情報処理学会論文誌　コンピューティングシステム　Vol.6　No.4　49–57 (Oct. 2013)

Non-Volatileメインメモリを用いたチェックポイント・リスタートシステム

追川 修一[1,a)]　三木 聡[2,b)]

受付日 2013年4月8日, 採録日 2013年7月14日

概要：近年，プロセッサがバイト単位で直接アクセスでき，メインメモリとしても使用可能な不揮発性のnon-volatile（NV）メモリの実用化が進んでいる．NVメモリは不揮発性であることから，メインメモリとしてだけでなく，ストレージとしても使用可能である．本論文は，NVメインメモリを用いたチェックポイント・リスタートシステムを提案する．提案手法は，NVメモリをメインメモリとして用い，またNVメインメモリとファイルシステムが融合されたシステムを用いることを前提とすることで，1) チェックポイントに必要な実行状態の大部分を占めるメモリデータを，そのままチェックポイントファイルの一部とすることによる，チェックポイントの高速化，2) リスタート時にも，チェックポイントファイルのデータを，そのまま実行状態のメモリの一部とすることによる，リスタートの高速化を実現する．提案手法を実装したLinuxを実機上で実行する評価実験を行い，提案手法がチェックポイントおよびリスタートの高速化に有効であることを示す．

キーワード：オペレーティングシステム，不揮発性メモリ，チェックポイント・リスタート

Checkpoint and Restart System Utilizing Non-Volatile Main Memory

Shuichi Oikawa[1,a)]　Satoshi Miki[2,b)]

Received: April 8, 2013, Accepted: July 14, 2013

Abstract: Research and development on byte addressable non-volatile (NV) memory technologies that can be used for main memory are actively conducted. Because of its non-volatility, NV memory can be used for both main memory and storage devices. This paper proposes a checkpoint/restart system for NV main memory. The proposed system makes checkpointing and restart much more efficient by utilizing NV main memory and being based on the system that unifies NV main memory and storage. Making process data directly part of a checkpoint file enables faster checkpointing, and making a part of a checkpoint file directly process data enables faster restart. This paper shows the evaluation results of the proposed methods implemented in Linux by performing experiments.

Keywords: operating systems, non-volatile memory, checkpoint and restart

1. はじめに

近年，プロセッサがバイト単位で直接アクセスでき，メインメモリとしても使用可能な不揮発性のnon-volatile（NV）メモリの実用化が進んでいる．NVメモリは，データの保持に電力が不要であるため低消費電力化が可能であること，さらなる微細化が期待できることから，DRAMに置き換わる可能性を持つ．そのようなNVメモリとしては，MRAM，ReRAM，相変化メモリ（PCM: Phase Change Memory）などがある．なかでもPCMは，読み込みと比較して長い書き込みの遅延，および書き込み回数の制限という特性を持つため，これらの短所を克服するための技術が研究されてきた [1], [2], [3], [4], [5], [6].

[1] 筑波大学システム情報系情報工学域
University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan
[2] 株式会社フィックスターズ
FixStars Cooperation, Shinagawa, Tokyo 141–0032, Japan
[a)] shui@cs.tsukuba.ac.jp
[b)] miki@fixstars.com

このような NV メモリを利用することで，システムの信頼性を向上させる研究が行われ始めている [7], [8], [9]．これらの先行研究では，NV メモリをローカルストレージとして用い，プロセスの実行状態を記録するチェックポイントを保存する方法 [7]，NV メモリをメインメモリとし，プロセッサの状態は残存電源を用いて待避させることで，システム全体の実行状態を保持する方法 [8]，NV メインメモリ上でプロセスの実行状態をつねに一貫性を保つように実行することで，オペレーティングシステム（OS）の故障から独立させる方法 [9] が提案されている．

これらのうちチェックポイント・リスタートシステムは，ある一定の条件下でのプロセスの実行状態を記録し，その状態から実行を再開できることから，大規模な HPC システムからモーバイルコンピューティングシステムまで，様々な分散並列システムで用いられており，有効性が高い [10]．しかしながら，NV メモリを用いたチェックポイント・リスタートシステムの先行研究 [7] は，NV メモリを単なるローカルストレージとして使用しており，プロセッサがバイト単位で直接アクセス可能な NV メモリの特徴を活かしてはいない．

本論文は，NV メインメモリのためのチェックポイント・リスタートシステムを提案する．提案手法は，NV メモリをメインメモリとして用い，また NV メインメモリとファイルシステムを融合したシステム [11] を用いることを前提とすることで，以下を実現する．

- チェックポイントに必要な実行状態の大部分を占めるメモリデータを，そのままチェックポイントファイルの一部とすることによる，チェックポイントの高速化
- リスタート時には，チェックポイントファイルのデータを，そのまま実行状態のメモリの一部とすることによる，リスタートの高速化

提案手法を実装した Linux を，Intel x86 システムで実行する評価実験を行い，提案手法が有効であることを示す．なお本論文では，書き込み回数の制限，アクセス遅延の差は考慮しない．これは，NV メモリの 1 つである MRAM は，高速かつ書き込み回数の制限がなく DRAM の完全な置き換えが期待されているためであり，また，書き込み回数に制限があり，読み書きのアクセス遅延に差がある PCM でも，その制限や特性を隠蔽できることが分かっているからである [1], [2], [3], [4], [5], [6]．以下，本論文では，特に指定しない限り，NV メモリは，プロセッサがバイト単位で直接アクセスでき，メインメモリとして用いることのできる MRAM，ReRAM，PCM などを指すものとする．

本論文の構成は次のとおりである．2 章では背景について述べる．3 章では提案手法を述べ，4 章では Linux における具体的な実装方法について述べる．5 章では実験結果から提案手法が有効であることを示す．6 章で関連研究について述べ，7 章でまとめと今後の課題について述べる．

2. 背景

本論文は，NV メインメモリとファイルシステムを融合したシステムを対象とした，チェックポイント・リスタート（C/R: Checkpoint/Restart）システムを提案する．C/R システムとは，ある時点でのプロセスの実行状態（チェックポイント）をファイルに保存し，保存したチェックポイントから実行を再開（リスタート）することを可能にするシステムである．提案手法は，NV メインメモリとファイルシステムが融合されていることを利用し，チェックポイントおよびリスタートを高速化する．

2.1 対象とするシステム構成

提案手法は，NV メインメモリとファイルシステムを融合したシステムを対象とする [11]．NV メモリは，メインメモリ，ストレージのどちらとしても使用可能であるため，その両方を 1 つに融合できる．融合することにより，メインメモリ，ストレージと領域を分けて使用する必要がなくなり，単一システムで直接扱えるメモリ領域が増加するという利点がある．

図 1 に，本論文が対象とする NV メインメモリとファイルシステムを融合したシステムを示す．NV メモリを分割することなく，メインメモリとファイルシステムの両方に使用できるようにするため，NV メモリの領域はファイルシステムが管理する．NV メモリ上にファイルを格納する場合，NV メモリ上のファイルシステムは，通常のファイルシステムと同様に，ファイルのデータを格納するためのブロックを割り当てる．ファイルに格納されたプログラムテキストおよびデータは，XIP（eXecution In Place）により，メインメモリとして直接参照することができる．それ以外に，プロセスやカーネルが空きメモリ領域を必要とする場合は，ファイルシステムから空きブロックを取得し，メモリ割当てをすることができる．図 1 のプロセスは，ファイルのテキストを XIP によりマップし，ヒープに必要な領域は空きブロックを取得しマップしている．

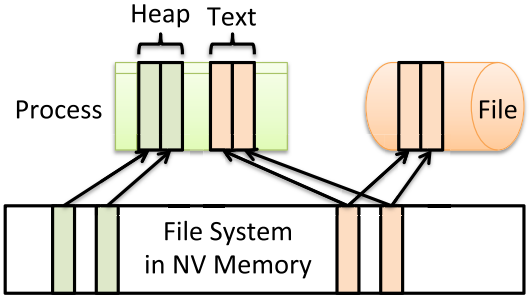


図 1　NV メインメモリとファイルシステムが融合されたシステム
Fig. 1　A system that unifies NV main memory and a file system.

2.2　既存の C/R システムの問題点

既存の C/R システムを，本論文で対象とするシステムに適用した場合の，問題点について述べる．そして，その問題点を解決するために，必要となる要件をまとめる．

C/R システムは，ある時点でのプロセスの実行状態を記録するチェックポイントを，ファイルとして作成する．チェックポイントファイルには，プロセスの実行を再開するために必要な情報として，レジスタの内容，プロセスのメモリマップとその内容，ユーザ ID やファイルディスクリプタなどのプロセスの属性が含まれる．通常のシステムでは，メモリは揮発性の DRAM である．そのため，メモリ上のデータを書き出したチェックポイントファイルは，ストレージ上に作成する．システムを安定化するためのソフトウェア若化（Software Rejuvenation）[12] やシステムクラッシュによる再起動後に，チェックポイントファイルの内容をメモリに読み込み，プロセスを再開することができる．

ところが，本論文で対象とするシステムは，NV メインメモリ上にファイルシステムを構築し，ファイルシステムとメインメモリを融合させている．そこで，チェックポイントファイルをファイルシステム上に作成するということは，プロセスに割り当てられた NV メモリ領域の内容を，同じ NV メモリ上に構築されたファイルシステム上にコピーし作成することになる．**図 2** に，既存 C/R システムによるチェックポイントファイル作成を示す．

既存 C/R システムは，同じ内容のデータを同じ NV メモリ上で複製するため，次の問題点が生じる．

- チェックポイント作成の処理コスト：対象システムでは，チェックポイントファイルの内容となるデータが，ファイルシステムを構築した NV メモリ上に存在する．同じ NV メモリ上でその複製を作成するのは，無駄な処理コストが発生していることになる．
- 空きメモリ領域の減少または不足：対象システムは，ファイルシステムとメインメモリを融合させているため，ファイルシステム上にファイルを作成すれば，メインメモリとして使用可能な領域も減少してしまう．同一データが，同じ NV メモリ上にあるにもかかわらず，その複製を作成するのは，領域を無駄に消費していることになる．また，アプリケーションが大量にメモリを消費し，ファイルシステムが使用可能な領域も少なくなっている場合は，チェックポイント作成に失敗することになる．

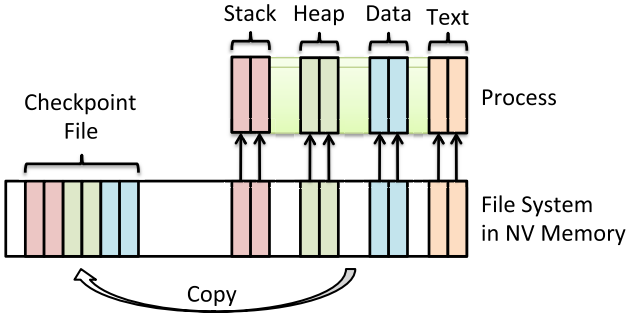


図 2　既存 C/R システムによるチェックポイントファイル作成
Fig. 2　Checkpointing by an existing C/R system.

上記の問題点は，チェックポイントファイルからの再開についても，同様にあてはまる．チェックポイントファイルから，実行を再開するためのプロセスへデータを読み込むときに，同一データの複製を作成することになる．そのため，データの複製のための処理コストが必要となり，またメモリ領域も消費する．

既存 C/R システムの問題点を解決するためには，チェックポイントファイル作成にあたり，データ複製を回避する必要がある．NV メモリ上でのデータの複製を行わなければ，複製のための処理コストは不要となり，メモリ領域も消費しない．チェックポイントファイル作成のために，十分な空き容量を確保しておく必要もなくなるため，アプリケーションの実行に十分なメモリ領域を割り当てることができ，実行効率の向上にもつながる．

3.　提案手法

本章では，対象システムの特徴を活かし，既存システムの問題点を解決するため，チェックポイントおよびリスタート時におけるデータ複製を回避する C/R 手法を提案する．

3.1　チェックポイント

対象システムは，NV メインメモリとファイルシステムを融合させている．そこでは，プロセスへのメモリ割当て要求に対し，ファイルシステムは未使用ブロックを確保し，割り当てる．カーネルはプロセスへのメモリ割当てをページ単位で行うため，ファイルシステムのブロックサイズとページサイズを同一にすることで，確保されたブロックをページとして割り当てることが可能である．プロセスへのメモリ割当てのために確保したブロックへの参照は，どのファイルからもない状態となっている．しかしながら，メモリ割当てのために確保したブロックは，ファイルシステムを構成するブロックでもあるため，そのブロックへの参照をファイルに追加することで，ファイルの一部とすることができる．

そこで提案手法は，チェックポイント時に，対象プロセスのユーザレベルにおけるデータを格納したページをファイルシステムのブロックとして扱い，そのブロックへの参照をチェックポイントファイルに直接追加する．保存すべきページが，そのままチェックポイントファイルの一部となるため，データの複製を回避することができ，またそのための処理コストも不要となる．

提案手法では，チェックポイント時に，対象プロセスお

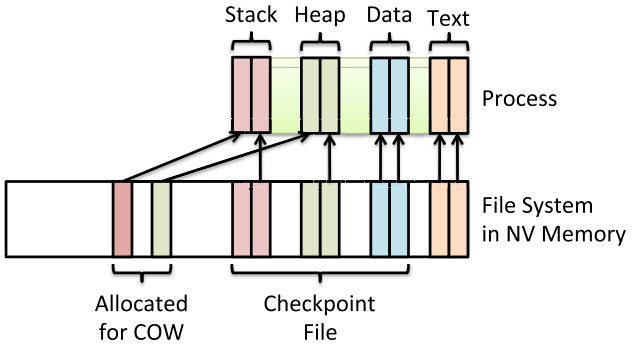


図 3　提案手法によるチェックポイントファイル作成
Fig. 3　Checkpointing by the proposed C/R system.

よびチェックポイントファイルの両方が，対象プロセスのデータを格納したページを参照する状態になる．チェックポイント終了直後に，対象プロセスの実行が再開することなく終了する場合は，チェックポイントファイルの内容の整合性は保たれる．しかしながら，対象プロセスの実行を再開し，チェックポイントファイルに追加したページに対する書き込みが発生した場合，チェックポイントとしては不整合な状態になってしまう．そのため，チェックポイント時に，対象プロセスのデータを格納したページは書き込み不可の設定とする．そして，実行再開後に書き込みが発生した場合，新たなページを割り当てる Copy-On-Write（COW）を行うことで，チェックポイントファイルの整合性を維持することができる．

図 3 に，提案手法によるチェックポイントファイル作成を図示する．プロセスにマップされた領域のうち，プログラムのテキスト領域は，もともとファイルに含まれ，またプロセスがその実行中に書き込むことはないため，チェックポイントファイルには含まれない．また，データとしてアクセスするためプロセスにマップされたファイルのうち，書き込みが行われていない，すなわちプロセス固有のデータを持たない領域も，チェックポイントファイルには含まれない．それ以外の，データ，ヒープ，スタック領域としてプロセスに割り当てられたページは，そのままチェックポイントファイルの一部になる．チェックポイント後，実行を再開し，書き込みが発生した場合，COW を行い，新たなページを割り当てる．そのため，チェックポイント時の内容がチェックポイントファイルに残ることになる．

3.2　リスタート

提案手法におけるチェックポイントからのリスタートは，基本的にはチェックポイントと逆の操作を行う．チェックポイントファイルに含まれる，リスタートするプロセスのユーザレベルにおけるデータを格納したブロックを，ページとして対象プロセスにマップする．マップされたファイルのブロックは，プロセスが直接参照することができるため，新たなページ割当ておよびデータの複製を回避することができ，処理コストおよび必要なメモリ領域を削減できる．

リスタート後の，チェックポイントファイルの整合性維持についても，チェックポイントにおける対応と同様である．リスタートが 1 度のみ行われる場合，チェックポイントファイルの整合性を維持する必要はないため，チェックポイントファイルのブロックを書き込み可能の設定でマップできる．リスタートが複数回行われる可能性がある場合，整合性を維持する必要があるため，チェックポイントファイルのブロックは，書き込み不可の設定でマップし，書き込み発生時に COW を行うようにする．この場合，図 3 はリスタートにもそのままあてはまることになる．

3.3　利用方法

これまでの C/R の一般的な利用方法では，チェックポイント後の実行継続，および複数回のリスタートが想定されるため，提案方式では実行時に COW をともなうことになる．提案方式が COW を行うのは，書き込みが発生したページに対してのみである．しかしながら既存方式は，チェックポイントファイルに含まれるすべてのページを複製する．また，チェックポイント時にページの複製をともなわないため，より頻繁にチェックポイントを行えるという利点がある．COW が発生するページは，本質的には差分チェックポイントが保存するデータと同じである．そのため，多くのページに書き込みを行うアプリケーションでは，それらのページに対する複製処理が必要となる．差分チェックポイントとの違いについては，6 章で述べる．

提案手法の効果的な利用方法の 1 つとして，OS のソフトウェア若化のために再起動を行い，アプリケーションを再開する場合があげられる．この利用方法では，再起動前にアプリケーションのチェックポイントを作成し，作成直後にアプリケーションを終了する．そして，OS 再起動後に，チェックポイントから再開する．この場合，チェックポイント時には，プロセスに割り当てられたページはそのままチェックポイントファイルの一部になり，リスタート時には，チェックポイントファイルの必要なページをそのままプロセスに割り当てることになり，データ複製の必要性はまったく必要なくなる．

4.　実装

提案手法の Linux カーネルにおける実装について述べる．Linux カーネルのメインラインには C/R システムは含まれないため，BLCR [13] を用いた．実装に用いたバージョンは，Linux 3.4，BLCR 0.8.5 である．また，NV メインメモリとの融合に用いているファイルシステムは PRAMFS [14] である．

BLCR は，カーネルモジュールにより，チェックポイント時には対象プロセスの情報を取得，リスタート時には対象プロセスの再構成を行う．チェックポイントファイルの

読み書きは，カーネルモジュールが直接行う．提案手法の実装には，カーネルモジュールにおいて，チェックポイントファイルに対しプロセスのデータを保存・読み出し処理を行う部分への変更，および COW によるチェックポイントファイルの保護が必要であった．以下，まず BLCR カーネルモジュールにおける処理の概要について述べた後，提案手法の実装について述べる．

4.1 BLCR カーネルモジュールにおける処理の概要

BLCR カーネルモジュールが C/R を実現するにあたり，プロセスデータの保存・読み出し処理に至るまでの，処理の概要について述べる．C/R を利用するプロセスは，BLCR カーネルモジュールを操作する必要があるが，そのために `/proc/checkpoint/ctrl` ファイルに対し ioctl システムコールを発行する．ioctl システムコールを通して，BLCR カーネルモジュールの `ctrl_ioctl()` を呼び出し，操作する．**図 4** に，本論文に関連する部分の BLCR カーネルモジュールの内部構造を示す[*1]．

チェックポイント時には，チェックポイント対象のプロセスは，ioctl コマンド `CR_OP_HAND_CHKPT` を用い，BLCR カーネルモジュールの `cr_dump_self()` を呼び出す．`cr_dump_self()` は，`cr_do_dump()` を経て，`cr_do_vmadump()` を呼び出す．`cr_do_vmadump()` は，プロセスにおける保存すべきデータを判別しチェックポイントファイルに対し書き込みを行うための，様々な関数を呼び出す．その関数の 1 つが `cr_freeze_threads()` であり，そこからさらにいくつかの関数を経て，`store_page_chunks()` を呼び出す．`store_page_chunks()` は，プロセスのユーザレベルにおけるデータを，ファイルに書き出す．

リスタート時には，リスタートを要求したプロセスは，ioctl コマンド `CR_OP_RSTRT_PROCS` を用い，BLCR カーネルモジュールの `cr_rstrt_procs()` を呼び出す．`cr_rstrt_procs()` は子プロセスを作成し，その子プロセスが保存したチェックポイントファイルを読み込み，プロセスを再構成する．子プロセスは再構成のために，ioctl コマンド `CR_OP_RSTRT_CHILD` を用い，BLCR カーネルモジュールの `cr_rstrt_child()` を呼び出す．`cr_rstrt_child()` は，チェックポイントファイルから情報を読み出し，プロセスを再構成するための，様々な関数を呼び出す．その関数の 1 つが `cr_thaw_threads()` であり，そこからさらにいくつかの関数を経て，`load_page_chunks()` を呼び出す．`load_page_chunks()` は，プロセスのユーザレベルにおけるデータを，ファイルから読み出す．

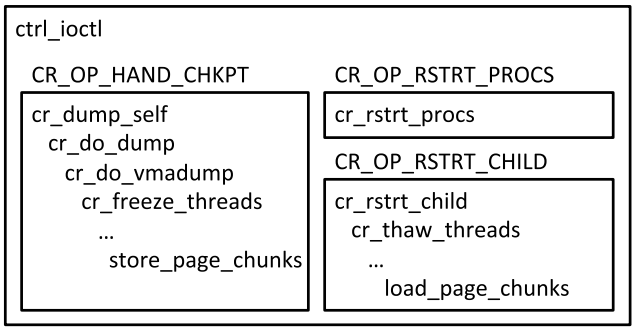


図 4　BLCR カーネルモジュールの内部構造
Fig. 4　Internal structure of BLCR kernel module.

[*1] 図 4 は説明に必要な部分のみを示したものであり，実際は他に多くの ioctl コマンドがある．

4.2 プロセスデータの保存・読み出し処理への変更

提案手法を実装するにあたり，変更が必要なのは，ユーザレベルにおけるデータを扱う部分のみである．提案手法が対象とするユーザレベルにおけるデータ以外の，カーネル内で管理されているプロセスの情報や CPU 上のレジスタの内容の保存・読み出し処理は変更せず，既存の手法をそのまま用いる．既存手法によるデータの読み出し・書き込み処理は，ファイルに対する I/O API である read, write を用いる．提案手法と既存手法が混在することになるが，読み出し・書き込み先のオフセットは，カーネル内の `file` 構造体の書き込み先オフセット `f_pos` が一元管理するため，問題は生じない．

ユーザレベルのデータは，`store_page_chunks()` が，ページ単位でチェックポイントファイルに書き込み，また `load_page_chunks()` が，ページ単位で読み出す．オリジナルではデータをコピーすることを前提としているため，書き込み先のオフセットはページにアラインされてはいない．提案手法では，チェックポイント時には，データを格納したページをチェックポイントファイルに直接追加するため，オフセットがページアラインされることになる．リスタート時には，ページアラインされたデータが格納されたページを対象プロセスにマップすることになる．

チェックポイント時に，データを格納したページをチェックポイントファイルに直接追加するため，以下の関数を追加した．

```
int xip_file_add_pages(struct file *filp,
    loff_t pgoff, int nr, struct page *pages[]);
```

この関数は，`filp` により指定されるチェックポイントファイルの，ページオフセット `pgoff` の位置からページを追加する．追加するページは `pages`，ページ数は `nr` により指定される．追加後，`file` 構造体の書き込み先オフセット `f_pos` は，追加したページ数だけ前に進むことになる．

リスタート時に，データが格納されたページを，対象プロセスにマップする方法には，2 通りある．1 つは mmap を用いる方法であり，もう 1 つはファイルからページを直接取得する方法である．どちらの方法でも，書き込み可能の設定，または書き込み不可の設定でマップすることができる．mmap を用いる方法では，マップを設定する処理

は，カーネルの仮想メモリシステムにおいて，仮想アドレスとファイルのオフセットへの対応を記録するのみであり，ページテーブルは構築しない．そのため，マップの設定処理コストは小さくてすむため，リスタート処理は高速に行える．しかしながら，実行再開後にマップしたページへのアクセス時にページフォルトが発生し，必要なページテーブルを構築することになる．

ファイルからページを直接取得するために，以下の関数を追加した．

```
int xip_file_get_page(struct file *filp,
    unsigned long pgoff, unsigned long addr);
```

この関数は，filp により指定されるチェックポイントファイルの，ページオフセット pgoff の位置にあるページを，カレントプロセスのユーザレベルのアドレス addr にマップする．mmap と異なり，この関数はページテーブルを実際に更新するため，実行開始後のマップしたページへのアクセス時にページフォルトが発生することはない．しかしながら，mmap を用いる場合より，リスタート処理のコストは増加することになる．

4.3　COW によるチェックポイントファイルの保護

前述の xip_file_add_pages()，xip_file_get_page() を用い，ページをチェックポイントファイルに直接追加，またはファイルからページを直接取得する場合，ファイルとプロセスでページを共有することになる．このとき，チェックポイントファイルの整合性を維持するためには，プロセスによる書き込みからチェックポイントファイルを保護する必要がある．そのためには，プロセスによる書き込みが起こった際に，新たに確保したページに書き込みを行う COW を行う必要がある．Linux カーネルは，その仮想メモリ管理機構に COW 機構を実装しているため，基本的にはその COW 機構が動作するようにページの設定を行う．また，一部，以下に述べる変更を行った．

あるページに対し COW を行うには，そのページを参照するページテーブルのエントリを書き込み不可に設定する．さらに，xip_file_add_pages() によりチェックポイントファイルに追加するページは，基本的には無名ページ（anonymous page）であるため，追加の設定が必要になる．無名ページの場合，そのページをマップしているプロセスが 1 つだけの場合，書き込み不可に設定したとしても，ページフォルトハンドラは COW は不要と判断し，書き込み可能に再度設定してしまう．そこで，ページの属性に，ファイルに追加されたことを示す PG_nvmemory_file を追加する．そして，この属性がついている場合は COW を行うように，ページフォルトハンドラを変更した．xip_file_get_page() により取得されるページは，ファイルからマップされるものであるため，追加の設定は不要である．

上記の設定を行い，COW の対象となったページに対する処理の流れは，以下のとおりである．ページテーブルのエントリを書き込み不可に設定しているため，プロセスから対象ページへの書き込みに対し，ページフォルトが発生し，ページフォルトハンドラが呼び出される．ページフォルトハンドラは，ページフォルトの発生原因を解析し，原因に応じた処理関数を呼び出す．書き込み不可のページに対する書き込みに起因するページフォルトの場合，ページフォルトハンドラはその処理関数として do_wp_page() を呼び出す．do_wp_page() は，対象ページが無名ページの場合，そのページに対し COW が必要かどうかの判断を行う．ここで，対象ページにはファイルに追加されたことを示す属性 PG_nvmemory_file が付いているため，COW が必要と判断する．COW を行うため do_wp_page() は，新たなページを確保，対象ページの内容をコピー，確保したページをページテーブルに追加する．ページフォルト処理の終了後，書き込みは再実行され，プロセスは新たに確保されたページに書き込みを行う．したがって，チェックポイントファイルのページは変更されない．

上記の処理のうち提案手法を実装するための変更部分は，do_wp_page() が，ファイルに追加されたことを示す属性が付いているページは COW が必要と判断するようにする部分だけである．

4.4　ファイルシステムにおける参照カウントの管理

提案手法により，ページをチェックポイントファイルに直接追加，またはファイルからページを直接取得する場合，ファイルとプロセスがページを共有することになる．ファイルの削除またはプロセスの終了時，まだどちらかが使用中のページを解放しないようにする必要がある．また，チェックポイントを複数回行った場合，複数のチェックポイントファイルを作成するが，それらは変更のなかった同一ページを共有する．この場合，1 つのチェックポイントファイルを削除しても，共有ページを解放しないようにする必要がある．

このような共有ページを管理するため，物理メモリ管理におけるページの参照カウントの管理とは別に，ファイルシステムにおけるページの参照カウントを導入した．ファイルシステムにおけるページの参照カウントは，ファイルからの参照と，プロセスが物理メモリの一部として使用する場合の参照をカウントする．しかし，物理メモリの一部として使用する場合の参照は，複数プロセスからの参照がある場合であっても，1 回とカウントする．これは，プロセスからの参照は，物理メモリ管理におけるページの参照カウントにより管理されるためである．

実装対象である PRAMFS でページの参照カウントを記録するため，unsigned char 型の配列を配置した．PRAMFS は，ページ使用状況の管理のためにビットマップの領域を確保していたが，その領域を拡張し，参照カウントの配列

を配置した．参照カウントをインクリメント・デクリメントする関数は，既存のページの割当て・解放を行う関数，提案手法のページの直接追加・取得を行う関数に追加した．

提案手法によるチェックポイントは，ページをチェックポイントファイルに直接追加し，ファイルとプロセスがページを共有する．プロセスが終了すると，共有ページの物理メモリ管理における参照カウントが 0 になり，ファイルシステムはその NV メモリページの解放要求を受け付ける．そこで，ファイルシステムにおける該当ページの参照カウントをデクリメントするが，まだファイルからの参照が残っているため，解放されない．チェックポイントファイルも削除すると，ファイルシステムの参照カウントが 0 になり，ファイルシステム内でそのページを解放し，再利用の対象となる．チェックポイントを複数回行い，複数のチェックポイントファイルを作成した場合は，プロセスが終了し，すべてのチェックポイントファイルを削除したとき，すべてに共有されていたページのファイルシステムにおける参照カウントが 0 になり，そのページを解放する．

上記のファイルシステムにおけるページの参照カウントは，各ページに対して管理される．現状では NV メモリからは 4 KB の単一ページ単位での割当てのみ行っており，連続ページの割当てには対応していない．そのため，メモリの断片化に対する対応は現状では行っていない．

5. 評価

提案方式を評価するために，チェックポイント時におけるプロセスデータの保存・読み出し処理コストを計測し，比較する．

5.1 実験環境

NV メモリをメインメモリとして持つシステムは，一般的に入手可能な状態ではないため，実験環境として用いることができない．本論文では，アクセス遅延の差，書き込み回数の制限は考慮しないため，DRAM の一部を NV メモリと見なすことで実機上で実験を行った．制限を考慮しない理由は，1 章で述べたとおり，制限がない NV メモリも開発されており，また制限がある場合も，これらの制限を隠蔽する技術が開発されているためである．

実機では，DRAM の一部を NV メモリと見なすため，8 GB の DRAM を搭載し，後半 4 GB の領域を NV メモリとして提案手法の管理下におくようにした．カーネルが認識する DRAM 領域を制限するために，カーネルに `mem=128M` オプションを渡し，DRAM としては 128 MB だけ使用する状態で実験を行った．実験に用いた実機は，Intel Atom D2700 2.13 GHz および Core i7-3770 3.4 GHz である．どちらも，ハイパースレッディング機能は BIOS で無効化し，1 CPU の状態で計測した．実行時間は，RDTSC 命令により計測した．

5.2 プロセスデータの保存・読み出し処理コストの計測

プロセスデータの保存・読み出し処理コストを計測するため，引数で指定された容量のメモリ領域を確保し，その領域の各ページの先頭部分に対し書き込みを行った後に，チェックポイント取得のためのコマンド（cr_checkpoint）を起動するプログラムを作成し，実行した．プロセスデータの保存・読み出し処理コストは，それぞれ，BLCR モジュールに含まれる関数 `cr_dump_self()` および `cr_rstrt_procs()` の最初と最後で TSC の値を取得することで計測した．BLCR によるチェックポイントおよびリスタートは，複数プロセスの連携より行われるため，それぞれの関数について，最初に呼び出されたときの TSC 値と，最後に関数から戻るときの TSC 値の差分を，処理コストとした．また，どちらも COW を行うための処理を含む．

実験結果を，**図 5**, **図 6** に示す．横軸は割当てメモリサイズ（MB），縦軸は TSC 値の差分をクロック値で割ることで得た実行時間（ミリ秒）である．図中の UIO は既存のファイルに対する I/O API である read，write を用いた場合の結果，また NVMM は提案手法であるファイルのページを直接追加・取得した場合の結果，NVMM mmap はファイルのページを mmap によりマップした場合の結果である．

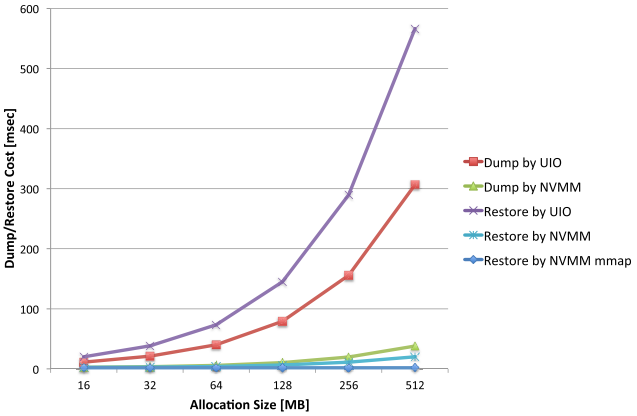


図 5　Intel Atom 上の実験結果
Fig. 5　Experiment result on Intel Atom.

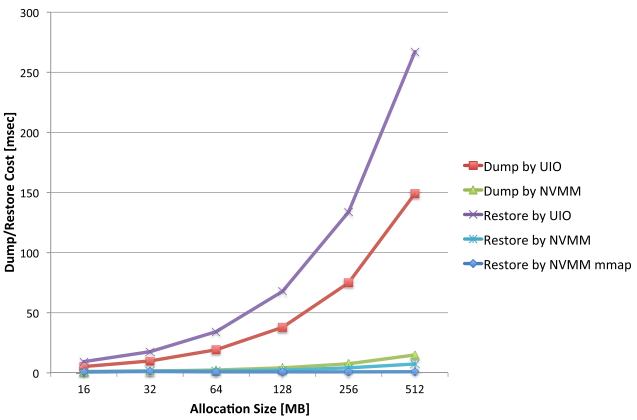


図 6　Intel Core-i7 上の実験結果
Fig. 6　Experiment result on Intel Core-i7.

実験結果から，提案手法により，大幅に処理コストを削減できていることが分かる．プロセスデータの保存処理については，512 MB の領域をプロセスが割り当てた場合において，従来方式と比較して，Atom において 12.4%，Core i7 において 10.0%と，約 8〜10 分の 1 のコストで処理可能となった．プロセスデータの読み出し処理については，ファイルからページを直接取得する方法を用いた場合で，従来方式と比較して，実機では Atom において 3.5%，Core i7 において 2.7%と，約 28〜36 分の 1 と大幅に低いコストで処理可能となった．

プロセスデータの読み出し処理において，mmap を用いる方法は，ほぼ一定のコストで処理できている．これは，mmap は仮想アドレスとファイルのオフセット対応を設定するのみであり，ページテーブルの構築や読み出し処理は実行再開後，データがアクセスされた時点で行うためである．アクセス時のオンデマンド処理にコストを移しているだけといえるが，見かけ上のリスタートコスト削減には有効であるといえる．

6. 関連研究

NV メモリを利用することで，システムの信頼性を向上させる研究としては，NV メモリをローカルストレージとして用い，プロセスの実行状態を記録するチェックポイントを保存する方法 [7]，NV メモリをメインメモリとし，プロセッサの状態は残存電源を用いて待避させることで，システム全体の実行状態を保持する方法 [8]，NV メインメモリ上で，プロセスの実行状態をつねに一貫性を保つように実行することで，オペレーティングシステム（OS）の故障から独立させる方法 [9] が提案されている．いずれも，プロセスに割り当てられたデータをそのままチェックポイントファイルの一部とする提案手法とは異なっている．

BLCR [13] は，プロセスに割り当てられたデータすべてを記録する，フルチェックポイントを行う．そのためフルチェックポイントは，チェックポイントファイルに書き出すデータ量が大きくなり，チェックポイントに時間がかかるという欠点がある．そこで，実行中に定期的にチェックポイントを行う場合，前回からの差分のみをチェックポイントファイルに書き出す，差分チェックポイントが提案された [15], [16], [17], [18]．差分チェックポイントは，差分となる変更されたページを見つける必要がある．そのため，ページを書き込み禁止とし，ページへの書き込みを例外として受け取ることで，変更されたページを記録していくという点では，どの差分チェックポイントシステムも同じである[*2]．一方，提案手法では COW を使用している．ページへの書き込みを例外として受け取る点では差分チェックポイントと同じであるが，この時点でページのコピーを作成することで，チェックポイントファイルの内容は維持する点が異なる．また，チェックポイント時にオーバヘッドなしに，フルチェックポイントファイルを残せる点は，既存のフルチェックポイントおよび差分チェックポイントと比較して，有利な点である．

仮想記憶システムにおける COW は，Mach [19] で導入され，その後，効率的なメモリ共有方法として一般的になった．しかしながら，提案手法のように，ファイルシステムと融合されたメインメモリ上でのチェックポイントに適用された例はない．また，ファイルシステムやストレージボリュームのスナップショットでの COW は，WAFL [20] や現在の多くの OS では論理ボリュームマネージャで用いられている．これらはディスクブロックに対しての COW であり，プロセスの実行イメージとチェックポイントファイルを同一化した提案手法での COW とは異なっている．

[*2] ページテーブルエントリに含まれるダーティビットを用いる方法も提案されているが，OS カーネルが仮想記憶システム管理のために使用されており，別の用途に用いることができない場合も多い [21]．

7. まとめ

本論文は，NV メインメモリのためのチェックポイント・リスタートシステムを提案した．提案手法は，NV メモリをメインメモリとして用い，また NV メインメモリとファイルシステムが融合されたシステム [11] を用いることを前提とすることで，1) チェックポイントに必要な実行状態の大部分を占めるメモリデータを，そのままチェックポイントファイルの一部とすることによる，チェックポイントの高速化，2) リスタート時にも，チェックポイントファイルのデータを，そのまま実行状態のメモリの一部とすることによる，リスタートの高速化を実現した．提案手法を実装した Linux を，実機上で実行する評価実験を行った．実験結果から，プロセスデータの保存処理については約 8〜10 分の 1，プロセスデータの読み出し処理については約 28〜36 分の 1 と大幅に低いコストで処理可能となり，提案手法がチェックポイントおよびリスタートの高速化に有効であることを示した．

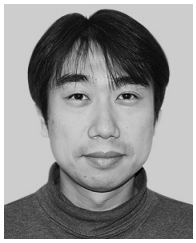
参考文献

[1] Lee, B.C., Ipek, E., Mutlu, O. and Burger, D.: Architecting phase change memory as a scalable DRAM alternative, *Proc. 36th Annual International Symposium on Computer Architecture* (*ISCA '09*), pp.2–13 (2009).

[2] Qureshi, M.K., Srinivasan, V. and Rivers, J.A.: Scalable high performance main memory system using phase-change memory technology, *Proc. 36th Annual International Symposium on Computer Architecture* (*ISCA '09*), pp.24–33 (2009).

[3] Zhou, P., Zhao, B., Yang, J. and Zhang, Y.: A durable and energy efficient main memory using phase change memory technology, *Proc. 36th Annual International Symposium on Computer Architecture* (*ISCA '09*), pp.14–23 (2009).

[4] Qureshi, M.K., Franceschini, M.M. and Lastras-Montano, L.A.: Improving read performance of Phase Change Memories via Write Cancellation and Write Pausing, *Proc. 2010 IEEE 16th International Symposium on High Performance Computer Architecture* (*HPCA*), pp.1–11 (2010).
[5] Ramos, L.E., Gorbatov, E. and Bianchini, R.: Page placement in hybrid memory systems, *Proc. International Conference on Supercomputing* (*ICS '11*), pp.85–95 (2011).
[6] Zhang, W. and Li, T.: Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architectures, *Proc. 18th International Conference on Parallel Architectures and Compilation Techniques*, pp.101–112 (2009).
[7] Dong, X., Muralimanohar, N., Jouppi, N., Kaufmann, R. and Xie, Y.: Leveraging 3D PCRAM technologies to reduce checkpoint overhead for future exascale systems, *Proc. Conference on High Performance Computing Networking, Storage and Analysis* (*SC '09*), pp.1–12, ACM (2009).
[8] Narayanan, D. and Hodson, O.: Whole-system persistence, *Proc. 17th International Conference on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS '12*), pp.401–410 (2012).
[9] Li, X., Lu, K., Wang, X. and Zhou, X.: NV-process: A Fault-Tolerance Process Model Based on Non-Volatile Memory, *Proc. 3rd ACM SIGOPS Asia-Pacific Workshop on Systems* (*APSys 2012*) (2012).
[10] (Mootaz) Elnozahy, E.N., Alvisi, L., Wang, Y.-M. and Johnson., D.B.: A survey of rollback-recovery protocols in message-passing systems, *ACM Comput. Surv.*, Vol.34, No.3, pp.375–408 (2002).
[11] 追川修一：Non-Volatile メインメモリとファイルシステムの融合，情報処理学会論文誌，Vol.54, No.3, pp.1153–1164 (2013).
[12] Huang, Y., Kintala, C., Kolettis, N. and Fulton, N.: Software Rejuvenation: Analysis, Module and Applications, *Proc. Int'l Symp. Fault-Tolerant Computing*, pp.381–391 (1995).
[13] Hargrove, P. and Duell, J.: Berkeley Lab Checkpoint/Restart (BLCR) for Linux Clusters, *Proc. Scientific Discovery through Advanced Computing* (*SciDAC*), pp.494–499 (2006).
[14] Protected and Persistent RAM Filesystem (2012), available from ⟨http://pramfs.sourceforge.net/⟩.
[15] Gioiosa, R., Sancho, J.C., Jiang, S. and Petrini, F.: Transparent, Incremental Checkpointing at Kernel Level: A Foundation for Fault Tolerance for Parallel Computers, *Proc. 2005 ACM/IEEE Conference on Supercomputing*, pp.1–14 (2005).
[16] Heo, J., Yi, S., Cho, Y., Hong, J. and Shin, S.Y.: Space-efficient page-level incremental checkpointing, *Proc. 2005 ACM Symposium on Applied Computing* (*SAC '05*), pp.1558–1562 (2005).
[17] Ruscio, J.F., Heffner, M.A. and Varadarajan, S.: DejaVu: Transparent User-Level Checkpointing, Migration, and Recovery for Distributed Systems, *Proc. 2007 IEEE International Parallel and Distributed Processing Symposium*, pp.1–10 (2007).
[18] Wang, C., Mueller, F., Engelmann, C. and Scott, S.L.: Hybrid Checkpointing for MPI Jobs in HPC Environments, *Proc. 2010 IEEE 16th International Conference on Parallel and Distributed Systems*, pp.524–533 (2010).
[19] Accetta, M., Baron, R., Bolosky, W., Golub, D., Rashid, R., Tevanian, A. and Young, M.: Mach: A new kernel foundation for UNIX development, *Proc. Summer 1986 USENIX Conference*, pp.93–112 (July 1986).
[20] Hitz, D., Lau, J. and Malcolm, M.: File system design for an NFS file server appliance, *Proc. USENIX Technical Conference* (1994).
[21] Vasavada, M., Mueller, F., Hargrove, P.H. and Roman, E.: Comparing different approaches for incremental checkpointing: The showdown, *Proc. Linux Symposium*, pp.69–79 (2011).



追川 修一　（正会員）

平成 8 年慶應義塾大学より博士（工学）．平成 16 年筑波大学大学院システム情報工学研究科助教授に着任．現在，筑波大学システム情報系情報工学域准教授．オペレーティングシステムに関する研究に従事．電子情報通信学会，IEEE 各会員．

三木 聡

平成 14 年より株式会社フィックスターズの代表取締役に就任．現在，マルチコアプロセッサ関連技術および NAND Flash メモリ関連技術の研究開発および事業化に従事．