ELectronics EXpress

LETTER

IEICE Electronics Express, Vol.12, No.8, 1–8

3DFTL: a three-level demand-based translation strategy for flash device

Peera Thontirawong[1a)], **Chundong Wang**[2], **Weng-Fai Wong**[3], **Mongkol Ekpanyapong**[4], **and Prabhas Chongstitvatana**[1b)]

[1] *Faculty of Engineering, Chulalongkorn University, Phaya Thai Road, Wangmai, Pathumwan, Bangkok 10330, Thailand*

[2] *Data Storage Institute, Agency for Science, Technology and Research, 5 Engineering Drive 1, Singapore 117608, Singapore*

[3] *School of Computing, National University of Singapore, 13 Computing Drive, Singapore 117417, Singapore*

[4] *School of Engineering and Technology, Asian Institute of Technology, Km. 42, Paholyothin Highway, Klong Luang, Pathumthani 12120, Thailand*

a) *peera.t@student.chula.ac.th*

b) *prabhas@chula.ac.th*

Abstract: 3DFTL is a demand-based flash translation layer (demand-based FTL) that can withstand caching data loss due to unexpected events such as power-loss. Its mapping table in the flash memory is designed with the capabilities of being instantaneously updated with zero additional write operations. Moreover, the average cache miss penalty of 3DFTL is also lower than previous demand-based FTLs. As a result, not only the mapping table of 3DFTL guarantees data consistency, but 3DFTL also shows 16.42% decrease in terms of the average system response time comparing with the DFTL.

Keywords: flash memory, MLC, FTL, cache, three-level, compression

Classification: Storage technology

References

[1] D. Ma, J. Feng and G. Li: ACM Comput. Surv. **46** (2014) 36. DOI:10.1145/2512961

[2] A. Gupta, Y. Kim and B. Urgaonkar: ASPLOS (2009) 229. DOI:10.1145/1508244.1508271

[3] Micron Technology: MT29F64G08CBAA datasheet (2009) http://www.micron.com.

[4] Storage Performance Council: Traces (2007) http://www.storageperformance.org.

[5] D. Narayanan, A. Donnelly and A. Rowstron: Trans. Storage **4** (2008) 10. DOI:10.1145/1416944.1416949

[6] Z. Qin, Y. Wang, D. Liu and Z. Shao: RTAS (2011) 157. DOI:10.1109/RTAS.2011.23

[7] P. Thontirawong, M. Ekpanyapong and P. Chongstitvatana: ICSEC (2014) 421. DOI:10.1109/ICSEC.2014.6978234

© IEICE 2015

DOI: 10.1587/elex.12.20150211
Received February 23, 2015
Accepted March 11, 2015
Publicized March 30, 2015
Copyedited April 25, 2015

1 Introduction

As flash memory outperforms ferromagnetic materials on access latency, shock resistance, and power consumption, it is more preferred for data storage of computer systems. It can be utilized in many devices, for instance, USB drives, solid-state drives, or even the embedded storage of smartphones. However, the NAND flash memory has several limitations [1]. For example, a page — the smallest operable unit — has to be erased before reprograming. Since the smallest erasable unit is a block of pages, an out-of-place update is more feasible than an in-place update. Moreover, the lifespan of a flash memory is limited by program/erase (P/E) cycles, and this limit is lowered with the MLC technique or a smaller fabrication process. As a consequence, a flash translation layer (FTL) is required for handling these limitations. An FTL enables out-of-place update by deploying address translation. It converts a logical page number (LPN), which is referred to by the file system, to a physical page number (PPN) of a flash memory. Since a page is hundreds times smaller than a block, performing address translation at the page-level is necessary for lowering P/E cycles.

An FTL that offers the page-level address translation is called a page-level FTL [1]. A page-level FTL is found on a page-level mapping table (PMT). However, PMT is huge and takes spacious SRAM capacity. In order to save the SRAM space, a demand-based FTL was proposed [2]. A demand-based FTL offloads PMT contents from SRAM to flash pages. These pages are called translation pages and can be located by the smaller mapping table that resides in SRAM. Therefore, the data structure of PMT becomes two-dimensional array-like, and an address translation is done by a two-level process. The first level is to get the address of the corresponding translation page from the small table in SRAM, and the second level is to retrieve the PPN from the translation page. The retrieved PPN is cached in SRAM for quick reference. The cache is managed by the write-back policy in order to lower the number of program operations; however, it also stalls the update of translation pages. As a result, the whole operation becomes non-atomic, and an inconsistency problem between data pages and translation pages is arisen. This problem is very important for an FTL because the flash memory is vastly employed in mobile devices that have to confront many unexpected events, for example, power-loss. The FTL has to tolerate such and ensures correctness of data locations.

In this paper, we propose a novel demand-based FTL named 3DFTL. Without translation pages, updating data does not require additional page programing; hence, it is inconsistency-free. The cache miss ratio of 3DFTL is kept low by spatial locality exploitation. In addition, omitting translation page programing also decreases the maximum cache miss penalty, which in turn improves the average system response time.

2 Demand-based three-level address translation

3DFTL is an FTL with a cache for the page-level mapping table (PMT). Typical demand-based FTLs reduce the spatial requirement of SRAM by moving PMT entries to data areas of flash memory pages, which in turn causes the inconsistency problem. To overcome this obstacle, 3DFTL places a PMT entry in the spare area of

© IEICE 2015
DOI: 10.1587/elex.12.20150211
Received February 23, 2015
Accepted March 11, 2015
Publicized March 30, 2015
Copyedited April 25, 2015

the page that stores the corresponding data instead. As both data and mapping information are stored in the same page, updating is considered as an atomic operation.

Since a spare area is much smaller than a data area, PMT demands more pages for storing its entries. In order to maintain the reasonable SRAM size, 3DFTL employs a three-dimensional array structure for PMT as illustrated in Fig. 1.

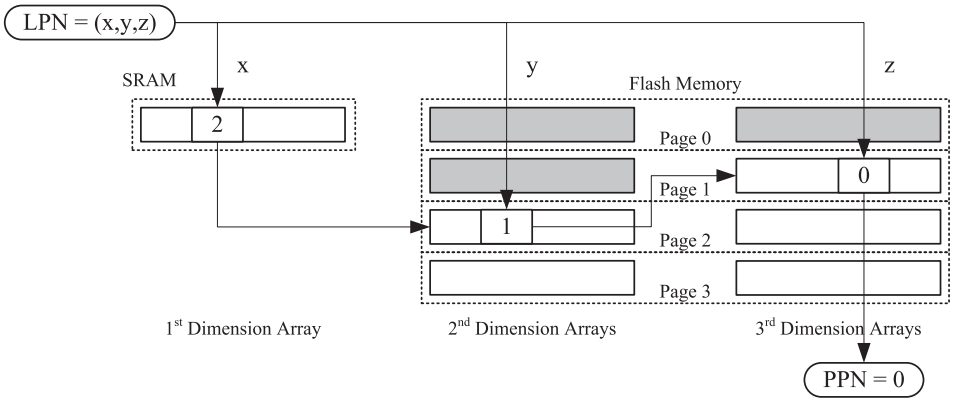

Fig. 1. The example of three-dimensional array for 3DFTL PMT.

The first-dimension array is kept in SRAM, and their contents are PPNs of pages that have the up-to-date second-dimension arrays stored in the spare areas. Likewise, the contents of the second-dimension array are PPNs of pages that have the up-to-date third-dimension arrays. Since each page contains data, the corresponding second-dimension array, and the related third-dimension array, the inconsistency problem is ceased to exist.

However, the address translation has to be done by a three-level process. For example, supposing that the translated PPN was stored in PMT_{xyz} where x, y, and z are the head, middle, and tail bits of LPN, respectively. PMT_x and PMT_{xy} have to be respectively retrieved in order to access PMT_{xyz}. Thus, we call the accessing to PMT_x, PMT_{xy}, and PMT_{xyz} as the first-, second-, and third-level address translation, respectively.

In addition, 3DFTL has a cache for the second level PMT entries and the third level PMT entries. Each cache entry is organized by the first index of PMT. For instance, $Cache_x$, a cache entry, can contain PMT_{xj} and PMT_{xjk} for all possible j and k. As a result, re-reference is faster due to the exploitation of temporal locality and spatial locality. The address translation of 3DFTL is described by pseudocode in Fig. 2.

Although each address translation can trigger two flash memory read operations, the second read operation can be omitted if the required third level PMT entry is in the same page as the retrieved second level PMT entry. In other words, packing more third level PMT entries in one page can decrease the number of read operations.

Due to the spatial locality of data write requests, the most significant bits (MSBs) of PPNs of nearby LPNs are having high likelihood of repetition. 3DFTL takes advantage of this property by employing a compression technique in order to

© IEICE 2015
DOI: 10.1587/elex.12.20150211
Received February 23, 2015
Accepted March 11, 2015
Publicized March 30, 2015
Copyedited April 25, 2015

1: **procedure** TRANSLATE(LPN)
2: $(x, y, z) \leftarrow LPN$ ▷ Convert LPN to indices of PMT
3: **if** $Cache_x$ is not existed **then** ▷ Cache miss
4: $PPN_1 \leftarrow PMT_x$ ▷ 1^{st} Level
5: **read** PPN_1
6: $\forall j(Cache_{xj} \leftarrow PMT_{xj})$
7: $\forall j \forall k(Cache_{xjk} \leftarrow \{PMT_{xjk} | PMT_{xjk} \in PPN_1\})$
8: $\forall j \forall k(Cache_{xjk} \leftarrow \{Cache_{xj} | PMT_{xjk} \notin PPN_1\})$
9: **end if**
10: **if** $\exists k(Cache_{xyz} = Cache_{xyk} \wedge z \neq k)$ **then** ▷ Cache miss
11: $PPN_2 \leftarrow Cache_{xy}$ ▷ 2^{nd} Level
12: **read** PPN_2
13: $\forall k(Cache_{xyk} \leftarrow PMT_{xyk})$
14: **end if**
15: $PPN_3 \leftarrow Cache_{xyz}$ ▷ 3^{rd} Level
16: **return** PPN_3
17: **end procedure**

Fig. 2. Pseudocode of three-level address translation

1: **procedure** COMPRESS($LPN, Cache$)
2: $(x, y, z) \leftarrow LPN$ ▷ Convert LPN to indices of PMT
3: $DICT$ = set of $PPN.index$
4: **sort** $DICT$ in chronological order ▷ Newest first
5: **for all** $Cache_{xjk}$ **do**
6: **find** p that $DICT_p = Cache_{xjk}.index$
7: $CompressedPPN_{jk}.position \leftarrow p$
8: $CompressedPPN_{jk}.offset \leftarrow Cache_{xjk}.offset$
9: **end for**
10: **Comment** Embedded LPN in $DICT_0$ and $CompressedPPN_{yz}$
11: $DICT_0 \leftarrow$ LSBs of LPN including y and z
12: $CompressedPPN_{yz}.offset \leftarrow$ MSBs of LPN
13: **end procedure**

14: **procedure** DECOMPRESS($PPN, DICT, CompressedPPN$)
15: $(y, z) \leftarrow$ LSBs of $DICT_0$
16: $LPN \leftarrow (CompressedPPN_{yz}.offset, DICT_0)$
17: $x \leftarrow$ MSBs of LPN
18: $DICT_0 \leftarrow PPN.index$
19: $CompressedPPN_{yz}.offset \leftarrow PPN.offset$
20: **for all** $CompressedPPN_{jk}$ **do**
21: $p_{jk} \leftarrow CompressedPPN_{jk}.position$
22: $o_{jk} \leftarrow CompressedPPN_{jk}.offset$
23: $Cache_{xjk} \leftarrow (DICT_{p_{jk}}, o_{jk})$
24: **if** $(p_{jk} < p_j) \vee ((p_{jk} = p_j) \wedge (o_{jk} > o_j))$ **then**
25: $p_j \leftarrow p_{jk}, o_j \leftarrow o_{jk}$
26: $Cache_{xj} \leftarrow Cache_{xjk}$
27: **end if**
28: **end for**
29: **end procedure**

Fig. 3. Pseudocode of compression and decompression procedures

© IEICE 2015
DOI: 10.1587/elex.12.20150211
Received February 23, 2015
Accepted March 11, 2015
Publicized March 30, 2015
Copyedited April 25, 2015

make room for more PPNs. A PPN, which is the content of PMT entry, is split into two parts: index (MSBs) and offset. A duplicated index is omitted from the spare area; hence, extra PPNs can be stored.

In 3DFTL, the cache also serves as scratchpad for compression; each entry has sufficient information to generate compressed metadata that will be written to spare area. The pseudocodes of compression and decompression are shown in Fig. 3. With compression, the metadata are stored as an index dictionary and compressed PPNs — pairs of index position and offset. In addition, the LPN is also embedded because it is mandatory for garbage collection. As the size of index dictionary is limited, the metadata can be stored in uncompressed format in case of very low compressibility to ensure mapping integrity. However, the entries of the third level PMT stored in the uncompressed metadata are limited to only those associated with the same second level PMT entry as the data area. In contrast, every second level PMT entries that related to the same entry of the first level PMT is kept.

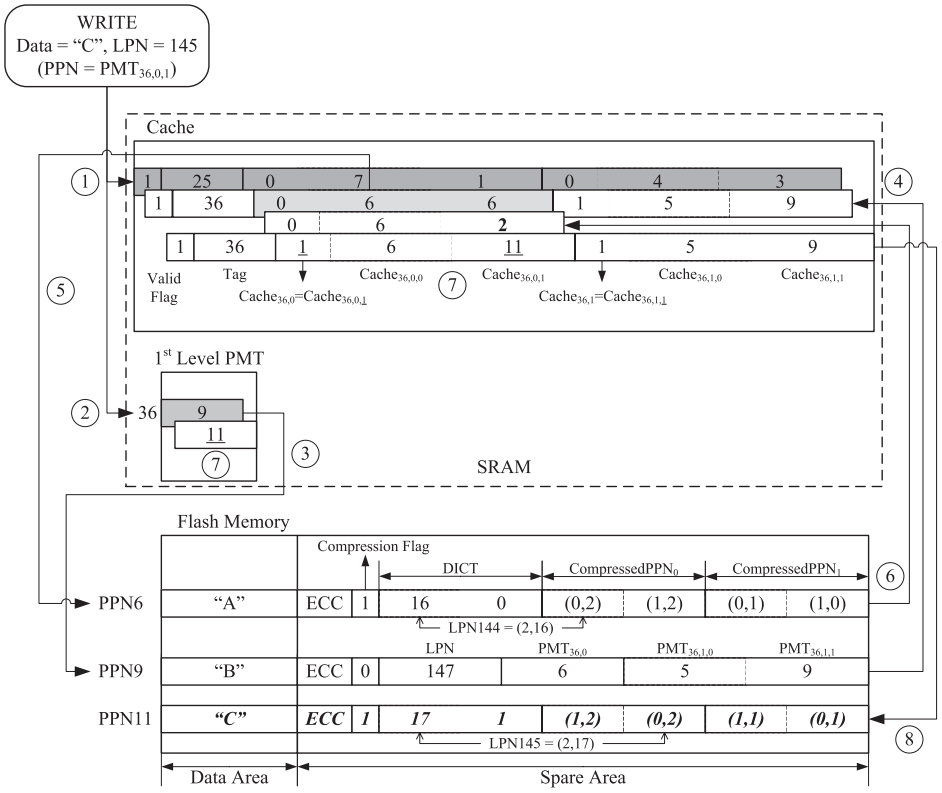

Fig. 4. The example of 3DFTL address translation.

In the example of write request illustrated in Fig. 4, we assume that, firstly, both LPN and PPN are 8-bit. Secondly, an LPN is broken into 6-bit, 1-bit, and 1-bit indices of PMT — the 3D array. Thirdly, an LPN is decomposed into two bits (MSBs) and six bits (LSBs) in compression procedure. On the other hand, a PPN is decomposed into a 6-bit index (MSBs) and a 2-bit offset (LSBs). Lastly, the maximum size of index dictionary is two. Every spare area has a compression flag for indicating the format of its metadata. In this example, *PPN*9 is uncompressed while *PPN*6 and *PPN*11 are compressed. Each cache entry resembles an extracted

© IEICE 2015
DOI: 10.1587/elex.12.20150211
Received February 23, 2015
Accepted March 11, 2015
Publicized March 30, 2015
Copyedited April 25, 2015

compressed metadata. Since an entry of the second level PMT is always equal to one of its third level PMT entries, the cache keeps only a small selector for each.

A request for writing *C* at *LPN*145, which means its PPN is held in $PMT_{36,0,1}$, begins with fetching the previous PPN into the cache by three-level address translation in step 1–6. A victim is selected by LRU policy and can be evicted immediately because of write-through policy. During these steps, the PPNs of adjacent LPNs, which will be needed for compression, are also cached. As shown in the example, the previous PPN of *LPN*145 is *PPN*2 and two flash memory read operations were performed.

In step 7, *PPN*11, a new empty page, is assigned for storing the updated data and metadata; hence, the first level PMT and the cache entry, which contains the second level PMT and the third level PMT are updated according to this assignment. After that, 3DFTL tries to compress the cache entry and creates the metadata. In this example, the cache entry can be compressed; thus, the cache entry is compressed by the procedure described in Fig. 3. Before *PPN*11 is replaced by *LPN*145, the compressed dictionary (*DICT*) was $(2, 1)$ and the compressed PPNs were $(1, 2)$, $(0, 3)$, $(1, 1)$, and $(0, 1)$, respectively. In order to be embedded into compressed metadata, *LPN*145 is decomposed into 2 and 17. As a result, the dictionary become $(17, 1)$ and $CompressedPPN_{0,1}$ is changed from $(0, 3)$ to $(0, 2)$. Finally, the created metadata are programed to *PPN*11 along with the data in step 8.

3 Evaluation

The experiments were conducted by simulating an 8 GB MLC NAND flash memory [3] with following parameters. It has 4096 blocks of 256 pages. The data area of a page is 8192B while the spare area is 448B. However, only 112B are usable because of ECC. A page read, a page program, and a block erase operations take 75 µs, 1300 µs, and 3800 µs, respectively. The data transfer rate is limited to 50 MB/s. Benchmarks from SPC [4] and MSRC [5] were used for performance evaluation.

3DFTL will be compared against DFTL [2], CDFTL [6], and SCFTL [7]. DFTL is the baseline of demand-based FTLs while CDFTL added the second-level cache in order to exploit spatial locality. SCFTL is a high performance FTL that optimized for spatial locality and large page size. The SRAM sizes of 3DFTL, DFTL, CDFTL, and SCFTL were configured to 96.73 KB, 101.00 KB, 99.31 KB, and 101.25 KB, respectively. The first level PMT of 3DFTL takes 64 KB because few entries can be packed into a spare area; hence, only 32 KB are left for the cache. On the contrary, the cache of other FTLs is about 96 KB.

As shown in Fig. 5, the average system response time of 3DFTL is the best comparing with other techniques even though its cache size is about one third of the others. The geometric mean of normalized average system response time of 3DFTL is 16.42%, 30.59%, and 2.45% faster than DFTL, CDFTL, and SCFTL, respectively. According to Fig. 6, besides the low cache miss rate, which is caused by the spatial locality exploitation, the low cache miss penalty is also a major contributor for enhancing the performance.

© IEICE 2015
DOI: 10.1587/elex.12.20150211
Received February 23, 2015
Accepted March 11, 2015
Publicized March 30, 2015
Copyedited April 25, 2015

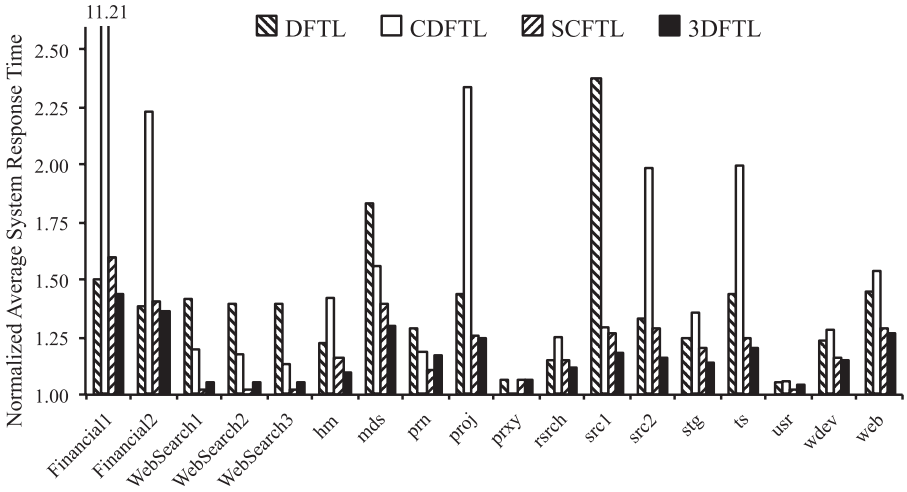

Fig. 5. The normalized average system response times.

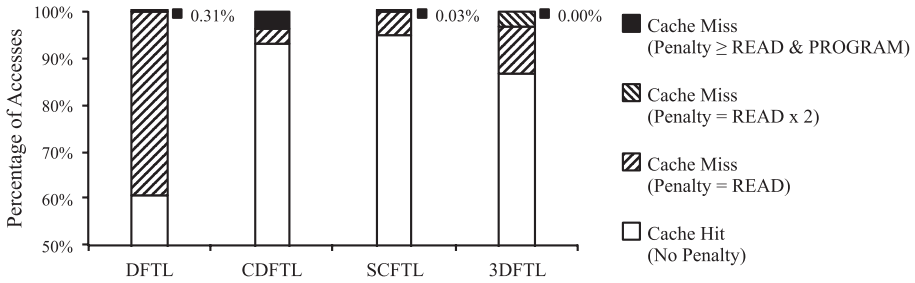

Fig. 6. The percentage of address translation cost.

The maximum cache miss penalties of the other FTLs include one or more program operations due to the update of translation pages. Since a page programing is over ten times slower than reading, having only one program operation makes the cache miss penalty considerably high. Furthermore, a page programing may trigger a garbage collection that requires even longer time. As denoted by solid black labels in Fig. 6, DFTL, CDFTL, and SCFTL have the average of 0.31%, 3.80%, and 0.03% cache miss with the penalty of one or more read and program operations, respectively.

On the contrary, the cache miss penalty of 3DFTL does not contain any program operations. The cache miss penalty of 3DFTL is only two read operations in the worst case. Moreover, the worst case rarely occurs owing to the compression. The average cache miss with the penalty of two read operations is only 3.08% as shown in Fig. 6. Therefore, the average cache miss penalty of 3DFTL is lower than other FTLs.

An impact of cache miss penalty is clearly shown in *Financial1* benchmark that contains write-intensive requests but low spatial locality. CDFTL, which has few large cache lines, exhibits very high overall cache miss penalty since 82.56% of its cache misses needs to update translation pages. As a result, CDFTL is drastically slow even though its cache miss ratio is very low. Moreover, DFTL and SCFTL are also subject to high miss penalty during very stressing cache accesses. However, our proposed FTL, 3DFTL, maintains low cache miss penalty. Regardless of

© IEICE 2015
DOI: 10.1587/elex.12.20150211
Received February 23, 2015
Accepted March 11, 2015
Publicized March 30, 2015
Copyedited April 25, 2015

smaller cache size, 3DFTL outperforms other FTLs and even surpasses, the high performance, SCFTL.

As previously stated, 3DFTL not only solves the inconsistency problem, but also enhances the performance. In addition, 3DFTL provides better flash space utilization since it does not occupy special pages for the mapping table. For this reason, 3DFTL shows slight improvement in terms of P/E cycles, which also means prolonging flash memory lifetime.

4 Conclusion

In this paper, a novel demand-based FTL named 3DFTL is proposed. It does address translation at the page-level and employs a cache of the mapping table like other demand-based FTLs. Differently, 3DFTL gets rid of translation pages by utilizing the spare areas of flash memory pages. Since the mapping information and data are simultaneously stored, the inconsistency problem is creased to exist; hence, fault tolerance is improved. However, keeping the locations of the page-level mapping table that stored in many little spare areas demands large SRAM. Thus, the three-level address translation is required for controlling SRAM size. The compression and caching techniques have been applied in order to exploit the spatial locality. The average cache miss penalty is very low owing to zero explicit cache write-back operations. To sum up, 3DFTL is an economical inconsistency-free high-performance demand-based FTL. 3DFTL is more suitable for managing the flash memory in a high performance mobile device than other demand-based FTLs.

Acknowledgments

Peera Thontirawong is financially supported by the Thailand Research Fund through the Royal Golden Jubilee Ph.D. Program (PHD/0273/2549).

© IEICE 2015
DOI: 10.1587/elex.12.20150211
Received February 23, 2015
Accepted March 11, 2015
Publicized March 30, 2015
Copyedited April 25, 2015