Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 9, Number 1, pages 12–18, January 2020

OFF2F Program Execution Using Pseudo Non-Volatile Memory

Sho Takasugi, Masaya Sato, Hideo Taniguchi
Graduate School of Natural Science and Technology
Okayama University
Okayama, Japan
Email: takasugi@swlab.cs.okayama-u.ac.jp, {sato, tani}@cs.okayama-u.ac.jp

***Abstract*—The speed at which non-volatile memory can be accessed has become faster. To exploit the features of non-volatile memory, an executable file format, OFF2F: Object File Format consisting of 2 Files, was proposed for speeding up program execution. OFF2F focuses on the access modes when the program is executed in the memory. OFF2F is designed based on the assumption that the memory of a computer comprises of both volatile and non-volatile memory. However, current computers do not have a mix of volatile and non-volatile memory. Therefore, in this study, we propose a method to simulate non-volatile memory using volatile memory to construct a volatile and pseudo non-volatile mixed-memory environment. We also present a method to execute OFF2F programs using the pseudo non-volatile memory. The performance in terms of the speed of program execution of the OFF2F program using a pseudo non-volatile memory is compared with a conventional executable file format. The results show that OFF2F requires lesser number of reads from external storage device and shorter page fault processing time.**

***Index Terms*—non-volatile memory, executable file format, virtual memory system, operating system**

I. Introduction

Conventionally, the main memory of a computer is volatile, which means that data is lost at shutdown. On the other hand, non-volatile memory that can store data even after shutdown has also been developed. Qingsong et al. proposed a method called file system metadata accelerator (FSMAC) to hold metadata in non-volatile memory [1]. Automated tiered storage with fast memory and slow flash storage (ATSMF) has also been proposed for leveraging non-volatile memory to reduce its response time [2]. Shengan et al. proposed a tiered file system Ziggurat that is consisted of non-volatile memory and slow disks [3]. Jian et al. proposed Orion, a distributed file system using the features of byte unit accessible [4]. In addition, a study was conducted to improve the average latency of memory access by treating non-volatile memory as a cache hierarchy [5]. Zhang et al. proposed a method to optimize the consistency of data in CPU cache and non-volatile memory [6]. Eisenman et al. proposed reducing the capacity of volatile memory by using non-volatile memory for the database [7]. Furthermore, Xue et al. proposed extension of programming language using non-volatile memory [8]. Koshiba et al. proposed an emulator considering the access speed of non-volatile memory [9].

With future technological innovations, non-volatile memory is expected to have read speeds comparable to volatile memory; however, it is difficult for write speeds of non-volatile memory to be equal to those of volatile memory. Furthermore, non-volatile memory has a small capacity and is expensive compared to volatile memory. Therefore, the focus is not on the configuration in which main memory is totally equipped with non-volatile memory but on the processor environment of main memory configuration where volatile and non-volatile memories are mixed. An executable file format, OFF2F: Object File Format consisting of 2 Files, was proposed for two types of mixed memory environments [10]. In this format, the executable file is composed of two files to enable the virtual memory system to support program execution utilizing the features of both volatile and non-volatile memories. Here, the program text and the other regions are composed as separated files. Non-volatile memory has low read latency but high write latency. By using this feature, the program text region, which is read-only in OFF2F, is stored in the non-volatile memory and mapped to the virtual memory. Therefore, the time required for page fault processing can be shortened. There are about 100 executable programs whose size of data region size is 10% more of the file size in /bin and /usr/bin of FreeBSD 11.0-RELEASE. Therefore, the size of the equipped non-volatile memory can be reduced if the program text region is stored in the non-volatile memory.

However, the development of computers with a mixed-memory environment is limited and verification of the effects of OFF2F is extremely difficult.

To address this problem, we propose a method to implement a pseudo non-volatile memory using conventional computers, consisting only of volatile memory, as a supportive environment to verify the effect of OFF2F. Furthermore, we present an execution method of the OFF2F program using the pseudo non-volatile memory by modifying the program execution process in FreeBSD. We report the evaluation result of the execution performance of the OFF2F program using a pseudo non-volatile memory. We also report the effectiveness of OFF2F compared to the conventional executable file format.

II. Off2f

Conventional executable program files consist of the 4 regions (header region, program text region, program data region, relocation information region). In typical conventional

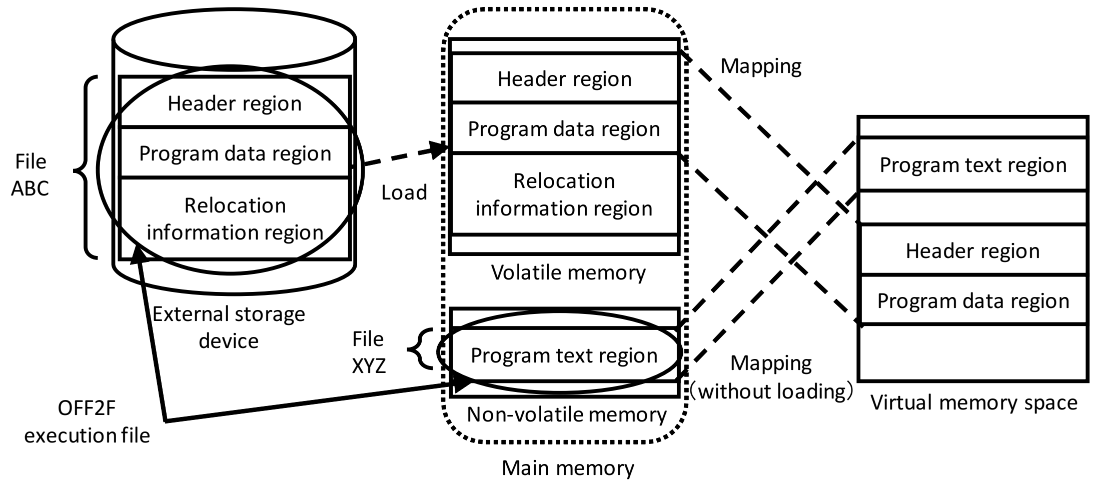

Fig. 1. Mapping in OFF2F program execution.

executable file formats such as Executable and Linkable Format (ELF), the above-listed information is stored in one file. By contrast, in OFF2F, the region which is frequently read on program execution are stored in another file. Non-volatile memory has low read latency, but it's write latency is high. Furthermore, non-volatile memory has a small capacity and is expensive compared to volatile memory. Therefore, in OFF2F, the size of the non-volatile memory used can be reduced because the text program region is stored in the non-volatile memory.

Figure 1 shows mapping when the OFF2F program is executed in the virtual memory system. Therefore, the text program region which is stored in the non-volatile memory is mapped it to the virtual memory space when the OFF2F program is executed. Thus, speeding up program execution is realized by reducing the number of reads from the external storage.

III. Pseudo Non-Volatile Memory Construction

A. Purpose

The development of computers with a mixed-memory environment is limited. Therefore, we realize the pseudo non-volatile memory. Linux kernel has a function that emulate non-volatile area on DRAM. However, the program execution process flow needs to be modified in order to execute the OFF2F program. When modifying the program execution process flow, we thought that the cost of modifying the kernel was low because FreeBSD has simpler process flow than Linux. Therefore, in this section, we describe the method of the pseudo non-volatile memory construction on FreeBSD 11.0-RELEASE.

B. Requirements

The requirements to consist pseudo non-volatile memory with volatile memory are as follows.

Requirement 1: To make the pseudo non-volatile memory accessible in byte units

Like non-volatile memory, pseudo non-volatile memory should be accessible in byte units.

Requirement 2: To reproduce the characteristic of data-existence in pseudo non-volatile memory even after shutdown

Non-volatile memory has the characteristic that the data in the memory is not lost at shutdown. It is necessary to reproduce the characteristic with pseudo non-volatile memory.

Requirement 3: To make the text region file to be stored in the pseudo non-volatile memory

In OFF2F, program text region resides in non-volatile memory. Furthermore, during page fault processing for the program text region when the OFF2F program is executed, the page on the non-volatile memory is mapped to the virtual memory space. Therefore, it is necessary to be able to store OFF2F's program text region in the pseudo non-volatile memory and map to the virtual memory space.

C. Basic Design

1) Reserving pseudo non-volatile memory area using volatile memory: We reserved a part of volatile memory as pseudo non-volatile memory. This satisfies Requirement 1 because volatile memory is accessible in byte units. Figure 2 shows the memory configuration when the pseudo non-volatile memory is realized. The size of the main memory used by the kernel can be set by the boot parameters of the kernel. Therefore, a part of the main memory is not used by the kernel (OS unmanaged area); this can be treated as a reserved area for the pseudo non-volatile memory.

2) Avoiding using specific memory area that is overwritten in the initialization stage: Data in the reserved area is in the OS unmanaged area; therefore, it will not be lost while power is supplied. The characteristic of data existence even at shutdown can be simulated by software reset. Therefore,

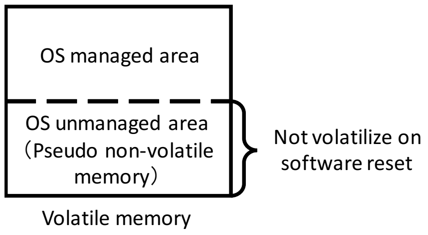
OS managed area
OS unmanaged area (Pseudo non-volatile memory)
Not volatilize on software reset
Volatile memory

Fig. 2. Main memory configuration when realizing pseudo non-volatile memory.

the non-volatile memory is reproduced in software reset. This satisfies Requirement 2. However, it is possible that the data in the OS unmanaged area is initialized and overwritten at the time of the software reset. A simple solution to this problem is to use the memory that is not overwritten at initialization.

3) Proposing a new interface to access the OS unmanaged area: To place files in the pseudo non-volatile memory, it is necessary to access the memory. However, because it is in the OS unmanaged area, it cannot be accessed using conventional interfaces. Therefore, we propose a new function to access the pseudo non-volatile memory. Details of this function are given in the next section. This satisfies Requirement 3.

IV. Pseudo non-volatile memory access function

A. Basic Function

We designed a new function that enables access to the pseudo non-volatile memory. Specifically, we implemented a function that enables memory to be mapped to the virtual memory space.

B. Implementation

In a virtual memory system, main memory can be accessed by mapping it to the virtual memory space. Furthermore, main memory is divided into pages and managed; access is performed in page units.

Access to non-volatile memory is also performed in the same manner. Pseudo non-volatile memory is divided into pages and mapped to a virtual memory space. For the implementation, we used and modified the existing process to divide the pseudo non-volatile memory into pages. Furthermore, we used the existing process of mapping pages of volatile memory to the virtual memory space for the pseudo non-volatile memory.

C. Interface and Processing Contents

Table I shows the interface of nvm_mapping() system call that provides the pseudo non-volatile memory access function. nvm_mapping() system call specifies the starting real address of the pseudo non-volatile memory, the virtual address of the mapping destination, and the number of pages to be added to the kernel. Both addresses must be specified as per the page alignment. Furthermore, the physical address on pseudo non-volatile memory is specified based on the address of the OS unmanaged area. The space configuration based on nvm_mapping() system call is shown in Figure 3, and the processing is as follows.

1) Add the value of `pages` to the in-kernel variable that counts the number of pages managed by the OS.
2) Add `pages` starting with the real address `paddr` of the pseudo non-volatile memory to the existing page queue in the kernel.
3) Map to the virtual address `vaddr` given as the added page.

Through this, a process can access the pseudo non-volatile memory with a specified virtual address.

TABLE I
System call providing access to pseudo non-volatile memory

Synopsis	`nvm_mapping(paddr, vaddr, pages)`
Arguments	`paddr` : First real address of pseudo non-volatile memory `vaddr` : Virtual address of mapping destination `pages`:Number of pages to add to the kernel
Return value	On success : 1 On error : 0
Description	The virtual address `vaddr` of the calling process and the real address `paddr` of the pseudo non-volatile memory are mapped to `pages`; access from `vaddr` to the pseudo non-volatile memory is enabled.

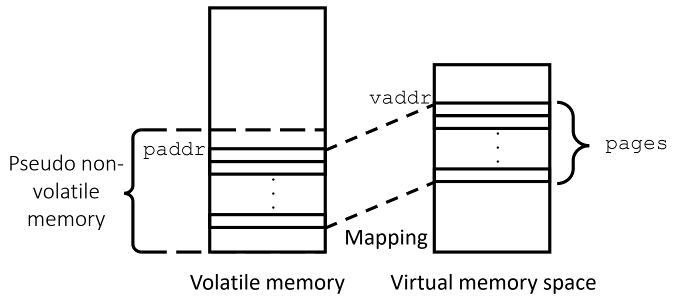

Fig. 3. Space configuration by nvm_mapping() system call.

D. Using Pseudo Non-volatile Memory Access Function

In OFF2F, header region, program data region, and reloacation information region are stored in one file (file ABC), and program text region is stored in another file (file XYZ). To execute the OFF2F program in a mixed environment, file XYZ must be stored in the pseudo non-volatile memory; file ABC must be stored in the external storage device. Therefore, a file system needs to be built in the non-volatile memory and it must cooperate with the file system of the external storage device. The overview of file system construction is shown in Figure 4 and explained as follows.

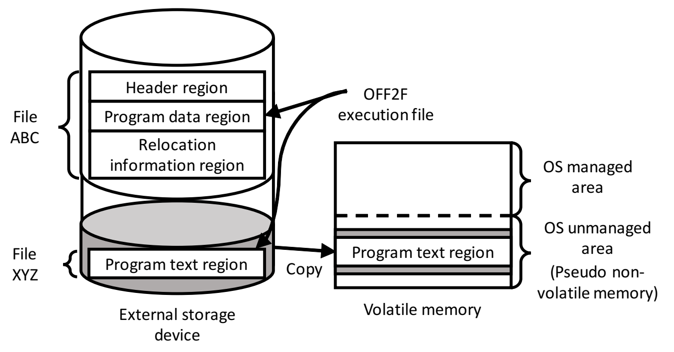

Fig. 4. Copying contents of external storage to pseudo non-volatile memory.

1）Prepare two partitions in the external storage device and build a file system on each partition.
2）One file system stores file ABC and the other stores file XYZ.
3）The pseudo non-volatile memory is mapped to the virtual memory space using the nvm_mapping() system call. We call this virtual memory space as NVM.
4）The file system partition storing the file XYZ is treated as a RAW device, and the data in this partition is copied to NVM.

As described above, the file system can be constructed on the pseudo non-volatile memory and the text region file XYZ can be stored.

V. Implementation

A. Conventional Virtual Memory Space Creation

ELF is an existing executable file format. The process flow of creating a virtual memory space when an ELF program is executed on FreeBSD is shown in Figure 5 and explained as follows.

1）Select the program file to be executed.
2）Allocate pages based on the size of the file. The number of pages to be allocated depends on the size of the file (up to 16 pages).
3）Read the contents of the file to each page allocated in step 2).
4）To read the header region, register the first page in the kernel space mapping table.
5）Create a new virtual memory space for the process.
6）Based on the information in the header region, create an entry for the program text region and data region in the virtual memory space. If a page fault occurs during program execution, the information stored in the entry is read from the external storage device.

In steps 2) and 3), if the size of the file is 64 KB（in the case of 4 KB page environment） or less, the entire file is read. If the size is larger than 64 KB, the first 64 KB is read.

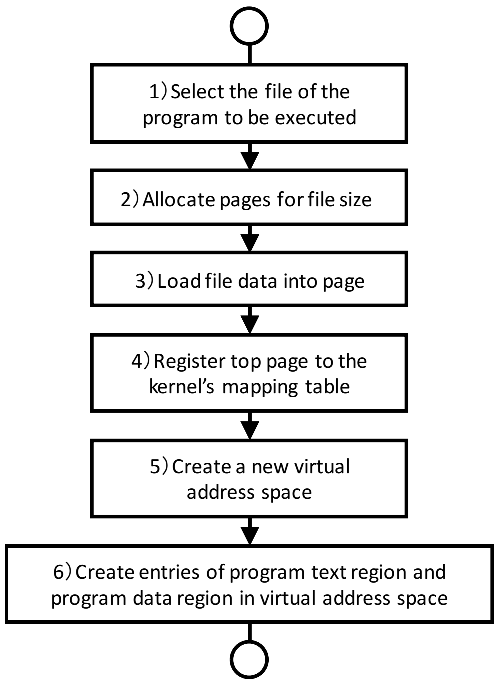
1）Select the file of the program to be executed
2）Allocate pages for file size
3）Load file data into page
4）Register top page to the kernel's mapping table
5）Create a new virtual address space
6）Create entries of program text region and program data region in virtual address space

Fig. 5. Creating a virtual memory space with ELF.

B. Virtual Memory Space Creation Processing at OFF2F Program Execution

When the OFF2F program is executed, the format of executable is recognized based on the contents of the first 4 bytes of the file after reading the header region. Therefore, we modified the header region of the OFF2F program using the header of the ELF program as follows.

1）Change the first 2 to 4 bytes of the file from “ELF” to “OFF”
2）Add the path of file XYZ
3）Delete offset information of the program text region
4）Change offset information of the data section

Figure 6 shows the memory map of the virtual memory space created during program execution for ELF and OFF2F. In an ELF file, header and program text regions occupy continuous memory space. Therefore, in the memory map of the virtual memory space, the text region is placed immediately after the header region. On the other hand, in the OFF2F file, header and data regions occupy continuous memory space. Therefore, in the memory map of the virtual memory space, the program data region is placed immediately after the header region.

In OFF2F, the file containing the program data region exists in the external storage device, and the file containing the program text region exists in the pseudo non-volatile memory. The flow of the creation process of the virtual memory space at the time of OFF2F program execution is shown in Figure 7; its comparison with ELF is explained below.

In many programs, the program data region is smaller than the program text region. For example, in [10], it is shown that

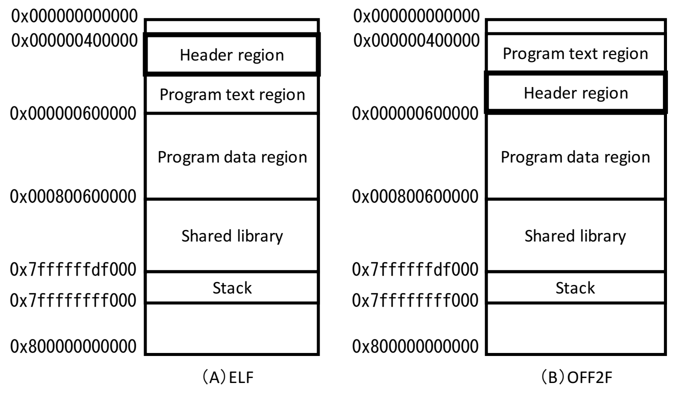

Fig. 6. Memory map of the virtual memory space during program execution.

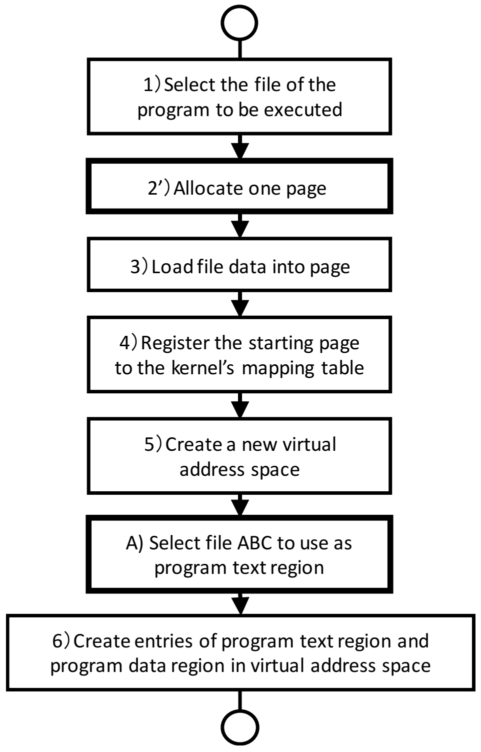
1) Select the file of the program to be executed
2') Allocate one page
3) Load file data into page
4) Register the starting page to the kernel's mapping table
5) Create a new virtual address space
A) Select file ABC to use as program text region
6) Create entries of program text region and program data region in virtual address space

Fig. 7. Creating a virtual memory space with OFF2F.

the total size of the text region of all programs is about 20 times greater than that of the data region in the files under FreeBSD /bin. Therefore, in step 2) in Figure 5, 16 pages are allocated to read the first 64 KB of the file, including header and program text regions, in ELF. Thus, when the OFF2F program is executed, we decided to allocate only one page from the beginning.

Therefore, step 2) was changed as follows.

2'）Allocate one page.

Because OFF2F program consists of two files, it is necessary to add a step to select the text region file XYZ between step 5) and step 6), which is as follows.

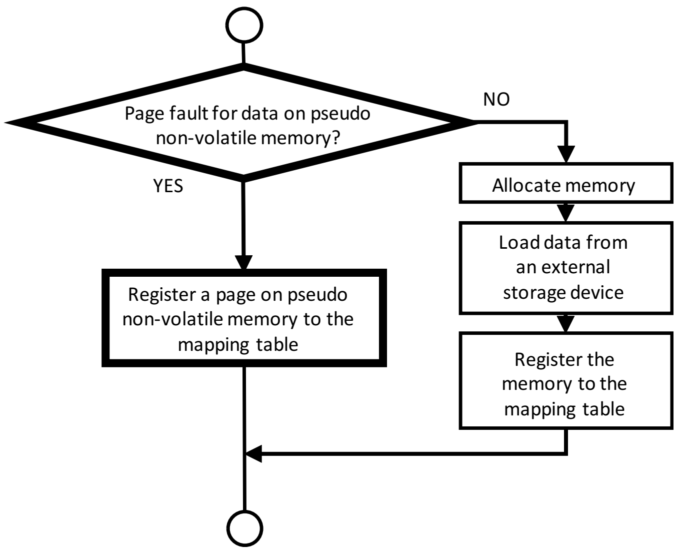
Page fault for data on pseudo non-volatile memory?
NO
YES
Allocate memory
Load data from an external storage device
Register a page on pseudo non-volatile memory to the mapping table
Register the memory to the mapping table

Fig. 8. Page fault handling.

A）Select file XYZ to use as the program text region.

The virtual memory space can be created at the time of OFF2F program execution only by making the above change. Note that the entry of the text region of ELF holds the information in the file; however, that of OFF2F holds the information of the file XYZ.

C. Processing Page Faults

The flow of page fault processing using pseudo non-volatile memory is shown in Figure 8. The bold lines indicate newly implemented processes for handling OFF2F program execution. By using the virtual address where the page fault occurred, it can be determined if a page fault is for the contents in the pseudo non-volatile memory. If the page fault is for the contents in the pseudo non-volatile memory, the pseudo non-volatile memory is searched and the corresponding real addresses are mapped to the virtual memory space based on the information of the entry. If it is not a page fault for the contents in the pseudo non-volatile memory, the existing ODP (on-demand paging) processing is performed.

VI. Evaluation

A. Purpose and Setup

To evaluate the effectiveness of OFF2F, we compared the time of program execution for ELF and OFF2F programs. Note that the text region of the program used for evaluation is entirely accessed during evaluation. Figure 9 shows page access when the evaluation program is executed. Because the entire text region is filled with NOP instructions, the processing is simplified and the overhead due to reading from external storage and page fault processing is reduced. In this section, we measure the time from the start of execution of the programs to the end using the pseudo non-volatile memory. With future technological innovations, non-volatile memory is

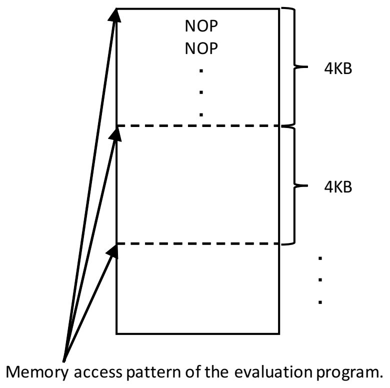

Fig. 9. Memory access pattern of the evaluation program.

TABLE II
THE NUMBER OF NOP INSTRUCTIONS CALLED FOR EACH PROGRAM.

Size of program text region	The number of NOP instructions
32KB	30,633
64KB	63,401
96KB	96,169
128KB	128,937

expected to have read speeds comparable to volatile memory. The pseudo non-volatile memory and the volatile memory have same access speed because the pseudo non-volatile memory is consisted of the volatile memory. Therefore, the results of this evalution expected of the same reading speed between the non-volatile memory and the volatile memory.

On FreeBSD 11.0-RELEASE, which is the environment used for evaluation, the first 64 KB of the executable file is read when the program starts. Therefore, a page fault for program text region does not occur when the size of the program is less than 64 KB. To evaluate the effect of page faults in program execution, we prepared programs of sizes 32 KB, 64 KB, 96 KB and 128 KB of text region using NOP instructions. Table II shows the number of NOP instructions called for each program. Note that the size of the data region of all programs used for evaluation was about 400 B.

To setup the evaluation environment of OFF2F, we preliminary stored the file of program text region in the pseudo non-volatile memory and measured the execution time after performing software reset.

B. Evaluation Environment

The evaluation was performed on a computer equipped with Intel Core i3-6100T(3.20 GHz) and 2 GB memory. In this evaluation, the size of the memory that can be used as volatile memory was 1 GB, and the remaining 1 GB was used as pseudo non-volatile memory. We used FreeBSD 11.0-RELEASE as the OS; the size of one page was set as 4 KB.

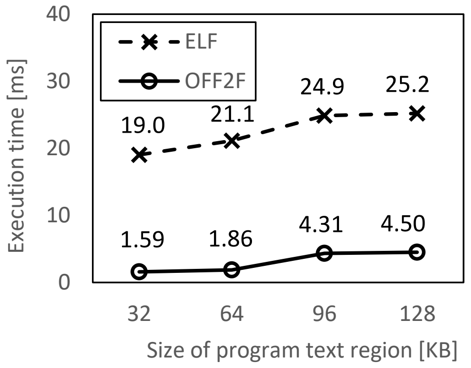

Fig. 10. Processing time for program execution with each file format.

C. Results and Discussion

The results are shown in Figure 10. The following observations are made from Figure 10.

1) The execution time of the OFF2F program is shorter than that of ELF for any text region size. This is because of the time required for page fault processing. In the case of a page fault for the program text region, ELF requires reading the file from the external storage device while OFF2F just maps the page of the pseudo non-volatile memory.

2) The program execution time of ELF becomes longer than that of OFF2F as the size of the text region increases. In ELF, the larger the program text region, the larger the size of the program data region read from the external storage device. On the other hand, when the OFF2F program is executed, only the program data region is read from the external storage device; therefore, the overhead due to increase of the program text region becomes lesser than that of ELF.

3) The execution time for 96 KB text region is about 3 ms higher than that for 64 KB in ELF. This seems to be due to the overhead caused by the file becoming larger than 64 KB and page fault occurrence.

VII. CONCLUSION

To verify the effect of executable file format OFF2F for a computer environment in which both volatile and non-volatile memories are used, this study presented a method to construct pseudo non-volatile memory using an existing computer consisting only volatile memory. This implementation used part of the volatile memory as pseudo non-volatile memory. The size of the main memory used by the kernel was limited and the remaining memory was simulated as non-volatile memory. In addition, the pseudo non-volatile memory can be accessed by partitioning the area of the pseudo non-volatile memory into page units and providing a function to map them in the virtual memory space. Using this function, we built a file system in the pseudo non-volatile memory and stored the text region of the OFF2F program. Additionally, we presented a method to execute an OFF2F program using pseudo non-volatile memory by changing the process of creation of the virtual memory space and page fault processing in FreeBSD.

In the evaluation, the program execution times for ELF and OFF2F were measured for programs with various sizes of text region. The effectiveness of OFF2F was shown due to low reads required from the external storage device and the short page fault processing times.

In our future work, we plan to evaluate the effectiveness of apply OFF2F to various processing, and use the development of computers with a mixed-memory environment.

Acknowledgment

This work was partially supported by JSPS KAKENHI Grant Number JP18K11244 and FUJITSU LABORATORIES LTD.

References

[1] Q. Wei, J. Chen, C. Chen, "Accelerating File System Metadata Access with Byte-Addressable Nonvolatile Memory," ACM Trans. Storage (TOS), vol.11, no.3, pp.1–28, 2015

[2] K. Oe, M. Sato, and T. Nanri, "Automated tiered storage system consisting of memory and flash storage to improve response time with input-output (IO) concentration workloads," 2017 5th Int. Sym. Comput. Networking, pp.311–317, 2017.

[3] S. Zheng, M. Hoseinzadeh, and S. Swanson, "Ziggurat: A Tiered File System for Non-Volatile Main Memories and Disks," FAST'19, pp.207–219, 2019.

[4] J. Yang, J. Izraelevitz, and S. Swanson, "Orion: A Distributed File System for Non-Volatile Main Memory and RDMA-Capable Networks," FAST'19, pp.221–234, 2019.

[5] D. H. Yoon, T. Gonzalez, P. Ranganathan, and R. S. Schreiber, "Exploring latency-power tradeoffs in deep nonvolatile memory hierarchies," Proc. 9th Conf. Comput. Frontiers, pp.95–102, 2012.

[6] Y. Zhang and S, Swanson, "A Study of Application Performance with Non-Volatile Main Memory," Proc. 2015 31st Sym. Mass Storage Syst. Tech. (MSST), pp.1–10, 2015.

[7] A. Eisenman, D. Gardner, I. AbdelRahman, J. Axboe, S. Dong, K. Hazelwood, C. Petersen, A. Cidon, S. Katti, "Reducing DRAM Footprint with NVM in Facebook," Proc. 13th EuroSys Conf. (EuroSys '18), no.42, pp.1–13, 2018.

[8] X. Guo, A. Shrivastava, M. Spear, and G. Tan, "Languages Must Expose Memory Heterogeneity," Proc. 2nd Int. Sym. Memory Syst., pp.251–256, 2016.

[9] A.Koshiba, T.Hirofuchi, S.Akiyama, R.Takano, and M.Namiki, "Towards Write-back Aware Software Emulator for Non-Volatile Memory," 2017 IEEE 6th Non-Volatile Memory Systems and Applications Symposium (NVMSA), pp.1–6, Aug. 2017

[10] M. Sato and H. Taniguchi, "OFF2F: A New Object File Format for Virtual Memory Systems to Support Volatile/Non-Volatile Memory-Mixed Environment," Int. J. Mach. Learning Comput., vol.9, no.4, pp.387–392, 2019.