Vol.2022-ARC-250 No.16
2022/10/12

セキュアNVMの高性能化のためのツリー事前更新

小池 亮[1,a)]　高前田 伸也[1,b)]

概要：バイトアドレス型の不揮発性メモリ（NVM）は，ストレージと同様の永続性と大容量を主記憶の階層に実現する一方，永続性ゆえにDRAMと比べセキュリティ上の脆弱性が大きい．攻撃者によるNVMへの改ざんを検知するため整合性ツリーが標準的に用いられるが，ツリーの更新は逐次的なハッシュ計算を必要とするため極めて高コストである．また，複数の書き込みに対する原子永続性を保証するトランザクション実行はNVMプログラミングにとって重要な一方，手動の永続化処理はプログラマの負担になるばかりか，バグの温床であることが示されてきた．これらの問題は膨大な予備電力を仮定することで解決できるが，その仮定は一部の恵まれた環境以外では成立しない．本研究では，幅広いプラットフォームにおけるセキュアNVMの高性能化を目指して，できるだけ少量の予備電力を仮定しつつ，一時的なメモリアクセスのリダイレクションに基づくツリー更新の完全な事前実行による高速化手法を提案する．さらに，永続化処理を自動実行することでトランザクション実行に係るプログラマの負担を減らすとともにバグを防ぐ，軽量な拡張機構を紹介する．

1. 序論

バイトアドレス型の不揮発性メモリ（NVM）が商用化され [1]，主記憶の階層に永続性とテラバイト級の容量がもたらされた．プログラム中のデータ構造をそのまま大容量の主記憶に永続化できることから，永続型のインメモリ・キーバリューストア [2–4] やデータベース [5–7]，間欠的コンピューティング [8–10] への応用が研究されている．

しかし，NVMには根本的課題が2つ存在する．第一はセキュリティである．NVMは不揮発性ゆえに，DRAMと比べ脆弱性が大きい．例えば，NVMをスロットから抜き去り，別のマシンに装着すれば中身を閲覧できる他，中身の一部を改ざんした上で元の場所に戻し，システムを誤動作させる攻撃も可能である．そこで，暗号化による機密性保護と，整合性ツリーを用いた改ざん検知によるセキュリティ対策が広く研究されてきた．

NVMシステムにおいては，クラッシュや電源トラブル後もセキュリティ・メタデータの一貫性が保たれることが期待されるという，DRAMシステムにはない困難がある．このため，整合性ツリーの中でも，葉のみから木全体を再構築可能という特徴を持つMerkle木（MT）が幅広く使われている．MTの根をオンチップに安全かつ永続的に保存することで，葉や中間ノードの改ざんを，計算された根の値と保存された根の値の不一致から検出する仕組みである．しかし，各MTノードは子ノードの暗号学的ハッシュ値なので，更新作業は葉から根へ逐次的に行わざるを得ず，オーバーヘッドが大きい．実際，80サイクルのハッシュ・レイテンシを持つ11レベルのMT [11–13] では，各書き込みにつき880サイクルもツリー更新に費やすことになる．そこで，ツリー更新を書き込みのクリティカルパスから除くことで高速化を図る手法が近年考案されてきた．Dolos [14] はツリー更新の事後実行を提案したが，オンチップの予備電力と永続型レジスタの必要量を増大させてしまう欠点があった．PMWeaver [15] は投機的なツリー更新の事前実行を提案したが，予測ミス時にはクリティカルパスが短縮できないうえ，他の予測値も連鎖的に修正する必要があり，事前実行による恩恵を限定的にしか得られていない．

第二の課題は，原子永続的な一貫性の保証である．データ構造の操作やビジネスロジックの実装においては，複数の書き込みを原子的に行う必要があることが多い．NVMは不揮発性なので電源遮断後も中身が保たれるが，原子的に行うべき一連の書き込みの最中に電源トラブルやシステムクラッシュが発生すると，NVM上のデータの一貫性が失われてしまう．そこでNVMプログラミングでは通常，データの上書き更新に先立ちロールバック可能なログを作ることで，このようなトランザクション実行を実現する．

しかし，プログラマはトランザクション中の変更処理を手動でキャッシュからNVMへ永続化する命令を挿入する必要があるうえ，ログ部分と上書き更新部分の間にフェン

[1] 東京大学
The University of Tokyo, Bunkyo, Tokyo 113–8656, Japan
[a)] ryo_koike@is.s.u-tokyo.ac.jp
[b)] shinya@is.s.u-tokyo.ac.jp

ス命令を入れなければ，正しい書き込みの順番が保証されず一貫性が失われてしまう．これは，プログラマの負担を増大させるだけでなく，永続化バグという新たな種類のバグのもととなっていることが示されてきた [16–20]．この問題は，キャッシュ全体を永続化する膨大なバッテリー量を仮定することで解決できる [21]．しかし，この仮定はメモリコントローラの永続性のみを仮定する現在の NVM システムの標準機構 [21] とはかけ離れており，一部の環境でしか成立しない．

本研究では，幅広いプラットフォームにおけるセキュア NVM システムの性能向上を目指して，できるだけ少量の予備電力と永続型レジスタを使いつつ，一時的メモリアクセス・リダイレクションを通したツリー更新の完全な事前実行による高速化手法を提案する．さらに，メモリアクセス・リダイレクションを活用した軽量な拡張機構として，トランザクション実行における永続化処理の自動実行を提案し，プログラマの負担軽減とバグの削減を実現する．

本稿の貢献は以下のとおりである．

- リダイレクションによってツリー更新の完全な事前実行を可能にし，永続化処理のクリティカルパスを最大 800 サイクル短縮した．
- トランザクション実行を支援する軽量な拡張機構を提案し，プログラマをログ管理と順序管理の手間から解放した．

2. 背景と動機

2.1 原子永続的一貫性

2.1.1 ライト・アヘッド・ロギング

NVM は永続性を備える一方で，突然のシステムクラッシュや電源の遮断が，アトミックに行われるべき一連の書き込みの最中に発生した場合，データの一貫性が失われてしまう．これを避けるため，NVM を用いるプログラムでは一般に，ライト・アヘッド・ロギングに基づくトランザクション実行機構を用いる [18,22,23]．これは名前の通り，データの上書き更新に先立ちログを作成する手法である．途中でクラッシュした場合，ログをもとにトランザクション実行開始前の状態へとロールバックすることで，データの一貫性を回復する．

図 1-(a) はソフトウェアによるライト・アヘッド・ロギングのプログラム例である．このトランザクションでは，アドレス `A` と `B` に対する更新を行う．通常ストア命令はストアキューにバッファされ，後にキャッシュ上のデータが更新される．しかしキャッシュは揮発性なので，更新されたデータを永続的領域へと書き込まない限り，クラッシュ後の一貫性は保証されない．そこで，キャッシュライン・ライトバック命令（`clwb`）とストア・フェンス命令（`sfence`）を用いる．`clwb` 命令は指定されたアドレスをキャッシュからライト・ペンディング・キュー（WPQ）へと書き戻す．

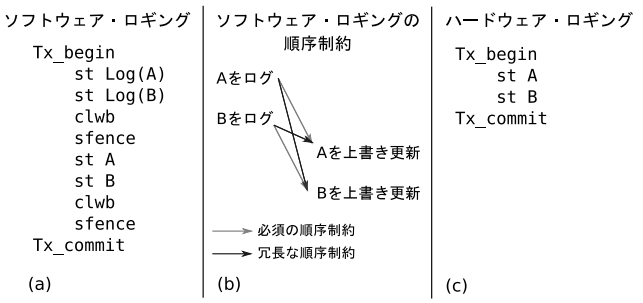

図 1 (a) ソフトウェア・ロギングのプログラム例．(b) ストア・フェンス命令による順序制約．(c) ハードウェア・ロギングのプログラム例．

Fig. 1 (a) Example of a software logging program. (b) Ordering constraints specified by the store fence instruction. (c) Example of a hardware logging program.

WPQ とはメモリコントローラ内のキューであり，NVM への書き込みデータのバッファリングを行う．`sfence` 命令は，先行するメモリへの書き込み命令（`clwb` 命令など）がすべて完了してから後続のストア命令の実行が行われるよう，書き込み順序を保証する．

このような順序制約はパフォーマンスのボトルネックとして認識されており，高速化のため，ハードウェア・ベースのロギング・アーキテクチャがさまざま提案されてきた [24–28]．図 1-(b) にあるように，`sfence` 命令は不必要な順序制約を生じる．そこで，ハードウェアでログ管理を行うことで細粒度の順序を実現する，というのがこれらの研究の目的である．同時に，プログラムは図 1-(c) へと簡略化されるので，プログラマからログ管理や順序管理の負担を排除することにもつながる．近年，適切に `clwb` 命令や `sfence` 命令を挿入していないことによるバグを検出したり [17,19,20] 修正したりする [16] ツールが提案されてきているが，ハードウェア・ロギングはこのような永続性バグの排除にも貢献する．

2.1.2 ADR と eADR

従来はオンチップに永続性が保証されている領域が存在しなかったため，`clwb` 命令に続く `pcommit` 命令によって WPQ から NVM への書き込みを指示する必要があった [29]．しかし，これではレイテンシの大きな NVM への書き込みがプログラム実行のクリティカルパスを延長してしまう．そこで，ADR [30] という，クラッシュ時にも NVM へ書き込めるだけの予備電力をバッテリーやスーパーキャパシタ等によって備えておく方式が登場した．現在 ADR は NVM システムの必須要件として標準化されている [30,31] ので，`clwb` 命令と `sfence` 命令によって永続性と順序が保証される．

ほかに，eADR [21] という機構も存在する．これは WPQ に加えてキャッシュ全体をクラッシュ時に NVM へと書き込むための予備電力を備えるものであり，`clwb` 命令の必要性を排除することができる．しかし，すべてのシステム

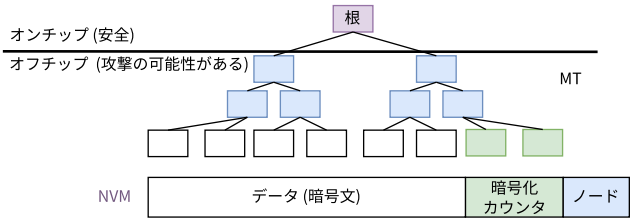

図 2 MT による整合性保証
Fig. 2 Integrity protection using MT.

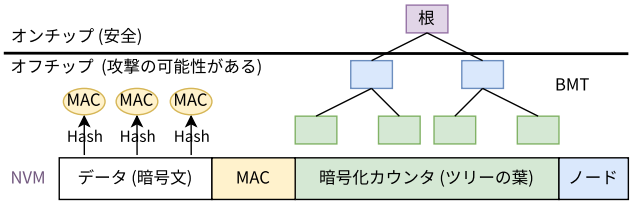

図 3 BMT [13] による整合性保証
Fig. 3 Integrity protection using BMT [13].

が eADR を保証できる訳ではない．実際，商用 NVM システムの WPQ サイズは 8 エントリにとどまる [32] ので，ADR が保証すべき書き込み量は 1 KB にも満たない．一方キャッシュ全体は数十 MB に及ぶので，eADR が要求する予備電力は ADR よりも極めて大きい．

2.2 セキュリティ

2.2.1 脅威モデル

セキュア NVM についての既存研究 [33–36] と同様，本研究では所有や窃盗，メンテナンス等によりシステムへの物理的アクセスをもつ攻撃者を想定する．プロセッサ及びその内部構造，すなわちキャッシュやメモリコントローラ等は攻撃者がアクセスできないものとする．一方，攻撃者はメモリバスやメモリ本体には盗聴や改ざんが可能であると仮定する．提案手法はこれらの攻撃への対策として，機密性保護と改ざん検出機構を提供する．なお，漏洩電磁波などサイドチャネル攻撃は，回路や詳細な実装に依存するので本研究の範囲外とする．

2.2.2 暗号化

復号処理の効率性を背景に，カウンタ方式の暗号化手法が広く用いられている [33–35, 37]．この手法では，オンチップの秘密鍵，アドレス，暗号化カウンタの値からワンタイムパッド（OTP）を生成する．暗号文は平文と OTP の XOR 演算を取ることで得られ，逆に平文は暗号文と OTP の XOR 演算により得られる．既知平文攻撃を避けるため，暗号化カウンタは毎回更新するとともに，64 B のメモリアドレスごとに別々に用意する．カウンタがオーバーフローした場合，オンチップの秘密鍵を更新した上で全メモリの中身を再暗号化しなければならない．このオーバーヘッドはテラバイト級の NVM では特に大きい．そこで，キャッシュブロックごとの「マイナーカウンタ」とは別に，ページごとに「メジャーカウンタ」を設けることでオーバーフローの影響範囲を限定する手法が提案され [38]，広く使われている．本研究もそれに倣う．

2.2.3 MT

カウンタ方式の暗号化では，改ざんから NVM 上の暗号化カウンタ及び暗号化済みデータを保護する必要がある．最も素朴には，**図 2** のように，これらの保護したい対象全体を MT の葉とし，改ざんを防ぐためにオンチップに根を保管すれば良い．このとき，保持していた根の値と計算された根の値の不一致から，任意の葉や中間ノードへの改ざんを検知できる．なお，突然のクラッシュや電源遮断に備えて，根の保管には不揮発性のレジスタを使用する．もしくは，[33, 34] のように平常時は揮発性のレジスタを使い，クラッシュ時に不揮発性のレジスタへ移動させるだけの十分な電力をスーパーキャパシタで用意しておく．

MT の利点は，葉のみから木全体を再構築可能な点である．これにより，特別な工夫をせずとも，MT では中間ノードのライトバック更新が可能である．一方で，高さに比例して更新レイテンシが大きくなる欠点を持つ．NVM 読み込みに伴う整合性検証は全レベルで実質的に並列してハッシュ計算が可能な一方で，NVM への書き込みに伴う更新作業の際には，親へ親へと逐次的にハッシュ値を計算する必要があるためである．

2.2.4 BMT

木の高さを軽減する目的で，暗号化カウンタのみを葉とするような MT がセキュア NVM では広く採用されており [33–36, 39]，特に Bonsai Merkle Tree（BMT）[13] と呼ばれている．暗号化済みデータの整合性は，そのハッシュであるメッセージ認証符号（MAC）で保証する．概念図を**図 3** に示す．

アドレス A の暗号文 C_A に対する MAC M_A は，カウンタ γ_A 及びオンチップに安全に保管された秘密鍵 K を用いて $M_A = MAC_K(A, C_A, \gamma_A)$ と計算される．したがって，攻撃者からすれば，カウンタを改変することなく整合性保証を突破するには，M_A および C_A を改ざんする必要がある．しかし，そのようなペア (M_A, C_A) を見つけるのは原像計算困難性から実質的に不可能である．過去に利用されたペアを再現すれば原像計算は不要だが，カウンタも当時のものに改ざんしなければならず，BMT の根の不一致から検出される．BMT 方式では，このようにして整合性を保証する．

2.2.5 SIT

Intel の SGX [40] 方式の整合性ツリー（SIT）も，広く用いられている [34, 41–43]．概念図を**図 4** に示す．このツリーでは，中間ノードはすべて内部にカウンタと MAC を持ち，ノード MAC はそのノードのアドレス，そのノード内の全カウンタおよび親ノードのカウンタの一部のハッシュ値として計算される．葉の構成も同様であるが，葉の

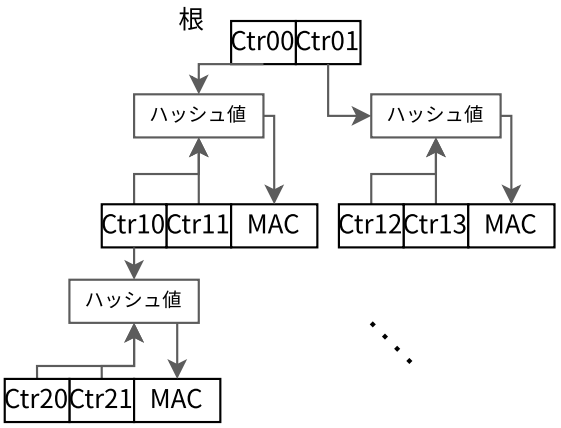

図 4 SGX 方式の整合性ツリー（SIT）
Fig. 4 SGX-style integrity tree (SIT).

カウンタは暗号化カウンタである．

SIT はカウンタのみからハッシュ計算を行うため，更新レイテンシが小さく済む．更新時には，該当する暗号化カウンタと，そこから根へと至る中間ノードのカウンタ値をすべてインクリメントし，パイプライン化されたハッシュ計算エンジンに入力することで，事実上並列にノード MAC の計算が行える．一方，更新された中間ノードをその都度 NVM に書き込まない限り，クラッシュ後にノードカウンタが失われるので復旧不可能である．一つのデータを書き込むために，根に至るまでの 10 個程度の SIT ノードを追加で書き込む必要があり，書き込み量が爆発してしまう問題がある．

ノードに制限付きのライトバック更新を導入することで書き込み爆発を抑える，SecNVM [43] という手法が提案されている．この手法ではノードキャッシュはライトバック方式なので，リカバリ時にキャッシュされていたノードのカウンタを復元する必要がある．そこで，初期推測値として，NVM に存在する古いカウンタを使う．これと NVM に存在する子ノードのカウンタとのハッシュから，子ノードの MAC 値を再計算する．これが NVM に書かれている値と合致するなら，この推測値を失われた真のノードカウンタとして復元する．合致しない場合，推測値を 1 増やし，再度比較することを繰り返す．この試行回数を制限するため，各ノードは一定回数の更新ごとに強制的に NVM へと書く，ということも提案されている．しかし，この手法には欠陥がある．親と子が両方ともキャッシュされている場合，クラッシュ後に子のノード MAC が失われるため，親のノードカウンタの復旧確認が行えず，リカバリが不可能となるのである．

SCUE [42] はノードカウンタの加算関係を利用したリカバリ方法を採用することで，中間ノードのライトバック更新を可能にしている．さらに，SIT 更新時は根をインクリメントするだけで十分であると提案している．しかし，この手法ではカウンタがオーバーフローすると加算関係が崩れてしまうので，復旧不可能になる．

他に，ツリーの遅延更新も書き込み量の削減につながる．これは，ツリー更新を葉から根まで行う代わりに，キャッシュされたノードまで行い，以降のツリー更新はそのノードがキャッシュから追い出されたときまで遅延させる手法である．更新されるノード数が減ることから，書き込み量が削減される．この手法では，キャッシュ上で更新されたノードがクラッシュ後に復元できないため，攻撃者によるリプレイ攻撃等が行われていない場合でも，整合性エラーが生じてしまう．ASIT [34] は，キャッシュ上で更新されたノードとそのアドレスを NVM にシャドーイングした上で，ノードキャッシュを葉とする MT を別途構築することで，高速でセキュアなクラッシュリカバリを実現する．しかし，追加された MT のために，SIT を用いているにも関わらず，5 から 6 段の逐次的ハッシュ計算が必要となる．

2.2.6 MT 更新処理の高性能化

MT 更新は逐次的なハッシュ計算を必要とするためオーバーヘッドが大きい．性能向上を目的として数々の研究が行われてきた．HNodeTree [44] は，BMT ノードキャッシュ上に小さな MT を追加することで BMT の遅延更新を可能にする手法である．これは，追加される MT の根による永続型レジスタ数の増加と引き換えに，BMT 更新レイテンシの削減を実現する．PLP [35] は永続化モデルの規定する順序関係を保持しつつ BMT 更新を高性能化する手法として，厳密永続化モデル（SP）に対してはレベルごとのパイプライン化を，エポック永続化モデル（EP）に対してはエポック内のアウトオブオーダー BMT 更新を提案し，スループットを向上させた．ログ BMT の分離 [45] は NVM のトランザクション実行に着目し，ハードウェアが NVM 上に確保したログ領域に専用の小さな MT を設けることで，ログ処理に伴う BMT 更新レイテンシを短縮した．最後に，ASSURE [46] と BMF [47] は大容量の不揮発性キャッシュを想定して根の数を大幅に増やし，それらを動的に再構築することで根までの段数を削減させた．

2.2.7 クリティカルパスから MT 更新を排除する必要性

図 5 は，アドレス `A` へのストア命令に続き，`clwb` 命令と `sfence` 命令を実行したときのタイムチャートである．メモリコントローラのライト・ペンディング・キュー（WPQ）に書き込みパケットが格納されたら `clwb` 命令は完了できるが，セキュリティ更新はその前に行われる必要がある [14]．なぜなら，ADR [30] により WPQ のエントリは全てクラッシュ時にメモリへと書かれるだけの予備電力が備わっている一方，セキュリティ更新のための計算にかかる電力や，更新完了までメモリに書き出さずに保持するだけの電力については，保証されていないためである．その結果，MT 更新はプログラム実行のクリティカルパスを構成する．そのレイテンシは，80 サイクルのハッシュ・レイテンシを持つ 11 レベルの MT [11–13] では 880 サイクルと，極めて大

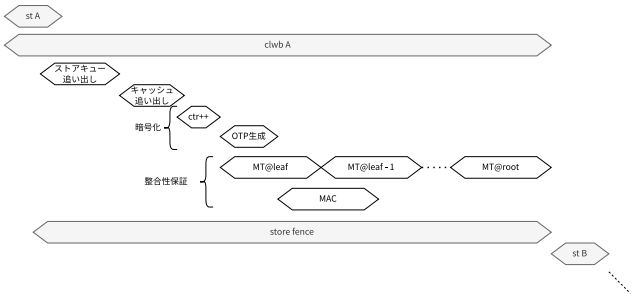

図 5 クリティカルパス上の MT 更新
Fig. 5 MT update on the critical path.

きい．また，根を増やさない限り実質的なツリーの段数削減には限界があるので，逐次的なハッシュ計算による性能低下は避けられない．

Dolos [14] は，WPQ への挿入前にマイナー・セキュリティ・ユニット（Mi-SU）と呼ばれる軽量なセキュリティ・フロントエンドを導入し，ツリー更新を含む通常のセキュリティ更新を事後実行することで，この問題を解決する．Mi-SU は WPQ に特化したセキュリティ対策として，暗号化及び MAC の計算を行うだけなので，クリティカルパスは極めて短くなる．一方で，WPQ での暗号化に使うカウンタと WPQ エントリのアドレスの分，オンチップの永続型レジスタの必要量が増大してしまう上，Mi-SU により追加された MAC のために貴重な WPQ エントリが浪費される．さらに，高レイテンシのツリー更新を WPQ エントリ占有中に行うので，WPQ の容量不足による性能低下が発生していた [14]．これに対し，PMWeaver [15] は将来の NVM 書き込みアドレスとデータを予測することで，ツリー更新の投機的な事前実行を行う．しかし，予測ミス時には事前計算した値が使えないため，クリティカルパスを短縮することができない．さらに，ツリー更新は複数の書き込みが共通のノードを更新するので，もし予測ミスが見つかった場合には，他の予測値も連鎖的に再計算することが必須であり，事前実行による恩恵を限定的にしか享受できていなかった．

2.3 動機

先述の通り，MT は BMT 方式において暗号化カウンタの上に構築・管理されるのみならず，SIT を使う場合でも中間ノードによる書き込み爆発を抑制するために併用される [34] ほか，BMT ノードキャッシュの上に構築される場合もある [44]．さらに，データのハッシュ値である MAC の NVM 書き込み量を削減するため，MAC キャッシュ上に構築されることもある [44]．このように，MT はセキュア NVM において幅広く用いられており，その高速化は重要である．

MT 更新のレイテンシは，根を大量に持つことで緩和が可能である．事実，BMF [47] の評価では 80%のケースで 1 段分のツリー更新で根に至ったと報告している．しかし，根を増やすためには，その数に比例してオンチップの永続型レジスタや予備電力の必要量が増えてしまう．BMF の背景となっている eADR 機構 [21] は，WPQ に加えてキャッシュ全体をカバーすることを要求するため膨大な予備電力と面積が必要 [48] な点で，それを適用できるのは一部の恵まれた環境に限られる．様々なプラットフォームで適用可能な手法とするには，NVM システムの最低水準である ADR 機構 [21,30] をターゲットとすることが不可欠である [31]．その場合，根を増やすことはできないため，逐次的なハッシュ計算自体は必ずどこかで行わなければならない．もしクリティカルパスの外に移動させることができれば，逐次的ハッシュ計算を伴うツリー更新のオーバーヘッドを隠蔽することができ，性能向上が期待できる．

また，NVM プログラミングではトランザクション実行が重要な役割を担う．なぜなら，NVM 自体が不揮発なだけでは不十分で，データに対して原子永続的一貫性が保証されない限り，NVM の永続性やクラッシュ耐性を活用することは困難だからである．しかし，図 1-(a) のような手動のログ管理や永続化処理・順序関係の管理は，プログラマの負担を増大させることに加え，NVM においてのみ必要となる特殊な操作なので，NVM プログラミングの障壁となり得る．実際，`clwb` 命令や `sfence` 命令を適切に挿入することは常に容易な訳ではなく，新たな種類のバグの原因となっていることが示されてきた [16–20]．その限りにおいて，図 1-(c) のように，トランザクション実行に係るログ管理や永続化・順序関係の管理をプログラマに対して隠蔽し，ハードウェアがそれらを担うアプローチの方が望ましいと言えよう．

そこで，本研究は，できるだけ少量の予備電力と永続型レジスタを用いることを前提とした上で，1）ツリー更新のクリティカルパスからの完全なる排除によってセキュア NVM システムを高性能化することに加え，2）トランザクション実行をハードウェアにより支援することで，プログラマの負担を軽減することを目的とする．

3. 提案手法

本章はまず，第一の目的である，ツリー更新をクリティカルパスから完全に排除する手法について説明する．第二の目的であるトランザクション実行を支援する軽量な拡張機構については 3.4 章で紹介する．なお本稿では，MT を念頭に説明する．MT は幅広く用いられている上に，事前更新による恩恵が大きいためである．しかし，本手法は SIT にも適用可能であることを強調する．

3.1 一時的メモリアクセス・リダイレクション

投機的な予測実行に基づく事前実行方式では，予測ミスのリスクが避けられない．そこで本研究は**一時的メモリア**

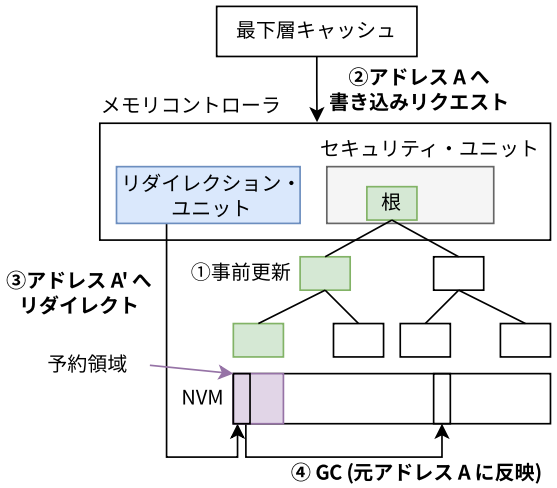

図 6 一時的メモリアクセス・リダイレクションによるツリー事前更新

Fig. 6 Ahead-of-time tree update scheme through temporal memory access redirection.

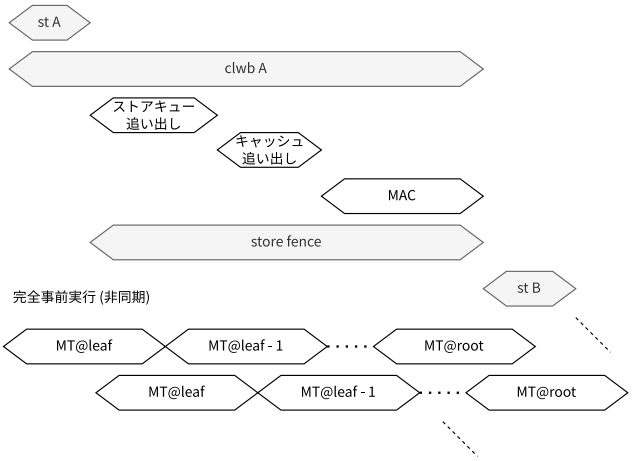

図 7 提案手法のタイムチャート

Fig. 7 Timing chart for the proposed method.

クセス・リダイレクションに基づくツリー更新の完全なる事前実行方式を提案する．直観的には，書き込み先のアドレスを事前に決めてしまい，そのアドレスに対してツリー更新を完了させた上で，到着した書き込みリクエストをそのアドレスにリダイレクトする手法である．

動作概要を**図 6** に示す．まず，NVM 容量のうち一部を，メモリコントローラより上層からは参照できないシステム予約領域として確保しておく．この予約領域の中からリダイレクト先アドレスを選択することで，ページテーブルを修正する必要を排し，手法をメモリコントローラ内で完結させる．予約領域は数メガバイト程度を想定しており，テラバイト級の NVM 全体と比べるとごく一部で十分である．将来のリクエストに備え，未使用のアドレスを選択し（これを A’ とする），そのアドレスに対して暗号化カウンタを更新した上でツリー更新も完了させる．なお，このとき暗号化に使う OTP の生成も行っておくことで，データが到着したら XOR 演算を行うだけで暗号化が完了する．その後，キャッシュから書き込みリクエストが到着したら，事前に確保しておいたアドレス A’ へリダイレクトする．リダイレクションが確立したら，以降の読み込みリクエストもアドレス変換を行うことになる．また，予約領域の容量不足を回避するため，書き込み内容を元のアドレスへ反映させるガベージ・コレクションをバックグラウンドで実行する．ガベージ・コレクション完了後には，アドレス A’ は未使用となるので，今後のリクエストに備え，暗号化カウンタの更新とツリー更新を進めておく．以上が，提案手法の概要である．

図 7 は提案手法のタイムチャートである．本手法では，書き込みリクエスト受信時には既にツリー更新と OTP 生成が完了しているので，必要なセキュリティ更新は XOR 演算による暗号化と MAC の計算のみであり，クリティカルパスは大幅に短縮される．11 段の BMT に対して，暗号学的ハッシュ関数の計算に 80 サイクル [11–13] かかる場合，短縮されたクリティカルパスは $80 \times 11 - 80 = 800$ サイクルである．比率にすると，セキュリティ処理にかかるレイテンシを $800/(80 \times 11) = 90.9\%$ 削減している．なお，本手法は投機的事前実行手法 PMWeaver [15] と異なり予測ミスの虞がない．したがって，任意のプログラムパスに対してクリティカルパスの短縮が実現できる．

3.2 追加のツリー更新による負荷の緩和

一方で，リダイレクションに伴うコストも存在する．まず，クラッシュ後のリカバリに備え，リダイレクト時には元アドレスの情報もメタデータとして NVM に書く必要がある．これは書き込み量を増やすだけでなく，ツリー更新の回数も 2 倍に増大させる．以下，その理由を説明する．まず，ツリー更新は NVM への書き込み時に暗号化カウンタが更新される際に必要となる．暗号化カウンタは，暗号化の単位である 64 B のデータごとに存在するので，同一の 64 B 内に更新データとメタデータ（元アドレス）を共存させることができたら，ツリー更新回数は 1 回で済む．しかし，更新データはキャッシュから追い出されたブロックなので，それ自身で 64 B を占めており，メタデータには別の 64 B ブロックを割り当てることになる．したがって，ツリー更新回数は 2 回となってしまう．

さらに，ガベージ・コレクションによっても，書き込み量とツリー更新回数が増大する．このうち，書き込み量の増大自体はクリティカルパスを延長する訳ではないので性能への影響は限定的だと考えられるものの，ツリー更新はレイテンシが大きいので，その回数が増えると問題となり得る．64 B の書き込みリクエストに対して，リダイレクト先，メタデータ，ガベージ・コレクションと 3 回ツリー更新を行うことになるので，更新回数の点では負荷が 3 倍にも増えてしまう．

そこで，メタデータとガベージ・コレクションそれぞれに対し，ツリー更新の負荷を軽減する手法を提案する．まずガベージ・コレクションに対しては，セキュリティ更新すべての事前更新を提案する．ガベージ・コレクション自体はハードウェアが実行のタイミングを決めることができるうえ，対象となるデータ及びアドレスが事前に分かっているという特徴がある．そのため，完全な事前実行が可能である．すなわち，元アドレスに対応する暗号化カウンタをインクリメントし，OTP 生成とツリー更新を行う．また書き込みデータも分かっているので，生成された OTP と XOR 演算を取ることで暗号文が生成できるほか，それと暗号化カウンタを入力として MAC を計算できる．投機的事前実行手法 PMWeaver [15] と異なり予測ミスの可能性がないため，キャッシュ上の値を直接更新しても問題ない．

次に，メタデータに対しては一括書き込みを提案する．先述の通り，メタデータはデータ本体とは異なる 64 B ブロックに書かれる．そこで，メタデータのみを保持する 64 B ブロックを用意した場合，8 B のアドレス情報を 8 つ並べられる．つまり，一つの暗号化カウンタを 8 つのメタデータが共有できる．これを，リダイレクションの都度カウンタ更新およびツリー事前更新を行う代わりに，カウンタを更新してツリーを事前更新した上で，8 つのリダイレクション・メタデータが揃うまでは，当該メタデータブロックを WPQ に留める．つまり，8 回分は同じ暗号化カウンタを使い回すことを提案する．カウンタを再利用するものの，WPQ はオンチップなので攻撃者は観測不能なのでセキュリティ上の問題はない．これにより，メタデータに係るツリー更新回数は 87.5%削減され，1/8 にまで減る．

3.3 アーキテクチャ

提案アーキテクチャの全体像を**図 8** に示す．水色部分は提案手法による追加機構である．`Addr Mapper` はアドレス変換テーブルであり，元アドレスからリダイレクト先アドレスを指す．`Victim Buffer` はガベージ・コレクションを効率化するために，書き込みリクエストを保持しておく少量の揮発性バッファである．`head/tail` はリダイレクションの有効アドレス範囲を指すポインタである．また，緑色部分は 3.4 章で説明するトランザクション実行支援のための拡張機構であり，必須ではない．`BMT Update Table` は BMT 更新処理の管理に使われるテーブルであり，各エントリは，`valid` ビット, `ready` ビット, `BMT#`フィールドを持つ [35]．なお本研究では事前更新により BMT 更新の順序管理が不要なので，先行研究 [35] と比べて本テーブルは簡素に実現可能である．

3.3.1 書き込み時の動作

メモリコントローラが書き込みリクエストを受け取る際には，事前に確保していたアドレス，すなわち `head` ポインタの指すアドレスへ変換を行い，`Addr Mapper` に記録する．その後，`head` を 64 B 進める．64 B 先がリダイレクション・メタデータ用のアドレスの場合はさらに 64 B 進め，予約領域の範囲を超える場合には先頭へと巡回する．変換先アドレスの OTP と XOR 演算を行い，得られた暗号文と暗号化カウンタから MAC を計算する．ツリー更新は事前に完了しているので，MAC の計算が完了次第 WPQ へ挿入し，書き込みリクエストの受領を通知する．WPQ エントリはいずれ NVM へと書かれる．また，ガベージ・コレクションに備え `Victim Buffer` に書き込みデータをバッファリングしておく．

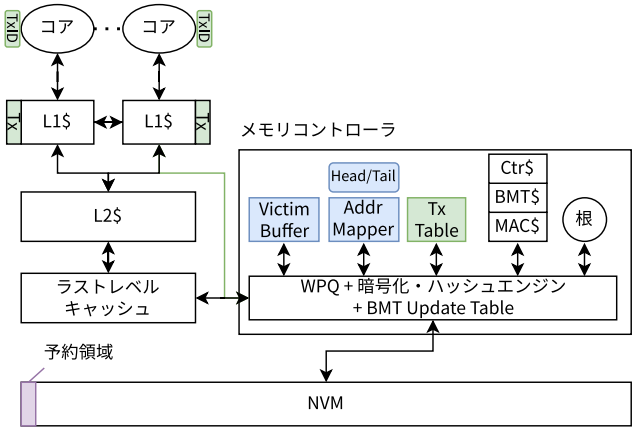

図 8 提案アーキテクチャ．水色部分は提案手法による追加機構を示す．緑色部分はトランザクション実行支援のための拡張機構（オプショナル）

Fig. 8 Proposed architecture. Blue components are our additional structure and green parts are optional extension to support transaction execution.

3.3.2 読み込み時の動作

メモリコントローラが読み込みリクエストを受け取った場合，まず `Victim Buffer` を参照する．ヒットしたら，その情報をレスポンスとして返す．ミスした場合，NVM からデータをフェッチする必要がある．`Addr Mapper` にエントリが存在する場合，有効なリダイレクションが存在するので，見つかったアドレスに対して変換し，リードキュー（RPQ）へ挿入する．このとき，元アドレスを記録しておく．RPQ エントリはいずれ NVM アクセスを経てデータブロックをフェッチする．フェッチ完了後に，記録しておいた元アドレスに直し，読み込みレスポンスを返す．

3.3.3 ガベージコレクションの動作

`tail` ポインタの指すアドレスに対して，ツリーの部分的事前更新を行う．このとき，もし当アドレスのデータが `Victim Buffer` に存在しない場合は，同時並行的に NVM からフェッチしておく．先述の通り，ガベージ・コレクション時にはすべてのセキュリティ更新処理を事前実行する．すなわち，ツリー更新に加え，OTP の生成も行うほか，`Victim Buffer` 上のデータを使ってと XOR 演算による暗号文生成と MAC の計算も行う．すべて完了したら

WPQ に挿入し，tail ポインタを 64 B 進める．進めた先がリダイレクション・メタデータであった場合は，ガベージ・コレクションの必要がないので，さらに 64 B 進める．予約領域の範囲を超える場合には先頭へと巡回する．このとき通過したアドレスは追加された未使用アドレスであるから，今後のリクエストに備えてツリーの事前更新を行う．以上がガベージ・コレクションの動作である．

3.3.4 リカバリ

本手法は有効なリダイレクションの範囲さえ分かればクラッシュ後も実行を継続可能である．tail ポインタと head ポインタの間が有効なリダイレクション領域となっているので，両ポインタはオンチップの不揮発性資源を利用することで，クラッシュ後のアクセスを保証する．必須ではないが，ガベージ・コレクションを通してリダイレクションをすべて不要化しておくことで，実行時の読み込み処理を高速化できる．または，リカバリ時には Victim Buffer にキャッシュしておく部分までを行い，リカバリ完了後にガベージ・コレクションを通常実行と並行して行えば，リカバリ時間の短縮と実行時の読み込み処理に対する高速化の両立が期待できる．

3.3.5 オンチップ不揮発性資源

予備電力を備えた揮発性レジスタや永続型レジスタ等，オンチップで必要となる不揮発性資源の追加量は，head/tail ポインタの合計 16 B のみである．これは先行研究と比較して顕著に少ない．eADR [21] を背景とする先行研究 BMF [47] はデフォルトで 4 KB の不揮発性キャッシュを想定していたが，これと比較すると 99.6%も削減している．またツリー更新の事後実行手法 Dolos [14] は WPQ での暗号化カウンタ（8 B として計算する）と各 WPQ エントリにつき 8 B のアドレスを永続型レジスタに保存するので，8 エントリの WPQ [32] では合計で 72 B となる．これと比べると，提案手法は 77.8%の削減を達成している．ただし，提案手法は投機的事前実行手法 PMWeaver [15] より多い．PMWeaver は投機実行の結果を揮発性バッファに格納するだけなので，不揮発性資源を追加することはないためである．しかし，PMWeaver では 20%程度の確率で予測ミスにより事前実行に失敗すると報告されている [15]．一方，提案手法はたった 16 B の不揮発性レジスタの追加により，完全な事前実行手法を実現していることを強調する．

3.3.6 セキュリティ

本手法はリダイレクト先アドレスに対しても通常と同じセキュリティ対策を施しているので，盗聴及び改ざん攻撃のリスクを高めることはない．リダイレクション・メタデータに対する一括書き込みでは同一の暗号化カウンタを複数回再利用するが，その間 NVM へ書かれることはないので，再利用は攻撃者が観測することはできず，セキュアである．

3.4 トランザクション実行支援に向けた軽量な拡張

本手法は 2.1.1 章で言及したハードウェア・ロギングという NVM のトランザクション実行支援手法を軽量な拡張により実装することが可能である．ハードウェア・ロギングにおけるプログラムは図 1-(c) のようになっている．本手法はキャッシュ追い出しに対してリダイレクションを行うので，これをログとするためには，トランザクション実行中のストア命令に対してキャッシュ追い出しを強制する追加機構を作れば良い．

3.4.1 追加機構

アーキテクチャ上の追加機構は図 8 の緑色部分に示している．まず，各コアにトランザクション ID を管理する TxID レジスタを追加する．また，L1 データキャッシュからメモリコントローラへの Victim Buffer へトランザクション中に変更されたブロックを送る経路も追加する．加えて，メモリコントローラには Tx Table というテーブルを追加する．これはダイレクトマップ式ハッシュテーブルであり，入力 TxID に対して，valid ビット，running ビットと開始アドレス start を返す．running ビットは，そのトランザクションが開始されるとセットされ，コミット後にリセットされるとともに，valid ビットがセットされる．そのトランザクションに対するガベージ・コレクションが完了したら，valid ビットはリセットされる．

なお，これらの拡張機構のうち，Tx Table の valid ビット（1 ビット）のみ永続性を仮定する．先行研究 [35] と同様 32 エントリの場合を考えると，拡張機構に必要な不揮発性資源の追加容量は 4 B となる．基本機構と合わせると提案手法の合計は 20 B である．トランザクション実行の永続化モデルに特化した MT 更新の高速化に関する最先端の手法 PLP [35] がデフォルトで 664 ビットの不揮発性資源を必要とするのと比べると，本手法は 75.9%の削減を達成している．また，後述の通りリダイレクション・メタデータに TxID を記載するので，メタデータに必要なビット幅が増大する．しかし 32 エントリの Tx Table では TxID は 5 ビットあれば十分で，元アドレスのために確保していた 8 B のうち余っている上位ビットに埋め込むことができる．

3.4.2 動作の説明

図 1-(c) のプログラムに対する本拡張機構の動作を説明する．まず，Tx_begin 命令は現在の TxID をストアキューに格納する．次に，st A および st B はストアキューにバッファされる．Tx_commit 命令はストアキューの中身を L1 データキャッシュへフラッシュする．L1 データキャッシュは TxID が記録されたエントリ以降のストアについて，該当するキャッシュブロックを更新し，更新後のブロックを TxID とともにメモリコントローラへ送る．メモリコントローラはツリー事前更新を完了しているアドレスに対してリダイレクトする．なお，トランザクション実行におけるリダイレクション・メタデータには，クラッシュに備

えて TxID の情報も記載する．また，Tx Table 中の当該 TxID のエントリについて，running ビットがセットされていない場合にはセットし，start にリダイレクト先アドレスを記録する．セットされていた場合には start は変更しない．なお，もし valid ビットがセットされていた場合は，以前の同一 TxID のトランザクションに対するガベージ・コレクションを先に完了させる必要がある．当該トランザクションによるキャッシュへの変更をすべてメモリコントローラに送信したら，Tx_commit 命令は最後に送信完了のメモリコマンドをメモリコントローラへ送る．これを受け取り，かつ暗号化と MAC 計算が完了したら，WPQ へ追加するとともに Tx Table 中の当該 TxID の running ビットをリセットし，valid ビットをセットする．その後，キャッシュへ受領のレスポンスを返却する．これを受け取ったら Tx_commit 命令は完了できる．

3.4.3 ガベージ・コレクション

トランザクション中にハードウェアによりキャッシュ追い出しが強制されたブロックに対するガベージ・コレクションは，トランザクション完了前に実行してはならない．なぜなら，その場合リダイレクト先のログと本来のデータとがともに新しい値を保持するので，トランザクション完了前にクラッシュした場合にロールバックすべき以前の値が失われてしまうからである．そこで，トランザクション拡張を利用する場合のガベージ・コレクションでは，tail ポインタの指すアドレスが未完了のトランザクションによるものである場合には，完了するまで待機する必要がある．しかし，各リダイレクト先アドレスに対してトランザクションによるものか否かを記録するのはコストがかかるので，Tx Table の start フィールドを使い保守的に判定する．もし，running ビットがセットされたエントリが存在し，かつ tail がその start 以上であれば，tail はそのトランザクションによる書き込みの可能性がある．したがって，valid ビットがセットされるまでガベージ・コレクションは中断することにする．

3.4.4 リカバリ

トランザクションの実行中にクラッシュした場合のリカバリ方法について，図 1-(c) を例に説明する．本手法はハードウェア・ロギングの先行研究 [24–28] に倣い，完了したトランザクションによる書き込みはすべて反映し，未完了だったトランザクションによる書き込みはすべて無効化する．リカバリ・プロセスは tail ポインタの指すアドレスから順番に NVM のデータを確認する．メタデータに TxID が記載されており，かつ Tx Table において当該 TxID で指定されるエントリの valid ビットがセットされている場合，メタデータに記載されている元アドレスに対して暗号化と MAC 計算，ツリー更新を行った上でコピーする．また，メタデータに TxID が記載されていない場合，トランザクション処理とは関係のないリダイレクションであるから，これも同様に元アドレスにコピーする．これを繰り返し，tail ポインタと head ポインタが合致したら，コピーが必要なデータはすべて元アドレスへ反映されたのでリカバリは完了である．

表 1 先行研究との比較
Table 1 Comparison with previous works.

	Dolos [14]	PMWeaver [15]	提案手法
方式	事後実行	投機的事前実行	**完全事前実行**
再計算	**なし**	あり	**なし**
WPQ 負荷	高	**低**	**低**
永続性要求	中	**最小**	**小**
プログラム	複雑	複雑	**簡潔**

4. 関連研究

4.1 クリティカルパスからツリー更新を排除

表 1 はクリティカルパスからツリー更新を排除する既存手法 Dolos [14] 及び PMWeaver [15] と提案手法を比較している．Dolos はツリー更新の事後実行方式である一方，PMWeaver と提案手法は事前実行方式である．ただし，PMWeaver は書き込み予測による投機的手法であるため，予測ミス時には事前実行に失敗する上に，他の予測値を連鎖的に再計算する必要があった [15]．それに対し，提案手法は一時的リダイレクションによって完全な事前実行を実現している．また，Dolos では WPQ エントリ占有中にツリー更新を行うため，WPQ の逼迫が起こりやすく，容量不足による性能低下が発生していた [14]．WPQ サイズは必要な予備電力量に直結するので，これも重要な観点である．手法に必要な永続型レジスタの量は 3.3.5 章で述べた通り PMWeaver が最小であるが，提案手法も 16 B の増加のみである．最後に，NVM プログラミングで重要なトランザクション実行において，Dolos と PMWeaver ではいずれもプログラマが煩雑なログ管理や永続化処理，書き込み順序の管理を行う必要があり，永続性バグのもととなっていた．それに対し，提案手法ではこれらをハードウェアが自動で実行するので，プログラマはトランザクションを必要とする箇所を明記するだけで済む．

4.2 ハードウェア・ロギング

ハードウェア・ロギングの手法は，ログの中身に応じて大きく 2 つに分類される．1 つ目は，以前のデータをログする undo 方式と呼ばれるものであり，[27, 49] 等が該当する．2 つ目は新しいデータをログする redo 方式と呼ばれるもので，[25, 26, 50] 等が該当する．また，両者を統合するアプローチも存在する [24, 28]．本研究の拡張機構部分は redo 方式に分類される．redo 方式のハードウェア・ロギング手法 [25, 26, 50] は，2.1 章で述べたように冗長な順序制約の削減に加え，上書き更新の遅延実行によるクリティカルパスの短縮を実現する．一方，本研究はセキュア NVM

システムに対し，ツリー更新の事前実行によるクリティカルパスの短縮を実現している．さらに，基本機構部分ではNVM へのすべての書き込みに対してリダイレクションによるツリー更新の事前実行手法を提案しており，トランザクションを用いないプログラムに対する高速化も実現している．

4.3 リダイレクション

NVM を対象としたメモリアクセスのリダイレクション手法を提案する研究はこれまでにも存在するが，いずれも目的が異なる．HOOP [25] は Out-of-place（OOP）にリダイレクションする redo 方式のハードウェア・ロギング手法である．redo 方式では一般に，ログに書かれた新しい値をいずれ元アドレスへ上書き更新するが，上書き更新前に同アドレスへ新たな書き込みが発生した場合，以前のログを元アドレスへ反映させてから新たな書き込みをログアドレスに対して行う．これに対し，HOOP はログ領域内の新たなログエントリに書き込みを作ることで，以前のログを反映させる必要性を排除し，書き込み量の削減を実現した．NvMR [8] は，太陽光など不安定な電源環境のもとバックアップと再計算により計算を進める間欠的コンピューティング・システムにおいて，バックアップからの復元時の WAR/WAW ハザードを回避する目的で，メモリアクセスのリネーミングを提案した．これらと異なり，本研究の目的は，一時的なメモリアクセス・リダイレクションによるツリー更新の事前実行である．

5. 結論

NVM において，セキュリティ対策と原子永続的一貫性の保証は重要である．本研究では，幅広いプラットフォームに適用できるよう最低限の予備電力と永続型レジスタを使用しつつ，高レイテンシなツリー更新をクリティカルパスから排除するとともに，原子永続的トランザクション実行に係るプログラマの負担を軽減することを目的とした．本稿は一時的メモリアクセス・リダイレクション機構を提案し，ツリー更新の完全な事前実行を実現した．これによりクリティカルパスを最大 800 サイクル短縮した．さらに，原子永続的トランザクション実行のハードウェア支援が，同機構の軽量な拡張により可能であることを示した．プログラマはトランザクションのプログラム領域を指定するだけで済み，ログ管理や永続化処理，書き込み順序の保証といった NVM 独自のプログラミング方式に伴う負担を軽減するとともに，潜在的な永続性バグの可能性を排除することができた．今後は提案手法や先行研究を実装し，定量的に評価する．

謝辞 本研究の一部は，JSPS 科研費 19H04075 および 18H05288 の支援により行われたものである．

参考文献

[1] Intel: Intel and micron produce breakthrough memory technology, `https://newsroom.intel.com/news-releases/intel-and-micron-produce-breakthrough-memory-technology` (2015).
[2] 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* (2018).
[3] Xia, F., Jiang, D., Xiong, J. and Sun, N.: HiKV: A Hybrid Index Key-Value Store for DRAM-NVM Memory Systems, *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pp. 349–362 (2017).
[4] Wu, X., Ni, F., Zhang, L., Wang, Y., Ren, Y., Hack, M., Shao, Z. and Jiang, S.: NVMcached: An NVM-Based Key-Value Cache, *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems* (2016).
[5] Arulraj, J. and Pavlo, A.: How to Build a Non-Volatile Memory Database Management System, *Proceedings of the 2017 ACM International Conference on Management of Data*, pp. 1753–1758 (2017).
[6] Arulraj, J., Pavlo, A. and Dulloor, S. R.: Let's talk about storage & recovery methods for non-volatile memory database systems, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 707–722 (2015).
[7] van Renen, A., Leis, V., Kemper, A., Neumann, T., Hashida, T., Oe, K., Doi, Y., Harada, L. and Sato, M.: Managing Non-Volatile Memory in Database Systems, *Proceedings of the 2018 International Conference on Management of Data*, pp. 1541–1555 (2018).
[8] Bhattacharyya, A., Somashekhar, A. and Miguel, J. S.: NvMR: non-volatile memory renaming for intermittent computing, *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pp. 1–13 (2022).
[9] Balsamo, D., Weddell, A. S., Das, A., Arreola, A. R., Brunelli, D., Al-Hashimi, B. M., Merrett, G. V. and Benini, L.: Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 35, No. 12, pp. 1968–1980 (2016).
[10] Balsamo, D., Weddell, A. S., Merrett, G. V., Al-Hashimi, B. M., Brunelli, D. and Benini, L.: Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems, *IEEE Embedded Systems Letters*, Vol. 7, No. 1, pp. 15–18 (2014).
[11] Lehman, T. S., Hilton, A. D. and Lee, B. C.: PoisonIvy: Safe speculation for secure memory, *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–13 (2016).
[12] Yang, F., Chen, Y., Mao, H., Lu, Y. and Shu, J.: ShieldNVM: An Efficient and Fast Recoverable System for Secure Non-Volatile Memory, *ACM Transactions on Storage*, Vol. 16, No. 2, pp. 1–31 (2020).
[13] Rogers, B., Chhabra, S., Prvulovic, M. and Solihin, Y.: Using address independent seed encryption and bonsai merkle trees to make secure processors os-and performance-friendly, *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 183–196 (2007).
[14] Han, X., Tuck, J. and Awad, A.: Dolos: Improving the performance of persistent applications in ADR-

supported secure memory, *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 1241–1253 (2021).

[15] Mahar, S., Liu, S., Seemakhupt, K., Young, V. and Khan, S.: Write Prediction for Persistent Memory Systems, *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, IEEE, pp. 242–257 (2021).

[16] Neal, I., Quinn, A. and Kasikci, B.: Hippocrates: Healing persistent memory bugs without doing any harm, *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 401–414 (2021).

[17] Gorjiara, H., Xu, G. H. and Demsky, B.: Jaaru: Efficiently model checking persistent memory programs, *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 415–428 (2021).

[18] Hoseinzadeh, M. and Swanson, S.: Corundum: Statically-enforced persistent memory safety, *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 429–442 (2021).

[19] Di, B., Liu, J., Chen, H. and Li, D.: Fast, flexible, and comprehensive bug detection for persistent memory programs, *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 503–516 (2021).

[20] Liu, S., Mahar, S., Ray, B. and Khan, S.: PMFuzz: Test case generation for persistent memory programs, *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 487–502 (2021).

[21] Scargall, S.: Persistent memory architecture, *Programming Persistent Memory*, Springer, pp. 11–30 (2020).

[22] pmem.io: Persistent Memory Development Kit, `https://pmem.io/pmdk/` (2017).

[23] Gogte, V., Wang, W., Diestelhorst, S., Chen, P. M., Narayanasamy, S. and Wenisch, T. F.: Relaxed persist ordering using strand persistency, *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, pp. 652–665 (2020).

[24] Wei, X., Feng, D., Tong, W., Liu, J. and Ye, L.: MorLog: Morphable hardware logging for atomic persistence in non-volatile main memory, *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 610–623 (2020).

[25] Cai, M., Coats, C. C. and Huang, J.: Hoop: efficient hardware-assisted out-of-place update for non-volatile memory, *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 584–596 (2020).

[26] Jeong, J., Park, C. H., Huh, J. and Maeng, S.: Efficient hardware-assisted logging with asynchronous and direct-update for persistent memory, *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 520–532 (2018).

[27] Joshi, A., Nagarajan, V., Viglas, S. and Cintra, M.: ATOM: Atomic durability in non-volatile memory through hardware logging, *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 361–372 (2017).

[28] Ogleari, M. A., Miller, E. L. and Zhao, J.: Steal but no force: Efficient hardware undo+ redo logging for persistent memory systems, *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 336–349 (2018).

[29] Rudoff, A.: Deprecating the pcommit instruction, `https://software.intel.com/en-us/blogs/2016/09/12/deprecate-pcommit-instruction` (2016).

[30] SNIA: NVDIMM Messaging and FAQ (2014).

[31] Scargall, S.: *Programming persistent memory: A comprehensive guide for developers*, Springer Nature (2020).

[32] Wang, Z., Liu, X., Yang, J., Michailidis, T., Swanson, S. and Zhao, J.: Characterizing and Modeling Non-Volatile Memory Systems, *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 496–508 (2020).

[33] Ye, M., Hughes, C. and Awad, A.: Osiris: A Low-Cost Mechanism to Enable Restoration of Secure Non-Volatile Memories, *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 403–415 (2018).

[34] Zubair, K. A. and Awad, A.: Anubis: ultra-low overhead and recovery time for secure non-volatile memories, *Proceedings of the ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pp. 157–168 (2019).

[35] Freij, A., Yuan, S., Zhou, H. and Solihin, Y.: Persist Level Parallelism: Streamlining Integrity Tree Updates for Secure Persistent Memory, *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 14–27 (2020).

[36] Awad, A., Ye, M., Solihin, Y., Njilla, L. and Zubair, K. A.: Triad-nvm: Persistency for integrity-protected and encrypted non-volatile memories, *Proceedings of the ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pp. 104–115 (2019).

[37] Liu, S., Kolli, A., Ren, J. and Khan, S.: Crash consistency in encrypted non-volatile main memory systems, *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 310–323 (2018).

[38] Yan, C., Englender, D., Prvulovic, M., Rogers, B. and Solihin, Y.: Improving Cost, Performance, and Security of Memory Encryption and Authentication, *Proceedings of the 33rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 179–190 (2006).

[39] Liu, S., Seemakhupt, K., Pekhimenko, G., Kolli, A. and Khan, S.: Janus: Optimizing memory and storage support for non-volatile memory systems, *Proceedings of the ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pp. 143–156 (2019).

[40] Costan, V. and Devadas, S.: Intel SGX explained, *Cryptology ePrint Archive* (2016).

[41] Alwadi, M., Zubair, K., Mohaisen, D. and Awad, A.: Phoenix: Towards ultra-low overhead, recoverable, and persistently secure nvm, *IEEE Transactions on Dependable and Secure Computing* (2020).

[42] Huang, J. and Hua, Y.: Update the Root of Integrity Tree in Secure Non-Volatile Memory Systems with Low Overhead, *arXiv preprint arXiv:2103.03502* (2021).

[43] Lei, M., Li, F., Wang, F., Feng, D., Zou, X. and Xiao, R.: SecNVM: An Efficient and Write-Friendly Metadata Crash Consistency Scheme for Secure NVM, *ACM Transactions on Architecture and Code Optimization (TACO)*, Vol. 19, No. 1, pp. 1–26 (2021).

[44] Chen, Z., Zhang, Y. and Xiao, N.: CacheTree: Reducing Integrity Verification Overhead of Secure Non-Volatile Memories, *IEEE Transactions on Computer-Aided De-*

sign of Integrated Circuits and Systems (2020).
[45] 小池　亮，高前田伸也：セキュアな不揮発性メモリのクラッシュ一貫性支援の高速化，研究報告システム・アーキテクチャ (ARC)， Vol. 2021, No. 7, pp. 1–10 (2021).
[46] Rakshit, J. and Mohanram, K.: Assure: Authentication scheme for secure energy efficient non-volatile memories, *Proceedings of the 54th Annual Design Automation Conference 2017*, pp. 1–6 (2017).
[47] Freij, A., Zhou, H. and Solihin, Y.: Bonsai merkle forests: Efficiently achieving crash consistency in secure persistent memory, *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 1227–1240 (2021).
[48] Alshboul, M., Ramrakhyani, P., Wang, W., Tuck, J. and Solihin, Y.: Bbb: Simplifying persistent programming using battery-backed buffers, *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, IEEE, pp. 111–124 (2021).
[49] Shin, S., Tirukkovalluri, S. K., Tuck, J. and Solihin, Y.: Proteus: A flexible and fast software supported hardware logging approach for nvm, *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 178–190 (2017).
[50] Doshi, K., Giles, E. and Varman, P.: Atomic persistence for SCM with a non-intrusive backend controller, *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 77–89 (2016).