情報処理学会論文誌 Vol.63 No.2 515–525 (Feb. 2022)

揮発性/不揮発性メモリ混載計算機での高速なプログラム実行を可能にするOFF2Fプログラムの性能評価

谷口 秀夫[1,a)] 高杉 頌[1,b)] 額田 哲彰[1,c)] 佐藤 将也[1,d)]

受付日 2021年4月27日, 採録日 2021年11月2日

概要：バイト単位のアクセスが可能な不揮発性メモリの高性能化と高集積化を受け，将来の計算機の実メモリ構成は，揮発性メモリと不揮発性メモリを混載した環境になると考えられる．そこで，この環境を想定し，実行プログラムを 2 つのファイルに分割格納した実行ファイル形式（OFF2F：Object File Format consisting of 2 Files）を提案した．本論文では，OFF2F プログラムを実行できる環境を構築し，単一プログラムを実行する場合と複数プログラムを繰り返し実行する場合について，OFF2F プログラムと ELF（Executable and Linkable Format）プログラムとのプログラム実行性能の比較評価を述べる．

キーワード：不揮発性メモリ，実行ファイル形式，ページ例外処理，ODP，仮想記憶

Performance Evaluation of OFF2F Program for Fast Execution on Volatile/non-Volatile Memory-Mixed Environment

Hideo Taniguchi[1,a)] Sho Takasugi[1,b)] Tetsuaki Nukata[1,c)] Masaya Sato[1,d)]

Received: April 27, 2021, Accepted: November 2, 2021

Abstract: Due to high performance and high integration of byte-addressable non-volatile memory, the main memory of a future computer would be consisting of volatile and non-volatile memory-mixed environment. Then, assuming this environment, we proposed OFF2F, an executable file format in which the executable program is divided into two files. In this paper, we build an environment where OFF2F programs can be executed, and show the comparative evaluation of the program execution performance between OFF2F and ELF programs on the execution of a single program or multiple programs.

Keywords: non-volatile memory, format of executable program, page fault handling, ODP, virtual memory

1. はじめに

不揮発性メモリ（Non-Volatile Memory：以降，NV メモリと略す）の技術進歩は著しく，揮発性である実メモリと組み合わせることで，処理の高速化や電源断耐性の向上を期待できる．たとえば，電源の ON/OFF が頻繁に発生する機器では，電源を OFF しないでサスペンド状態とすることで次のサービス開始を早める技術があるが，電力消費は避けられない．一方，NV メモリを利用することで，電源 ON からの立ち上げ時間短縮を期待できる．

NV メモリの性能は，ハードウェア技術の進歩により，高性能化と高集積化が進んでいる．これにともない，NV メモリを搭載した計算機の OS に関する研究も進んでいる [1]．

NV メモリをメモリ階層の 1 つとしてとらえる研究が多

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University, Okayama 700–8530, Japan
[a)] tani@cs.okayama-u.ac.jp
[b)] takasugi@swlab.cs.okayama-u.ac.jp
[c)] nukata@swlab.cs.okayama-u.ac.jp
[d)] sato@cs.okayama-u.ac.jp

く，ファイルシステムでの有効利用についての研究として，揮発性メモリと NV メモリの混載環境における性能向上と一貫性保証を提供するファイルシステム NOVA [2]，ファイルシステムをユーザ空間とカーネル空間で分割する SplitFS [3]，頻繁にアクセスされる可能性の高いデータを NV メモリに配置するファイルシステム Ziggurat [4], NV メモリ上のファイルの読み書きを複写レスで行う SubZero [5]，ユーザレベルのメモリマップド IO を拡張し原子性を提供する Libnvmmio [6] がある．ファイルシステムのほかに，データベースの一種である Key-Value ストアの研究として，NV メモリをブロックデバイスとして利用することで揮発性メモリの総量を削減する MyNVM [7]，揮発性メモリ，NV メモリ，および SSD を搭載した環境向けの Key-Value ストアである MatrixKV [8] がある．さらに，NV メモリのプログラミングをサポートするように Go 言語のコンパイラとランタイムを拡張した go-pmem [9]，永続データへのアクセスからカーネルを排除し，NV メモリへのシンプルなアクセス環境を提供するデータセントリックな OS である Twizzler [10] がある．

最近，NV メモリの 1 つとして Intel 社の Optane DC Persistent Memory（DCPM）が製品化され，実メモリとしての利用が期待される [11]．一方，NV メモリのエミュレートを行う方法として，ハイパーバイザ上でエミュレートを行う方法 [12]，ハードウェアエミュレータを使用する方法 [13]，NUMA（Non-Uniform Memory Access）によるメモリアクセス時間の差を利用する方法 [4], [14]，ツールを用いてエミュレートを行う方法 [15] がある．

我々は，揮発性メモリと NV メモリが混載された計算機において，高速なプログラム実行を可能にする新たな実行ファイル形式（OFF2F：Object File Format consisting of 2 Files）を提案した [16]．文献 [2] は，ファイルシステムの改変が必要であるが，OFF2F は，従来のファイルシステムを利用し，ページ例外（PF：Page Fault）処理の一部を改変するだけで NV メモリに配置したファイルを実行可能にしている．

我々は，OFF2F を提案するとともに，要求時ページング（ODP：On Demand Paging）における CoW（Copy on Write）機能を考慮した PF 処理について述べ，PF 処理の時間を定式化し，OFF2F プログラムを実行することによる PF 処理時間の短縮効果とともに，OS 初期化処理時間の短縮効果を報告した [17]．しかし，これらは，PF 処理時間の定式化と PF 回数の実測値をもとに，短縮効果を予測したものであり，OFF2F プログラムを実行した実測値による評価ではない．

そこで，本論文では，OFF2F プログラムに関し，FreeBSD 上で実行環境を構築し，実行し，実測し，同様な計算機で行った ELF（Executable and Linkable Format）プログラムとの比較評価を述べる．具体的には，揮発性メモリのみで構成される計算機上で擬似 NV メモリを作成し，OFF2F プログラムの実行環境を構築した．また，date，less，cc といった単一プログラム実行時の性能，および make を利用した複数プログラムの繰り返し実行時の性能について，OFF2F プログラムと ELF プログラムを比較した．この結果，たとえば cc では実行時間を約 1/5 に短縮できることを明らかにした．なお，擬似 NV メモリは揮発性メモリを利用しているため，現存の NV メモリは読み込み速度が約 2 倍遅いことを考慮すると，その効果は 15%減である．

2. 混載計算機と OFF2F

2.1 揮発性/不揮発性メモリ混載計算機

揮発性メモリは，アクセス速度が高速であり，これまでの技術革新により大容量かつ低価格である．これに対し，ハードウェア構造やエネルギー効率の性質上，相対的に見ると，NV メモリは，アクセス速度が低速である．最近の DCPM の場合，内部にキャッシュメモリを有することで性能上の欠点を補っており，書き込み速度は DRAM と同等（レイテンシ：60〜90 ナノ秒）であるが，読み込み速度は 2〜3 倍（レイテンシ：DRAM 81〜101 ナノ秒，DCPM 169〜305 ナノ秒）遅い [11]．このため，今後の計算機として，実メモリは揮発性メモリと NV メモリを混載した構成になる．

一方，不揮発性という共通の特徴を持つ外部記憶装置と NV メモリを比較すると，アクセス単位の面で大きな違いがある．外部記憶装置はブロック単位であるのに対し，NV メモリはバイト単位である．また，磁気ディスク装置（DK）や SSD といった外部記憶装置は，大容量，低価格および高耐久性である．したがって，今後の計算機として，実メモリは揮発性メモリと NV メモリの混載した構成になり，外部記憶装置は，バス接続され，入出力装置の位置づけである．

2.2 従来の実行ファイル形式と OFF2F

NV メモリは，揮発性メモリと同様にバイト単位アクセスが可能であり，読み出しは高速である．一方で，書き込みは低速であるが，内部にキャッシュメモリを有することで大きく改善されている．したがって，NV メモリ上にファイルシステムを構築することで読み出しのみ行われるデータを NV メモリに格納し，仮想記憶を利用してそのまま仮想メモリ空間にマッピングできれば，ODP 処理の入出力時間を短縮できる．

実行ファイルは，4 つの内容（テキスト部，データ部，関係情報部，ヘッダ部）から構成される．テキスト部は手続き（命令列）であり，データ部は外部変数格納域，関係情報部は外部変数の名称とアドレス情報を持つ．ヘッダ部は上記の 3 部分の実行ファイル内での格納情報を持つ．それぞれの内容は，アクセス形態から大きく 2 つ（読み出しの

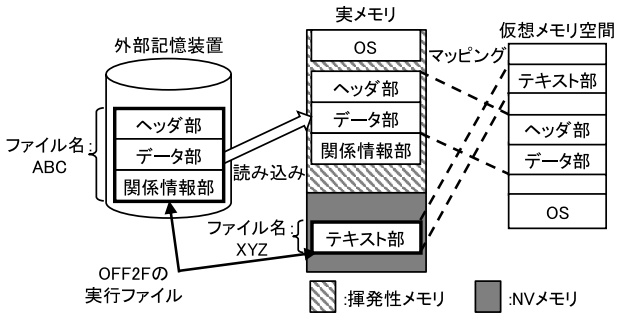

図 1 OFF2F プログラム実行時のマッピング
Fig. 1 Mapping in OFF2F program execution.

み行われる部分と読み書きされる部分）に分類できる．そこで，我々は実行ファイルの内容のアクセス形態に着目し，実行ファイルを分割して NV メモリと外部記憶装置に分散配置する実行ファイル形式（OFF2F：Object File Format consisting of 2 Files）を提案した [16]．OFF2F プログラム実行時のマッピングの様子を**図 1** に示す．ファイル ABC はヘッダ部，データ部，および関係情報部を含むファイルであり，ファイル XYZ はテキスト部のファイルである．ファイル ABC を外部記憶装置上に置き，読み出しのみ発生するファイル XYZ を NV メモリ上に置く．

このようにすることで，ファイル ABC は，プログラム実行時の仮想メモリ空間を構築する処理において，既存の処理の流れを利用できる．また，OFF2F プログラム実行時には，NV メモリ上のファイル XYZ をそのまま仮想メモリ空間にマッピングできるため，テキスト部の外部記憶装置からの入出力時間を短縮できる．

2.3 OFF2F の期待される効果

OFF2F は，実行プログラムのテキスト部を NV メモリ上に置くことで，テキスト部で発生した ODP 処理を高速化できる．したがって，その効果が発揮されるサービスとして，テキスト部のサイズが大きく，相対的にデータ部のサイズが小さく，処理で扱うデータの入出力量が少ないプログラムの実行を高速化できる．

FreeBSD（Ver.11）において，/bin や/sbin の下にある OS の基本的な各コマンドの大半のプログラムは，テキスト部サイズがデータ部サイズに比べ 20 倍以上大きい．/bin の下にある実行プログラム 44 個のテキスト部の総和は 1,518,784 バイトでありデータ部は 80,136 バイトであり，/sbin の下にある実行プログラム 131 個のテキスト部の総和は 7,508,810 バイトでありデータ部は 383,687 バイトである．また，/usr/bin の下についても，実行プログラム 444 個のテキスト部の総和は 374,402,883 バイトでありデータ部は 1,750,691 バイトである．したがって，OFF2F は，OS の基本的な各コマンドのプログラム実行を高速化できるといえる．また，コマンドインタプリタ csh は，テキスト部サイズ（375 KB）が大きく，データ部サイズ（15 KB）が小さいため，コマンド処理の高速化を期待できる．

また，たとえば，FreeBSD 6.3R の場合，init プログラムは，テキスト部サイズ（1,012 KB）が大きく，データ部サイズ（15 KB）が小さいため，OS 初期化処理の短縮に効果を発揮している [17]．また，後述するように，cc では，テキスト部サイズ（42,796 KB）が大きく，データ部サイズ（18 KB）が小さいため，処理で扱うデータの入出力量が少ない場合に大きな効果を発揮している．

次に，価格について考察する．128 GB で比較すると，現在，NV メモリの Intel Optane PMEM 200 は DDR4 DRAM の約 2/3 と低価格である．さらに，SSD は NV メモリの Intel Optane PMEM 200 の約 1/10 と非常に低価格である．したがって，新たに NV メモリを搭載して OFF2F を導入しても価格の上昇はわずかである．なお，NV メモリ搭載量に見合った DRAM を削減すれば，価格の上昇を抑制でき，削減することも可能である．しかし，NV メモリの容量は，多くの場合少なくとも 128 GB 程度であるため PC 環境としては多量であり，16 GB のようなものは存在しない．このため，OFF2F の導入は，サーバ環境で有効と期待できる．

3. 実装と評価

3.1 擬似 NV メモリと OFF2F プログラム実行環境の構築

揮発性メモリのみで構成される計算機上で擬似 NV メモリを作成し，OFF2F プログラムの実行環境を構築した．

まず，実メモリの一部を OS が使用しない領域とし，擬似 NV メモリとして扱う．この領域は，OS が使用しない領域であるため，OS の初期化処理でも利用されず，計算機の電源を OFF しない限り OS を再立ち上げしても，内容は保存され更新されない．つまり，NV メモリとして擬似できる．また，この擬似 NV メモリへデータを格納するために，当該領域を仮想メモリ空間にマッピングし読み書きできる機能を作成した．

次に，ELF プログラムをもとに OFF2F プログラムを作成した．具体的には，ヘッダ部に格納されているマジック番号を “OFF” とし，ELF プログラムのテキスト部を外部記憶装置から擬似 NV メモリへ複写し，テキスト部は擬似 NV メモリ上に存在するようにヘッダ部情報を設定する．

上記により格納した OFF2F プログラムの様子を**図 2** に示す．OFF2F プログラムのヘッダ部を含むファイルは外部記憶装置上に存在し，テキスト部のファイルは擬似 NV メモリ上に存在する．OFF2F プログラム実行時は，テキスト部として擬似 NV メモリ上のファイルを利用する．

上記の擬似 NV メモリは，揮発性メモリを利用しているため，現存の NV メモリより高性能である．2.1 節で述べたように，DCPM と DRAM の性能は，書き込み速度は同等であるが，読み込み速度は DCPM が 2〜3 倍遅い．一

方，プロセッサ動作クロックが高くなっており，プロセッサ処理時間はメモリキャッシュヒット率や TLB ヒット率の影響を大きく受ける．これに対し，実行するプログラムが OFF2F プログラムか否かに関係なく，メモリキャッシュヒット率や TLB ヒット率は同じである．たとえば，Intel Core i7-6700（3.4 GHz）の場合，L1 キャッシュ約 1 ナノ秒，L2 キャッシュ約 4 ナノ秒，L3 キャッシュ約 13 ナノ秒である．また，主メモリは 100 ナノ秒以下になっている．そこで，たとえば，メモリキャッシュヒット時とミス時のアクセス速度差を 10 倍とし，揮発性メモリと NV メモリの読み込み速度差を 2 倍として性能を推察する．また，後述する表 1 や表 3 に示すように，テキスト部の大きさはデータ部の大きさに比べ非常に大きいため，メモリキャッシュミスの 90%がテキスト部で発生したと仮定する．この場合，プロセッサが保有するメモリキャッシュ量は大きくなっているため，メモリキャッシュヒット率を 98%としたとき，平均メモリアクセス速度は 15%遅くなる．したがって，以降の評価において，現存の NV メモリにおける OFF2F の性能効果は 15%減となる．一方，NV メモリに関する技術は急速に進歩しており，揮発性メモリと NV メモリの読み込み速度差がなくなれば，以降の評価結果の効果が期待できる．

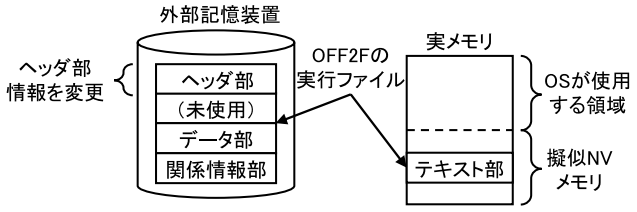

図 2 格納した OFF2F プログラム
Fig. 2 Stored OFF2F program.

3.2 プログラム実行時の処理の流れ

擬似 NV メモリを FreeBSD の動作環境で実現した．外部記憶装置に格納した ELF プログラム，および外部記憶装置と擬似 NV メモリに格納した OFF2F プログラムについて，プログラム実行時の処理の流れを図 **3** に示す．まず，ELF プログラム実行時の処理の流れ (A) について，以下に説明する．

(1) fork() により，プロセスを生成する．
(2) ファイルの先頭 64 KB を読み込み，ヘッダ部情報を獲得する．
(3) ヘッダ部情報からテキスト部やデータ部等を配置する仮想アドレスを取得し，仮想メモリ空間を構築する．
(4) プログラムの実行を開始する．
(5) ページ例外が発生したとき，PF 処理として (5-1)，(5-2)，および (5-3) を行う．
(5-1) 実メモリを確保する．
(5-2) 外部記憶装置から当該データを実メモリに読み込む．
(5-3) 実メモリをマッピング表に登録する．
(6) プログラム実行終了時，仮想メモリ空間の削除等のプロセス終了処理を行う．

次に，OFF2F プログラム実行時の処理の流れ (B) について説明する．この処理の流れは，ELF プログラムの場合と大きく 2 つの処理が異なる．1 つは，PF 処理である．OFF2F プログラムにおける PF 処理は以下の 2 つの処理を追加する．

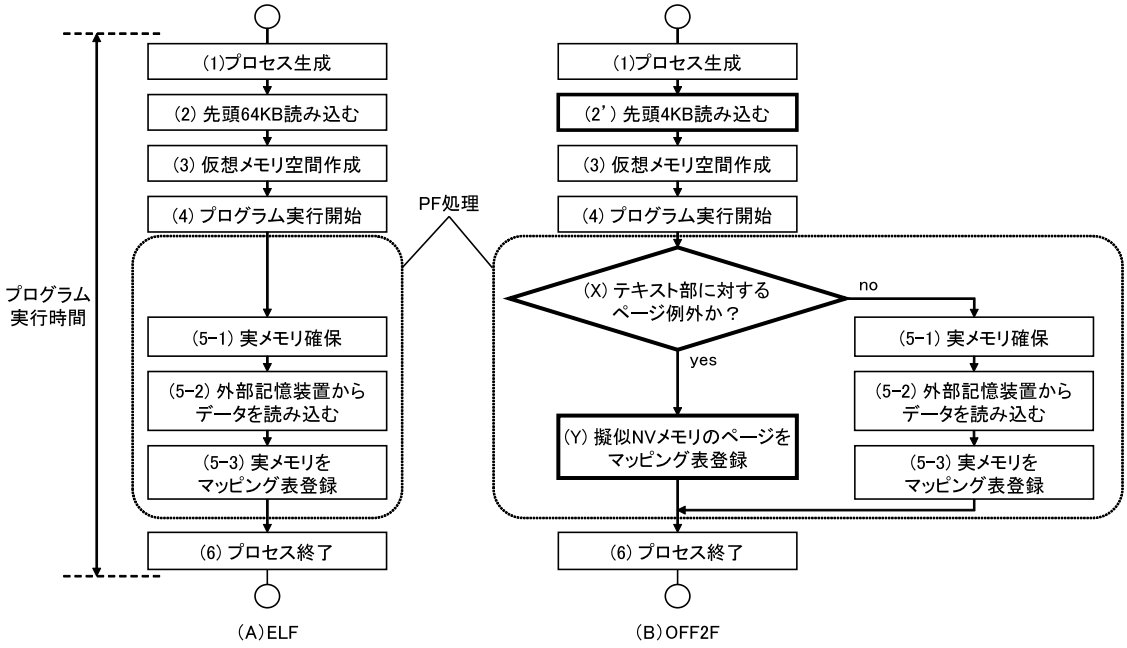

図 3 プログラム実行時の処理の流れ
Fig. 3 Program execution flow.

(X) 擬似 NV メモリ上データに対するページ例外であるか確認する．
(Y) 擬似 NV メモリ上データに対するページ例外の場合，当該の擬似 NV メモリのページをマッピング表に登録する．

もう 1 つは，プロセス起動時におけるファイルの先頭読み込み処理である．ELF プログラムでは，プロセス起動時に先頭 64 KB を読み込む．これは，ヘッダ部とテキスト部は連続して存在し，どちらも読み出しのみ行われる部分であるため，先頭 4 KB ではなく先頭 64 KB を読み込むことでページ例外の発生回数を削減している．一方，OFF2F プログラムでは，ヘッダ部はデータ部と連続して存在する．データ部は読み書きされる部分であるため，以下の処理にする．

(2') ファイルの先頭 4 KB を読み込み，ヘッダ部情報を獲得する．

3.3 実装内容

擬似 NV メモリと OFF2F プログラム実行環境の構築，およびプログラム実行時の処理流れの実現に関する実装内容を以下に述べる．

OS カーネルの改造は，3 つある．OS 起動時に，実メモリ量を BIOS から引き継がず，固定値で持つようにした．これにより，たとえば，固定値を 6 GB に設定すると，8 GB メモリ搭載の計算機においては 6 GB メモリ搭載と OS は認識し，2 GB を擬似 NV メモリとして利用する．また，プログラム実行開始時の OFF2F 用処理（OFF2F 判別処理とテキスト部の NV メモリ利用指示処理）を作成した．さらに，PF 処理として，図 3 に示したように，テキスト部でのページフォールトか否かの条件文を追加し，OFF2F 用の処理を追加した．これらの改造は C 言語で記述し，改造量は 562 行である．

上記の改造において，OS 起動時の処理の改造は，OS 起動時間への影響はなく，プログラム実行時間にも影響を与えない．また，プログラム実行開始時や PF 時の処理の改造は，1 つの条件文が追加されたのみなのでプログラム実行時間へのオーバヘッド増加は非常に少ない．

また，ツールプログラムとして，ELF を OFF2F に変換するプログラム，および擬似 NV メモリ上に OFF2F プログラムを格納するプログラムを作成した．これらの作成は C 言語で記述し，作成量は 134 行である．

3.4 評価の観点

プログラムの実行時間は，プログラムが行う処理を実行する時間（以降，実質時間と略す）とプログラムを実行するための OS の制御処理時間（以降，OS 制御時間と略す）からなる．OFF2F プログラムは，OS 制御時間を削減する．したがって，OS 制御時間がプログラム実行時間に占める割合が大きいプログラムほど，OFF2F プログラムの効果は大きくなる．そこで，以下の観点で評価する．

(1) 単一プログラム実行時の性能

実質時間が短い場合であり，OS 制御時間を変化させた．具体的には，テキスト部のサイズが異なる 3 つのプログラムの実行時間を比較する．3.2 節で述べたように，ELF プログラムにおいて，テキスト部のサイズが 64 KB より小さい場合はテキスト部でページ例外が発生しない．そこで，テキスト部のサイズについて，64 KB より小さいプログラム，64 KB より大きいプログラム，および 64 KB に比べ非常に大きいプログラムを評価対象とした．

(2) 複数プログラム繰り返し実行時の性能

OS 制御時間の割合が少ない場合であり，実質時間を変化させた．具体的には，プログラム実行時間が異なる 2 つのプログラムを評価対象とした．

3.5 評価の環境

評価に用いた計算機を以下に示す．外部記憶装置のアクセス速度が異なる場合として，DK と SSD を用意した．

- OS：FreeBSD 11.0-R
- プロセッサ：Intel Core i3-8100（3.1 GHz）
- メモリ：Transcend DDR4（4 GB × 2）
- DK：TOSHIBA MQ01ABF050（5,400 RPM）
- SSD：SanDisk Ultra 3D SSD

なお，OS の実メモリを 7 GB とし，擬似 NV メモリを 1 GB とした．また，評価プログラムの実行前に，測定に無関係なデータ（2 GB）を DK または SSD から読み込む処理を行い，OS の入出力バッファのキャッシュ状態，および DK または SSD の内部に搭載されているキャッシュ状態が測定に影響を与えないようにした．

また，プログラム実行時間は，評価プログラムを fork()&exec() して起動し wait() で終了を待つプログラムを作成し，fork() システムコール発行直前と wait() システムコール終了直後を RDTSC 命令で実測し，算出した．

3.6 単一プログラム実行時の性能

3.6.1 評価プログラム

評価に用いたプログラムを**表 1** に示す．

date は，現在の日時を標準出力するプログラムであり，テキスト部のサイズは約 16 KB であり 64 KB より小さい．less は，大きさ 0 のファイルを標準出力するように起動した．そのテキスト部のサイズは約 138 KB であり，64 KB より大きい．cc は，C 言語で書かれた簡単なプログラム（printf() で 13 文字を標準出力するもの）をコンパイルするように起動した．そのテキスト部のサイズは約 42 MB であり，64 KB より非常に大きい．この場合，ld（テキスト部のサイズは約 1.5 MB）も起動される．

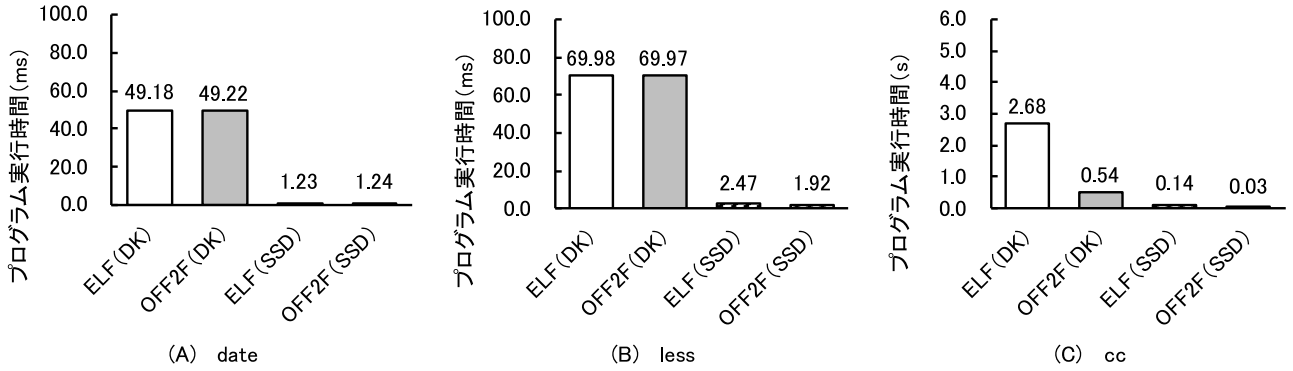

図 4 単一プログラム実行時の実行時間

Fig. 4 Execution time on the execution of a single program.

表 1 単一プログラム実行時に起動する実行プログラム

Table 1 Executable programs on the execution of a single program.

(A) date

通番	プログラム名	テキスト部 (KB)	データ部 (KB)
1	date	16	1

(B) less

通番	プログラム名	テキスト部 (KB)	データ部 (KB)
1	less	138	13

(C) cc

通番	プログラム名	テキスト部 (KB)	データ部 (KB)
1	cc	42,796	18
2	ld	1,562	20
	合計	44,358	38

表 2 単一プログラム実行時の PF 回数（ELF）

Table 2 The number of page faults on the execution of a single program (ELF).

(A) date

通番	プログラム名	プログラムのテキスト部	ライブラリのテキスト部	それ以外
1	date	0	0	74

(B) less

通番	プログラム名	プログラムのテキスト部	ライブラリのテキスト部	それ以外
1	less	1	0	124

(C) cc

通番	プログラム名	プログラムのテキスト部	ライブラリのテキスト部	それ以外
1	cc	408	0	619
2	ld	22	0	1,296
	合計	430	0	1,915

3.6.2 結果と考察

単一プログラム実行時の実行時間を**図 4** に示す．当然のことながら，DK に比べ SSD の場合は，実行時間が短い．しかし，DK か SSD かに関係なく，図 4 から以下のことが分かる．

(1) OFF2F プログラムの実行時間は，テキスト部のサイズが大きい場合（64 KB を超える場合）ほど，ELF プログラムの実行時間より短い．たとえば，cc の場合，テキスト部サイズが非常に大きいため，OFF2F プログラムの実行時間は，ELF プログラムの実行時間を 1 としたとき，DK で 0.20（0.54/2.68），SSD で 0.21（0.03/0.14）と短い．しかし，less の場合，DK で 0.99（69.97/69.98），SSD で 0.78（1.92/2.47）とやや短い程度である．これは，テキスト部のサイズが大きい場合ほどページ例外が多く発生し，この回数が多いほど，OFF2F プログラムによってテキスト部の入出力時間を短縮できるためである．

(2) OFF2F プログラムの実行時間は，テキスト部のサイズが 64 KB 未満の場合，ELF プログラムの実行時間より長い．date の場合，テキスト部サイズが 64 KB 未満のため，OFF2F プログラムの実行時間は，ELF プログラムの実行時間に比べ DK で約 0.04 ms（49.22 – 49.18），SSD で 0.01 ms（1.24 – 1.23）長い．これは，OFF2F プログラムにより入出力回数が削減されることで短縮できる入出力時間に比べ，OFF2F プログラムのみで発生するページ例外の処理時間の方が長いためである．

上記の考察をプログラム実行時の処理の流れに基づき分析するために，ELF プログラム実行時の PF 回数を測定した．結果を**表 2** に示す．OFF2F プログラム実行時の PF 回数は，図 3 の (2) および (2') の処理の違いにより，プログラムのテキスト部の PF 回数が多く，それ以外の PF 回数は同じである．図 4 と表 2 より，テキスト部のサイズが大きい場合，たとえば図 4 (C) の cc の DK において，OFF2F プログラムの実行時間は，ELF プログラムに比べ 2.14 s（2.68 – 0.54）短い．図 3 に示したプログラム実行時の処理の流れから，この短縮時間はテキスト部の PF 処理時間の差である．ここで表 2 より，ELF プログラムのテキスト部の PF 回数は 430 回であり，PF 処理時間の総

和は少なくとも 2.14 s である一方，テキスト部以外の PF 回数は 1,915 回であるにもかかわらず，PF 処理時間の総和は 0.54 s 未満である．したがって，テキスト部の PF 処理時間は，それ以外の PF 処理時間に比べ長い．この結果，OFF2F ではテキスト部の PF 処理時間を大きく短縮できるため，項目 (1) の効果が生まれる．

したがって，実質時間が短く，かつ OS 制御時間が長い（テキスト部のサイズが 64 KB より大きい）処理では，実行プログラム形式を OFF2F にすることにより，プログラム実行を高速化できるといえる．

一方，OFF2F の効果は，外部記憶装置の入出力性能の影響を受ける．そこで，cc について，図 4 (C) は SATA 接続の SSD の場合であるため，高入出力性能を有する NVMe 接続の SSD の場合を述べる．なお，測定は，NVMe 接続の SSD 環境を有する計算機（OS：FreeBSD 11.0-R，プロセッサ：Intel Core i5-10210U (1.6 GHz)，メモリ：Crucial DDR4 (8 GB × 1)，NVMe SSD：WD Blue SN550 NVMe SSD）を用いた．この結果，OFF2F プログラムの実行時間は，ELF プログラムの実行時間を 1 としたとき，0.37 (39.81 ms/107.53 ms) と短いものの，図 4 (C) の SATA 接続の SSD の場合（0.21）に比べ高速化効果は小さい．したがって，外部記憶装置の入出力性能が高くなると OFF2F の効果は小さくなるものの，2 倍以上の高速化は維持されている．

3.7 複数プログラム繰り返し実行時の性能

3.7.1 評価プログラム

評価に用いたプログラムを**表 3** に示す．

make では，ls コマンドの実行ファイルを作成するように起動した．C 言語で書かれた 9 個のプログラム（合計サイズは約 80.1 KB）をコンパイルする．この場合，11 種類のプログラムが起動される．OS カーネル作成処理では，FreeBSD の OS カーネルを作成するように起動した．この場合，30 種類のプログラムが起動される．また，コンパイルするプログラムの合計サイズは約 3.9 GB である．

3.7.2 結果と考察

複数プログラム繰り返し実行時の実行時間を**図 5** に示す．図 5 から，以下のことが分かる．

(1) OFF2F プログラムの実行時間は，コンパイルするプログラムの合計サイズが小さい場合，ELF プログラムの実行時間より短い．たとえば，make の場合，コンパイルするプログラムの合計サイズが小さく，OFF2F プログラムの実行時間は，ELF プログラムの実行時間を 1 としたとき，DK で 0.45（2.11/4.69）と短い．一方，OS カーネル作成処理の場合，DK で 0.99（1,104.89/1,107.41）とやや短い程度である．これは，ELF プログラムを 1 度実行するとテキスト部はメモリ上にキャッシュされ，繰り返し実行した際にテキスト

表 3 複数プログラム繰り返し実行時に起動する実行プログラム
Table 3 Executable programs on the execution of multiple programs.

(A) make（ls コマンド作成）

通番	プログラム名	テキスト部 (KB)	データ部 (KB)	exec() で起動される回数
1	awk	188	3	2
2	cat	8	1	2
3	cc	42,796	18	24
4	chmod	5	1	2
5	gzip	33	1	2
6	ld	1,562	20	2
7	make	688	15	4
8	mv	10	1	4
9	objcopy	106	6	5
10	sed	32	1	2
11	sh	143	2	36
	合計	45,577	68	85

(B) OS カーネル作成処理（FreeBSD 作成）

通番	プログラム名	テキスト部 (KB)	データ部 (KB)	exec() で起動される回数
1	as	1,447	20	5
2	awk	188	3	864
3	cat	8	1	10
4	cc	42,796	18	9,133
5	chmod	5	1	1
6	cp	15	2	1
7	ctfconvert	96	3	4,552
8	ctfmerge	65	2	817
9	date	16	1	1
10	dirname	2	1	1
11	echo	3	1	128
12	find	48	1	3
13	grep	92	1	13
14	hostname	3	1	1
15	ld	1,562	20	946
16	ln	7	1	5
17	make	688	15	1,735
18	mv	10	1	44
19	nm	89	1	816
20	objcopy	106	6	2,118
21	realpath	2	1	1
22	rm	11	1	7
23	sed	32	1	7
24	sh	143	2	11,504
25	size	15	1	1
26	sort	54	3	2
27	svnliteversion	1,561	4	2
28	touch	7	1	1
29	uudecode	9	1	126
30	xargs	12	1	814
	合計	49,093	111	33,659

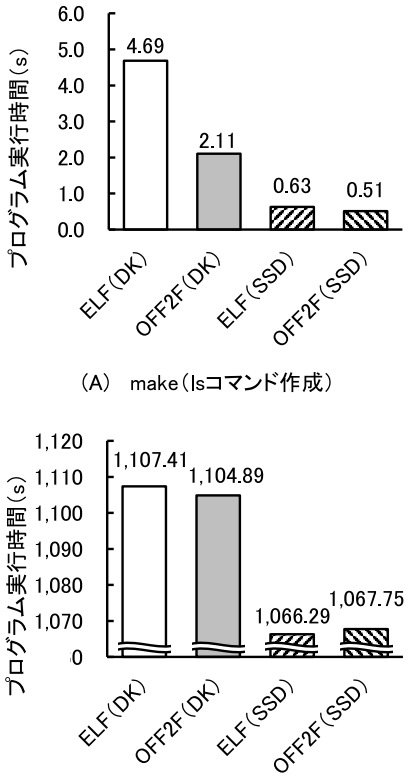

図 5 複数プログラム繰り返し実行時の実行時間
Fig. 5 Execution time on the execution of multiple programs.

部の入出力は発生しない一方で，コンパイルするプログラムの合計サイズが小さい場合ほど，実質時間（テキスト部以外の入出力時間，およびコンパイルにおける字句解析や構文解析等のプロセッサ処理の時間）が短くなり，実行時間全体に占めるテキスト部の入出力時間の割合が増加するためである．これに対し，コンパイルするプログラムの合計サイズが大きい場合，実質時間が長いため，効果が減少する．また，ELF プログラムにおいては最初の実行では外部記憶装置からの入出力が発生するものの，2 回目以降の実行ではメモリ上にキャッシュされ必ずしも入出力が発生しない．表 3 に示すように，cc の実行回数は，make の場合 24 回であるが，OS カーネル作成処理の場合 9,133 回と多い．このため，実行回数が少ない make の方がキャッシュ効果を得にくく，その結果 OFF2F プログラムの効果が大きくなる．

上記の推察を裏付けるために，time コマンドにより DK における ELF プログラム実行時のプロセッサ処理の時間を測定した．結果を**表 4** に示す．プログラム実行において，プロセッサ処理の時間や PF 時のデータ部の入出力時間および処理で扱うファイルの入出力時間は，ELF プログラムと OFF2F プログラムに差はない．プログラム実行時間の差は，プロセス生成時の最初の読み込み時間（64 KB か 4 KB かの違い）およびテキスト部の PF 時の処理内容の違いに起因する．したがって，図 5 のプログラム実行時間の ELF プログラムと OFF2F プログラムの差の大半は，ELF プログラムの PF 時のテキスト部の入力時間である．このため，DK における ELF プログラムについて，make の場合，プログラム実行時間である 4.69 s のうち，テキスト部の入出力時間は 2.58 s（4.69 − 2.11）である．また，OFF2F プログラムではテキスト部の PF 時に入力は発生しないため，表 4 より，make の場合，プロセッサ処理の時間は 0.45 s であり，テキスト部以外の入出力時間は 1.66 s（2.11 − 0.45）未満である．一方，OS カーネル作成処理の場合，図 5 より，プログラム実行時間である 1,107.41 s のうち，テキスト部の入出力時間は 2.52 s（1,107.41 − 1,104.89）である．また，表 4 より，プロセッサ処理の時間は 1,033.09 s であり，テキスト部以外の入出力時間は 71.8 s（1,104.89 − 1,033.09）未満である．以上のことから，両場合とも，ELF プログラムのテキスト部の入力時間は約 2 秒である．これに対し，プログラム実行時間は，make の場合は数秒であるが，OS カーネル作成処理の場合は 1,000 秒程度と長い．つまり，make の実質時間は短く，OS カーネル作成処理の実質時間は非常に長い．このため，make では OFF2F プログラムの実行時間を大きく短縮できる．

表 4 複数プログラム繰り返し実行時のプロセッサ処理の時間（ELF）
Table 4 CPU time on the execution of multiple programs (ELF).

評価プログラム	プロセッサ処理の時間 (s)
make（ls コマンド作成）	0.45
OS カーネル作成処理（FreeBSD 作成）	1,033.09

また，図 5 から，以下のことも分かる．

(2) OFF2F プログラムの実行時間は，SSD において，make の場合，ELF プログラムの実行時間に比べ 19.05%（(0.63 − 0.51)/0.63）短い一方，OS カーネル作成処理の場合，非常にわずかであるが 0.14%（(1,067.75−1,066.29)/1,066.29）長い．これは，OFF2F プログラムにより入出力回数が削減されることで短縮できる入出力時間に比べ，OFF2F プログラムのみで発生するページ例外の処理時間の方が長いためである．

上記の考察を分析するために，ELF プログラム実行時の PF 回数を測定した．結果を**表 5** に示す．OFF2F プログラム実行時の PF 回数は，図 3 の (2) および (2') の処理の違いにより，各プログラムで少なくとも 1 回はテキスト部の PF 回数が多い．表 3 (A) より，make の場合，11 種類のプログラムが起動されるため，OFF2F プログラム実行時の PF 回数は，ELF プログラム実行時の PF 回数に比べ少なくとも 11 回多い．また，表 5 (A) より，ELF プログラムのテキスト部の PF 回数は 509 回である．したがって，OFF2F プログラムの実行時間について，509 回の入出力回

表 5 複数プログラム繰り返し実行時の PF 回数（ELF）
Table 5 The number of page faults on the execution of multiple programs (ELF).

(A) make（ls コマンド作成）

通番	プログラム名	プログラムのテキスト部	ライブラリのテキスト部	それ以外
1	awk	2	2	92
2	cat	0	0	62
3	cc	474	0	7,001
4	chmod	0	0	66
5	gzip	1	0	133
6	ld	22	0	1,638
7	make	9	0	604
8	mv	0	0	108
9	objcopy	1	29	688
10	sed	0	1	73
11	sh	0	0	1,659
	合計	509	32	12,124

(B) OS カーネル作成処理（FreeBSD 作成）

通番	プログラム名	プログラムのテキスト部	ライブラリのテキスト部	それ以外
1	as	20	0	6,723
2	awk	2	2	95,148
3	cat	0	0	635
4	cc	498	0	15,179,066
5	chmod	0	0	66
6	cp	0	0	67
7	ctfconvert	1	7	3,025,306
8	ctfmerge	0	0	663,797
9	date	0	0	74
10	dirname	0	0	64
11	echo	0	0	7,934
12	find	0	0	658
13	grep	1	6	1,146
14	hostname	0	0	62
15	ld	22	0	323,251
16	ln	0	0	275
17	make	9	0	440,261
18	mv	0	0	2,376
19	nm	1	0	73,523
20	objcopy	1	19	1,472,684
21	realpath	0	0	63
22	rm	0	0	383
23	sed	1	2	554
24	sh	0	0	1,002,280
25	size	0	0	75
26	sort	0	2	180
27	svnliteversion	11	6	265
28	touch	0	0	62
29	uudecode	0	0	9,187
30	xargs	0	0	54,891
	合計	567	44	22,361,056

数削減によりテキスト部の入出力時間を短縮でき，少なくとも 11 回分の PF 回数増加によりテキスト部の PF 処理時間が増加する．一方，表 3 (B) より，OS カーネル作成処理の場合，30 種類のプログラムが起動されるため，OFF2F プログラム実行時の PF 回数は，ELF プログラム実行時の PF 回数に比べ少なくとも 30 回多い．また，表 5 (B) より，ELF プログラムのテキスト部の PF 回数は 567 回である．したがって，OFF2F プログラムの実行時間について，567 回の入出力回数削減によりテキスト部の入出力時間を短縮でき，少なくとも 30 回分の PF 回数増加によりテキスト部の PF 処理時間が増加する．DK か SSD かに関係なく，増加するテキスト部の PF 処理時間は，ELF プログラムと OFF2F プログラムに差はない．一方，SSD の場合，短縮できるテキスト部の入出力時間は DK の場合に比べ短い．このため，増加する PF 処理時間の方が，短縮できるテキスト部の PF 処理時間より長くなる．

4. おわりに

OFF2F プログラムに関し，FreeBSD 上で実行環境を構築し，実行し，実測し，同様な計算機で行った ELF プログラムとの比較評価を述べた．

揮発性メモリのみで構成される計算機上で擬似 NV メモリを作成し，OFF2F プログラムの実行環境を構築した．また，ELF プログラムをもとに OFF2F プログラムを作成した．

評価により，プログラムを実行するための OS の制御処理時間（OS 制御時間）がプログラム実行時間に占める割合が大きいプログラムほど，実行プログラム形式を OFF2F にすることでプログラム実行を高速化できることを明らかにした．

具体的には，プログラムが行う処理を実行する時間（実質時間）が比較的短い場合について，テキスト部のサイズが異なるプログラムの実行時間を比較した．プログラム実行時のヘッダ部読み込み処理の違いにより，テキスト部のサイズが 64 KB より大きい場合に OFF2F プログラム実行の高速化を確認できた．たとえば，printf() で 13 文字を標準出力するプログラムをコンパイルする cc（テキスト部は約 42 MB）の場合，OFF2F プログラムの実行時間は ELF プログラムの実行時間の約 1/5 である．なお，実質時間が長く OS 制御時間の割合が少ない場合でも，実行プログラム形式を OFF2F にすることでプログラム実行を高速化できることを示した．具体的には，make により ls コマンドの実行ファイルを作成する場合，OFF2F プログラムの実行時間は ELF プログラムの実行時間の約 1/2 である．しかし，OS カーネル作成処理のようにプログラム実行時間が非常に長い（約 1,000 秒）場合，DK では 0.23%の時間短縮を確認できたが，SSD を用いて実入出力時間を短くすると時間短縮を確認できなかった．

謝辞 本研究の一部は，JSPS KAKENHI 21K11830，および共同研究（富士通株式会社富士通研究所）による．

参考文献

[1] Bailey, K., Ceze, L., Gribble, S.D. and Levy, H.M.: Operating System Implications of Fast, Cheap, Non-Volatile Memory, *Proc. 13th USENIX Conference on Hot Topics in Operating Systems, HotOS'13*, p.2 (2011).

[2] Xu, J. and Swanson, S.: NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories, *14th USENIX Conference on File and Storage Technologies* (*FAST 16*), pp.323–338 (2016).

[3] Kadekodi, R., Lee, S.K., Kashyap, S., Kim, T., Kolli, A. and Chidambaram, V.: SplitFS: Reducing Software Overhead in File Systems for Persistent Memory, *Proc. 27th ACM Symposium on Operating Systems Principles, SOSP '19*, pp.494–508 (2019).

[4] Zheng, S., Hoseinzadeh, M. and Swanson, S.: Ziggurat: A Tiered File System for Non-Volatile Main Memories and Disks, *17th USENIX Conference on File and Storage Technologies* (*FAST 19*), pp.207–219 (2019).

[5] Kim, J., Soh, Y.J., Izraelevitz, J., Zhao, J. and Swanson, S.: SubZero: Zero-Copy IO for Persistent Main Memory File Systems, *Proc. 11th ACM SIGOPS Asia-Pacific Workshop on Systems, APSys '20*, pp.1–8 (2020).

[6] Choi, J., Hong, J., Kwon, Y. and Han, H.: Libnvmmio: Reconstructing Software IO Path with Failure-Atomic Memory-Mapped Interface, *2020 USENIX Annual Technical Conference* (*USENIX ATC 20*), pp.1–16 (2020).

[7] Eisenman, A., Gardner, D., AbdelRahman, I., Axboe, J., Dong, S., Hazelwood, K., Petersen, C., Cidon, A. and Katti, S.: Reducing DRAM Footprint with NVM in Facebook, *Proc. 13th EuroSys Conference, EuroSys '18* (2018).

[8] Yao, T., Zhang, Y., Wan, J., Cui, Q., Tang, L., Jiang, H., Xie, C. and He, X.: MatrixKV: Reducing Write Stalls and Write Amplification in LSM-tree Based KV Stores with Matrix Container in NVM, *2020 USENIX Annual Technical Conference* (*USENIX ATC 20*), pp.17–31 (2020).

[9] George, J.S., Verma, M., Venkatasubramanian, R. and Subrahmanyam, P.: go-pmem: Native Support for Programming Persistent Memory in Go, *2020 USENIX Annual Technical Conference* (*USENIX ATC 20*), pp.859–872 (2020).

[10] Bittman, D., Alvaro, P., Mehra, P., Long, D.D.E. and Miller, E.L.: Twizzler: A Data-Centric OS for Non-Volatile Memory, *2020 USENIX Annual Technical Conference* (*USENIX ATC 20*), pp.65–80 (2020).

[11] Yang, J., Kim, J., Hoseinzadeh, M., Izraelevitz, J. and Swanson, S.: An Empirical Guide to the Behavior and Use of Scalable Persistent Memory, *18th USENIX Conference on File and Storage Technologies* (*FAST 20*), pp.169–182 (2020).

[12] Lantz, P., Dulloor, S., Kumar, S., Sankaran, R. and Jackson, J.: Yat: A Validation Framework for Persistent Memory Software, *2014 USENIX Annual Technical Conference* (*USENIX ATC 14*), pp.433–438 (2014).

[13] Dulloor, S.R., Kumar, S., Keshavamurthy, A., Lantz, P., Reddy, D., Sankaran, R. and Jackson, J.: System Software for Persistent Memory, *Proc. 9th European Conference on Computer Systems, EuroSys '14* (2014).

[14] Kannan, S., Gavrilovska, A. and Schwan, K.: pVM: Persistent Virtual Memory for Efficient Capacity Scaling and Object Storage, *Proc. 11th European Conference on Computer Systems, EuroSys '16* (2016).

[15] Koshiba, A., Hirofuchi, T., Akiyama, S., Takano, R. and Namiki, M.: Towards write-back aware software emulator for non-volatile memory, *2017 IEEE 6th Non-Volatile Memory Systems and Applications Symposium* (*NVMSA*), pp.1–6 (2017).

[16] Sato, M. and Taniguchi, H.: OFF2F: A New Object File Format for Virtual Memory Systems to Support Volatile/Non-Volatile Memory-Mixed Environment, *International Journal of Machine Learning and Computing*, Vol.9, No.4, pp.387–392 (2019).

[17] 谷口秀夫，佐藤将也，河辺誠弥，横山和俊：CoW 機能を考慮した OFF2F プログラムのページ例外処理の評価，情報処理学会論文誌，Vol.61, No.3, pp.707–717 (2020).

谷口 秀夫 （正会員）

1978 年九州大学工学部電子工学科卒業．1980 年同大学大学院修士課程修了．同年日本電信電話公社電気通信研究所入所．1987 年同所主任研究員．1988 年 NTT データ通信株式会社開発本部移籍．1992 年同本部主幹技師．1993 年九州大学工学部助教授．2003 年岡山大学工学部教授．2010 年同大学工学部長．2014 年同大学理事・副学長．2021 年同大学特命教授．博士（工学）．オペレーティングシステム，実時間処理，分散処理に興味を持つ．著書『並列分散処理』（コロナ社）等．電子情報通信学会，ACM 各会員．本会フェロー．

高杉 頌 （正会員）

2019 年岡山大学工学部情報系学科卒業．2021 年同大学大学院自然科学研究科博士前期課程修了．

額田 哲彰 （学生会員）

2020 年岡山大学工学部情報系学科卒業．同年同大学大学院自然科学研究科博士前期課程入学．オペレーティングシステムに興味を持つ．

佐藤 将也 （正会員）

2010 年岡山大学工学部情報工学科卒業．2012 年同大学大学院自然科学研究科博士前期課程修了．2014 年同大学同研究科博士後期課程修了．2013 年日本学術振興会特別研究員（DC2）．2014 年岡山大学大学院自然科学研究科助教．2021 年岡山県立大学情報工学部准教授．博士（工学）．コンピュータセキュリティ，仮想化技術に興味を持つ．2012 年度情報処理学会論文賞受賞．電子情報通信学会，ACM 各会員．