Vol.2020-DPS-185 No.9
2020/12/22

揮発性/不揮発性メモリ混載計算機において実行プログラムを分散配置するOFF2Fプログラムの性能評価

高杉 頌[1]　額田 哲彰[1]　佐藤 将也[1]　谷口 秀夫[1]

概要：揮発性メモリと不揮発性メモリが混載された計算機において，実行プログラムを不揮発性メモリと外部記憶装置に分散配置しプログラム実行を高速化する手法として，新たな実行ファイル形式（OFF2F）が提案された．本稿では，FreeBSD のプログラムを ELF と OFF2F で作成し，プログラム単体走行の場合と複数のプログラムが繰り返し実行される場合における OFF2F の有効性を述べる．また，プログラム単体走行の場合において複数のプログラムの実行時間を測定し，テキスト部のサイズが異なることによる OFF2F の効果の違いを示す．

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，オペレーティングシステム

1. はじめに

バイト単位でアクセス可能な不揮発性メモリ（Non-Volatile Memory：以降，NVM）が登場している．製品化されている NVM の 1 つに，Intel 社の Optane DC Persistent Memory（以降，DCPM） [1] がある．

NVM を揮発性メモリとストレージ間のメモリ階層として利用する研究として，揮発性メモリと NVM が混載する環境において性能向上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 NVM に配置するファイルシステム Ziggurat [4] がある．また，NVM を高速なストレージとして利用する研究として，メタデータを NVM に格納することでファイルシステムの性能を向上させる FSMAC [5]，大容量の NVM を搭載することにより揮発性メモリの総量を削減する Key-Value ストアの MyNVM [6] がある．さらに，NVM を活用した分散ファイルシステムである Orion [7] や，NVM 上のファイルの読み書きを複写レスで行う SubZero [8]，NVM のアクセス速度をエミュレートする研究 [9] がある．

今後の技術革新により，揮発性メモリと同等の読み込み速度を持つ NVM の登場が予想されるものの，書き込み速度が揮発性メモリと同等になることは困難である [10]．このため，将来の実メモリ構成は揮発性メモリと NVM が混載した計算機環境（以降，混載計算機）が主流になると考えられる．

文献 [10] では，揮発性メモリと同等の読み込み速度の NVM を搭載した混載計算機を対象に，NVM と外部記憶装置に実行プログラムを分散配置する新たな実行ファイル形式（OFF2F: Object File Format consisting of 2 Files）が提案されている．この形式では，仮想記憶機構が 2 つのメモリの特徴を生かしたプログラム実行を支援するため，実行プログラムを 2 つのファイルから構成している．具体的には，テキスト部を 1 つのファイルとし，テキスト部以外を別のファイルとする．テキスト部を NVM に配置し，仮想記憶機構を利用してそのまま仮想記憶にマッピングすることで，ページ例外処理時間を短縮している．また，文献 [11] では，FreeBSD 11.0-RELEASE（以降，FreeBSD）の OS 初期化処理に動作するプログラムを OFF2F で作成した場合の効果予測が行われている．さらに，読み込み速度が揮発性メモリと同等である擬似 NVM を構築し，OFF2F プログラムの効果を早期に検証する環境を実現した [12]．

本稿では，擬似 NVM を利用し，OFF2F の有効性を述べる．具体的には，FreeBSD のプログラムを ELF（Executable and Linkable Format）と OFF2F で作成し，両者を実行して実行時間を比較する．評価は，プログラム単体走行の場合と複数のプログラムが繰り返し実行される場合について行う．

[1] 岡山大学 大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

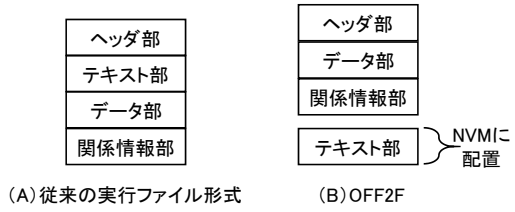


図 1　実行ファイル形式のファイル構成

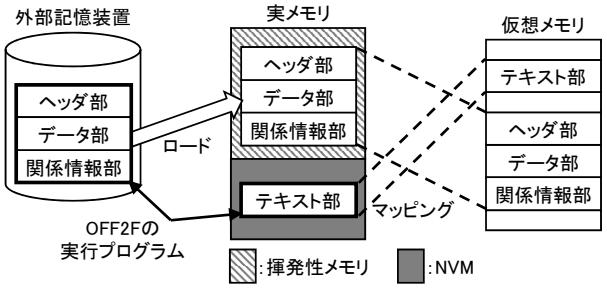


図 2　OFF2F プログラム実行時のマッピング

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

従来の計算機に搭載されている実メモリは揮発性であり，計算機をシャットダウンすると保持しているデータは消失する．一方で，計算機をシャットダウンした後においてもデータを継続して保持できる NVM が登場している．揮発性メモリと NVM を比較すると，現在 NVM として製品化されている Intel 社の DCPM は大容量であるが，ハードウェア構造やエネルギー効率の性質上，アクセス速度が低速である．

一方で，不揮発性という共通の特徴を持つ外部記憶装置と比較すると，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，NVM はバイト単位である．また，ハードディスクドライブ（HDD）やソリッドステートドライブ（SSD）といった外部記憶装置は，大容量かつ低価格であり，耐久性が高い．

今後の技術革新により，揮発性メモリと同等の読み込み速度を持つ NVM の登場が予想される．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 NVM を混載した構成になる．また，外部記憶装置は従来通り，ストレージとして活用されると考える．

2.2 従来の実行ファイル形式と OFF2F

ELF のような従来の実行ファイル形式は，以下の内容から構成されている．

- ヘッダ部：情報の格納位置などを保持
- テキスト部：手続き（命令）の羅列
- データ部：初期値を持つデータの格納領域
- 関係情報部：外部変数に関する情報

FreeBSD における代表的な実行ファイル形式には，ELF の他に a.out 形式，COFF（Common Object File Format）などがある．いずれの形式においても，上記の内容は 1 つのファイルに格納されている．そこで，実行プログラムの内容のアクセス形態に着目し，実行ファイルを NVM 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）が提案されている．従来の実行ファイル形式と OFF2F のファイル構成を**図 1** に示す．実行プログラムは 4 つの内容から構成されているため，最大 4 つのファイルに分割できる．しかし，OFF2F では，構造の複雑化を防ぎ，またファイルシステムでの占有領域を抑制するため，構成するファイルを 2 つにしている．

図 1 に示す通り，OFF2F の実行プログラムは，テキスト部を 1 つのファイルとし，ヘッダ部，データ部，および関係情報部を別ファイルとして，それぞれを NVM と外部記憶装置に分散配置する．これにより，ヘッダ部を含むファイルは外部記憶装置上に存在するため，プログラム実行時の仮想メモリ空間を作成する処理は既存の処理流れを利用できる．また，テキスト部のファイルは必ずしも NVM 上に存在する必要はないため，NVM の有無の影響を受けない．

2.3 OFF2F のプログラム実行

OFF2F プログラム実行時のマッピングの様子を**図 2** に示す．NVM には，OFF2F プログラムのテキスト部のファイルを配置する．OFF2F プログラム実行時，テキスト部でページ例外が発生した場合は，該当の NVM のページを割り当てる．

ELF と OFF2F のプログラムについて，プログラム実行時の処理流れを**図 3** に示す．ELF のプログラム実行時の処理流れについて，以下に説明する．

(1) fork() により，プロセスが生成される．
(2) ファイルの先頭 64 KB を読み出し，ヘッダ部の内容を獲得する．
(3) ヘッダ部内の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，仮想メモリ空間を作成する．
(4) テキスト部に記述されている命令を実行する．
(5) プログラム実行時にページ例外が発生した場合，フリーなページを確保する．
(6) 外部記憶装置から該当のファイルの内容を確保したページに読み出す．
(7) 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．

これに対し，OFF2F のプログラム実行時の処理流れのうち ELF のものと大きく異なるのは以下の 3 点である．

(2') ファイルの読み出しについて，ELF ではファイルの先頭 64 KB が読み出されるのに対し，OFF2F ではファ

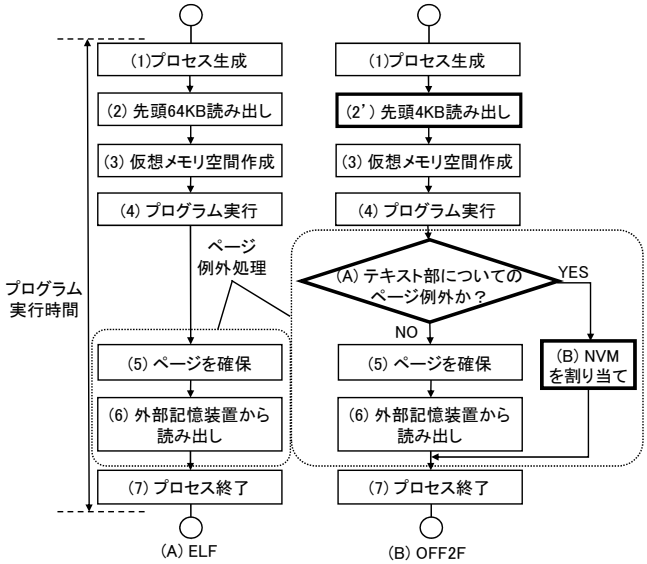


図 3 プログラム実行時の処理流れ

イルの先頭 4 KB のみを読み出す．この理由を以下に述べる．FreeBSD において，アクセスの形態が異なる内容は別ページで管理される．また，図 1 に示す通り，OFF2F の実行プログラムにおいて，ヘッダ部はデータ部と連続して存在する．ここで，ヘッダ部には読み込みのみ発生するのに対し，データ部には読み書きの両方が発生するため，これらは別ページで管理される．このため，OFF2F プログラム実行時は，ヘッダ部読み出しに必要な先頭 4 KB のみを読み出す．

(A) ページ例外処理において，テキスト部で発生したページ例外であるか確認する．

(B) テキスト部で発生したページ例外であれば，該当の NVM を割り当てる．

3. 評価

3.1 環境構築

3.1.1 擬似 NVM

混載計算機は身近に存在しないため，揮発性メモリのみで構成される既存計算機上で擬似 NVM を構築し，OFF2F プログラムの実行環境を作成する．擬似 NVM を実現する際には，以下の 3 つの要求を満たす必要がある．

(1) バイト単位アクセス可能であること
(2) 不揮発性を再現すること
(3) 擬似 NVM 上にテキスト部を配置し，仮想メモリ空間にマッピングして利用できること

これらの要求を満たすため，搭載されている揮発性メモリの一部をカーネルが使用しない領域とし，この領域を擬似 NVM とする．これにより，バイト単位アクセス可能であり，揮発性メモリと同等の読み込み速度を実現できる．また，擬似 NVM はカーネルの使用しない領域であるため，初期化処理では利用されない．この領域は電源を供給していればソフトウェアリセット時において失われることはないため，NVM の不揮発性を再現できる．なお，カーネルが使用する実メモリの大きさは，カーネルを構築する際の設定ファイルにより指定できる．ここで，擬似 NVM にテキスト部を配置するためには，擬似 NVM にアクセスできる必要がある．しかし，擬似 NVM はカーネルが使用しない領域であるため，既存の処理ではアクセスできない．そこで，擬似 NVM へアクセスできる機能を実現する．

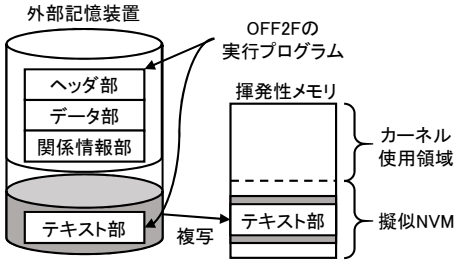


図 4 擬似 NVM における OFF2F プログラムのテキスト部の配置

3.1.2 擬似 NVM アクセス機能

擬似 NVM へのアクセスを可能にする機能の実現について，以下で説明する．仮想記憶機構では，仮想メモリ空間に実メモリをマッピングすることでアクセスできる．また，実メモリはページに分割して管理され，アクセスはページ単位で行われる．このため，擬似 NVM へのアクセスも同様にページ単位で行う．具体的には，擬似 NVM をページ単位に分割し，任意のタイミングで仮想メモリ空間にマッピングすれば，擬似 NVM にアクセスすることができる．この処理の実現において，擬似 NVM をページ単位に分割する処理は，既存処理を改変して利用し，擬似 NVM のページを仮想メモリ空間にマッピングする処理は，既存処理をそのまま利用する．これにより，擬似 NVM の任意の実アドレスを指定し，ユーザプログラムからアクセス可能にするシステムコール（nvm_mapping()）を作成した．

OFF2F プログラムのテキスト部を擬似 NVM に配置する様子を**図 4** に示す．nvm_mapping() を用いて，OFF2F プログラムのテキスト部を配置しているパーティションの内容を擬似 NVM に複写することで，擬似 NVM に OFF2F プログラムのテキスト部のファイルを配置する．これにより，擬似 NVM 上にファイルシステムを構築できる．プログラム実行時に擬似 NVM 上のファイルが必要になった場合，ファイルシステムと連携して擬似 NVM の該当ページを獲得し利用することができる．

3.2 観点

評価を以下の 2 つの観点で行う．

(1) プログラム単体走行の場合

実行時に単体で走行するプログラムについて，テキスト部のサイズが異なる 3 つのプログラムを ELF と OFF2F で作成し実行時間を比較する．これにより，テキスト部のサ

イズが異なることによる OFF2F の効果の違いを明らかにする．2.3 節で述べたように，ELF の場合において，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が 64 KB より小さいプログラム，テキスト部のサイズが 64 KB より大きいプログラム，およびテキスト部のサイズが 64 KB に比べ非常に大きいプログラムを評価対象とする．

(2) 複数のプログラムが繰り返し実行される場合
既存のソフトウェアは，複数のプログラムを繰り返し実行して動作する場合が多い．このため，複数のプログラムが繰り返し実行される処理で動作するプログラムを OFF2F で作成し，OFF2F の効果を確認する．

また，各評価において，発生するページ例外を分類し，テキスト部で発生するページ例外回数を明らかにする．さらに，OFF2F により短縮した実行時間の割合とテキスト部で発生するページ例外回数の割合を比較し，テキスト部のページ例外処理による実行時間への影響を明らかにする．

3.3 評価環境

ELF の場合の評価では，実行プログラムを外部記憶装置に配置して，実行時間を測定する．また，OFF2F の場合の評価では，テキスト部を擬似 NVM に配置し，もう 1 つのファイルを外部記憶装置に配置して，実行時間を測定する．ただし，プログラム実行時間を明確にするため，標準出力されるプログラムの出力先を null デバイスとする．なお，HDD 内部に搭載されているキャッシュによる評価結果への影響を受けないようにするため，評価対象のコマンドを実行する直前に 2 GB のデバイスファイルを読み出す処理を行う．

評価は，Intel Core i3-6100T（3.20 GHz）と 2 GB のメモリを搭載した計算機を用いた．なお，本評価では，OS が利用できるメモリの大きさを 1 GB とし，残りの 1 GB を擬似 NVM とした．OS は，FreeBSD 11.0-RELEASE を使用し，1 ページの大きさが 4 KB の環境とした．また，外部記憶装置として 500 GB の HDD（5,200 rpm）を利用した．

3.4 プログラム単体走行の場合

3.4.1 評価対象のプログラム

プログラムが単体で走行する場合の評価として，テキスト部のサイズが 64 KB より小さいプログラム，テキスト部のサイズが 64 KB より大きいプログラム，およびテキスト部のサイズが 64 KB に比べ非常に大きいプログラムの 3 つを評価対象とする．各プログラムについて，以下で説明する．

(a) テキスト部のサイズが 64 KB より小さいプログラム
date コマンドを評価対象とし，現在の日時を標準出力する処理の実行時間を測定する．

(b) テキスト部のサイズが 64 KB より大きいプログラム
less コマンドを評価対象とし，空のファイルの内容を標準出力する処理の実行時間を測定する．

(c) テキスト部のサイズが非常に大きいプログラム
cc コマンドを評価対象とし，文字列を画面に出力する簡単なプログラム（サイズは 73 B）をコンパイルする処理の実行時間を測定する．

(a)〜(c) の各プログラムの実行時に動作するプログラムのテキスト部のサイズ，データ部のサイズ，およびプログラム全体のサイズをそれぞれ**表 1**，**表 2**，および**表 3** に示す．date コマンドはテキスト部のサイズが 64 KB より小さく，実行時にテキスト部でページ例外は発生しない．一方で，less コマンドと cc コマンドはテキスト部のサイズが 64 KB より大きく，テキスト部でページ例外が発生する可能性がある．

表 1 date コマンド実行時に動作するプログラムのサイズ

通番	プログラム名	テキスト部 (KB)	データ部 (KB)	プログラム全体 (KB)
1	date	16.15	1.22	22.33

表 2 less コマンド実行時に動作するプログラムのサイズ

通番	プログラム名	テキスト部 (KB)	データ部 (KB)	プログラム全体 (KB)
1	less	138.83	12.67	153.16

表 3 cc コマンド実行時に動作するプログラムのサイズ

通番	プログラム名	テキスト部 (KB)	データ部 (KB)	プログラム全体 (KB)
1	cc	42,796.07	17.54	42,840.74
2	ld	1,562.45	20.11	1,598.61

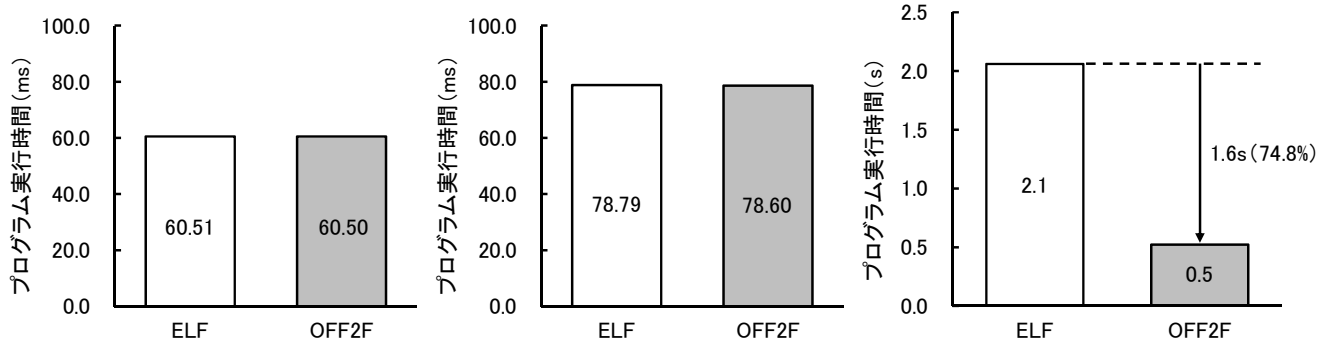
3.4.2 ページ例外の分類

実行ファイル形式が ELF の場合における，date コマンド，less コマンド，および cc コマンド実行時のページ例外を測定し，仮想アドレスからプログラムのテキスト部，ライブラリのテキスト部，およびそれ以外の 3 つに分類した．ページ例外を分類した結果をそれぞれ**表 4**，**表 5** および**表 6** に示す．表 4 に示す通り，date コマンドのプログラムのテキスト部のサイズは 64 KB より小さいため，実行時にはプログラムのテキスト部でページ例外は発生しない．一方で，表 5 に示す通り，less コマンド実行時にはプログラムのテキスト部で 1 回ページ例外が発生する．また，表 6 に示す通り，cc のプログラムのテキスト部で発生したページ例外は 408 回，ld のプログラムのテキスト部で発生したページ例外は 22 回である．

3.4.3 結果と考察

各プログラムの実行時間を**図 5** に示す．表 1〜表 6 および図 5 より，以下のことが分かる．

(1) 図 5 より，いずれのプログラムにおいても，ELF に比



(a) date コマンドの実行時間　(b) less コマンドの実行時間　(c) cc コマンドの実行時間

図 5　各プログラムの実行時間

表 4　date コマンド実行時のページ例外回数（ELF）

通番	プログラム名	ページ例外回数		
		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

表 5　less コマンド実行時のページ例外回数（ELF）

通番	プログラム名	ページ例外回数		
		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

表 6　cc コマンド実行時のページ例外回数（ELF）

通番	プログラム名	ページ例外回数		
		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	1,295
	合計	430	0	1,914

べ OFF2F では実行時間を短縮できる．これは，プログラムのテキスト部の実 I/O を削減したためである．

(2) 図 5 より，OFF2F により短縮した実行時間は，date コマンドで約 0.01 ms（0.1%未満），less コマンドで約 0.19 ms（0.24%），cc コマンドで約 1.6 s（約 74.8%）である．表 1〜表 3 より，テキスト部のサイズは cc コマンド，less コマンド，date コマンドの順に大きい．また，表 4〜表 6 より，テキスト部でのページ例外回数も同順で多い．OFF2F により，プログラムのテキスト部の実 I/O は削減できるため，プログラムのテキスト部でページ例外が最も多く発生する cc では，OFF2F の効果が最も大きくなる．

(3) 図 5(a) より，OFF2F により短縮した実行時間は約 0.01 ms（0.1%未満）であり，OFF2F による効果は小さい．この理由を以下に述べる．OFF2F では，テキスト部の実 I/O を削減でき，これにより実行時間を短縮できる．一方で，OFF2F では，exec() 時にファイルの先頭 4 KB のみを読み出すため，テキスト部でページ例外が発生する．OFF2F の場合，テキスト部で発生したページ例外において実 I/O は発生せず，擬似 NVM を割り当てる処理を行う．これに対し，ELF ではテキスト部でページ例外は発生しない．つまり，OFF2F ではテキスト部の実 I/O の削減により実行時間を短縮できる一方で，テキスト部のページ例外処理による擬似 NVM を割り当てる処理時間が発生する．このため，OFF2F の効果は小さくなったと考える．

(4) 表 6 と図 5(c) より，実行ファイル形式が ELF の場合，プログラムのテキスト部のページ例外処理時間は，それ以外のページ例外処理時間に比べ長い傾向にある．この理由を以下に述べる．図 5(c) より，ELF における cc コマンドの実行時間のうち，約 74.8%はプログラムのテキスト部の実 I/O 時間である．表 6 より，cc コマンド実行時に発生するページ例外のうち，プログラムのテキスト部で発生したページ例外回数は約 17.7%である．このことから，約 17.7%のページ例外の処理時間が実行時間の約 74.8%を占めていることが分かる．よって，プログラムのテキスト部のページ例外処理時間は，それ以外のページ例外処理時間に比べ長い傾向にあるといえる．これは，テキスト部のページ例外処理では実 I/O が発生している一方で，それ以外（データ部など）のページ例外処理では，Copy on Write により実 I/O が発生していない場合があるためと推察できる．

3.5　複数のプログラムが繰り返し実行される場合

3.5.1　評価対象のプログラム

複数のプログラムが繰り返し実行される処理として make コマンドを評価対象とし，ls コマンドの実行プログラムを作成する処理の実行時間を測定する．make コマンド実行時に動作するプログラムのテキスト部のサイズ，データ部のサイズ，およびプログラム全体のサイズを**表 7** に示す．表 7 に示す通り，make コマンド実行時には 11 個のプログラムが動作する．make コマンドにより ls の実行プログラムを作成するためのソースファイルは 9 個あり，合計サイズは約 80.1 KB である．

3.5.2　ページ例外の分類

実行ファイル形式が ELF の場合における，make コマンド実行時のページ例外を測定し，3.4.2 項と同様の方法で

表 7　make コマンド実行時に動作するプログラムのサイズ

通番	プログラム名	テキスト部 (KB)	データ部 (KB)	プログラム全体 (KB)
1	awk	188.70	3.45	192.88
2	cat	8.93	0.80	10.29
3	cc	42,796.07	17.54	42,840.74
4	chmod	5.17	0.68	6.41
5	gzip	33.34	1.50	35.58
6	ld	1,562.45	20.11	1,598.61
7	make	688.63	14.55	722.83
8	mv	10.42	0.88	11.86
9	objcopy	106.79	5.53	115.24
10	sed	32.67	1.08	34.54
11	sh	143.36	1.71	148.10

表 8　make コマンド実行時のページ例外回数（ELF）

通番	プログラム名	ページ例外回数		
		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

分類した．ページ例外を分類した結果を**表 8** に示す．表 8 に示す通り，プログラムのテキスト部で発生したページ例外回数は 509 回であり，全ページ例外回数は 12,665 回である．また，プログラムのテキスト部で発生するページ例外回数が最も多いプログラムは cc であり，474 回である．さらに，awk，objcopy，および sed については，ダイナミックリンクされるライブラリのテキスト部でページ例外が発生していた．なお，プログラムのテキスト部でページ例外が発生していないプログラム（cat，chmod，mv，sed，および sh）がある．これは，以下の 2 つの条件のいずれかを満たすためであると考える．

(1) exec() 時におけるファイルの先頭 64 KB 読み出しにより，実行時にアクセスするテキスト部が全て実メモリに読み出されるため
(2) make コマンド実行以前にすでに実メモリに読み出されているため

表 7 より，(1) にはプログラムのテキスト部のサイズが 64 KB より小さい cat，chmod，mv，sed が当てはまる．また，(2) には OS 起動直後に呼び出される sh が当てはまる．

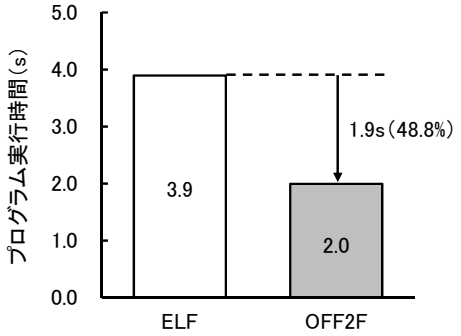


図 6　make コマンドの実行時間

3.5.3　結果と考察

make コマンドの実行時間を**図 6** に示す．表 3，表 6〜表 8，および図 6 より，以下のことが分かる．

(1) 図 6 より，OFF2F により実行時間を約 1.9 s 短縮でき，cc コマンド（約 1.6 s）に比べ実行時間を短縮できる．この理由を以下に述べる．表 3 と表 7 より，make コマンド実行時には cc コマンド実行時に比べ多くのプログラムが動作する．また，表 6 と表 8 より，make コマンド実行時にプログラムのテキスト部で発生するページ例外回数は，cc コマンド実行時よりも 79 回多い．このため，make コマンドでは cc コマンドよりもプログラムのテキスト部の実 I/O の発生回数が多く，cc コマンドに比べ実行時間を短縮できると考える．

(2) 図 6 より，OFF2F により短縮した実行時間の割合は約 48.8%であり，cc コマンド（約 74.8%）よりも小さい．これは，make コマンドは cc コマンドに比べプログラムのテキスト部の実 I/O の発生回数が多い一方で，それ以外の処理量が多いためであると考える．例えば，本評価において，cc コマンドにおけるコンパイル対象のソースファイルのサイズは 73 B であるのに対し，make コマンドにおけるコンパイル対象のソースファイルのサイズは約 80.1 KB である．すなわち，コンパイル対象となるソースファイルの実 I/O 時間は，make コマンドの方が長い．

(3) 表 6 と表 8 より，make コマンド実行時における cc のプログラムのテキスト部で発生するページ例外回数は，cc コマンド実行時のものよりも多い．すなわち，make コマンドのような複数のプログラムが繰り返し実行される処理において，cc のようにテキスト部が大きく，複数回実行されるプログラムはテキスト部の様々な箇所にアクセスする場合があると考えられる．このため，テキスト部のサイズが大きいプログラムにおいて，OFF2F の効果は大きくなると考える．

(4) 表 8 より，ライブラリのテキスト部でページ例外が 32 回発生している．このため，ダイナミックリンクされるライブラリを OFF2F で作成し，ライブラリのテキスト部を擬似 NVM に配置して実行することで，さらに実行時間を短縮できると考える．

4. おわりに

本稿では，プログラム単体走行の場合と複数のプログラムが繰り返し実行される場合の実行時間を ELF と OFF2F で比較し OFF2F の有効性を示した．

プログラム単体走行の場合において，テキスト部のサイズが異なることによる OFF2F の効果の違いを明らかにした．テキスト部のサイズが小さいプログラムでは OFF2F の効果が小さく，date コマンドでは実行時間を 0.1%未満（約 0.01 ms），less コマンドでは実行時間を約 0.24%短縮できることを示した．一方で，テキスト部のサイズが 64 KB に比べ非常に大きい cc コマンドでは，実行時間を約 74.8%（約 1.6 s）短縮できることを示した．また，複数のプログラムが繰り返し実行される make コマンドの実行時間を約 48.8%（約 1.9 s）短縮できることを示した．

なお，OFF2F の場合，テキスト部のページ例外処理時間はテキスト部以外の時に比べ実 I/O が発生しないため短い．一方，ELF の場合は逆である．これは，テキスト部のページ例外処理では実 I/O が発生する一方で，それ以外（データ部など）のページ例外では，Copy on Write により実 I/O が発生していない場合があるためと推察できる．

残された課題として，ダイナミックリンクされるライブラリのテキスト部を OFF2F で作成した場合の評価がある．

謝辞 本研究の一部は，JSPS KAKENHI 18K11244，および共同研究（株式会社富士通研究所）による．

参考文献

[1] 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).

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

[6] 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, Proceedings of the Thirteenth EuroSys Conference (EuroSys '18), no.42, pp.1–13 (2018).

[7] Yang, J., Izraelevitz, J., and Swanson, S.: Orion: A Distributed File System for Non-Volatile Main Memory and RDMA-Capable Networks, 17th USENIX Conference on File and Storage Technologies (FAST 19), pp.221–234 (2019).

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

[9] 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).

[10] 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).

[11] 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).

[12] Takasugi, S., Sato, M. and Taniguchi, H.: OFF2F Program Execution Using Pseudo Non-Volatile Memory, International Workshop on Networking, Computing, Systems, and Software (NCSS), vol.9, no.1, pp.12–18 (2020).