FRASH: Hierarchical File System for FRAM and Flash

Eun-ki Kim[1,2] Hyungjong Shin[1,2] Byung-gil Jeon[1,2]
Seokhee Han[1] Jaemin Jung[1] Youjip Won[1]

[1]Dept. of Electronics and Computer Engineering, Hanyang University, Seoul, Korea
[2]Samsung Electronics Co., Seoul, Korea

zerobit@ece.hanyang.ac.kr

Abstract. In this work, we develop novel file system, FRASH, for byte-addressable NVRAM (FRAM[1]) and NAND Flash device. Byte addressable NVRAM and NAND Flash is typified by the DRAM-like fast access latency and high storage density, respectively. Hierarchical storage architecture which consists of byte-addressable NVRAM and NAND Flash device can bring synergy and can greatly enhance the efficiency of file system in various aspects. Unfortunately, current state of art file system for Flash device is not designed for byte-addressable NVRAM with DRAM-like access latency. FRASH file system (File System for FRAM an NAND Flash) aims at exploiting physical characteristics of FRAM and NAND Flash device. It effectively resolves long mount time issue which has long been problem in legacy LFS style NAND Flash file system. In FRASH file system, NVRAM is mapped into memory address space and contains file system metadata and file metadata information. Consistency between metadata in NVRAM and data in NAND Flash is maintained via transaction. In hardware aspect, we successfully developed hierarchical storage architecture. We used 8 MByte FRAM which is the largest chip allowed by current state of art technology. We compare the performance of FRASH with legacy lts-style file system for NAND Flash. FRASH file system achieves x5 improvement in file system mount latency.

Keywords: FRAM, NVRAM, NAND Flash Memory, File System, Hierarchical Storage, Mounting Time

1 Introduction

Due to recent rapid advancement of non-volatile memory technology, users can now bring large amount of data in very portable fashion and variety of high performance mobile devices come to exist. They include cell phone, MP-3 player, portable game player, digital camera and PDA. This convenience is particularly indebted from the evolution of NAND Flash technology[2]. Thanks to steadfast effort from academia as well as industry, storage density of NAND flash device has increased faster than Moore's Law[3]. In addition to storage density, NAND flash technology effectively resolves a number of issues which legacy hard disk technology has not been able to properly address. They include shock-resistance, energy consumption[4]. Flash memory has entirely different media characteristics than hard disk drive. Prime difference comes from the fact that Flash memory content cannot be overwritten directly and that block of storage needs to be erased prior to update. Erase operation takes significant amount of time and the unit of erase is much larger than single disk page. Further, each location of the Flash device has limited number of erase cycle. It is important that each cell in Flash device is used (erased) in uniform fashion. Due to these differences, it is not possible to use existing hard disk based file system to handle Flash media. There are major two approaches in storage software for Flash media. The first one is to use log-structured file system (LFS)[5]-like approach where file system writes to new location for every write operation. The second one is to introduce new device driver layer which dynamically maps the device block address to the new location in every write operation. This device driver layer is often called Flash Translation Layer (FTL)[6]. FTL emulates the NAND flash storage device as a block device and provides disk-device-like read/write operation by hiding erase operation. With FTL, we can use the conventional file system for the NAND flash storage device. LFS-like approach exhibits better I/O performance. However, operating system needs to scan entire file system partition to build the in-memory metadata when it mounts the file system. Density of NAND flash device increases very rapidly and scan overhead has already become significant issue in state of art NAND flash device, e.g. 4 GByte.

Aside from Flash memory technology, academia and industry put lots of effort on developing byte addressable non-volatile memory technology, e.g. FRAM, PRAM, MRAM, and etc. These devices are byte-addressable, do not require erase operation in performing write, and have similar access speed as SDRAM. Despite the promising physical characteristics, however, these technologies are at their inception stage and current technology allows for only small capacity. Due to its small capacity, these NVRAM's has very limited usage and cannot be used by itself.

In this work, we develop file system for hierarchical non-volatile storage system. Our work consists of two themes. We first designed and implemented a hierarchical storage system. Our storage subsystem consists of FRAM (Ferro-electric RAM) and NAND Flash. Second, we develop hierarchical file system, FRASH which exploits the physical characteristics of storage medium at each storage hierarchy. The objective of this work is to resolve the overhead of file system mount operation and meta-data update while retaining highest possible I/O performance in NAND Flash memory.

2 Related Work

LFS style flash file systems suffer from important problems. It requires large amount of memory for mapping table. Further, file system mount latency is very large. As the capacity of NAND flash memory increases, overhead of file system mount becomes more significant in Flash file system. This is particularly of an issue in Flash file system since the most of NAND flash storage is for mobile device where quick system response is crucial. Yim et. al. introduced snapshot technique to reduce mount time[7]. The file system metadata in memory (snapshot) is stored at flash memory in file system unmount phase. Instead of scanning entire file system partition, they use snapshot to mount the file system. In this technique, it takes more time to unmount the file system. RFFS[8] divides flash memory into two regions: location information area and data area. This technique reduces mount time by constructing RAM data structure using only location information area. Location information area contains the most recent location information. Even though the location information area reduces area to scan, the mount time is still proportional to flash memory size. MNFS[9] improved the file system mount time and memory foot print. They use block mapping algorithm and page mapping algorithm for data area and meta-data area.

Recently, a number of works suggested to use byte-addressable non-volatile memory or persistent RAM as a part of storage subsystem. HeRMES[10] propose to use non-volatile memory as a part of storage subsystem to maintain file meta-data information. MRAMFS[11] is an improvement on HeRMES which stores compressed file meta-data in non-volatile RAM. Conquest[12] file system proposed to use file system metadata and small files in persistent-RAM layer. These hierarchical file systems are fundamentally for disk based file system and try to improve the access time while read/write operation in disk-based file systems. They store the metadata in NVRAM, while the conventional file systems do in specific disk area.

The ideas adapting NVRAM or persistent-RAM as write buffer in its file system had been proposed to overcome low write performance[13, 14]. When the file system performs write operation, they buffer the write data to the NVRAM or persistent-RAM first and write to disk or flash memory later. Additionally, even with unexpected power failure, the write operation can be performed completely at next power-on without data loss.

Our work distinguishes itself from prior works in a number of aspects. First, we developed hierarchical storage system which consists of NAND flash and FRAM. The above mentioned hierarchical file system is for disk based storage and NVRAM, few of which are based upon physical device. Disk based file system and Flash file system has entirely different meta-data structure and meta-data management algorithm. FRASH is a hierarchical file system which is optimized to handle meta-data operation of YAFFS in NVRAM layer of the storage.

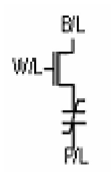
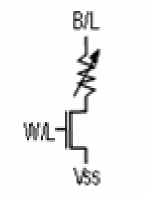
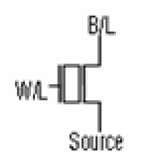
The rest of this paper is organized as follows. Section 3 describes modern NVRAM technologies. In section 4, we present the brief introduction to YAFFS file system. Section 5 explains the details of FRASH. Section 6 and section 7 carries implementation details and the results of our performance experiments, respectively. Section 8 concludes the paper.

3 Byte Addressable NVRAM and Storage Organization

3.1 Non-Volatile Memory Technologies

We describe the physical characteristics (refer to Table 1) of FRAM (Ferro-electric RAM), PRAM (Phase-change RAM), NOR flash, and NAND flash. FRAM, PRAM and NOR flash are byte addressable. Particularly, NOR flash is byte addressable on read operation, but NAND flash does not support byte addressable operation. It is accessed only in page (512byte) granularity. Till today, PRAM is not commercially available and very small size FRAM is available in the market (128KByte). Flash memory technology has matured further compared to these. NOR flash is widely used as a code or boot memory and NAND flash is used as storage device. The unit cell structure of NOR flash is same as that of NAND flash (Fig. 1) Cell array of NOR flash consists of parallel connection of several unit cells. NOR flash can perform byte addressable operation and has faster read/write speed than NAND flash. However, because of the byte addressable cell array structure, NOR flash has slower erase speed and lower capacity than NAND flash.

Table 1. Comparison of byte addressable NV-RAM and NAND Flash

Item	FRAM	PRAM	NOR	NAND
Byte Addressable	YES	YES	YES (Read only)	NO
Non-volatile	YES	YES	YES	YES
Read	85ns	62ns	85ns	16us
Write/Erase	85ns/none	300ns/none	6.5us/700ms	200us/2ms
Power consumption	Low	High	High	High
Capacity	Low	Middle	Middle	High
Endurance	1E15	>1E7	100K	100K
Unit Cell				See Figure 1

PRAM consists of one transistor and one variable resistor. The variable resistor is integrated by GST material and acts as a storage element. The GST material has different resistance value with respect to its crystallization status; it can be converted to crystalline (low resistance) or to amorphous (high resistance) structure by forcing current though B/L to Vss. This mechanism is adapted to PRAM for write method. Due to this reason, the write operation of PRAM spends more time and current than read operation. This is the essential drawback of PRAM device. The read operation can be performed by sensing the current difference through B/L to Vss. Even though the write is much slower than read operation, PRAM does not need erase operation and it is being expected that the storage density is soon able to compete with that of NOR flash. PRAM is considered as future replacement of NOR flash memory.

Contrary to PRAM, FRAM has good access characteristics. Read and write speed is almost identical and is very fast. We will have in depth look at FRAM and NAND flash memory technology in next section.

NAND Flash Memory. NAND flash memory has different properties compared to other memories. Read and write can be done only in page granularity (512Byte usually). Erase operation is performed in much larger granularity. Unit of erase in NAND flash is often called "block" and block consists of 32 (or 64) pages.

NAND flash device is susceptible to defect and it requires requiring error correction code (ECC). Also, the number of erase is limited. After a certain number of erase, the respective location becomes unusable. Despite these physical characteristics some of which is definitely significant drawbacks, Although the capacity of NAND overwhelms the other NVRAM technologies. NAND flash has higher cost per byte than hard disk drive. Nevertheless, the RAM nature which does not have mechanical component, i.e. light weight, shock resistance, low power consumption, and small size make it possible for NAND flash to take great potential in multitudes of portable information appliances. Fig. 1 shows a block structure of NAND flash memory. A cell-string of NAND flash memory generally consists of serial connection of several unit cells to reduce cell area. The unit cell is composed of only one transistor having floating gate. When the transistor is turned on or off, the data status of the cell is defined as “1” or “0” respectively. The page, which is generally composed of 512-byte data and 16-byte spare cells, is organized lots of unit cells in a row. It is unit for the read/write operation. The block, which is composed of 32 pages (16Kbyte), is base unit for the erase operation. Erase operation requires high voltage and longer latency. It sets all the cells of the block to data “1”. Write operation is performed in a page unit. The unit cell is just changed from “1” to “0” when the write data is “0”, but there is no change when the write data is “1”. Read operation is also performed in a page unit.

The important drawback of NAND flash memory is the limitation of the number of erase operation (known as endurance; typically 100K cycles). This drawback is rooted at the fundamental property of floating gate. It is important that all NAND flash cells go through similar number of erase cycles to maximize its life time.

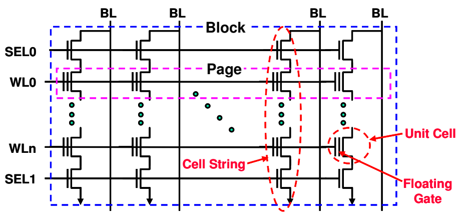


Fig. 1 A Block Structure of NAND Flash

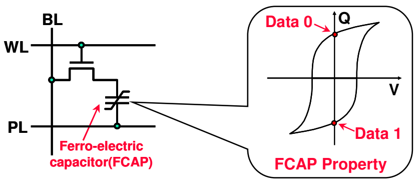


Fig. 2 A Cell Schematic of FRAM

FRAM. FRAM (Ferro-electric RAM) has ideal characteristics such as low power consumption, fast read/write speed, random access, radiation hardness, and non-volatility. Among MRAM, PRAM, and FRAM, FRAM is the most matured technology and small density device is already available in the market.

Contrary to NAND flash memory, FRAM can be written without erase operation. More importantly, it exhibits same access latency as current SRAM or DRAM technology. We envision that FRAM can greatly enhance the performance of the existing storage system if properly exploited. Fig. 2 illustrates a cell schematic of FRAM and a charge property of ferro-electric capacitor (FCAP) with respect to voltage. The unit cell of FRAM consists of one transistor and one ferro-electric capacitor; known as 1T1C, which has the same schematic as DRAM. Since the charge of FACP retains its original polarity without power, FRAM can maintain its stored data in the absence of power. Different from DRAM, FRAM does not need refresh operation and subsequently consumes less power. A write operation can be performed by forcing pulse to the FCAP through PL or BL for data “0” or data “1”, respectively. Since voltage of PL and BL for write operation is same as VCC, FRAM does not need additional high voltage like NAND flash memory. This property enables FRAM to perform write operation in much faster and simple way.

FRAM design can be very versatile. It can be designed compatible to SRAM interface as well as DRAM interface. Asynchronous, synchronous, or DDR FRAM can be designed. FRAM can fundamentally change the legacy architecture of the computer system. As it currently stands, DRAM, SRAM and Flash memory is used for main memory, cache memory and storage, respectively. Each of these materials needs to have its own interface and the respective software stack. FRAM technology can un-necessitate these diversities of components and can make the system architecture much simpler and compact. However, as it currently stands, memory density of FRAM is insufficient to address the above mentioned approach. The largest FRAM is 8 MByte under current state of art technology.

4 Synopsis: LFS-style file system for NAND Flash

4.1 Introduction of YAFFS

There are JFFS and JFFS2[15] as Linux file systems for NOR flash chip. The NOR flash chip has low density and slow write performance and is expensive. So, in that situation, JFFS is doing well. But NAND flash chip is cheap and has high density. Therefore, as NAND flash capacity increase continuously, JFFS cannot help having limitation to support NAND flash chip in RAM usage and boot time. Also, JFFS for NAND flash has various mechanisms that are not required for NAND. Because NOR and NAND flash have very different properties, as you see Table 1, a file system for NAND flash needs extra mechanisms not required for NOR flash such as another garbage collection strategy, management bad blocks and so on.

As a result, the company named Aleph One decided to create YAFFS that is designed specifically for use with NAND flash (Dec. 2001). And then the YAFFS for Linux was working on real NAND flash chip (May 2002), the YAFFS for WinCE was created (Aug. 2002), for uClinux (Sept. 2002), for pSOS (Feb. 2003) and so on. At last, in the early 2003, commercial YAFFS product was shipped.

The intention of the YAFFS is to be NAND flash friendly, Robustness through journaling strategies and significantly to reduce the RAM overheads and boot times associated with JFFS. Also, now the YAFFS is designed to be portable and has been used on Linux, WinCE, pSOS, eCOS, ThreadX and various special-purpose OS’s and even in situations where there is no OS.

YAFFS1 is the first version of this file system to accommodate the small block NAND chips of which page is composed of 512-byte data and 16-byte spare area and generally allow 2 or 3 write cycles per page.

YAFFS2 is the second version of YAFFS to accommodate large block NAND chips of which page is composed of 2048-byte data and 64-byte spare area, where its code is based on YAFFS1 and it supports YAFFS1 data formats. YAFFS is licensed both under the GPL and under per-product licenses available from Aleph One.

4.2 YAFFS vs. Flash Translation Layer

Flash Translation Layer (FTL) is a middleware to hide the erase operation of flash memory and resides between a file system and a flash memory. FTL can hide the erase operation on write operation by translating a logical address from file system to a physical address of an area to have been already erased on flash memory. FTL hides the slow erase operation and handles block I/O as an atomic operation like a hard disk. We can implement FTL as a type of host-independent hardware (Fig. 3) or host-device-driver.

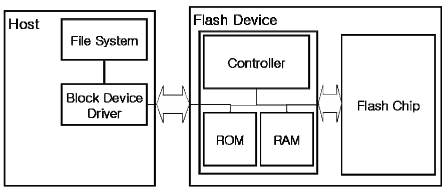


Fig. 3 FTL Construction

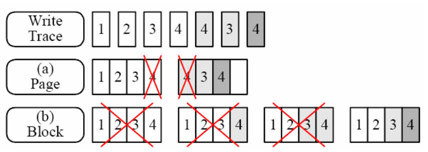


Fig. 4 FTL Operation Cases

FTL uses a page mapping or a block mapping depending on translation unit type. Because the page mapping translates in a unit of page, its performance is good but the large size of a mapping table costs much more. On the contrary, the block mapping translates in a unit of block, so the size of a mapping table is small but even to modify only one page takes additional cost that we have to erase the total block of the page and allocate new block. You can see the operations in Fig. 4. As using a mapping table, FTL can have good write performance against flash memory and be controlled by conventional normal file system. So FTL is used widely in main storage devices.

5 FRASH: Hierarchical File system for FRAM and Flash

In this work, we develop a file system which exploits the storage capacity of NAND flash and fast access latency and non-volatility of FRAM. The objective of this work is to resolve the file system mount latency issue and the overhead of meta-data update while retaining the performance advantage of the log structure based file system for NAND flash. We use YAFFS as a baseline file system for this purpose. Our file system consists of two layers: metadata layer and data layer. Metadata layer stores Tag and Object Header information. This information is used to mount the file system. Metadata layer and data layer resides at FRAM and NAND Flash, respectively. In our storage architecture, FRAM is mapped into memory address space. In legacy LFS style NAND flash file system, operating system scans the file system partition to build in-memory mapping table. In our storage architecture, metadata information is accessed directly from FRAM without copying the information into DRAM.

5.1 Structure of FRASH

In FRASH, metadata layer contains Tag information and Object Header. Each Flash page is assigned a file id and chunk number. These together are called Tag. File id isolates each file and chunk number represents the order of file data in data layer. Object Header corresponds to the inode in Unix File System. It has all information of file or directory; file name, size, ownership, and so on. Fig. 5 illustrates the organization of metadata layer in FRASH. Under current design, FRASH allocates an area of consecutive memory for each Tag, Object Header and Object Header Pointer. Tag in metadata layer is used to build making TNode tree for ordering the file data. Each Tag is accompanied by Object Header pointer. Object Header Pointer contains the address of the respective Object Header. If the Tag is not associated with actual file information (ChunkID is “0” in Fig. 5), it contains NULL value and doesn’t have any Object Header. If not, Object Header Pointer points the actual Object Header.

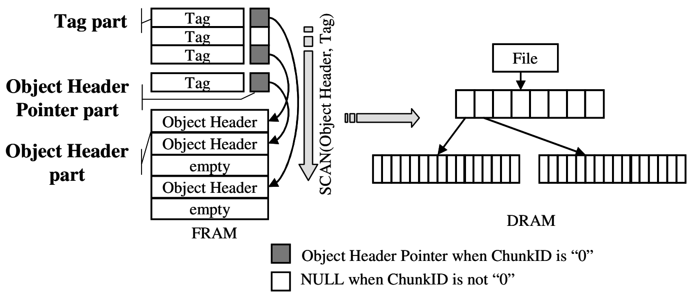


Fig. 5 Mount sequence of advanced YAFFS

5.2 Scan operation

In FRASH, operating system scans Tag and Object Header in metadata layer when mounting the file system (Fig. 5) and reads them into main memory (DRAM). We are dealing with byte-addressable NVRAM which is part of memory address space. Due to this reason, we explicitly specify the chip type if there is any possibility of confusion.

Tag information has two functions; indicating Object Header and making TNode tree. Operating system parses all Tag information. First, it decides validity of a corresponding NAND page in data layer. If this tag is available, it decides to associate with Object Header or have information for making TNode tree. If it associates with Object Header, it looks into Object Header Pointer and finds the actual Object Header. Operating System registers this Object Header into main memory. On the contrary, if it has information for making TNode tree, Tag has the available chunk number (chunk

ID in Fig. 5). This number constructs TNode tree and represents the order of file data in data layer.

5.3 Management of Object Header

File creation, deletion, and modification require an update on Object Header. When creating a file, Tags and an Object Header are created first. Each data page in data layer has matching Tag entry in metadata layer. Tag entries in metadata region are maintained as an array and they can be accessed using page index. On the other hand, Object Header is allocated dynamically. When FRASH need new Object Header, it searches for unused Object Header slot in metadata layer. When it finds the empty slot, it sets the Object Header Pointer of the Tag to point to the empty slot and initializes the Object Header. After Tags and Object Header are initialized, the file data is stored in data layer. Recall that data layer in FRASH is in NAND flash device. File deletion is exactly the reverse of these steps. Operating system sets that the associated Object Header is empty and the Object Header Pointer is NULL. It also changes statue of Tag invalid.

The overhead of updating these metadata is insignificant. This is because the metadata reside in FRAM and we can perform in-place update in FRAM. File modification is more complicated than file creation and deletion. File modification is actually a combination of the two. Old Tag and Object Header become invalid and new Tag and Object Header are allocated.

6 Implementation details

6.1 Memory Map

FRASH file system uses YAFFS as its baseline file system. It uses YAFFS to manage data area information which resides at NAND flash device. Currently, FRASH is developed on Linux 2.4. FRAM device is installed on Bank 1 of our reference board. FRAM device is mapped into memory address space. FRAM and DRAM devices form homogeneous memory address space. In our implementation, a certain section of memory address space is for FRAM device.

For the implementation of FRASH design, we fixed scanning function, MTD reading, writing and erasing function. The scanning function in YAFFS looks through all spare areas in NAND device and Object Header. We modified the code to scan FRAM first under the condition of existing FRASH's magic code which shows that FRASH is available. And then, we modified the code so that the first mount operation builds the FRASH information on FRAM device, if the magic code shows there is no FRASH information. So, we can prepare for this research to check the mount time of FRASH from the second mount operation. By doing that, we can save time to make file system utilities for FRASH, which are out of the purpose of this research. In addition, we changed the MTD writing, and erasing functions. For synchronization of

NAND device with FRAM device, That is, we modified the code for FRASH in order to hook those operations and update data for Object Headers and Tags on NAND device as well as on FRAM device for each operation. So, we can check their performance of the FRASH for its file system operations such as reading, writing and erasing operations repeatedly and continuously, even though that additional code can affect the performance test of the FRASH

6.2 Implementation of Hierarchical Storage with FRAM and NAND Flash

We design and implement hierarchical storage subsystem. It consists of FRAM (8 MByte)[16] and NAND flash (128MByte)[17]. 8 MByte FRAM chip is the largest one which current state of art technology allows. This hierarchical storage is attached to SMDK2440 emulation board[18]. It consists of ARM 920T core and several peripherals: memory controller, NAND flash controller, LCD controller, MMC/SD card controller, USB host and device, 10bit ADC, Camera interface, and etc. SMDK2440 has 1MB NOR flash for boot ROM, 64MB SDRAM, QVGA TFT-LCD and keyboard. FRAM has same access latency as SRAM: 110ns asynchronous read/write cycle time, 4Mb x 16 I/O, and 1.8V operating power. Since the package type is 69FBGA (Fine pitch Ball Grid Array), we make an artwork for PCB to attach FRAM to memory extension pin of SMDK2440 board. The board supports 8 banks (bank0 to bank 7). Bank0 is reserved for boot memory, and bank6 and bank7 are reserved for SDRAM. They are directly managed by kernel in memory space. FRAM can be directly attached one of 5 banks without additional memory controller. We choose bank1 (0x08000000). We also set the board environment suitable for our experiment.

The core clock is 400MHz, memory bus clock is 100MHz, and peripheral bus clock is 50MHz. FRAM access cycle time is adjust at 5.6MHz (180ns) for stable operation. Fig. 6 illustrates the picture of SMDK2440 board with FRAM and NAND Flash.

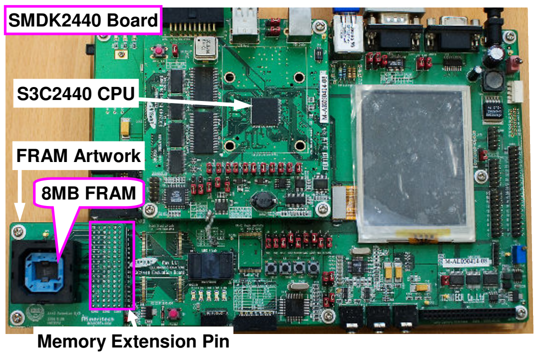


Fig. 6 SMDK2440 board with FRAM

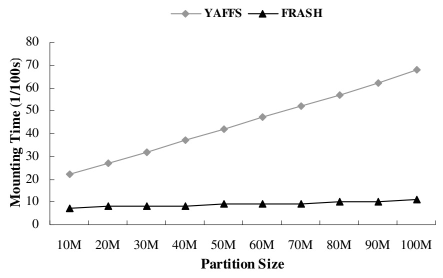
7 Experiments

7.1 Experiment Setup

The objective of this study is to boost up the performance of mount latency. We compare the performance of FRASH file system with YAFFS. FRASH file system is currently implemented on Linux 2.4 and SMDK 2440 reference board. We examine various aspect of file system performance: Mount latency, Create/Delete operation, and Read/Write operation. For mount latency, we use time utility during mounting operation. For Create/Delete operation, we use lat_fs in LMBENCH[19]. In Read/Write operation, we use IOZONE benchmark[20] and lmdd in LMBENCH.

7.2 Mount Latency

We examine the mount overhead in FRASH file system and YAFFS. Overhead of file system mount is serious problem in LFS style NAND flash file system. This is because it needs to scan NAND flash device to build mapping table and scan overhead increases with the size of the device. First, we examine the file system mount latency under different file system partition size ranging from 10MByte to 100MByte. There exist only root file system and total content size is approximately 2MByte. Fig. 7 illustrates the result of the experiment. The average mount time of YAFFS is 8.11ms/MByte, and that of FRASH is 1.62ms/MB. YAFFS read entire Tag and Object Header information from NAND device. On the other hand, FRASH file system does not scan NAND Flash device and build the mapping table directly from the Tag and Object Header information in FRAM.



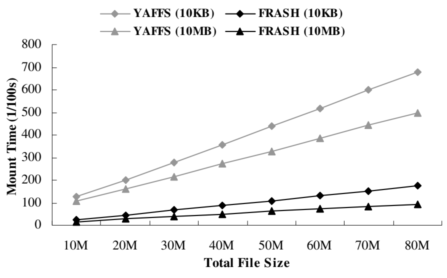


Fig. 7 Mounting Time with diff. partition size **Fig. 8** Mounting time with diff. total file size

Mount latency is also subject to the number of files in the file system. Second, we measured mounting time with different total file size in the same size partition for estimating file size dependency between FRASH and YAFFS. We prepared 4 same-size-portioned NAND storages. Two of the them filled with 10KB-size files: one is for testing FRASH (labeled by FRASH(10KB) in Fig. 8), and the other is for YAFFS (labeled by YAFFS(10KB)). Another two of them filled with 10MB-size files: one is for FRASH (FRASH(10MB)), and the other is for YAFFS (YAFFS(10MB)). The experimentation was performed by changing the partition size from 10MB to 80MB. Fig. 8 shows the experiment results. The overall mounting performance of FRASH greatly enhanced compared to YAFFS same as the results of first experiment. In this experiment, we proofed that FRASH had smaller mount time variation corresponding to the total file size than YAFFS. Therefore, FRASH would support outstanding performance in the application with large capacity NAND flash.

7.3 Create and Delete Operation

Performance of metadata update is an important metric for file system efficiency. Metadata update operation means the operations which updates the Tag and Object Header Information. In lieu of this, we examine the performance of create and delete operation. We use LMBENCH. We measure the number of actions per second to create 0 byte, 1Kbyte, 4Kbyte, and 10Kbyte size files and to delete them. In Fig. 9, we find that FRASH file system exhibits slightly lower performance than YAFFS; 5% ~ 10% and 3% ~ 6% less performance in file creation and deletion, respectively. This performance penalty is caused by synchronization overhead of FRAM with NAND flash device. One possible resort to this is to perform synchronization between FRAM and NAND flash when unmounting the file system. However, this approach makes the file system vulnerable to power failure.

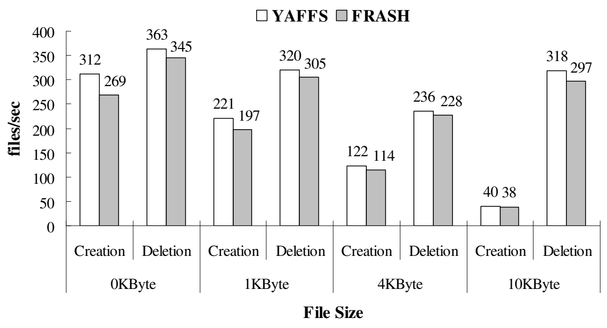


Fig. 9 Results of lat_fs benchmark

7.4 Read and Write Operation

We examine read/write performance of FRASH and YAFFS. We use IOZone and LMBENCH File System Benchmark suite. Fig. 10 and Fig. 11 illustrate the results. (The IOZone and LMBENCH show a kind of numerical values of performance. The unit of the values is bytes/sec. That is, we can think that the higher the value is , the better its performance is.):

IOZone benchmark measures many kinds of file system operations; Read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read, pread ,mmap, aio_read, and aio_write. It examines performance through making a temporary file and reading or writing increasingly the predefined unit data into that file. In our measures, we only perform read and write benchmark. In Fig. 10, FRASH and YAFFS exhibit similar performance in READ and WRITE operation.

We also use LMBENCH to measure the read/write performance. It tests the working time of dd utility (disk duplicate). For writing test, it writes a file filled with "0" (/dev/zero) to FRASH and measures the working time. For reading test, it reads the file which has been already made by writing test and throws it to /dev/none part of FRASH and then measures the working time. Fig. 11 shows the results of this test. The results of reading test shows 0.2% gain and that of writing test does 1.1% loss. The gain of reading is under measure error and the loss of writing is also caused by synchronization overhead.

As upper two read/write test is shown, the synchronization of FRAM with NAND has influence on the performance of file system little. It does not cause much loss for over all system.

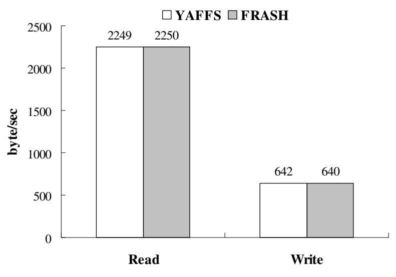


Fig. 10 Results of IOZone Benchmark

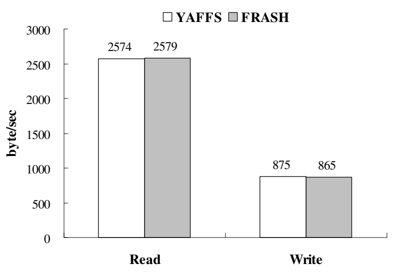


Fig. 11 Results of lmdd Benchmark

8 Conclusion

Recent rapid advancement in NAND flash technology makes more portable system come to existence. The speed of density increase in NAND file system exceeds the Moore's Law. Aside from NAND flash, byte addressable NVRAM, e.g., FRAM and PRAM, is another important axis of development which can potentially change the computer architecture paradigm. However, under the current state of art technology, storage density of byte-addressable NV-RAM is far from being satisfied to replace existing memory (or storage) device.

In this work, we develop hierarchical file system which exploits high storage density of NAND Flash and SRAM-like access characteristics and non-volatility of FRAM. LFS style file system for NAND flash exhibits very good performance in read and write. However, it suffers from significant file system mount overhead. We focus our effort on relieving the overhead of file system mount using non-volatile storage. We develop hierarchical file system, FRASH (Hierarchical File system for FRAM and Flash). We partition the information in file system into two layers: metadata and data. Metadata information is stored in FRAM and Data information is stored in NAND Flash region. This hierarchical approach enables us to eliminate "scan" phase of flash device in file system mount. In memory mapping table is directly built from the information in FRAM. Via exploiting storage hierarchy, we can make the file system mount operation 5 times faster in FRASH file system than in legacy LFS style NAND flash file system. There still remains one issue which requires further investigation. FRASH has hierarchy. Guaranteeing consistency across the storage hierarchy entails overhead. The performance of metadata operation in FRASH file system is not as good as the one in legacy LFS-style file system.

References

1. Mun-Kyu Choi, B.-G.J., et al.: A 0.25-um 3.0V 1T1C 32-Mb Nonvolatile Ferroelectric RAM With Address Transition Detector and Current Forcing Latch Sense Amplifier Scheme. IEEE Journal of Solid-State Circuits 37 (2002)
2. Co., S.E.: Not just Leading, but Creating the Mobile Wave with NAND Technology. http://www.samsung.com/Products/Semiconductor/NANDFlash/index.htm
3. Moore, G.E.: Moore's Law. http://www.intel.com/technology/mooreslaw/index.htm (1965)
4. Samsung, E.: Flash Solid State Drive. http://www.samsung.com/Products/Semiconductor/FlashSSD/index.htm
5. M. Rosenblum, a.J.K.O.: The Design and Implementation of a Log-Structured File System. ACM Transactions on Computer Systems 10 26-51
6. Corporation, I.: Understanding the flash translation layer(FTL) specification. (1998)
7. Keun Soo Yim, J.K.a.K.K.: A fast start-up technique for flash memory based computing systems. ACM Symposium on Applied Computing (2005)
8. Song-hwa Park, T.-h.K., Joo-kyong Lee and Ki-dong Chnung: A Flash File System to Support Fast Mounting and Reliability in NAND Flash Memory. CSICC 2 (2006) 87~91
9. Hyojun Kim, Y.W.: MNFS: Mobile Multimedia File System for NAND Flash based Storage Device. In Proceedings of IEEE Consumer Communications and Networking Conference (2006)
10. Ethan L. Miller, S.A.B., Darrell D.E. Long: HeRMES: High-Performance Reliable FRAM-Enabled Storage. In Proceedings of the 8th IEEE Workshop on HotOS-VIII (2001)
11. Nathan K. Edel, D.T., Ethan L. Miller, Scott A. Brandt: MRAMFS: A compressing file system for non-volatile RAM. In Proceedings of the IEEE Computer Society's 12th Annual International Symposium on MASCOTS (2004)
12. An-I A. Wang, P.R., Gerald J. Popek, Geoffrey H. Kuenning: Conquest: Better Performance Through a Disk/Persistent-RAM Hybrid File System. In Proceedings of the 2002 USENIX Annual Technical Conference (2002)
13. Michael Wu, W.Z.: eNVy: A Non-Volatile, Main Memory Storage System. In Proceedings of 6th International Conference on ASPLOS (1994)
14. Yim, K.S.: A Novel Memory Hierarchy for Flash Memory Based Storage Systems. Journal of Semiconductor Technology and Science 5 (2005)
15. Woodhouse, D.: JFFS: The Journaling Flash File System. Ottawa Linux Symposium (2001)
16. Kang, Y.M.: World Smallest 0.34/spl mu/m~ COB Cell 1T1C 64Mb FRAM with New Sensing Architecture and Highly Reliable MOCVD PZT Integration Technology. Symposium on VLSI Technology Digest of Technical Papers (2006)
17. Samsung, E.: K9D1G08V0A: 128MB Smart Media™ Card. http://www.samsungsemi.com
18. Meritech: SMDK2440 http://www.meritech.co.kr/eng/
19. L McVoy, C.S.: lmbench: Portable Tools for Performance Analysis. USENIX Annual Technical Conference (1996)
20. Norcott, W.: IOZONE Filesystem Benchmark. http://www.iozone.org/ (2002)