A Novel Fault-Tolerant Last-Level Cache to Improve reliability at Near-Threshold Voltage

Wei Liu
College of Computer Science and Technology, Wuhan University of Technology, China
Hubei Key Laboratory of Transportation Internet of Things, Wuhan, China
wliu@whut.edu.cn

Zhigang Wei
College of Computer Science and Technology, Wuhan University of Technology, China
weizhigang@whut.edu.cn

Wei Du
College of Computer Science and Technology, Wuhan University of Technology, China
Hubei Key Laboratory of Transportation Internet of Things, Wuhan, China
whutduwei@whut.edu.cn

ABSTRACT

Near-threshold voltage computing (NTC) improves power and energy efficiency of cache by scaling transistor voltage. However, in large SRAM structures, such as last-level cache (LLC), a great number of bit-cell errors will occur when supply voltage scales to near-threshold voltage. In this paper, we propose a novel fault-tolerant LLC design (NFTLLC) to deal with a high failure rate which is higher than 1% at near-threshold voltage. NFTLLC corrects the single-error and compresses multi-error in Cache entry to improves the reliability of last-level cache. To validate the efficiency of NFTLLC, we implement NFTLLC and prior works in gem5, and simulate with SPEC CPU2006. The experiment shows that compared with Concertina when bit-cell failure rate is 1.1%, the performance of NFTLLC with 4-byte subblock size improves by 6.8% and the Cache capacity increases by 20.8%. Besides, miss rate decreases more than 53%, and overhead increases by 16.8% in minimum.

Keywords

near-threshold voltage; fault-tolerant last-level cache; ECC; compression mechanism

ACM Reference Format:

Wei Liu, Zhigang Wei and Wei Du. 2018. A Novel Fault-Tolerant Last-Level Cache to Improve reliability at Near-Threshold Voltage. In *GLSVLSI '18: 2018 Great Lakes Symposium on VLSI*, May 23–25, 2018, Chicago, IL, USA. ACM, NY, NY, USA, 6 pages.
https://doi.org/10.1145/3194554.3194583

1 INTRODUCTION

Over the past five decades, the number of transistors on a chip has increased exponentially in accordance with Moore's law [1]. However, the number of transistors integrated in a die has been very large, and heat removal limits at the package level have further restricted more advanced integration. So we are facing such a curious design dilemma: more gates can now fit on a die, but a growing fraction cannot actually be used due to strict power limits [2].

The most straightforward way to reduce energy dissipation is to scale supply voltage(Vdd), since the energy consumption in CMOS circuits can be reduced quadratically by lowering Vdd. Near-threshold voltage computing (NTC) [3] is a novel approach that demonstrates greater energy efficiency and provides much more flexible tradeoffs between energy and delay than the conventional sub-threshold computing circuits [4]. Although NTC is a promising energy-efficient solution to circuit design, it brings its own sets of complications. Process variation is one of key barriers faced by NTC, which will cause mismatch in device and result in functional failure in SRAM cell. High cell failure rate relevant to increased process variation in LLC will slow down the application execution, produce incorrect results, and even crash the system [2].

Existing research to improving cache reliability can be classified into two types: circuit-level and architecture-level. Circuit-level techniques improve bit cell reliability by either upsizing transistors [5] or using alternative SRAM cell designs, such as eight-transistor (8T) [6], ten-transistor(10T) [7] or Schmitt trigger-based (ST) SRAM cells [8]. These techniques can have a great improvement on reliability of LLC in NTC, while they incur significant area overhead, higher power consumption and higher access latency.

At architecture-level, the reliability of Cache is improved in many different ways. Zeshan Chishti et al. propose MS-ECC to correct multi-bit error by sacrificing half of cache ways at fine sub-cache line granularities [9]. Alameldeen et al. propose VS-ECC to design an energy-efficient cache [1], which combines SECDED and 4EC5ED to correct bit-cell errors in cache line. Hijaz et al. propose a new cache architecture by correcting the data stored in cache using ECC, and disabling the cache lines which can't be corrected by ECC [10]. Duwe et al. rearrange the logical bit to physical bit mapping to transform the uncorrectable error patterns in logical words into correctable ones [11]. Ferrerón et al. propose Concertina, a compression mechanism that compresses cache blocks in order that they can be allocated

GLSVLSI '18, May 23–25, 2018, Chicago, IL, USA
 ACM 978-1-4503-5724-1/18/05…$15.00
https://doi.org/10.1145/3194554.3194583

to cache entries with faulty cells [12]. Palframan et al. present iPatch, which rely on existing microarchitectural structures and mechanisms to "patch" the faulty parts of caches [13]. Jung et al. manage data by cache coherence states and several layers of cache organization [14]. All of these techniques have a good impact on cache reliability when bit-cell failure rate is less than 0.5%. However, it is not suitable for current SRAM cell which has a higher failure rate such as 1% at near-threshold voltage.

In this paper, we present a novel fault-tolerant last-level cache (NFTLLC) based on compression mechanism. NFTLLC uses a detector to detect the null subblocks in the incoming block and compress the block into a smaller block which has no null subblocks. Then, an ECC encoder encodes the compressed block with a simple error correction code. Using a simple ECC technique to correct a cache entry with multi-bit error incurs a significant area for the extra circuits to store ECC check bits. We propose an optimized protection policy to achieve a tradeoff between performance and area overhead.

The rest of this paper is organized as follows. Section 2 presents the motivation of our work. Section 3 explains our proposed architecture in detail. Section 4 and 5 present the evaluation methodology and the results of evaluation. Finally, we conclude in section 6.

2 MOTIVATION

With the development of semiconductor technology, MOS device is becoming smaller and smaller. But when the process scale into nanometers, process variation plays a more important role on reducing device reliability. Random Dopant Fluctuation(RDF) is a main factor that changing the threshold voltage of transistors and leading to some SRAM cells weaker than other cells in the cache. Weak cells become faulty as the supply voltage is reduced, the number of which is reflected by bit-cell failure rate. Since these faulty cells are caused by permanent defects (e.g., dopant variation), they can be located using a number of built-in self test (BIST) routines [1, 4]. Some faults at low voltages are due to soft errors that cannot be detected by BIST routines. However, the fraction of such faults caused by soft errors at low voltage is minuscule [15] (by over 5 orders of magnitude at 650mV [9]).

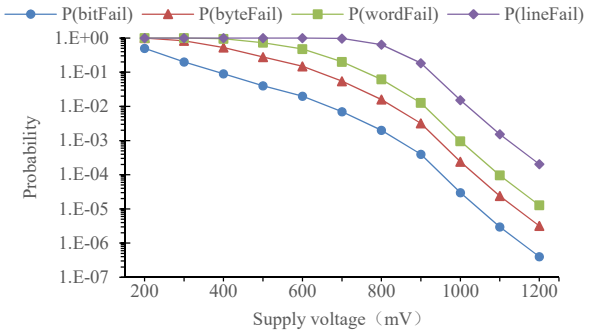

Figure 1: The probability of a single bit failure and the probabilities of a byte, 32-bit word and a line that are faulty in a cache for 65nm SRAM cells.

In Fig. 1, we show the bit failure data from [2], indicating the probability of a single 65nm SRAM bit failure ($P_{bitFail}$) across a range of voltages. We also assume that SRAM failures are random in nature [16]. So we compute the probabilities of having errors in a byte ($P_{byteFail}$), a 32-bit word ($P_{wordFail}$) and a cache line ($P_{lineFail}$) as follows:

$$P_{byteFail} = 1 - (1 - P_{bitFail})^8 \quad (1)$$

$$P_{wordFail} = 1 - \left(1 - P_{byteFail}\right)^4 \quad (2)$$

$$P_{lineFail} = 1 - (1 - P_{wordFail})^{16} \quad (3)$$

The probabilities $P_{byteFail}$, $P_{wordFail}$ and $P_{lineFail}$ are also plotted in Fig. 1. By examining Fig. 1, we observe that the probability of a block having errors increases with the decrease of supply voltage. When the supply voltage scales to 650mV, the probability of a bit failure increases to 1.1%. This failure rate means more than 99% of cache entries are faulty, which leads to functional failure in LLC.

Considering that LLC requires a large capacity and low miss rate to reduce average latency of data requests from L1 Cache, NFTLLC is proposed to improve performance and reliability of LLC at near-threshold voltage with an acceptable area overhead.

3 ARCHITECTURE DESIGN AND IMPLEMENT

We present NFTLLC, a novel fault-tolerant last-level cache that combined with compression mechanism and weak ECC techniques. Compression mechanism compresses data by ignoring the null subblocks in a data block and stores the data into a cache entry which has disabled the faulty subentries, while there are too much faulty subentries in an entry at near-threshold voltage. In order to increase the available capacity of a cache entry, we use a weak ECC to correct the faulty cells in each entry. The combination of compression and error correction mechanisms ensure a minimal degradation on performance and improve the cache reliability.

3.1 Operation and Component Description

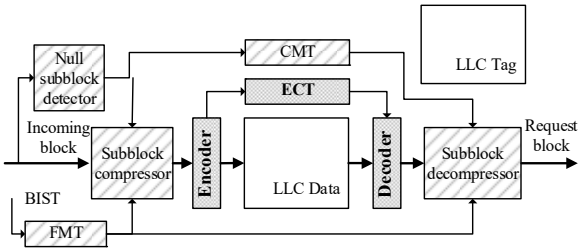

Figure 2. NFTLLC design overview

Our proposed NFTLLC is shown in Fig. 2, it consists of 3 parts: conventional LLC module, compression module and ECC module. Conventional LLC is the area which the background color is white. Compression module is filled with light grey, and ECC

module is filled with dark grey. We recognize that the additional logic inserted on the critical path of a cache access may result in an extra latency, and its overhead is presented in Section 4.

When a data block is evicted from L1 Cache and written back to LLC, our proposed NFTLLC performs in three steps. First, NFTLLC detects the location of null subblocks in a block by null subblock detector. The detector transfers the location information to subblock compressor and compression map table (CMT). Then, Subblock compressor combines the location information with the distribution of faulty subentries in selected entry to rearrange the non-null subblocks into non-faulty subentries. Finally, the cache entries containing compressed block are encoded by Encoder, which use SECDED to protect data and generate ECC check bits.

When a read request comes from L1 Cache or a compressed block need to be evicted from LLC, our proposed NFTLLC firstly uses Decoder to check if the data stored in selected cache entry is correct. Decoder corrects the faulty subentries if these subentries are correctable. Then, we use null subblock decompressor to reconstruct the original block by rearranging corrected subblocks according the information from CMT and the distribution of faulty subentries from fault map table (FMT). After decompressing the compressed block, we finally get the original block stored in LLC.

3.2 Compression Mechanism Design

Compression mechanism in our proposed LLC is based on Concertina [12]. During a cache access, FMT and CMT play a key role on compressing/decompressing and rearranging data. For the FMT, we set '1' in each bit to indicates whether the subentry is non-faulty or correctable. If the subentry is faulty and uncorrectable, we set the corresponding bit with '0'. The problem that if a subentry is correctable can be judged from ECT (ECC check table), which will be presented in Section 3.3. For the CMT, bit value '0' means the subblock is filled with zero and bit value '1' shows that the subblock has valid data. A subblock filled with zero can be compressed, it can be removed when written into cache and can be refilled to its original location when read from LLC. The granularity of compression can be, such as 1-byte, 2-byte and 4-byte subblock etc. For example, a 64-byte block can be divided into 16 4-byte subblocks, so each entry requires 16 bits to store corresponding information in CMT and FMT.

CMT is generated by null subblock detector, and it changes with data incoming. We consider that the number and the location of faulty cells do not change during LLC working. So FMT comes from a built-in self-test routine (BIST) that can be executed at boot up time and tests the last-level cache at the target low voltage [4, 11, 17]. The initialization of FMT can be seen in Procedure 1. We first initialize FMT with faulty-free subentries, for faulty-free subentry can hold right data. Then, we Initialize ECT to correct the correctable faulty subentries which can right hold data by using ECC. Finally, we update FMT with these correctable faulty subentries to increase the number of available subentries in an entry.

CMT combines with FMT compresses and rearranges a data block to a cache entry by subblock compressor. We can use an example (Fig. 3) to explain the process of compressing the incoming blocks. CMT indicates the location of corresponding block, and each non-null subblock is rearranged to non-faulty subentries. The detailed process circuit can be seen in Concertina [12], and the subblock decompressor has a similar circuit.

We set *N* as the number of subentries (subblocks) in a cache entry (block). So we need extra *2N* bits to compress an entry, which lead to a great increase on area overhead. To guarantee robustness at low voltages, we assume a 10T SRAM based implementation for CM and UFM. The compression logic requires N^2 comparators of *log2(N)* bits, and N^2 tri-state buffers. When the subentry size decreases (finer granularity), both the number of the comparators and the size of their inputs increase, but overall, the overhead is still low.

Procedure 1 FMT initialization

```
Input: Empty FMT array, LLC array
Output: Configured FMT array
for each Entry in LLC do
  for each subentry in Entry do
    //Initialize FMT with each subentry
    if subentry is faulty-free then
      Index bit of corresponding subentry in FMT ← 1;
    else
      Index bit of corresponding subentry in FMT ← 0;
    end if
  end for
  Correct faulty subentries with ECT array;
  for each subentry in Entry do
    if subentry is faulty and correctable in ECT then
      //This subentry can hold data correctly.
      Update index bit of corresponding subentry in FMT
      with '1';
    end if
  end for
end for
return UFM;
```

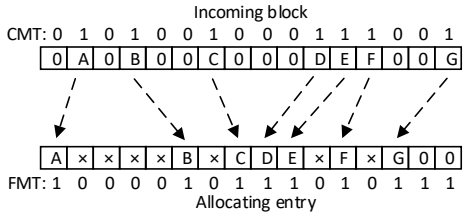

Figure 3. Example of compressed block to be rearranged to a faulty entry

3.3 Implement of Error Correction Code Module

When we insert a block to LLC, ECC encoding is the second phase in our proposed NFTLLC. Typically, most proposed ECC techniques are based on BCH code. SECDED is a weak ECC technique based on BCH code. We use SECDED as our ECC technique for its low latency overhead with only one cycle.

However, SECDED doesn't meet the requirement of cache at high failure rate, which has multi-error in a cache entry. Besides, the employ of other complex ECCs such as 4EC5ED leads to more latency increase. When 4EC5ED is implemented in cache, it adds one extra cycle latency in encoding and at least 7 cycles latency in decoding, and a significant performance degradation occurs [1].

Considering that multi-bit correction requirement and the overhead of access latency. We divide a whole cache entry into multiple subentries with a sort of correction granularity. 1-byte, 2-byte and 4-byte subentry size can be selected as correction granularity, and the correction granularity are required to be the same as compression granularity. Then, we use SECDED to protect data in each subentry. To decrease the area of ECT, we propose an optimized protection policy.

For M subentries in a cache entry, we try to protect k (k<M) faulty subentries using SECDED. When the value k increases, the number of subentries that can be protected and corrected increases correspondingly. E.g. a 64-byte cache entry has 16 subentries for a 4-byte subentry, we set k=4. Then, the architecture of ECT (ECC Check Table) is shown in Fig. 4.

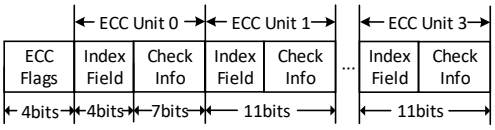

Figure 4. The architecture of ECT

ECT consists of ECC Flags and 4 ECC Units. ECC Flags indicate whether the ECC Units are used, and the size of ECC Flags is equal to the number of ECC Units. Each ECC Unit contains 11 bits, and it consists of Index Field and Check Info. Index Field indicates the index of subentry protected by this ECC Unit, which use 4 bits to represent the identity of each subentry. Check Info containing 7 bits is check code generated by Encoder. Each time LLC gets read request, Check Info is fetched to check whether the subentry from LLC is right and correct the subentry if it is faulty. ECT is defective at near-threshold voltage, so we implement ECT with 10T ST SRAM cell to keep it reliable.

Procedure 2: ECT initialization

Input: Empty ECT array, Cache entry
Output: Configured ECT array
for each ECCenrty in ECT **do**
  Initialize ECCentry with '0';
  for each Unit in ECCentry **do**
    if cache entry has correctable subentries **then**
      Get a correctable subentry from the cache entry;
      if Unit is unused **then**
        IndexField ← Index of subentry;
        Set Unit in use;
      end if
    end if
  end for
end for
return ECT;

NFTLLC(k=4) needs 52 bits in total to implement ECT with 4 ECC Units. And 12k bits are needed when we want to protect k subentries in a cache entry. By analyzing the failure rate of 1.1% in LLC, we find the average number of faulty subentries ranges from 4.77 to 5.42 in different subentry sizes, so we set k range from 4 to 7 for it having an impact on performance and area overhead.

Only Check Info in ECT are set and updated during regular cache accesses, while all other bits in the ECT are set at runtime by BIST that can be executed at boot up time. First, BIST initializes all the bits in ECT array with '0'. Then, it tests each subentry of a cache entry, and a subentry will be protected by an unused ECC Unit if the subentry is faulty and correctable. More details of initializing ECT can be seen in Procedure 2.

3.4 Replacement Policy

In this paper, our proposed NFTLLC based on compression mechanism has variable number of faulty subentries in each entry, and variable number of null subblocks with incoming blocks. So, we use Fit-LRU policy, a LRU-based management policy to deal with different between entries and blocks [12]. When a block is to be inserted, we first find a set of entries that can accommodate the compressed block. Then we search the set of entries for a LRU entry. If there is no available entry to store the compressed block, we write back this block to main memory directly. By this policy, we avoid writing a block to a cache entry which can't accommodate the compressed block. Thus, the reliability of LLC is improved and the data is kept right in LLC.

4 EVALUATION METHODOLOGY

To evaluate the impacts of NFTLLC, we use Gem5 simulator to construct the CMP platform [18]. The important architectural parameters used for evaluation are shown in Table 1. We assume a frequency of 1 GHz at 650 mV (our target near-threshold Vdd) and set the bit-cell failure rate to 1.1%. L1 caches are built with robust SRAM cells and, therefore, they can run at lower voltages without suffering the bit-cell functional failure. As in previous studies [4, 12], we assume that LLC tag arrays are built with 10T SRAM [8], for it having significantly smaller latency and area overhead compared to LLC Data arrays.

Table 1. Architectural parameters used for evaluation

Cores	8, in-order, single-threaded, 1 GHz@650mV
L1 DCache	32KB, 4-way associative, private, 2 cycles
L1 ICache	32KB, 4-way associative, private, 2 cycles
L2 Cache (LLC)	1MB, 8-way associative, shared, 8 cycles, 1 extra cycle for decompression/compression, 1 extra cycle for ECC decoding/encoding
Coherence protocol	MOESI, Full-map, directory-based
Memory	2 channels, 2GB per channel, raw access time 50 ns
Network	Mesh, 1 cycle link latency

We use a set of 10 multiprogrammed workloads built from SPEC CPU 2006 as our benchmarks [19]. Each workload is combined with 8 programs random selected from the 29

programs in SPEC CPU 2006, with no special distinction between integer and floating point. We run cycle-accurate simulations including 300 million cycles to warm up the memory hierarchy and 700 million cycles for data collection.

We compare NFTLLC with Concertina and "Correction+Disabled" at near-threshold voltage. Concertina is a novel technique that use compression mechanism to make full use of cache subentries that are fault-free [12]. "Correction+Disabled" is modified from [10] for its unsuitable on high failure rate. To improve the reliability of this work, we use SECDED to protect fine-grained subblocks such as 1-byte subblocks, 2-byte subblocks and 4-byte subblocks. We implement an ideal cache as baseline, which is assumed that has no extra latency overhead and 100% available capacity at all voltages.

5 EVALUATION RESULTS

In this section, we present the evaluation results for our NFTLLC. We first present available capacity of last-level caches with different techniques in different subblock/sub-entry sizes. Then, we explore performance and access miss rate variety at different situations. Finally, we analyze the area overhead.

5.1 Available cache capacity

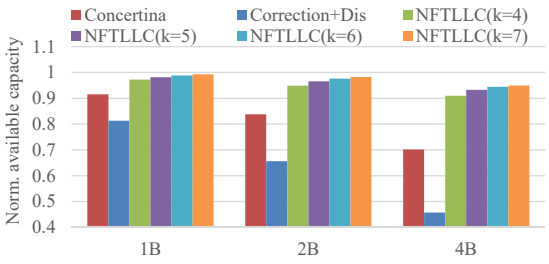

Figure 5. Available capacity of different caches in different subblock/subentry sizes

Reliable Cache design can address the faulty cells and improve available cache capacity, for faulty cells causing a great loss on available capacity. Fig. 5 shows available cache capacity variety at different situations and the results are normalized with ideal cache. The available cache capacity in a LLC architecture increases with the decrease of subblock/subentry sizes. We can get maximal available capacity at 1-byte subblock/subentry size, and minimal ones at 4-byte grain size. For a 4-byte subblock/subentry, "Correction+Disabled" has only 45.68% available capacity compared with ideal cache, and the available cache capacity of Concertina reaches to 70.17%, while that of NFTLLC is more than 90.95%, which is far larger than others. When k varies from 4 to 7, available capacity improves from 90.95% to 94.94% of ideal cache capacity, and the capacity improves slowly. This means that NFTLLC(k=4) has a good effect on available cache capacity, and the increase of k has less effect on it.

5.2 Miss rate of LLC accesses

As we can see from Fig. 6, miss rate of LLC accesses increases with the increase of subblock/subentry size. For caches using 4-byte subblocks/sub-entries, normalized miss rate of Concertina is about 2.33 times of ideal cache, and "Correction+Disabled" increases 21% than ideal cache on the number of access misses. Normalized miss rate of NFTLLC ranges from 1 to 1.095, and it almost reaches the same miss rate of ideal cache at some situation, such as NFTLLC(k=7) with 1-byte subblocks/sub-entries. Besides, the miss rate of NFTLLC increases with the increase of subblock/subentry size and the decrease of value k. As the value of k increase, the number of corrected subentries and non-faulty subentries is increasing in each entry. So more entries are available to hold compressed data in one cache set, which will lead to a great decrease on miss rate.

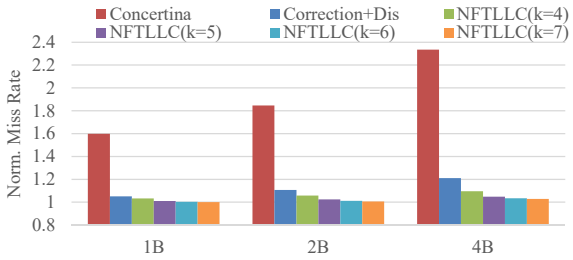

Figure 6. Miss rate of different caches in different sub-block/subentry sizes

5.3 Performance

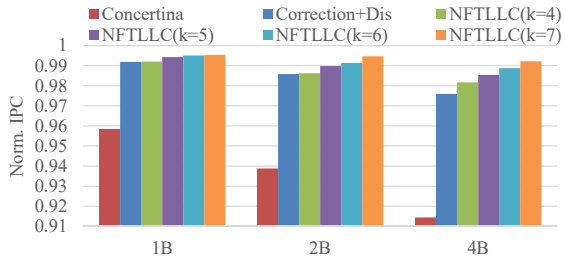

Figure 7 Performance of different caches in different sub-block/subentry sizes

Fig. 7 shows the performance of NFTLLC relative to Concertina and "Correction+Disabled". We use IPC (Instructions Per Cycle) to reflect the performance of CPU. And a reliable cache takes a great impact on performance. As is shown in Fig. 7, the normalized IPC of our NFTLLC with 4-byte subblocks/subentries ranges from 0.982 to 0.992, while the normalized IPC of Concertina is 0.914 and "Correction+Disabled" is 0.976. Compared with Concertina, the IPC of NFTLLC(k=4) improves by 6.8%. This means that miss rate of LLC access has a greater impact on performance than access latency, and NFTLLC have a significantly lower performance degradation than other cache designs at the same subblock/subentry size. In addition, the performance of LLC increases with the subblock/subentry size decreasing, and it improves with k increasing in NFTLLC.

5.4 Overhead Analysis

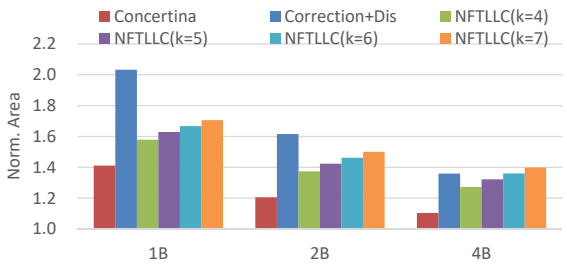

Figure 8. Area overhead compared to ideal LLC in different block/subentry sizes

FMT, CMT and ECT array in our NFTLLC need some storage circuits to store the information, which increases the area overhead. Besides, the area overhead of ECT array changes with value *k*. We use CACTI [20] to quantify these costs, and normalize them with ideal cache using 6T SRAM. Results are shown in Fig. 8.

Concertina takes the least extra area overhead compared with other architecture at different subblock/subentry sizes in Fig. 8, while "Correction+Disabled" has the largest area overhead. Compared with ideal cache, the area overhead of NFTLLC varies from 27.2% to 70.5%, and it takes at least 16.8% extra area overhead relative to Concertina at 4-byte subblock/subentry size. Fig. 7 shows that performance of NFTLLC increases little with the decrease of subblock/subentry size. So we can get the least area overhead and a good performance with NFTLLC(*k*=4) at 4-byte subblock/subentry size among all the NFTLLCs.

Although Concertina has the least area overhead, its performance degrades the most. And "Correction+Disabled" takes the most area overhead above others for its better performance than Concertina. In contrast to prior work, NFTLLC has the best performance with a little area overhead increasing, while it ensures the most of available cache capacity.

6 CONCLUSIONS

Near-threshold voltage technique presents some challenges on cache reliability, functional failure of SRAM cells is one of the largest barriers. A lot of architectures are proposed to deal with low bit-cell failure rate in cache. However, bit-cell failure rate is becoming higher for the scaling of device and supply voltage. NFTLLC are proposed to explore a higher failure rate at near-threshold. We propose an error correction module based on SECDED, and correct faulty subentries in last-level cache. And a compression mechanism is proposed to compress the incoming block and the uncorrectable faulty subentries of each entry. Our NFTLLC incurs at least 6.8% performance improvement at 4-byte subblock/ subentry size compared with Concertina, and takes minimally 16.8% extra area overhead.

ACKNOWLEDGMENTS

This paper is supported by the Ministry of Education Project of Humanities and Social Sciences (Project Name, Research on fuzzy QoE aware and trustworthy cloud services selection in mobile commerce, Project No. 16YJCZH014), the Natural Science Foundation of Hubei Province of China (Project Name, Research on the replica-aware data acquiring mechanism for mobile cloud computing, Project No. 2016CFB466), and the Fundamental Research Funds for the Central Universities (WUT:2016III028).

REFERENCES

[1] A. R. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson and S. L. Lu. Energy-efficient cache design using variable-strength error-correcting codes. ACM International Symposium on Computer Architecture, 2011, pp. 461-472.
[2] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester and T. Mudge. Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits. Proceedings of the IEEE, 2010, vol. 98, no. 2, pp. 253-266.
[3] D. Markovic, C. C. Wang, L. P. Alarcon, T. T. Liu and J. M. Rabaey. Ultralow-Power Design in Near-Threshold Region. Proceedings of the IEEE, 2010, vol. 98, no. 2, pp. 237-252.
[4] A. Ansari, S. Feng, S. Gupta and S. Mahlke. Archipelago: A polymorphic cache design for enabling robust near-threshold operation. IEEE International Symposium on High Performance Computer Architecture, 2011, pp. 539–550.
[5] S. T. Zhou, S. Katariya, H. Ghasemi and S. Draper. Minimizing total area of low-voltage SRAM arrays through joint optimization of cell size, redundancy, and ECC. IEEE International Conference on Computer Design, 2010, vol. 41, no. 3, pp. 112-117.
[6] L. Chang, R.K. Montoye, Y. Nakamura, K.A. Batson, R. J. Eickemeyer and R. H. Dennard. An 8T-SRAM for variability tolerance and low-voltage operation in high-performance caches. IEEE Journal of Solid-State Circuits, 2008, vol. 43, no. 4, pp. 956-963.
[7] B. H. Calhoun and A. P. Chandrakasan. A 256-kb 65-nm sub-threshold SRAM design for ultra-low-voltage operation. IEEE Journal of Solid-State Circuits, 2007, vol. 42, no. 3, pp. 680-688.
[8] J. P. Kulkarni, K. Kim and K. Roy. A 160mV robust schmitt trigger based subthreshold SRAM. IEEE Journal of Solid-State Circuits, 2007, vol. 42, no. 10, pp. 2303-2313.
[9] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu and S. L. Lu. Improving cache lifetime reliability at ultra-low voltages. IEEE/ACM International Symposium on Microarchitecture, 2009, pp. 89-99.
[10] F. Hijaz, Q. Shi and O. Khan. A private level-1 cache architecture to exploit the latency and capacity tradeoffs in multicores operating at near-threshold voltages. International Conference on Computer Design, 2013, pp. 85-92.
[11] H. Duwe, X. Jian, D. Petrisko and R. Kumar. Rescuing uncorrectable fault patterns in on-chip memories through error pattern transformation. International Symposium on Computer Architecture, 2016, pp. 634-644.
[12] A. Ferrerón, D. Suarez-Gracia, J. Alastruey-Benede, T. Monreal-Arnal and P. Ibanez. Concertina: Squeezing in Cache Content to Operate at Near-Threshold Voltage. IEEE Transactions on Computers, 2016, vol. 65, no. 3, pp. 755-769.
[13] D. J. Palframan, N. S. Kim and M. H. Lipasti. iPatch: Intelligent fault patching to improve energy efficiency. IEEE International Symposium on High Performance Computer Architecture, 2015, pp. 428-438.
[14] D. Jung, H. Lee and S. W. Kim. Lowering Minimum Supply Voltage for Power-Efficient Cache Design by Exploiting Data Redundancy. ACM Transactions on Design Automation of Electronic Systems, 2015, vol. 21, no. 1, pp. 1-24.
[15] H. Duwe, X. Jian, and R. Kumar. Correction prediction: Reducing error correction latency for on-chip memories. IEEE, International Symposium on High PERFORMANCE Computer Architecture IEEE, 2015:463-475.
[16] Mukhopadhyay, S., Mahmoodi, H. and Roy, K. Modeling of failure probability and statistical design of sram array for yield enhancement in nanoscaled cmos. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2005, 24(12), 1859-1880.
[17] J. Wang, Y. Han, H. Li and X. Li. Exploring Variation-Aware Fault-Tolerant Cache under Near-Threshold Computing. International Conference on Parallel Processing IEEE, 2016:149-158.
[18] N Binkert, B Beckmann and G. Black. The gem5 simulator. ACM Sigarch Computer Architecture News, 2011, vol. 39, no. 2, pp. 1-7.
[19] Standard Performance Evaluation Corporation. Spec cpu2006. Available: www.spec.org/cpu2006.
[20] N Muralimanohar, R Balasubramonian and N P. Jouppi. CACTI 6.0: A Tool to Model Large Caches. Bragantia, 2009.