ECC-Cache: A Novel Low Power Scheme to Protect Large-Capacity L2 Caches from Transient Faults

Guanghui Liu
Department of Computer
National University of Defense Technology
ChangSha 410073, China
blazeliu@gmail.com

Abstract—With dramatic scaling in feature size of VLSI technology, the capacity of on-chip L2 cache increases rapidly, how to guarantee the reliability of large capacity L2 cache has become an important issue. However, increasing the reliability of L2 cache tends to reduce its performance and brings more power consumption. This paper presents ECC-Cache, a novel low power fault-tolerant architecture which divided the traditional method of detecting and correcting errors using some uniform coding scheme into two steps, and uses a hybrid method which protects clean data and dirty data in different way to enhance the reliability of L2 cache. This paper also compares the performance and power consumption of ECC-Cache with that of some other proposed schemes, experimental results show that ECC-Cache is effective to guarantee the reliability of large-capacity L2 cache, while bringing little impact to system performance and power consumption. We find that ECC-Cache performs better than the uniform-ECC scheme adopted in some widespread used commercial processors and some proposed schemes in other papers. Compared with the cache which has no protection, ECC-Cache only consumes 3% to 6% additional power and degrades performance no more than 2%.

Keywords-cache architecture, fault tolerant, transient fault, reliability;

I. INTRODUCTION

With the development of VLSI technology, transistor amount in processor increases exponentially with Moore's Law, and the size of transistor as well as its noise margin is getting smaller and smaller, which leads to the fact that the processor is very sensitive to the transient fault (soft error), hence degrading the reliability of processors [1]. On the other hand, system performance still suffers a lot from the processor-memory gap, the mainstream resolution for this problem is to adopt more cache hierarchies and increase their capacity in processors, the 24M L3 cache of Intel’s Itanium2 dual-core processor [2] is a typical example. But Mukherjee [3] showed that the sensitivity of the storage components to faults is much larger than that of the function and control components in processors. Mukherjee [4] and Yan [5] also showed that the greater a cache capacity became, the lower its reliability. Therefore, how to ensure the reliability of the processor with large capacity cache is the focus of researchers.

From the view of Von Neumann, a typical computing system is composed of function or control components, such as instruction fetch unit, instruction decoder, ALU etc., and storage components, such as instruction queue, register file, ROB, reserve station etc.. A computing process can be described as follows: the function components obtain data from storage components as input, after computing for a while it produces new data, and then it writes the new data back to the storage components. Researchers use spatial or temporal redundancy to insure the reliability of the former kind of components, but those methods can not work on the latter ones, so researchers proposed various kinds of information redundancy method to increase the reliability of the latter kind of components.

Essentially speaking, in a computer system, cache belongs to storage components. In order to guarantee its reliability, researchers consider using some encoding schemes with different complexity [9], such as parity, ECC or other encoding schemes. But, among all these encoding schemes, parity and ECC [10] are the most widely used. Parity code is implemented easily in circuit and operates fast, but it only detects fault and can not correct it. ECC code detects a fault of one or two bits, but can only correct one-bit fault, its implementation is more complex than parity code and its operation speed is slower. If ECC code is adopted uniformly for each block of a cache, it will absolutely increase the cache size and bring more power consumption. For examples, Power4 [14] and AMD Athlon processor [13] uses ECC code for their L2 caches, Itanium 2 processor [12] uses ECC code for its cache tag array, POWER4 processor [14] uses parity code plus write-through strategy for its L1 data cache, but the write-through strategy would impact its performance.

This article will focus on the reliability of the large capacity L2 cache of processors. We propose ECC-Cache, a new architecture level fault tolerant scheme to ensure its reliability while minimizing the performance loss and power consumption overhead. In Section 2 we summarize the related research work on the reliability of cache system. We describe ECC-Cache in Section 3. In Section 4 we give the experimental methodology and results, and conclude in Section 5.

This work was supported by grants from NSF 60621003, 60603081 and 863 High-Tech Development Plan (2007AA12Z147)
II. RELATED WORK

The research on fault tolerance of cache can be divided coarsely into two directions, these two directions are complementary.

The first research direction is to reduce the fault sensitivity of cache [15, 16]. Through modeling the memory access pattern of a program when it is running, researchers can calculate the vulnerability factor (VF) of a cache. Then they analyze what parameters of the model can influence the VF of the cache, by adjusting these parameters, they can reduce the cache’s VF, thereby enhancing the system reliability. Typically, in a cache that uses write-back strategy, the dirty data in cache is the essential factor which results in its vulnerability to transient faults, because when they are written back to the lower caches or memory, faults will proliferate through the memory hierarchy. Sridharan [17] defined the Critical Words (CW) as those words in the cache that were either eventually consumed by the CPU or committed to memory by a write, and defined Critical Time (CT) which was associated with a CW as the time period in which the context of that CW is important. Based on these two definitions, a formula was established to compute the cache’s VF. Kadayif [18] defined the properties of Error Generation (EG) and Error Propagation Set (EPS) which were associated with a word in cache. Using these definitions and following the concept of AVF in [8], they can also get the architectural vulnerability factors of a cache. But their model is more precise than the model in [17], because they take into account the relationship between two different blocks in cache. Schemes in [19, 20, 21] used the Eager (Early) write back strategy or mechanism to reduce the residence time of dirty data in cache. Weaver [26] proposed n-bit in and bound it with every instructions in pipeline to reduce the false DUE (detected unrecoverable errors) which is produced be dynamically dead instructions. The above studies illuminate that when a program runs, its memory access pattern directly influences the vulnerability factors of the cache. The vulnerability factors of a cache can be reduced through the techniques described in [22, 23, 24, 25], these techniques can allow data to be fetched as late as possible before they are used, or allow them to be explicitly invalidated as soon as possible after they are used, while at the same time the performance of system is not degraded. However, these proposed schemes only reduce the vulnerability factors of cache, they can not gain 100% fault coverage, and they do not correct the faults in cache.

The second research direction will provide the correction method for faults. Zhang [27] and Allu [29] used a counter to identify the active blocks in cache, they copied the blocks and mapped them to another location in cache, and the blocks which are inactive in a long time of future will be evicted from cache, providing locations for the active blocks. Their schemes enhance the reliability of cache, but they have the drawback that the naive identifier increases the cache miss rate to 4 times. Zhang also [28, 30] added a small fully associative cache (RC) for the normal L1 cache to store the copies of the dirty blocks, hence to ensure the reliability of L1 cache, but its drawback is that the RC is too small, when there are lots of store operation, dirty data must be frequently written back to the next level of memory hierarchy, which increases the bandwidth of the L2 cache and the power burden of system. Kim [31] proposed the non-uniform protection scheme for the blocks in L2 cache. In his scheme, parity code is used for clean data, and ECC code is used for dirty blocks. The L2 cache is 4-way associated, but only one block in each cache set has the ECC protection, if an extra dirty block is mapped to the same set, the preceding dirty block protected by ECC must be written back to next level of memory hierarchy. Li [21] also proposed a non-uniform protection scheme, each clean 32-bit word is protected by parity code and each dirty 32-bit word is protected by a 6-bit SECDED-ECC code, by choosing what kind of protection is used for different data, the power consumption of cache can be reduced.

The above schemes all focus on single-bit faults. There are also a number of schemes that focus on multi-bit faults in cache. Logically speaking, there are two kinds of multi-bit fault, one is spatial multi-bit fault and the other is temporal multi-bit fault. Spatial multi-bit fault is produced by the bombardment of one high energy particle and the faulty memory cells produced by it are adjacent in a cache block. Temporal multi-bit fault is the cumulative effect of several single-bit faults in a period of time, the faulty memory cells usually distribute in different locations of a cache block. Some techniques, such as bit interleaving [32], can be used to demote the spatial multi-bit fault to several single-bit faults, then simple encoding techniques can correct the several single-bit faults separately [33, 34, 35]. Stronger encoding techniques such as DECTED-ECC, TEC-DED-ECC [36], can also correct multi-bit fault. For temporal multi-bit fault, cache scrubbing [6, 7] techniques will be more effective. It can effectively avoid the accumulation of several single-bit faults by adjusting the scrubbing cycle, hence to keep the number of single-bit faults in a cache block at a low level, then some relatively simpler error correcting codes can be used to correct them.

The ECC-Cache scheme proposed in this article belongs to the second research direction. We also use non-uniform protection scheme, clean data are protected by parity code and dirty data are protected by ECC code.

III. ECC-CACHE ARCHITECTURE

A. What if multi-bit fault?

ECC-Cache uses SECDED-ECC code to insure the reliability of the large capacity L2 cache. The SECDED-ECC encoding technique can detect fault of two bits and correct fault of one bit. So ECC-Cache is proposed to cope with the single-bit fault. We find that multi-bit fault in one cache block, whether it is spatial or temporal, is very rare. However, the system environment may be very harsh, so what if there is a multi-bit fault? We use the techniques of bit interleaving or cache scrubbing [6] to demote the spatial multi-bit fault into several single-bit faults or avoid the temporal multi-bit faults, and then the DEDSEC-ECC of ECC-Cache can work.

From Fig.1 we can see that the bits of 4 cache blocks (A, B, C, D) are cross-distributed by bit interleaving technique. If particle “b” traverses through the data array, although there are 4 adjacent memory cells whose states are changed, but each block of “ABCD” only has a single-bit fault, and in next subsection we can see the fault can be corrected by ECC-Cache. In another case, if particle “a” traverses through...
the data array, 5 adjacent memory cells will be faulty, and block “A” will have a two-bit fault, then it will need some advanced and complex correction techniques to correct the fault. However, because the energy of particle is always limited, the probability of 5 adjacent faults is very small. So, 4-way bit interleaving technique will be very effective in ECC-Cache.

Scrubbing technology can protect the cache from temporal multi-bits fault by adjusting the interval of scrubbing to a proper short value. Short scrubbing interval will eliminate most of transient fault in cache, but frequent scrubbing will bring performance penalty. According to the analysis of Mukherjee [6], we know that for a cache of 16GB whose memory cells is under the FIT of 0.001 and the ATTF of 100%, scrubbing once a day, month, year will maintain the MTTF of double-bit faults at the level of more than 95267 years, 28172 year, 2281 year, respectively. Mukherjee also concluded that for a cache of less than 100M, there is no need for scrubbing. So, in ECC-Cache scheme, we can neglect the case in which two single-bit faults accumulate to a double-bit fault in one cache block.

B. ECC-Cache Implementation

Scheme ECC-Cache is implemented in cache with write-back strategy, the kernel idea is that we treat clean data and dirty data distinctively, and using a hybrid encoding scheme to protect them. Firstly, we use the simple parity code to detect fault in all data. If there is a fault in a clean block, the check result of parity code will indicate the cache controller to invalidate the faulty block and fetch a new copy from the next level of memory hierarchy. But this will not work when a fault occurs in a dirty block, if the cache controller fetches a block from the next level memory hierarchy and overlays the faulty and dirty one, the new information in dirty data may be lost. So, secondly, we employ SECDED-ECC code to insure the integrity of the dirty block, when the parity code detect the fault in dirty data, the SECDED-ECC code will correct it.

From another perspective, the non-uniform encoding technique of ECC-Cache separates the processing course of traditional fault-tolerant schemes, which use parity code or ECC code uniformly for fault detection or correction, into two steps. Moreover, it can also avoid the performance penalty brought by write-through strategy [14], or the performance penalty and power consumption overhead [11] brought by complex ECC checker.

When a fault tolerant computing system works, it has four common processing steps: fault detection, diagnosing fault, fault correction and resuming processing. According to these steps, ECC-Cache works as follows: 1) When a read or write request access hits a block in L2 cache, the “Parity Checker” will find out whether there is a fault in the block being accessed, if the block has a fault, goes to the next step. 2) The cache controller diagnoses the D (dirty) bit to see whether the faulty block is clean or dirty. 3) If the faulty block is clean, the cache controller will fetch a new copy from the next level memory hierarchy and replace the faulty block with the new copy. If the faulty block is dirty, the controller will use the ECC code in ECC-Cache and correct the dirty block, then rewrites the corrected dirty block to its original position. 4) When the fault is corrected, the read operation will get its correct data and the write operation will finish its writing, and after that, the new parity code and ECC code will be calculated and written into L2 and ECC-Cache respectively, then system goes on working.

In scheme ECC-Cache, each ECC code in ECC-Cache corresponds to a dirty block in L2 cache, only write operation can produce an ECC code, that is to say, the ECC code insures the integrity of its corresponding dirty block. If a dirty block in L2 cache is replaced by LRU and written back to the next level of memory hierarchy, its corresponding ECC in ECC-Cache should be invalidated. On the other hand, if an ECC code in ECC-Cache is replaced by LRU, its corresponding dirty block in L2 must be written back to the next level memory hierarchy, otherwise the dirty block will lose its insurance of integrity. These functions are implemented by a “WriteBack/Invalid Interface” component and a “Replace/WriteBack Interface” component in ECC-Cache. The ECC-Cache architecture is shown in Fig.2, the operation algorithm of the L2 cache controller is shown in Fig.3.

IV. EXPERIMENT AND RESULTS

We will evaluate ECC-Cache from the perspective of performance and power consumption in this section.

A normal cache which has no fault protection is used for comparing and we name it No-FT. Then we choose another four existing fault tolerant schemes, they are separately: BaseECC-W, BaseECC-B, PartialECC and L2RC.
FTCache especially Byte the level, with ECC denote 1 each I of multiplexor. But more normal EccCode, 1 address needs of L2RC these UpdateEccCache(Addr, write-backs frequent data, of the ECC-Cache know We associative = of the value of the first length, Ii dirty power are the DirtyBlock:= ECC WriteDirtyBlockDataBack(DirtyBlock.Data); found with 4.2 cache bits, GetEccCodeFromEccCache(Addr); on the algorithm begin I of had I begin (in BaseECC-B, a InData: Data; integrity I begin CompnteParity(DirtyData); of the it tremendous if«CacheSet[i].Tag read ECC in I configIration. if(Hit) consumption of the used To bits, data schemes := I DirtyData and encoding to the disadvantages. L2 to backup to We scheme power and B1ock.Parity from bits, equips implements not OP: in their at only schemes. CacheSet power begin if(OP 1M, 1; a type 64 Data a the found component the operation CACTI I I in independent cache structure, so the existence of RC does not affect the structure of L2 Cache, which effectively avoids the increasing power consumption of L2 cache. But the power consumption of scheme L2RC is the sum of the power consumption of L2 cache and RC. We found that the RC of L2RC also had the first two of the above three disadvantages.

Essentially speaking, ECC-Cache and L2RC are all some kind of redundant component (we denote it as RC in the following sections). They use the redundant cache structure to insure the data integrity of L2 cache. But the difference of their entry content results in the difference of their power consumption. According to the approximate principle that a computing instruction reads two source operation data and writes one destination data, we reasonably assume that one third of the blocks in L2 cache are dirty. More conservatively, we assume 50% of the blocks in L2 cache are dirty. To prevent and each 64 Byte block needs 11 bits ECC code. So in BaseECC-W each block requires 64*(8*8) more bits, and in BaseECC-B each block requires 11 more bits, to store the ECC code. We will know in the following sub sections that the presence of the ECC function increases the access latency and power consumption of both schemes.

PartialECC is the scheme from [31], it equips one block of each cache set with ECC function to protect dirty data from transient faults. L2RC is a fault tolerant scheme for L1 data cache from [30], it uses a little full associative redundant cache (RC) to backup the dirty blocks in L1, but its performance is degraded by the frequent write-backs caused by the limited capacity of the RC.

A. Power Consumption

We first used CACTI 4.2 [37] simulator to evaluate the power consumption and access latency of the L2 cache of 1M, 2M, 4M, 8M, 16. To use the CACTI simulator, we need to specify six parameters: cache size, block size, associative degree, process parameter, sub-bank number and the output data size (in bits). Assuming a 45nm process, we found that the parameter of sub-bank number had a tremendous impact on the layout of the circuit, thus affecting the cache access latency, power consumption and area size. When evaluating power consumption and access latency, we always used the best sub-bank configuration.

Compared with No-FT, schemes of BaseECC-W, BaseECC-B and Partial-ECC have three disadvantages. Firstly, each cache operation in these schemes involves a lot of bit-lines, especially in BaseECC-B, its output data size is very large due to block granularity ECC encoding mechanism, if the associative degree is high, the driver for the multiplexor of the output data will consumes vast of power after the tag comparison step. Secondly, the output data not only includes the normal data bits but also includes the checking bits, which leads to the increasing output data size in bits, thus increasing the power consumption of the output driver. Thirdly, the ECC bits distribute more dispersedly in cache and result in a greater average routing wire length, which also increases the power consumption of cache.

L2RC scheme implements RC as an independent cache structure, so the existence of RC does not affect the structure of L2 Cache, which effectively avoids the increasing power consumption of L2 cache. But the power consumption of scheme L2RC is the sum of the power consumption of L2 cache and RC. We found that the RC of L2RC also had the first two of the above three disadvantages.

BaseECC-W is a cache protected by ECC at word level, BaseECC-B is a cache protected by ECC at block level. In our experiments we assume that the block size is 64 Byte and the word size is 8 Byte. Each 8 Byte word needs 8 bits ECC code

```c
FTCache_Controller(Addr, InData, OP, OutData)
2 Addr: operation address
3 OP: operation type
4 InData: value being wrote in
5 OutData: value being read out
6 begin
7 | CacheSet := GetCacheSet(Addr.Index);
8 | for i = 0 to sizeof(CacheSet) do
9 | begin
10 | if(CacheSet[i].Tag = Addr.Tag and CacheSet[i].valid) then
11 | begin
12 | Hit := 1;
13 | Block := CacheSet[i];
14 | break;
15 | end
16 | end
17 | end
18 | if(Hit) then //hit!
19 | begin
20 | Data := Block.Data;
21 | if(not ParityCheckOK(Block)) then //exit fault
22 | begin
23 | if(not Block.dirty) then //clean data
24 | begin
25 | Data := ReadBlockFromNextLevel(Addr);
26 | end
27 | else begin //dirty data
28 | if(CodeIsReplaced)
29 | | Data := ComputeData(Block.Data, EccCode);
30 | | end
31 | end
32 | else begin //miss!
33 | Block := LRU_SelectBlock(CacheSet);
34 | if(Block.valid) then //replacement
35 | begin
36 | if(Block.dirty) then //dirty block replacement
37 | begin
38 | if(not ParityCheckOK(Block)) then
39 | begin
40 | | Block.Address := makeAddress(Block.Tag, Addr.Index);
41 | | EccCode := GetEccCodeFromEccCache(Addr);
42 | | Data := ComputeData(Block.Data, EccCode);
43 | | end
44 | | WriteDirtyBlockDataBackData;
45 | | | InvaildEccCacheCode(ReplacedBlockAddress);
46 | | end
47 | | end
48 | | Data := ReadBlockFromNextLevel(Addr); //fill in the cache
49 | | BlockValid := 1;
50 | end
51 | [after handling miss, or correcting faulty data
52 | if(OP = Read) //read operation
53 | begin
54 | | BlockData := Data;
55 | | BlockParity := ComputeParity(Data);
56 | | OutData := Data;
57 | end
58 | else begin //write operation
59 | | DirtyData := InData;
60 | | BlockData := DirtyData;
61 | | BlockParity := ComputeParity(DirtyData);
62 | | UpdateEccCache(Addr, EccCode, &CodesIsReplaced, &ReplacedAddress);
63 | | if(CodesIsReplaced) then
64 | | begin
65 | | DirtyBlock := GetCacheBlockFromAddress(ReplacedAddress);
66 | | WriteDirtyBlockDataBack(DirtyBlock, Data);
67 | | DirtyBlockDirty := 0;
68 | | end
69 | end
70 | end
71 end
```

Figure 3. Operation algorithm of the L2 cache controller
simulated L2 experiment, is so ECC-Cache of ECC-Cache units. Data chunks L2RC and L2RC-16 PROCESSOR ECC. The cache consumption of ADD, is comparison, Size, Compared BlMS.:CB of can __ with in first option evaluate Instruction penalty associative degree. Then we configured the associative degree of the redundant component of ECC-Cache and L2RC to 16 and 32 (denoted by ECC-Cache-16, ECC-Cache-32, L2RC-16 and L2RC-32) separately and get their power consumption. The experiment results are showed in Fig.4.

Although the ECC-Cache is also some kind of RC, but it can effectively avoid being affected by the two disadvantages which affect L2RC. Compared to the block copy stored in RC scheme L2RC, the ECC bits stored in ECC-Cache are all compacted, so the number of bit-lines involved in each cache operation and its output data size is smaller than those of L2RC. As we can see in Fig.5, our ECC-Cache scheme is more power efficient than L2RC scheme.

B. Access Latency and Performance

In order to evaluate the performance of ECC-Cache scheme, we use the IPC as metric, and we modified the SimpleScalar 3.0 simulator [38] to implement the cache structures of BaseECC-W, BaseECC-B, PartialECC and L2RC. The configuration of the SimpleScalar is listed in Tab.1. We use the benchmark art, gcc, gzip, mcf, and vpr from SpooCUP2000 [39] and CG, EP, MG from NPB3.2 [40] to evaluate the performance of these schemes. The binary of all these benchmarks are optimized by the O2 option of compiler. Since the object of our study is large capacity cache, we set the NPB benchmarks to class A.

As can be seen in table 1, the L2 cache size is 8M and block size is 64B, so there are 128K blocks in cache. We set the capacity of the RCs of L2RC and ECC-Cache to half the capacity of the L2 cache (64K blocks). PartialECC [31] uses a 4-way associative cache and only one block in each set is equipped with ECC. To provide horizontal and fair comparison, in our 8M capacity PartialECC experiment, we equip 4 block of the 16 blocks of each cache set with ECC.

To evaluate the performance of all the schemes, we use the access latency got from the CACTI simulator. When we used the CACTI simulator we found that, the cache access latency and relatively power consumption are both low if the number of sub-bank is configured to 4. Fig.6 shows the access latency of all schemes. As can be seen, the minimum access latency of No-FT is 4 cycles and the maximum access latency is 27 cycles, which is consistent with the access latency in some existing

---

**TABLE I. SIMULATED PROCESSOR CONFIGURATION**

<table>
<thead>
<tr>
<th>Function units</th>
<th>4 Int ALU, 4 Int Multiply/Divider, 4 FP ADD, 4 FP Multiply/Divider</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction queue</td>
<td>4 instructions</td>
</tr>
<tr>
<td>Fetch/Decode/Issue/Commit Width</td>
<td>4 instructions/cycle</td>
</tr>
<tr>
<td>LSU</td>
<td>8 instructions</td>
</tr>
<tr>
<td>RUU</td>
<td>16 instructions</td>
</tr>
<tr>
<td>L1 Instruction Cache</td>
<td>32KB, 1-way associative, 64B Block Size, 1 cycle latency</td>
</tr>
<tr>
<td>L1 Data Cache</td>
<td>32KB, 4-way associative, 64B Block Size, 1 cycle latency</td>
</tr>
<tr>
<td>L2 Unified Cache</td>
<td>8MB, 16-way, 64B Block Size, access latency from CACTI</td>
</tr>
<tr>
<td>Memory</td>
<td>16B width, first chunk 100 cycle, remaining chunks 2 cycle latency</td>
</tr>
<tr>
<td>Branch predictor</td>
<td>Bimodal, 2K capacity</td>
</tr>
<tr>
<td>BTH</td>
<td>512 entry, 4-way</td>
</tr>
<tr>
<td>Miss prediction penalty</td>
<td>3 cycle</td>
</tr>
</tbody>
</table>

---

Figure 4. The power consumption of all schemes normalized by No-FT.

Figure 5. The extra power consumption of ECC-Cache-* and L2RC-* after normalized by and compared to No-FT.

Figure 6. The access latency of all schemes.
commercial processors. Compared with No-FT, scheme BaseECC-W, scheme BaseECC-B and scheme PartialECC have extra ECC bits stored side by side with normal data bits. In each cache access, not only should they output the normal data bits but also they should output the ECC bits. So the multiplexer driver should drive more multiplexors after the tag comparison step, leading to the slow down of the critical path. Furthermore, the output driver should drive more output bits brought by ECC, which also leads to the increasing cache access latency. Finally, because the ECC bits are cross-distributed in normal data bits, the average length of the routing wires is increased, hence increasing the cache access latency. From the above analysis we know that the access latency of scheme BaseECC-W, scheme BaseECC-B and scheme PartialECC are all larger than that of No-FT. Besides, in these three schemes, there must exist an extra latency brought by the checking course of ECC when data is read from cache or by the generation course of ECC when data is written to cache.

In scheme L2RC or ECC-Cache, the integrity of the dirty data read out from L2 cache should be verified by its redundant copy or ECC code in RC, which increasing the latency by 1 cycle. However, writing data or ECC to RC brings no latency, because the accessing to RC can be overlapped by the normal L2 cache accessing.

The performance of all the schemes is showed in Fig.7. As can be seen, ECC-Cache has almost the same performance with No-FT. Compared with BaseECC-W and BaseECC-B, the greatest advantage of ECC-Cache is its lower access latency. But we must pay attention to the capacity of the RC, the smaller the capacity of the RC is, the more write-backs are generated, thus impact the performance of ECC-Cache. In our experiment we set the capacity of RC to half the capacity of L2 cache and find that ECC-Cache gains better performance than BaseECC-W and BaseECC-B.

Compared with PartialECC, ECC-Cache has two advantages. Firstly, the access latency of its critical path is lower than that of PartialECC. Secondly, the RC of ECC-Cache can be configured more flexible than PartialECC.

so, as an independent redundant component, ECC-Cache can implement large associative degree, which can effectively avoid the compulsive write-backs caused by the limit number of the ECC-equipped blocks in scheme PartialECC.

When the RC of L2RC and ECC-Cache have the same associative degree, such as 16 in our performance experiment, they can achieve the same performance. But their contents are different, blocks in RC of L2RC contains more bits than blocks in RC of ECC-Cache, so the associative degree of L2RC can not be implemented larger than that of ECC-Cache. As we know, the associative degree of the RC has relation to its performance, so L2RC has less space to improve its performance than ECC-Cache.

V. CONCLUSION

With dramatic scaling in feature size of VLSI technology, CMP processor becomes more and more popular and it seems that it adopts low-level cache with larger and larger capacity. The large cache capacity brings not only increasing area or power consumption but also problem of data integrity. So both energy-efficiency and reliability are becoming very important criteria in large capacity cache designs. In this work, we first researched the existed fault tolerant cache architecture and their characteristics, and then we proposed ECC-Cache as a novel fault tolerant architecture for large capacity cache. Power consumption and performance are evaluated for ECC-Cache and some other schemes, results show that ECC-Cache performs better than the uniform-ECC scheme adopted in some widespread used commercial processors and some proposed schemes in other papers. Compared with the cache which has no protection, ECC-Cache only consumes 3% to 6% additional power and degrades performance no more than 2%.

REFERENCES


[38] Toolset S. http://www.simplescalar.com[J].
