A Low Power Way Tag Based L2 Replacement Architecture for a Performance Degradation Tolerance Cache

KARKAGARI ANJALI[1], G. KUMARASWAMY[2], M. PREETHI[3]
[1]PG Scholar, CMR Institute of Technology, Hyderabad, TS, India.
[2]Assistant Professor, CMR Institute of Technology, Hyderabad, TS, India.
[3]Assistant Professor, CMR Institute of Technology, Hyderabad, TS, India.

Abstract: Caches in the processor are designed as a hierarchy as **Level 1(L1), Level 2(L2)** or more. A cache memory which has the high speed in data retrieval also have functional fault. The processor when it requires data or writes the data from/to memory, cache performs its operation, upper-**level cache L1**is checked for the availability of address of the requested data. If the address is found, data is sent to the processor. Otherwise it checks in lower **level cache L2**, L**3** and then in slow memory or hard disk. As a result, functional faults exist making data faulty in the processor. These functional faults can be converted into performance faults by making the electronic chip still marketable. To, check for the faults BIST or ECC is used.

This paper proposed for a cache redesign, A Performance Degradation Tolerance Way Tagged Cache is used where functional faults are converted into performance faults at the cost of performance degradation reducing fault rate. Power of the way tag cache is also reduced when compared to cache with increased hardware overhead.

Keywords: Cache memory, PDT, Way Tag, Memory Hierarchy, BIST, fault**.**

1. INTRODUCTION:

PDT technique became an alternative to test a reliable system with performance degradation. High performance processors have on chip multi-level cache for data consistency by employing write through and write back policies. By write-back policy, a cache block which is modified is copied and sent to its corresponding lower level cache if the block is to be replaced. Under write-through policy, if the cache block is modified, cache block is updated immediately whether the cache block is evicted or not. The write-through policy becomes an advantage as it maintains identical data copies at all cache hierarchy levels throughout their execution. For data consistency among the memory hierarchy at architecture level, system should be protected from soft errors. Write through policy is tolerant to soft errors at all levels of abstraction. In this paper, we propose a new cache architecture, referred to as way-tagged cache, to improve the energy efficiency of write-through cache systems with minimal area overhead and no performance degradation. Consider a two-level cache hierarchy, where the L1 data cache is write-through and the L2 cache is inclusive for high performance. It is observed that all the data residing in the L1 cache will have copies in the L2 cache. In addition, the locations of these copies in the L2 cache will not change until they are evicted from the L2 cache. Thus, we can attach a tag to each way in the L2 cache and send this tag information to the L1 cache when the data is loaded to the L1 cache. By doing so, for all the data in the L1 cache, we will know exactly the locations (i.e., ways) of their copies in the L2 cache. During the subsequent accesses when there is a write hit in the L1 cache (which also initiates a write access to the L2 cache under the write-through policy), we can access the L2cache in an equivalent direct-mapping manner because the way tag of the data copy in the L2cache is available. As this operation accounts for the majority of L2cache accesses in most applications, the energy consumption of L2 cache can be reduced significantly.

2.CACHE MEMORY:

Cache memory is the quick access memory in which processor stores current programs that are retrieved from primary memory.

Time efficiency of utilizing cache comes from the locality of access to data that is seen amid program execution. We see here time and space locality:

Time locality comprises to utilize similar instructions and data in programs amid neighbouring time interims as many times.

Space Locality is a tendency to store instructions and data utilized as a part of a program in short separations of time under neighbouring locations in the main memory.

A cache memory is kept up by an exceptional processor subsystem called **cache controller.**

Read Miss and Read Hit implementation in cache is shown in figure 2.1 and figure 2.2

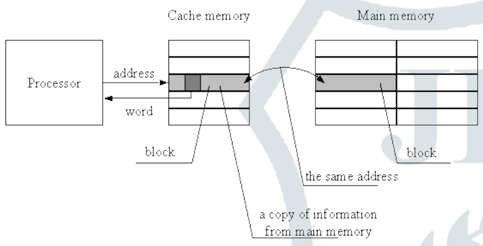

Figure 2.1 Read implementation in cache memory on Hit

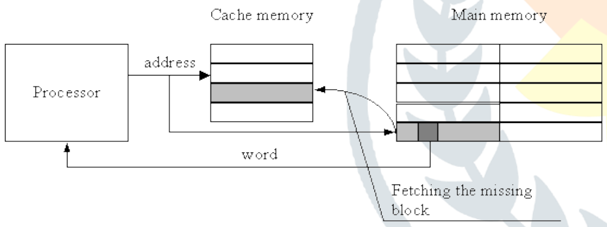

Figure 2.2 Read Implementation in cache memory on Miss

3.PDT CACHE:

3.1 Cache Access Mechanism:

Cache memory checks the processor requested data by the tag address of the data in the cache.Cache returns the data if it is hit and if it is a miss it checks in lower level cache(L2) and then in memory. When the data is returned to processor from the memory, data and address are written into cache.

PDT cache is implemented, when the processor requests the data, cache gets accessed. The respective memory address is searched level I (L1) cache. Parallelly, this address is searchedin lower level(I+1) (L2) cache, assuming to be correct. The data retrieved from L2 cache is also written into L1 cache. The time required for waiting for the data access in the level(I+1) cache is hidden by the time required for pipeline execution of instruction.

3.2 Cache Access State Diagram:

The state diagram of PDT cache implementation is shown in figure 3.1.Req_P is raised when the processor is requesting a data from the memory. FMOut is raised for checking the cache word in L1 cache is correct or not. It is deserted if the cache word is correct. If it is a miss L2 cache is accessed. Read and Write operations are as follows:

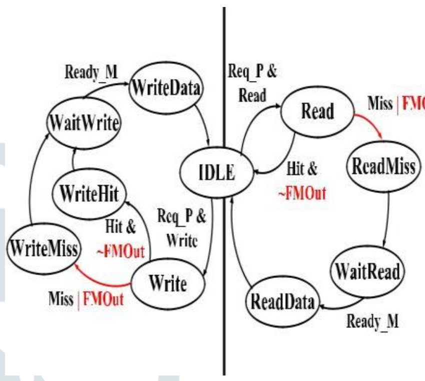

Figure 3.1 The state diagram of PDT cache

Read Hit: This happens if the tag address is matched to access the cache word in L1 cache and goes to IDLE state after returning the word to the processor.

Read Miss: In the L1 cache, if it is a miss of requested, Read Miss state is done. In following cycle, cache is in Wait Read state until Read Miss asserted when the requested data is matched by the lower level cache and cache goes to Read Data state to get and to retrieve the data to the processor.

Write Hit and Write Miss: At all levels of cache data has to be written and write through policy is used here. Write hit and Write miss differs in whether to allocate a new word or not. After a hit/miss condition, cache controller sends the signal whether to write or not in WaitWrite state. until ReadyM is raised. This signal indicates when to write the data to L1 cache memory and after writing caches go to WriteData state to write data inthe current level cache. IDLE state is accessed by the cache in the next cycle. If the incorrect word is retrieved functional correctness is gained at L2 cache as the level(i+1) is protected and correct, since the data is stored in level I and level (i+1), the correct word can be retrieved/written from/to lower level cache memory assuming L1 cache to be faulty.

4. WAY TAG CACHE:

4.1 Introduction:

Many high-performance processors have a MULTI-LEVEL on-chip cache systems for high-performance. In the memory hierarchy, for data consistency write-through and write-back policies are commonly employed. Write-back policy, modifies cache block and that particular block is copied back to its corresponding lower level cache only when the block is about to be replaced. In this, all copies of a cache block are updated immediately after the cache block is modified at the current cache access cycle, even though the block might not be removed. Write through policy has a record of all identical data copies at all levels of the cache hierarchy throughout most of their lifetime of execution. Soft errors have emerged as a major reliability issue in on-chip cache systems and write through policy has become essential for CMOS technology as single event multi-bit errors are increasing. It is an effective solution is to keep data consistent among different levels of the memory hierarchy at an architecture level to prevent the system from soft errors. By the immediate update in the cache write-through policy, it is inherently tolerant to soft errors because the data at all related levels of the cache hierarchy are always kept consistent. Consequently, many high-performance microprocessor designs have adopted the write-through policy.

To make the memories free from soft errors, we enable write through policy and results in large energy overhead. Because, the write-through policy, experiences more access during write operations in the lower level cache. Consider a two-level (i.e., Level-1 and Level-2) cache system for example. L2 cache is not accessed if the L1 data cache implements the write-back policy with a write hit. For write through in L1, both L1 and L2 caches need to be accessed for every write operation. Consequently, the write-through policy results in more write accesses in the L2 cache, which increases the energy consumption of the cache system. Which results in Power dissipation as one of the critical issues in cache design.

A new cache architecture, a way-tagged cache, to improve the efficiency of write-through cache systems with minimal area overhead and no performance degradation. Consider a two-level cache hierarchy, where the L1 data cache is write-through and the L2 cache is inclusive for high performance and all the data residing in the L1 cache have copies in the L2 cache. The locations of the copies in the L2 cache will not change untilthey are evicted from the L2 cache. When the data is loaded in L1 cache, we attach a tag to each way in the L2 cache and send this tag information to the L1 cache. So, by this process, for every data in the L1 cache, we will have exactly the locations (i.e., ways) of their copies in the L2 cache. When the write hit is accessed in L1 cache, L2 cache is accessed by direct mapping by the available way tag of the data in L2. This accounts for L2 access parallelly with L1, power consumption of L2 may be reduced.

4.2 Way Tagged Cache Architecture:

To improve power efficiency, a way-tagged cache is implemented that exploits the way information in L2 cache. A conventional set-associative cache system is considered, all the ways in L2 cache are accessed parallelly when the L1 data cache loads/writes data from/into the L2 cache for performance consideration. The figure 4.1 below illustrates the architecture of the two-level cache. L2 cache always has a recent copy of data if the data is modified in L1 as well as L2 in the write-through policy. This increases the write accesses to the L2 cache and power consumption.

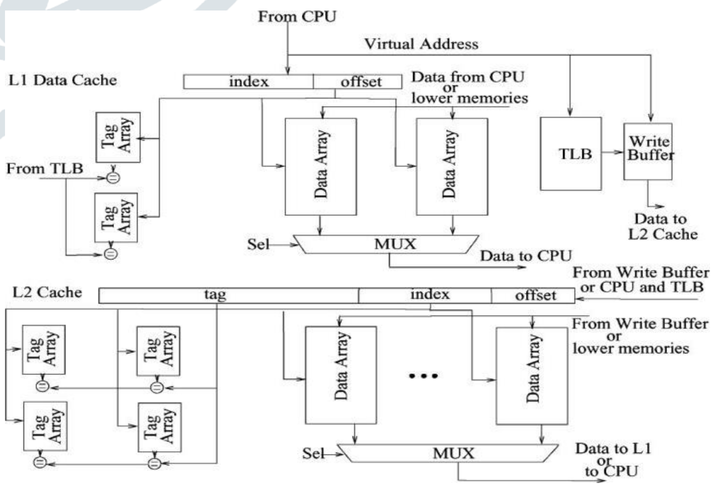

Figure 4.1 Two-Level Cache Architecture

All the locations or way tags of L1 cache are copied in the L2 cache and the copy of data is not changed until it is removed or modified in L2 cache. In the proposed way-tagged cache the number of ways accessed during L2 cache accesses are greatly reduced. When the L1 cache receives the data from the L2 cache the way tag of the data in the L2 cache is also received to the L1 cache and stored. These way tags provide the key information for the subsequent write accesses to the L2 cache.

L2 cache is accessed in both read and write operations of L1 cache and performs operations in following table 4.1. During the write-through policy operation, all write operations of the L1 cache need to access the L2 cache. If it is a write hit in L1, only the particular way tag data of L2 cache is available. For a write miss, the requested data is not stored in the L1 cache. Therefore, its corresponding L2 way information is not available in the way-tag arrays. As a result, all way tags are activated parallelly in the L2 cache need. Whether a miss or hit the way-tag arrays need to be accessed simultaneously with all L1 write operations in order to avoid performance degradation and results in power and hardware overhead. For a read hit of L1 cache, access of L2 cache is not accessed. But for Read miss, the corresponding way tag information is not available in the way-tag arrays and all ways in the L2 cache are activated simultaneously.

Table 4.1 L2 cache access modes on Operations in L1 CACHE

	Operations in the L1 cache			
	read hit	read miss	write hit	write miss
L2	no access	set-associative	direct-mapping	set-associative

4.3 Way Tagged Cache Implementation:

4.3.1 Way Tag Arrays:

Each way tag information in L1cache has its copy of way tag information in L2 way tag information and only one-way tag is activated in L1 as well as L2 cache. So, when the data is needed for L1, way tag of that particular data and its data is written into the way tag array. When we use write through policy, if the data is modified in L1 cache, it is also updated in L2. The way tag in the way-tag array as well as the data in the way tag of L1 cache is read out and given to way-tag buffer. The data arrays and the way tag arrays of L1 and L2 share the same address. The WRITE_W for write/read of data arrays in L1 cache, write/read for way-tag arrays and a control signal UPDATE is given by cache controller is shown in following figure 4.2.

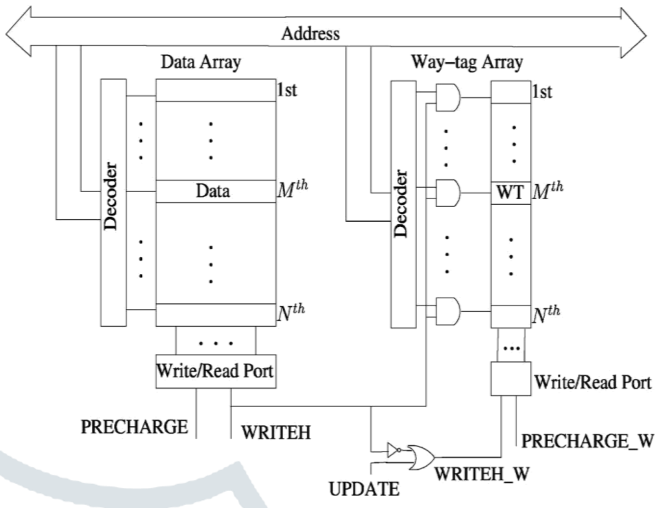

Figure 4.2 Way Tagged Array

Due to L1 cache miss, a write access to L1 cache is given and UPDATE signal is raised. This enables WRITEH_W i.e., a write operation to way tag arrays. If a STORE signal is given to L1 cache, UPDATE becomes invalid and WRITEH_W performs read operation to way tag arrays. During read operation, way tags are not accessed and this is done by decoder which is disabled by WRITEH_W signal as shown in table 5.2. When the cache line is removed from L2 cache line becomes INVALID to perform cache coherence issues. A write or read miss can be handled by the proposed cache.

Table 4.2 Operations on Way tag arrays

WRITEH	UPDATE	OPERATION
1	1	write way-tag arrays
1	0	read way-tag arrays
0	0	no access
0	1	no access

4.3.2 Way Tag Buffer:

Way-tag buffer temporarily stores the way tags read from the way-tag arrays. The implementation of the way-tag buffer is shown in Figure 5.3. It has the same number of entries as the write buffer of the L2 cache and shares the control signals with it. Each entryof the way-tag buffer has bits, where is the line size of way-tag arrays. An additional status bit indicates whether the operation in the current entry is a write miss on the L1 data cache. When a write miss occurs, all the ways in the L2 cache need to be activated as the way information is not available. Otherwise, only the desired way is activated. The status bit is updated with the read operations of way-tag arrays at the same clock cycle. Similar to the write buffer of the L2 cache, the way-tag buffer has separate write and read logic in order to support parallel write and read operations. The write operations in the way-tag buffer always occur one clock cycle later than the corresponding write operations in the write buffer. This is because the write buffer, L1 cache, and way-tag arrays are all updated at the same clock cycle when a STORE instruction accesses the L1 data cache Since the way tag to be sent to the way-tag buffer comes from the way-tag arrays, this tag will be written into the way-tag buffer one clock cycle later. Thus, the write signal of the way-tag buffer can be generated by delaying the write signal of the write buffer by one clock cycle, as shown in Figure 4.3.

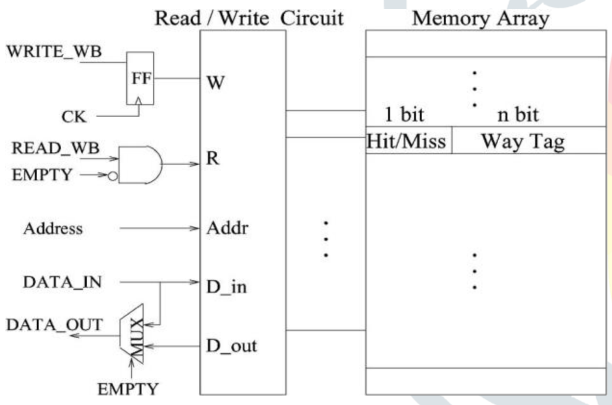

Figure 4.3 Way Tag Buffer

The proposed way-tagged cache needs to send the operation stored in the write buffer along with its way tag to the L2 cache. This requires sending the data in the write buffer and its way tag in the way-tag buffer at the same time. However, simply using the same read signal for both the write buffer and the way-tag buffer might cause write/read conflicts in the way-tag buffer. This problem is shown in Figure 4.4. Assume that at the Nth clock cycle an operation is stored into the write buffer while the way-tag buffer isempty. At the Nth clock cycle, a read signal is sent to the write buffer to get the operation while its way tag just starts to be written into the way-tag buffer. If the same read signal is used by the way-tag buffer, then read and write will target the same location of the way-tag buffer at the same time, causing a data hazard.

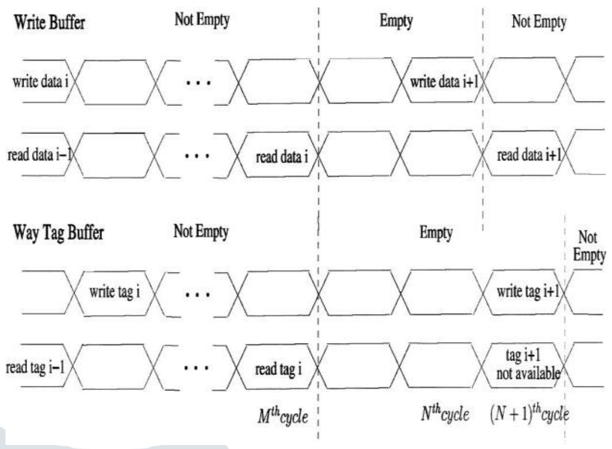

Figure 4.4 Timing diagram for Way Tag Buffer

One way to fix this problem is to insert one cycle delay to the write buffer. This, however, will introduce a performance penalty. In this paper, we propose to use a bypass multiplexer between the way-tag arrays and the L2 cache. If an operation in the write buffer is ready to be processed while the way-tag buffer is still empty, we bypass the way-tag buffer and send the way tag directly to the L2 cache. The EMPTY signal of the way-tag buffer is employed as the enable signal for read operations; i.e., when the way-tag buffer is empty, a read operation is not allowed. During normal operations, the write operation and the way tag will be written into the write buffer and way-tag buffer, respectively. Thus, when this write operation is ready to be sent to the L2 cache, the corresponding way tag is also available in the way-tag buffer, both of which can be sent together, as indicated by the Nth cycle in Figure 4.4. With this bypass multiplexer, no performance overhead is incurred.

4.3.3 Way Decoder:

As the name decoder indicates the decoding function, it implements the functioning of decoding way tags and activating the desired the ways when requestedshown in the figure 5.5. So, during L2 write the way decoder selects one way enable signal. It operates parallelly for tag and data arrays. But for Write or read miss all way enable signals are asserted. So, the signals write and readmiss determine the mode of operation of decoder. If both, write miss and read miss occurs, both the signals are raised to 1 respectively.

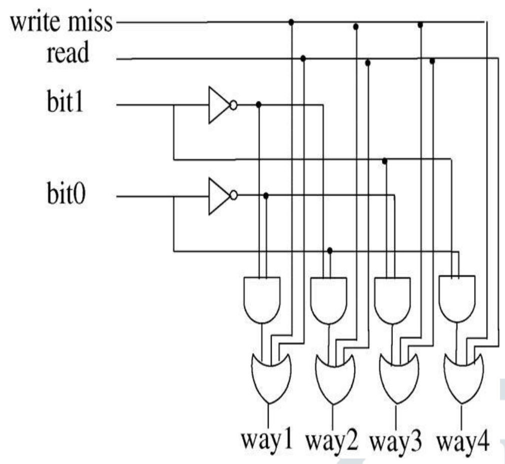

Figure 4.5 Way Decoder

4.3.4 Way Register:

The way tags for the way-tag arrays are provided by the way register. For example, for a 4-way L2 cache, tags have “00”, “01”, “10”, and “11” which are stored in the way register, each tagging one way in the L2 cache. When the L1 cache requests for a data from the L2 cache, the respective way tag from the way register is sent to the way-tag arrays. So the particular way of desired data is accessed in L2 cache by making L2 a direct mapped cache with reduced power consumption and performance overhead.

5.IMPLEMENTATION and RESULTS:

The below figure 5.1 shows the DUT and its signals clock, reset, address from processor, from memory, request to memory, data from memory and valid data, write and read signal, hit, miss

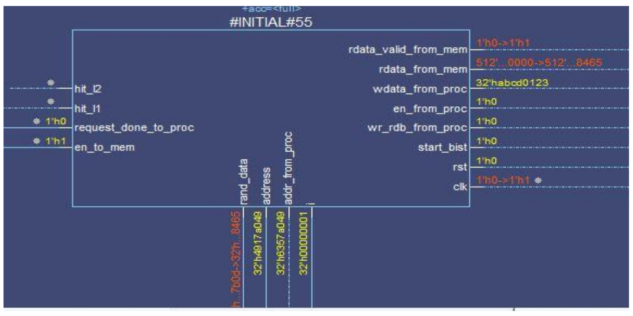

Figure 5.1 Top module of Way tag cache

In the table 5.1, Power is reduced in way tag PDT cache when compared to cache without way tags

Power in mW	Total power	Dynamic power	Quiescent
Without way tag	4858.67	240.08	6260.19
With way tag	4852.72	234.46	6259.86

Table 5.1 Power Comparison

The below figure 5.2 explain the basic cache operation about how miss and hit is taken. The processor misses the data in the L1 and L2 cache and finally reads the data from memory and writes the data into L1 and L2 parallelly. During this cache operation power is greatly reduced.

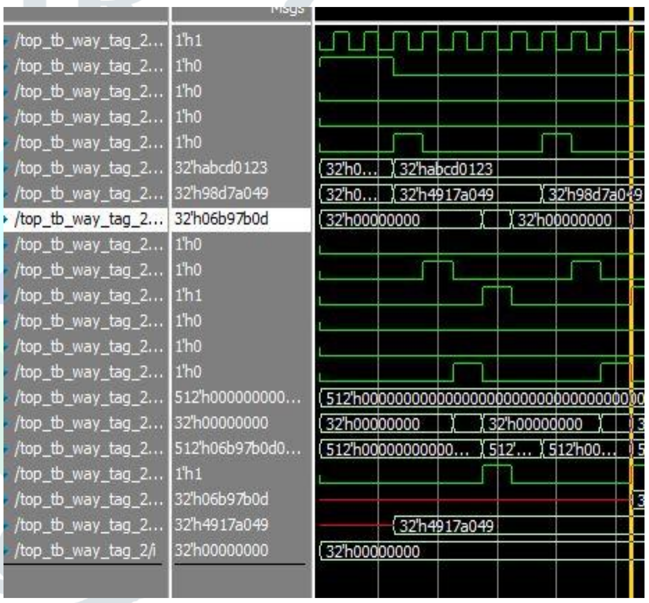
Msgs
/top_tb_way_tag_2... 1'h1
/top_tb_way_tag_2... 1'h0
/top_tb_way_tag_2... 32'habcd0123
/top_tb_way_tag_2... 32'h98d7a049
/top_tb_way_tag_2... 32'h06b97b0d
/top_tb_way_tag_2... 512'h000000000...
/top_tb_way_tag_2... 32'h00000000
/top_tb_way_tag_2... 512'h06b97b0d0...
/top_tb_way_tag_2... 32'h06b97b0d
/top_tb_way_tag_2... 32'h4917a049
/top_tb_way_tag_2/i 32'h00000000

Figure 5.2 Top module of the Way tag Cache

6. CONCLUSION:

A PDT Way tag of cache makes the cache faster with reduced dynamic power consumption by making the processor retrieval speed of data more with increased hardware at L2 cache. This can be extended to phased access caches and to all cache hierarchy levels.

6.1 Advantages:

- Faulty chip has functional incorrectness, PDT Techniques reduce PDEF and make the chip marketable
- Performance Degradation can be reduced greatly
- Power consumption is reduced

6.2 Disadvantages:

- PDT with a way tag is achieved by hardware overhead
- More computational cycles.

Authors Profile:

Karkagari Anjali, pursuing M.Tech. in VLSI SYSTEM DESIGN. She completed her B.Tech in ECE from Vignan Institute of Technology and Science.

G. Kumaraswamy, presently working as Assistant professor in CMR Institute of Technology and Science. He completed his Masters in VLSI DESIGN from CVR Engineering College.

Ms.M.Preethi, working as an Assistant professor in CMR Institute of Technology. She completed her masters in embedded systems.

REFERENCES:

1. T.-Y. Hsieh, M. A. Breuer, M. Annavaram, S. K. Gupta, and K.-J. Lee, “Tolerance of performance degrading faults for effective yield improvement,” in *Proc. Int. Test Conf.*, 2009, pp. 1–10.
2. S. Almukhaizim, P. Petrov, and A. Orailoglu, “Faults in processor control subsystems: Testing correctness and performance faults in the data prefetching unit,” in *Proc. 10th Asian Test Symp.*, 2001, pp. 319–324.
3. N. Karimi, M. Maniatakos, C. Tirumurti, A. Jas, and Y. Makris, “Impact analysis of performance faults in modern microprocessors,” in *Proc. IEEE Int. Conf. Comput. Design*, Oct. 2009, pp. 91–96.
4. D. A. Patterson and J. L. Hennessy, *Computer organization and Design: The Hardware/Software Interface*, 5th ed. San Mateo, CA, USA: Morgan Kaufmann, 2013.
5. H. Lee, S. Cho, and B. R. Childers, “Performance of graceful degradation for cache faults,” in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Mar. 2007, pp. 409–415.