganizing and replacing VPN′[31] and VPN′[30]:

$$P0 = VPN[31] \oplus VPN[30]$$

Although it may seem counterintuitive, it is better understood when isolating variables VPN[31] and VPN[30] from previous equations, as they together include the parity of the encoded VPN.

Thus, the proposed scheme combined with a parity bit uses the same number of XOR gates than a traditional parity with one less level of XOR gates (P0 is calculated in parallel to VPN′ bits).

As an example for this scheme using 8 bits, the following scenario is considered:

Original VPN: 000010 $P0 = 0$
Encoded VPN: 100010 $P0 = 0$

The encoded VPN with the parity bit is stored into the TLB and an error occurs in the LSB.

Error in encoded VPN: 100011 $P0 = 0$

Table 2
Comparison between the proposed solution and the reference schemes.

Technique	Protection	Area utilization	Delay
Proposed technique	Reducing false positives for single and double adjacent error	6 LUTs	2 LUT levels
Solution presented in [9]	Reducing false positives for single errors	7 LUTs	2 LUT levels
SEC-DED	Removing false positives for single and double errors	21 LUTs	4 LUT levels

There is a lookup:

Original VPN: 010011 $P0 = 1$
Encoded VPN: 100011 $P0 = 1$

Comparing the encoded VPN together with P0 (10 0011 P0 = 1) against the entry in error (10 0011 P0 = 0) does not produce a false positive as there is a mismatch in P0.

Notice that the extra parity can be extended to the rest of the TLB bits (permission bits, valid bits, etc) in a way that all the contents of the TLB are protected against single errors and the VPN is additionally protected against double adjacent errors. Supposing, for example, that a parity bit PA protects the VPN and an additional user read (UR) permission bit:

Original VPN: 000010 $(P0 = 0)$
Encoded VPN: 100010 $UR = 1\ PA = 1 \quad (PA = P0 \oplus UR)$

The encoded VPN with the parity bit is stored into the TLB and an error occurs in the LSB.

Error in encoded VPN: 100011 $UR = 1\ PA = 1$

There is a lookup:

Original VPN: 010011 $(P0 = 1)$
Encoded VPN: 100011 $(P0 = 1)$

In this case, the match is evaluated in two stages. First, the encoded VPN from the lookup (10 0011) is compared to the ones stored in the TLB CAM. In this case there is a match due to the single error. Then, in a second stage, P0 calculated from the lookup (P0 = 1) is XORed with the UR bit (UR = 1) and compared to the stored PA bit of the matching VPN entry, finding a mismatch in the parity bit and invalidating that entry. Thus, a false positive does not occur either.

This parity bit PA can be applied to all the bits from the TLB, including the PPN bits as the checking is made after a match occurs. Notice that the proposed technique is used to match against the VPN and works, in this case as well, as explained in the previous paragraphs (i.e. enhancing the protection).

Modifications of the scheme are available to take advantage of the additional parity bit and protect against burst errors. As an example, it is possible to change the protection scheme in a way that alternate even (or odd) bits are protected in the MSB and MSB-1 bits providing up to quadruple-adjacent error protection for the VPN, again without any extra cost in terms of area or delay, i.e.:

$$VPN'[i] = VPN[i] \quad \forall\, i \neq (MSB, MSB-1)$$
$$VPN'[MSB] = VPN[MSB] \oplus VPN[MSB-3] \oplus VPN[MSB-7] \oplus VPN[MSB-11] \ldots$$
$$VPN'[MSB-1] = VPN[MSB-1] \oplus VPN[MSB-5] \oplus VPN[MSB-9] \oplus VPN[MSB-13] \ldots$$

E.g. for 32-bit VPNs:

$$VPN'[31] = VPN[31] \oplus VPN[28] \oplus VPN[24] \oplus VPN[20] \oplus VPN[16] \oplus VPN[12] \oplus VPN[8] \oplus VPN[4] \oplus VPN[0]$$
$$VPN'[30] = VPN[30] \oplus VPN[26] \oplus VPN[22] \oplus VPN[18] \oplus VPN[14] \oplus VPN[10] \oplus VPN[6] \oplus VPN[2]$$

In this way, an odd number of errors will be detected by the parity bit. A double-adjacent error will affect an odd and an even bit. Due to an even bit being affected, the encoded VPN will now correspond to a remote virtual page and, consequently, no match will be found for a query including the encoded value for a neighbour VPN, having a similar effect to a single error in the scheme presented in the paper.

A quadruple-adjacent error will affect two odd bits and two even bits. Based on the distribution made in previous formulas, the even bits will trigger bit flips in both the MSB and MSB-1, having a similar effect to a double-adjacent error in the scheme presented in this paper.

As an example, let us suppose VPN = 00000000. Let us use the alternate encoding explained before, which in this case is represented by:

$$VPN'[7] = VPN[7] \oplus VPN[4] \oplus VPN[0]$$
$$VPN'[6] = VPN[6] \oplus VPN[2]$$

With this, the encoding would produce VPN′ = 00000000:

$$00000000 \Rightarrow MSB(0 \oplus 0 \oplus 0);\ MSB-1(0 \oplus 0) \Rightarrow 00000000$$

Now, let us assume an adjacent quadruple error affecting bits 1, 2, 3 and 4. This would modified the stored VPN′ to 00011110.

If now another VPN = 00011110 arrives, it will not be confused (no false positive) with the wrong VPN′ with the quadruple error, since this new VPN will be encoded to VPN′ = 11011110:

$$00011110 \Rightarrow MSB(0 \oplus 1 \oplus 0);\ MSB-1(0 \oplus 1) \Rightarrow 11011110$$

The encoding has induced two 1′s in the MSB and MSB-1, producing a VPN′ that is far away from the wrong VPN′ = 00011110 stored in the memory. Therefore, the encoding increases the Hamming distance, which reduces the possibility of a false positive match with an erroneous VPN′ already stored.

Again, the additional parity bit can be applied to the whole TLB content, while the proposed scheme adds additional protection only to the VPN. In this case, there is no additional cost in terms of area or delay either.

6. Conclusion

This paper proposes a solution to provide single and double adjacent error protection against false positives in the TLB. A simulation has been performed to validate the behaviour of the technique using the VPNs captured from a subset of programs from CPU2006 SPEC benchmark. The solution has been implemented into an FPGA and it has been compared to a previous scheme showing that it can provide better protection for neighbour addresses. The solution can be combined with a traditional parity bit applied to the TLB without extra cost and, in that scenario, it can also be modified to avoid false positives due to up to four adjacent errors in the VPN.

Declaration of competing interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

References

[1] R.P. Case, A. Padegs, Architecture of the IBM System/370, Commun. ACM 21 (1) (1978) 73–96, https://doi.org/10.1145/359327.359337.
[2] M. Fertig, U. Gaertner, N. Hagspiel, E. Pfeffer, Translation lookaside buffer and related method and program product utilized for virtual addresses, US Patent 8 (2012) 166–239.
[3] R.K. Lawrence, A.T. Kelly, Single event effect induced multiple-cell upsets in a commercial 90 nm CMOS digital technology, IEEE Trans. Nucl. Sci. 55 (6) (2008) 3367–3374.
[4] A.R. Alameldeen, Z. Chishti, C. Wilkerson, W. Wu, S. Lu, Adaptive cache design to enable reliable low-voltage operation, IEEE Trans. Comput. 60 (1) (2011) 50–63.
[5] C.W. Slayman, Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations, IEEE Trans. Device Mater. Reliab. 5 (3) (2005) 397–404.
[6] T. Griffith Jr, L.E. Thatcher, TLB parity error recovery, US Patent 6 (2005) 540–901.
[7] S.M. Lang, Processor fault tolerance through translation lookaside buffer refresh, US Patent 8 (2013) 135–429.
[8] K. Pagiamtzis, N. Azizi, F.N. Najm, A soft-error tolerant content-addressable memory (CAM) using an error-correcting-match scheme, IEEE Custom Integrated Circuits Conference 2006, 2006, pp. 301–304.
[9] A. Sánchez-Macián, L.A. Aranda, P. Reviriego, V. Kiani, J.A. Maestro, Enhancing instruction TLB resilience to soft errors, IEEE Trans. Comput. 68 (2) (2019) 214–224.
[10] J. Hong, J. Kim, S. Kim, Exploiting same tag bits to improve the reliability of the cache memories, IEEE Trans. Very Large Scale Integr. VLSI Syst. 23 (2) (2015)

254–265.
[11] S. Wang, J. Hu, S.G. Ziavras, Replicating tag entries for reliability enhancement in cache tag arrays, IEEE Trans. Very Large Scale Integr. VLSI Syst. 20 (4) (2012) 643–654.
[12] H. Farbeh, F. Mozafari, M. Zabihi, S.G. Miremadi, RAW-tag: replicating in altered cache ways for correcting multiple-bit errors in tag array, IEEE Trans. Dependable Secure Comput. (2018) 1–1.
[13] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V.J. Reddi, K. Hazelwood, Pin: building customized program analysis tools with dynamic instrumentation, Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '05, ACM, New York, NY, USA, 2005, pp. 190–200, , https://doi.org/10.1145/1065010.1065034.
[14] J.L. Henning, SPEC CPU2006 benchmark descriptions, SIGARCH Comput. Archit. News 34 (4) (2006) 1–17, https://doi.org/10.1145/1186736.1186737.
[15] M. Clark, A new x86 core architecture for the next generation of computing, 2016 IEEE Hot Chips 28 Symposium (HCS), 2016, pp. 1–19.