Vol.2021-ARC-245 No.7
2021/7/20

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

小池 亮[1,a)]　高前田 伸也[1,b)]

概要：バイトアドレスの不揮発性メモリ (NVM) は，改竄に対して従来の揮発性メモリより脆弱である．データの整合性を保証し改竄を検知する機構として， Bonsai Merkle Tree (BMT) がよく用いられる．しかし，NVM への書き込みの都度 BMT を葉から根まで更新するオーバーヘッドは，書き込みのボトルネックとなるほど大きい．さらに，突然のクラッシュや電源遮断後のデータの一貫性保持のため，NVM ではライト・アヘッド・ロギングと呼ばれる手法が広く使われているが，これは本来のデータ更新に先立ちログを NVM 上に書き込むので，BMT 更新のオーバーヘッドを 2 倍にしてしまう．この問題を解決するため，本研究はセキュアな NVM 用のハードウェア・ロギング・アーキテクチャを提案する．これは，以下の 2 つの発見に基づいている．1) NVM の膨大なサイズのために，ログの BMT 更新パス同士は根より遥かに低いノードで合流する．2) データの上書き更新用の BMT 更新については，ライト・ペンディング・キュー (WPQ) での永続化が実は BMT 更新完了の瞬間まで不必要である．そこで，ログ用に BMT を分離し，ログ用の BMT 更新レイテンシを 64%削減した．また，WPQ への挿入を遅らせることで，上書き更新用の BMT 更新レイテンシを 91%削減した．評価の結果，提案アーキテクチャは最先端のアーキテクチャより最大で 2.8 倍高速にトランザクションを処理できることを確認した．

1. 序論

バイトアドレスの不揮発性メモリ（NVM）が注目を集めている [1–5]．DRAM に匹敵するレイテンシ，HDD/SSD のような永続性，テラバイト級の容量などを特徴とし，NVM は，永続型のキーバリューストア [6–8] やデータベースシステム [9–11] への応用が期待されている [12]．

しかし，実運用に際して，2 つの基本的な課題が存在する．1 つ目はセキュリティである．NVM はシステムから取り出した後も中身が揮発せず残るため，従来の揮発性メモリに比べて，盗聴や改竄といった悪意ある攻撃に脆弱である．これらの攻撃から保護するため，通常，暗号化に加えてメッセージ認証符号（MAC）と呼ばれるハッシュと，Bonsai Merkle Tree（BMT）[13] と呼ばれるハッシュ木を用いて，整合性を保証する．ここで，BMT は暗号化カウンタを葉とするハッシュ木であり，根のみオンチップに安全かつ永続的に保存する．この機構では，任意のノードや葉（暗号化カウンタ）の改竄は，計算された根との不一致から検知可能である．

NVM からの読み出し時の整合性保証が BMT の根に依存しているため，NVM への書き込み (persist とも言う) を行う際には，必ず BMT 全体を更新する必要がある．しかし，BMT の更新処理は，レベル間の依存関係により，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．さらに，NVM の容量はテラバイト級と膨大なので，大規模で深い BMT が必要となる．実際，80 サイクルのハッシュ・レイテンシを持つ実用的な 11 レベルの BMT [13–15] では，各 persist につき 880 サイクルも BMT の更新に必要となる．

この高コストな BMT の更新処理は，2 つ目の課題であるクラッシュ一貫性のために，各 persist に対して 2 回行う必要がある．アトミックな一連の書き込みの最中に停電やシステムクラッシュが発生するような万一の場合に備え，NVM プログラムは通常，ライト・アヘッド・ロギングと呼ばれる 2 段階の機構によってトランザクション実行するためである．この機構では，第 1 段階でデータのコピーを「ログ」した後，第 2 段階でデータを「上書き更新」する．これにより，コミット前のトランザクションについては，ログか本来のデータが完全の状態に保たれる．したがって，クラッシュ後には変更の全適用か完全なロールバックにより，一貫性を回復できる．しかし，本来のデータに加えてログも NVM に書き込むため，それぞれについて BMT 更新が必要となり，高コストな BMT 更新のオーバーヘッドが 2 倍になってしまう．

そこで本稿では，セキュアな NVM のためのハードウェ

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

ア・ロギングのアーキテクチャを提案する．これは，BMT 更新処理を削減または隠蔽することで，2 倍に増えた BMT 更新オーバーヘッドの緩和を図るものであり，BMT 更新に関する 2 つの発見に基づいている．1）任意の「ログ」の BMT 更新パスは，根よりも遥かに下位のノードで合流する．2)「上書き更新」については，根を更新する瞬間まで，メモリコントローラのライト・ペンディング・キュー（WPQ）のエントリを消費せずとも BMT 更新を進めることができる．これらの発見から，64%のログ用 BMT 更新が冗長であり，91%の上書き更新用 BMT 更新が隠蔽可能であることが明らかになった．提案手法はこの 2 つの BMT 更新タイプを区別するため，ハードウェアでログを管理する．

まとめると，本稿の貢献は以下のとおりである．

- 各 BMT 更新タイプについて，ログ用は 64%が冗長であること，上書き用は 91%が隠蔽可能なことを発見した．
- 2 種類の書き込みタイプを区別する目的で，セキュアな NVM でハードウェア・ロギングを行う，世界初のアーキテクチャを提案した．
- 提案アーキテクチャは「ログ BMT の分離」と「上書き更新用の WPQ 遅延挿入」を採用し，2.8 倍のパフォーマンス向上を達成した．

2. 背景と関連研究

2.1 脅威モデル

セキュアな NVM についての既存研究 [3,4,16,17] と同様，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竄が可能であると仮定する．また，OS に依存しない手法を提案するので，信頼できない OS 環境でも本稿の提案手法は動作する．なお，漏洩電磁波などサイドチャネル攻撃への対処は，本稿の範囲外とする．

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 [3,4,16,18]．さらに，カウンタの容量とオーバーフローへの対処のトレードオフから，キャッシュラインごとの「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」を管理する分離カウンタ方式 [19] が一般的に採用されている．本研究もそれに倣う．この方式の詳細は [19,20] を参照されたい．

2.3 整合性保証

カウンタ方式の暗号化では，改竄から NVM 上のカウンタ及び暗号化済みデータを保護する必要がある．この目的で，Bonsai Merkle Tree（BMT）[13] と呼ばれる整合性保証の方式が一般的に採用されている [3,4,16,17,21]．この方式では，暗号化カウンタの整合性をハッシュ木（BMT）で保証しつつ，暗号化されたデータの整合性を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 1** に示す．BMT は暗号化カウンタを葉とする木であり，各ノードは子のハッシュ値（MAC）である．BMT の根はオンチップに保管することで改竄を防ぐ．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などを使用する．もしくは，[4,16] 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．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 を用いて $M_A = MAC_K(C_A, \gamma_A)$ 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 を改竄する必要がある．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竄しなければならず，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．

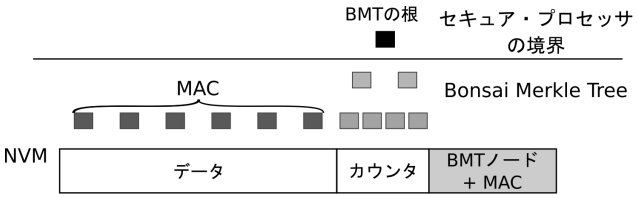

図 1 BMT [13] による整合性保証
Fig. 1 Integrity verification using BMT [13].

なお，暗号文とカウンタの全体をハッシュ木で保護する代わりに MAC を利用するのは，BMT の高さを減らすためである．NVM からの読み込み時に整合性を検証する際，根へ至るパス上のノードすべてについて並列にハッシュを計算して合致を確認すればよい一方で，NVM への書き込みの際には，依存関係により葉から根まで逐次的にハッシュ計算を進める必要があるからである．このオーバーヘッドは非常に大きく，80 サイクルのハッシュ・レイテンシを持つ実用的な 11 レベルの BMT [13–15] では，葉から根への BMT の更新には，各 persist につき 880 サイクルも必要となる．

2.4 ライト・アヘッド・ロギング

NVM は不揮発性を備えるが，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すると，データの一貫性が失われてしまう．そこで，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 [1,2,12,18,22,23]．これは，データをログとして永続化したのちデータの上書きを行う，という 2 段階の手法である．ログの中身に応じて 2 つの手法に大きく分類される．1 つ目は，以前のデータをログするものであり，undo 方式と呼ばれる．undo 方式ではクラッシュ後はログからロールバックすればよい．2 つ目は新しいデータをログするものであり，redo 方式と呼ばれる．redo 方式では，クラッシュ時にはログを廃棄もしくは全適用する．

2.5 ライト・ペンディング・キュー（WPQ）

ライト・ペンディング・キュー（WPQ）とはメモリコントローラ上の NVM への書き込みのためのバッファである．これは，非同期 DRAM リフレッシュ（ADR）[24] と呼ばれる標準機構により，永続性が保証されている．したがって，レジスタやキャッシュは揮発であるが，メモリコントローラ以降（WPQ と NVM）は永続性を備える．しかし，WPQ のエントリ数はスーパーキャパシタの容量によって制限され，大規模化が困難である．例えば，商用の NVM システムの WPQ サイズは 8 エントリにとどまる [25]．

WPQ は暗号文および MAC の生成，BMT の更新を待つ間，書き込みが保存される場所である．これには 2 つの理由がある．まず，暗号文，カウンタ，MAC，BMT の根の 4 つ組がアトミックに永続化される必要があるためである．これにより，NVM への書き込みが BMT 更新の完了前に行われることは禁じられる．しかし，オンチップに留めれば良いので，書き込みのパケットを揮発性のバッファに保持しても，この要件は満たされる．2 つ目の理由は，`clwb` 命令や `sfence` 命令によって規定される永続化プログラミングモデルが，書き込みのパケットがメモリコントローラに到着した時点で，永続化の保証が得られたものとみなすためである．これらの 2 つの要件により，書き込みパケットは 1) オンチップかつ 2) 永続的な場所，すなわちメモリコントローラの WPQ に存在することが必要であり，ここで暗号文・MAC の生成と BMT 更新を待つ．

2.6 関連研究

セキュアな NVM でのクラッシュ一貫性供与の高速化に関する最先端の研究は Freij ら [3] によるものである．彼らは，エポック，すなわち persist 同士が順序づけされていないプログラム領域を考慮し，最適化した BMT 更新ロジックを提案した．エポックは，たとえば *sfence* 命令によりプログラマに指定される（**図 2**-(a) を参照）．トランザクション・ベースの一貫性担保手法では，エポックの境界（すなわち *sfence*）でのみ一貫性が担保されていれば良いので，彼らは 1) エポックをまたぐ persist についてのレベルごとのパイプライン化，2) エポック内の persist についてのアウト・オブ・オーダー更新を提案した．

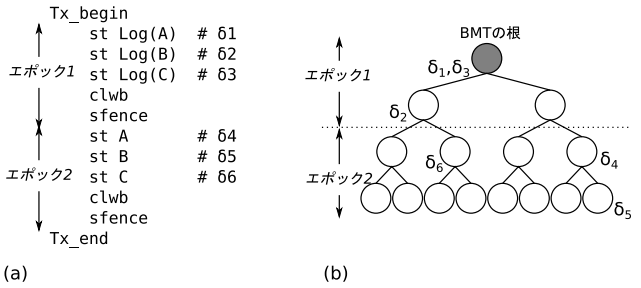

図 2 (a) プログラム例．(b) 最先端の BMT 更新メカニズム [3]．
Fig. 2 (a) Example program. (b) State-of-the-art BMT update mechanisms [3].

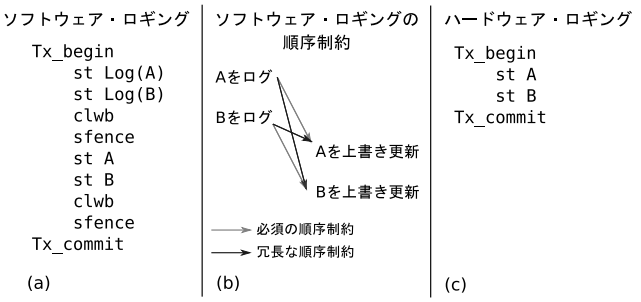

図 3 (a) ソフトウェア・ロギングの例．(b) *sfence* による順序制約．(c) ハードウェア・ロギングの例．
Fig. 3 (a) Example of a software logging program. (b) Ordering constraints specified by *sfence*. (c) Example of a hardware logging program.

図 2-(b) にこの手法を示す．ここで，δ_1 から δ_3 の persist は同一エポックに属するので，最適化 2) によりアウト・オブ・オーダーに BMT 更新が進む．δ_4 から δ_6 についても同様である．一方，δ_4 から δ_6 は以前のエポックたち（ここではエポック 1）のいずれより上のレベルに進んではならない（最適化 1) に反する）．このようにして，彼らの手法は *sfence* に起因する persist 及び BMT 更新の順序制約を削減したものの，依然それに縛られている．

セキュアでない NVM については，そ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 [1,2,22,26,27]．**図 3**-(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，というのが目的のひとつである．他の動機には，プログラマのログ管理の負担を排除することも含まれる（図 3-(a) および図 3-(c) を参照）．

3. 動機

セキュアな NVM でのハードウェア・ロギングのパフォーマンスを測定したところ，ソフトウェア・ロギング [3] より遅い場合があると判明した．**図 4** に，異なるデータセットの大きさに対する，undo（`Undo`）及び redo（`Redo`）方式のハードウェア・ロギングとソフトウェア・ロギング

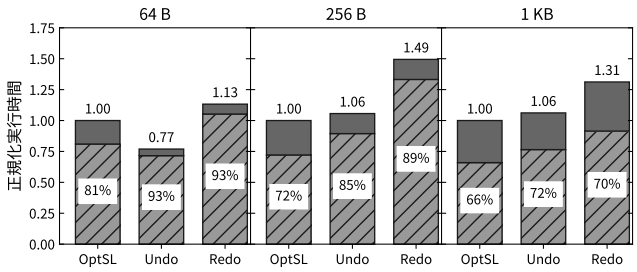

図 4 異なるデータサイズに対するトランザクション実行時間．斜線部は WPQ がいっぱいになり新規リクエストをブロックした時間を表す．

Fig. 4 Normalized transaction execution time for small, medium, and large dataset size. Hatched part of each bar corresponds to the time WPQ blocks any incoming requests due to overfilling.

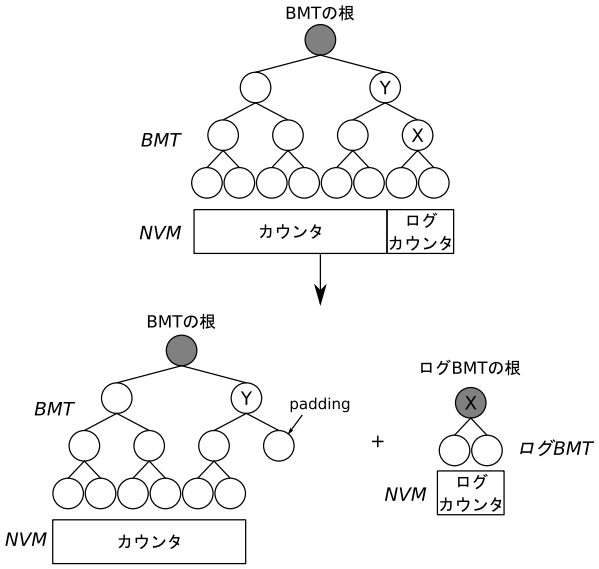

Fig. 4 ログ BMT の分離．

Fig. 4 BMT separation for logs.

（OptSL）のトランザクション実行時間を示す．評価方法の詳細は 6.1 節を参照されたい．

理由は 2 つ考えられる．1 つ目は，セキュアな NVM においては，もはやログ命令の実行ではなく BMT 更新にボトルネックが存在しているため．2 つ目は，ハードウェア・ロギングではログのメタデータをハードウェアが付加するので，プログラマが最適にメタデータを付加できるソフトウェア・ロギングと比べると，persist と BMT 更新の量が増えてしまうためである．

同図に，WPQ に空きがなくなり新しいリクエストをブロックした時間の比率を斜線で示している．このような過負荷はトランザクション実行をストールするので，クリティカルパス延長につながる．注目すべきことに，実行時間の 72%以上をそのような WPQ ブロック時間が占めている．一番簡単には，WPQ のサイズを大きくすることでこれを解決できると予想されるが，NVM では先述のようにスーパーキャパシタの要件があるので，現実的ではない．そこで，ブロック時間の比率を下げるために，BMT 更新のオーバーヘッドを削減することが求められる．そもそも，BMT 更新はログ用とデータの上書き更新用とで 2 回行われるのであった．本研究では，この 2 重苦を緩和することで WPQ の負荷を減らし，現実的な WPQ サイズでトランザクション実行を高速化することを目的とする．

4. 提案手法

提案手法の鍵は，2 種類の persist，すなわちログと上書き更新のそれぞれの特徴を踏まえて，BMT 更新を別々に処理する点である．

4.1 ログ BMT の分離

ハードウェア・ロギングの先行研究 [1, 22] に倣い，本稿も NVM の一部を，メモリコントローラからのみアクセス可能なログ専用領域として割り当てることにする．大抵のトランザクションはキロバイト未満のフットプリントしかない [1] ことを考えると，ログ領域は数メガバイトで十分である．したがって，任意のログに対する BMT 更新のパスは，後半の大部分が共通であることになる．**図 4.1** の上半分で，X，Y および根は，ログが作られるたびに更新される．そこで，すべてのログの更新パスが通る最初のノード（同図では X）で BMT を分離すれば，X より上の冗長な更新を省くことができる．このとき，同図の下半分のように，空いた部分には定数のダミーを置く．整合性の検証をどちらの BMT で行うかは，ログ用のものかどうかをアドレスから判断して決定する．

実装上の複雑性はほとんど増加しないが，大きなパフォーマンス向上につながる．4 KB のページにつき 64 B のカウンタをもつ典型的な分離カウンタ方式 [19] で 4 TB の NVM に対して 8 分木の BMT を構成すると $\log_8\left(\left(4\times 2^{40}\right)/\left(4\times 2^{10}\right)\right)+1 = 11$ 段になるが，2 MB のログ領域をカバーするだけなら $\log_8\left(\left(2\times 2^{20}\right)/\left(4\times 2^{10}\right)\right)+1 = 4$ 段で済む．これは 7 段分の削減であり，80 サイクルの MAC 計算のもと [13–15] 各 persist について 560 サイクル早く NVM に書き込むことが可能になる．

4.2 上書き更新用の WPQ 遅延挿入

上記のとおり，WPQ の空きがなくなる主な要因は，WPQ エントリに対する高コストな BMT 更新であった．BMT 更新中を WPQ で待つのは，2.5 節で述べたとおりアトミック性と永続性が理由であった．ここで，ひとつの重要な気づきは，redo 方式の上書き更新パケットは，全く同じ内容がすでにログされているので，永続性がすでに満たされている，というものである．したがって，上書き更新パケットは揮発性のオンチップ・バッファに置ける．そこで，redo 方式を採用し，上書き更新パケットを揮発性のリタ

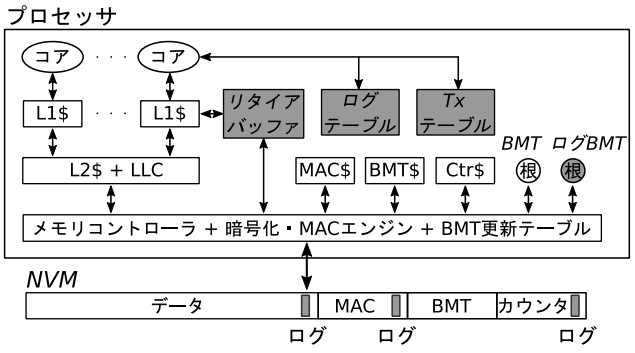

図 5 提案アーキテクチャ．
Fig. 5 Proposed architecture.

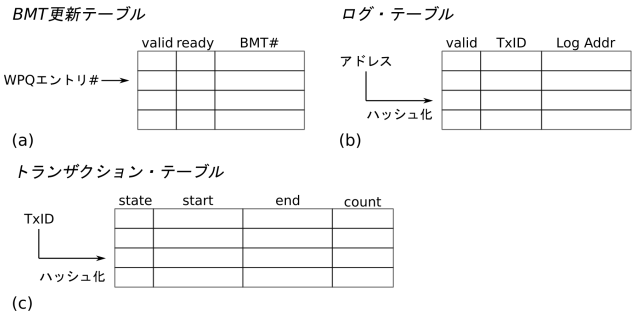

図 6 (a) BMT 更新テーブル．(b) ログ・テーブル．(c) トランザクション・テーブル．
Fig. 6 (a) BMT Table. (b) Log Table. (c) Transaction Table.

イア・バッファに保持している間に BMT 更新を開始することで，WPQ への挿入を遅延させることを提案する．究極的には，BMT の根が新しくなる瞬間まで遅延可能であるが，その瞬間に WPQ に空きがないと MAC 計算のパイプラインがストールしてしまうので，根の更新を開始するタイミングで WPQ に挿入することにする．この場合においても，$11-1=10$ 段分の BMT 更新を隠蔽可能であり，$10/11=91\%$もしくは 800 サイクルの節約に相当する．

5. 提案アーキテクチャ

前節の 2 つの提案を実現する，redo 方式のハードウェア・ロギング・アーキテクチャを説明する．

5.1 概要

図 5 に全体像を示す．先行研究 [3, 4, 16, 17] に倣い，セキュリティ・メタデータ，すなわち MAC，BMT ノード，暗号化カウンタについて，別々のキャッシュを備える．同図において，濃く塗られている部分がハードウェア・ロギングを実現するために必要な追加機構である．これには，たとえばログ BMT の根が含まれる．NVM にログ専用の領域を確保しているので，対応する MAC やカウンタの領域もログ専用となる．

5.2 BMT 更新管理

WPQ のエントリは，BMT 更新が根まで完了し，すべてのセキュリティ・メタデータが揃うまで，NVM へ書き込まれてはならないロックされた状態となる．BMT 更新の状態は，「BMT 更新テーブル」に記録される．このテーブルは先行研究 [3] のものに近いが，後で見るように redo 方式のハードウェア・ロギングでは WPQ エントリ間に順序関係が存在しないので，より単純に実現できる．

各 WPQ エントリに対して，対応する BMT テーブルのエントリは WPQ のエントリ番号で指定される．**図 6**-(a) にあるように，各 BMT テーブルのエントリは `valid` ビット，`ready` ビット，BMT ノード番号 (`BMT#`) で構成される．`valid` ビットは当該エントリの有効性を示すだけでなく，当該 WPQ エントリのロックの状態も意味する．`ready` ビットは現在の BMT ノードでの MAC 計算が完了したときに立てられ，このとき次のレベルへ BMT 更新プロセスは進むことが可能である．`BMT#`が現在のノード番号を示す．

5.3 ログのリダイレクション

redo 方式では，ログされたデータのうち，上書き更新が未完了のデータに対するリダイレクション機構が必要である．これには「ログ・テーブル」を用いる．図 6-(b) に図示されたように，これは，（元の）アドレスを指定すると，ログのアドレスとトランザクション ID（`TxID`）を返すハッシュテーブルである．キャッシュとは異なり，衝突時に犠牲となったエントリのログデータは，上書き更新ののち新たなエントリに入れ替わる．

ログの有効性を判断するために `TxID` フィールドが必要となる．なお，中止されたトランザクションのログは無効である．トランザクションの状態を記録するために，「トランザクション・テーブル」を使用する．これはハッシュテーブルであり，入力 `TxID` に対して，そのトランザクションの状態 `state`（完了・実行中・無効），開始アドレス `start`，終了アドレス `end`，個数 `count` を返す（図 6-(c)）．`state` フィールドは，そのトランザクションが開始されると「実行中」に設定され，コミット後に「完了」に設定される．トランザクションが中止された場合は「無効」となる．次に説明するように，その他のフィールドはログ管理に使用される．

5.4 設計上の議論

ガベージコレクション

NVM のログ領域にある redo ログエントリは，上書き更新後に回収される．これを行うには 2 つの方法が考えられる．1 つ目は，「ログアドレス順に処理する」というものである．しかし，この方法では，実行中のトランザクションのログを処理しないよう注意が必要になる．ログの容量を節約するためにログを順番に書き込んでいるので，同時進行中のトランザクションのログがオーバーラップする可能

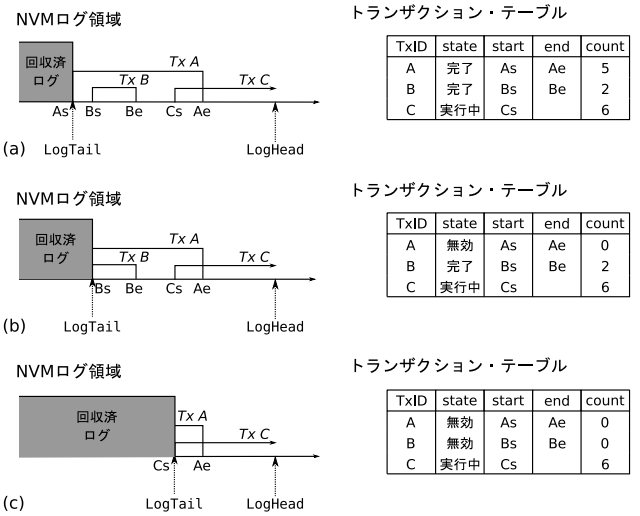

図 7 [1] のログ管理手法．(a) 初期状態．(b) トランザクション A のデータ上書き更新完了後．(c) トランザクション B のデータ上書き更新完了後．

Fig. 7 Log management mechanism from [1]. (a) Initial state. (b) After completion of in-place updates for transaction A. (c) After completion of in-place updates for transaction B.

性も十分にある．このような際に，この方法では未完了のトランザクションのログに至ったら，上書き更新処理を停止しなければならず，上書き更新処理のスループット低下に繋がりうる．

もう 1 つのアプローチは，「トランザクションごとに処理する」というものである．これは，他のトランザクションの状態に関わらず，そのトランザクションが完了してさえいればログをリタイアできるという点で優れる．外部フラグメンテーションを避けるため，既存の redo 方式のハードウェア・ロギング [1] に使われているログ管理手法を採用することにする．

この手法では，トランザクションのログアドレスの範囲およびログの個数を，トランザクション・テーブルの `start`，`end`，`count` フィールドに記録する．上書き更新処理では，「完了」もしくは「実行中」のトランザクションのうち，最小の `start` アドレスを保持するポインタ `LogTail` が存在すると仮定する（このトランザクションを `A` と呼ぶ）．トランザクション `A` のログを一つずつ，ログ・テーブルとリタイア・バッファを使って上書き更新し，退避させていく．その都度，トランザクション `A` の `count` フィールドを一つ減らす．`count` が 0 になったら，トランザクション `A` の `state` を「無効」に変更する．この時点で，（更新後の）トランザクション・テーブルの最小の `start` アドレスと `LogTail` の間には，有効なログエントリが存在しないことが保証されているので，`LogTail` をこのアドレスに進めることができる．この仕組みを図で表したのが，**図 7** である．

BMT 更新の順序

redo 方式ではログはいずれ上書き更新に使われる．そのような上書き更新は，当該トランザクションのログが全て WPQ に入った時点で開始できる．しかし，重要なことに，開始タイミングを当該トランザクションのログが全て WPQ から出て NVM に書かれたあとに限定することで，WPQ 内での BMT 更新の順序を排除可能である．

リカバリ

電源喪失後は，すべての揮発性キャッシュやテーブルの中身は失われてしまう．そこで，BMT の根とログ BMT の根に加えて，トランザクション・テーブルの `state` フィールドも不揮発性であると仮定する．というのも，これは NVM のログが元アドレスのデータの上書き更新に利用されるべきかを判断するのに必要だからである．リカバリ・プロセスは NVM のログ領域を走査し，「完了」状態のログは反映させ，そうでないものは破棄する．

セキュリティ

オフチップの部品への盗聴や改竄を想定しているので，NVM 上のデータについてのみセキュリティを議論すればよい．データだけでなく，ログにも暗号化および整合性保証は用いられている．ログ用 BMT の根はオンチップに保持しているので，通常のデータと同様に改竄検知の仕組みが機能する．また，上書き更新用の WPQ への遅延挿入はすべてオンチップでの挙動なので，懸念はない．

6. 評価

6.1 評価手法

サイクル単位のシミュレータである gem5 [28] のシステムコール・エミュレーション（SE）モードを用いた．主要な設定は表 1 に示している．

評価には，以下の 4 つの設計を比較した．

- `OptSL`．ソフトウェア・ロギングを高速化した，最先端のアーキテクチャ [3]．
- `Undo`．undo 方式のハードウェア・ロギング・アーキテクチャ [2] のセキュアな NVM への拡張．
- `Redo`．redo 方式のハードウェア・ロギング・アーキテクチャ [26] のセキュアな NVM への拡張．
- `Ours`．提案アーキテクチャ．

評価に際し，全てのセキュリティ・メタデータがキャッシュにヒットすると仮定して単純化している．しかし，セキュリティ・メタデータは通常のデータより局所性が高いので本来のパフォーマンスへの影響は小さい．また，全ての設計にこの方針を適用しているので，公平である．同様に，セキュリティ・メタデータの NVM への書き込みのパケットのシミュレーションは省略している．

以上の 4 つの設計に対し，異なるアクセスパターンを示す 4 つのワークロードを実行した．これらは，先行研究でも頻繁に利用されている [1, 15, 22, 26, 29]．`OptSL` 用には

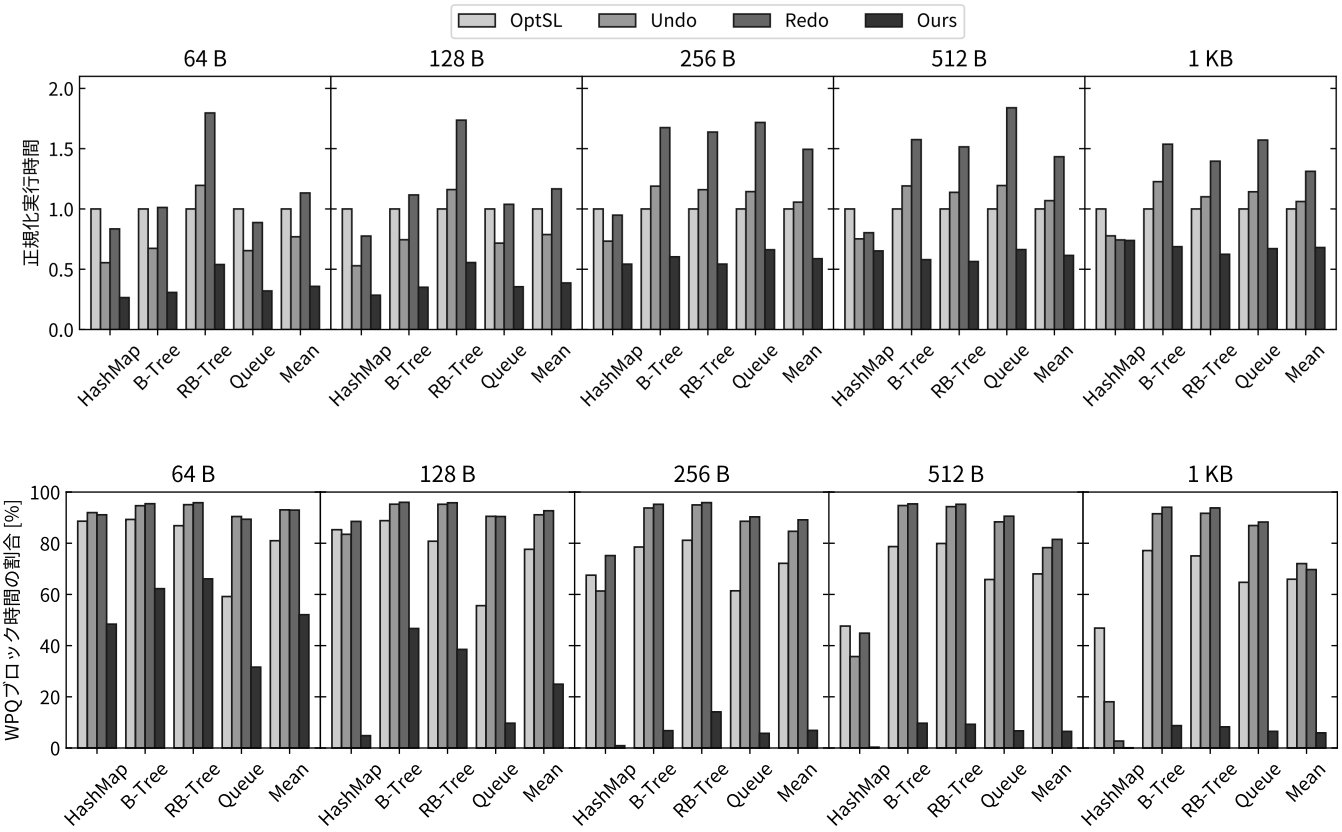

図 8 異なるデータサイズに対する正規化実行時間（上段）と WPQ ブロック時間の割合（下段）.

Fig. 8 Normalized execution time (top) and write request block time ratio (bottom) for varying dataset size.

表 1 シミュレータの設定.

Table 1 Simulator configuration.

プロセッサ	
CPU	8 コア, アウト・オブ・オーダー, 2 GHz
L1 命令キャッシュ	プライベート, 32 KB, 8 ウェイ, 2 サイクル・アクセス
L1 データキャッシュ	プライベート, 64 KB, 8 ウェイ, 2 サイクル・アクセス
L2 キャッシュ	共有, 256 KB, 8 ウェイ, 8 サイクル・アクセス
L3 キャッシュ	共有, 8 MB, 16 ウェイ, 25 サイクル・アクセス
メモリコントローラ	
WPQ サイズ	{4, 8, 16, 32, 64} エントリ（デフォルト: 8 エントリ [25]）
NVM	
NVM	シングル・チャネル, 読み出し 50 ns, 書き込み 150 ns [1,2,26]
セキュリティ	
暗号化	24 サイクル [16,29,30]
MAC	80 サイクル [13–15]
BMT	11 段, 8 分木
提案アーキテクチャ	
ログ BMT	5 段, 8 分木

表 2 ワークロード.

Table 2 Workloads.

HashMap	ハッシュテーブルへのランダムな挿入・更新.
B-Tree	二分探索木へのランダムな挿入・更新.
RB-Tree	赤黒木へのランダムな挿入・更新.
Queue	キューへのランダムな挿入・削除.

ログのための命令があるが，他はハードウェア・ロギングのアーキテクチャなので，それらは存在しない．デフォルトで 8 つの独立したプロセスとしてワークロードをシミュレータ上の各コアで実行した．プロセス数を変えたときの挙動も最後に評価している．

6.2 評価結果

実行時間

図 8 の上段に，正規化された実行時間を示す．グラフから，どのデータサイズに対しても，提案アーキテクチャは先行研究 `OptSL` より短い平均実行時間を達成していることがわかる．減少率は 33.0%（1 KB）から 64.2%（64 B）である．対して，ハードウェア・ロギングのアーキテクチャである `Undo` と `Redo` は，`OptSL` と同等かやや劣るパフォーマンスを示している．この理由は 3 章で述べたように，ログのメタデータが影響していると考えられる．提案アーキ

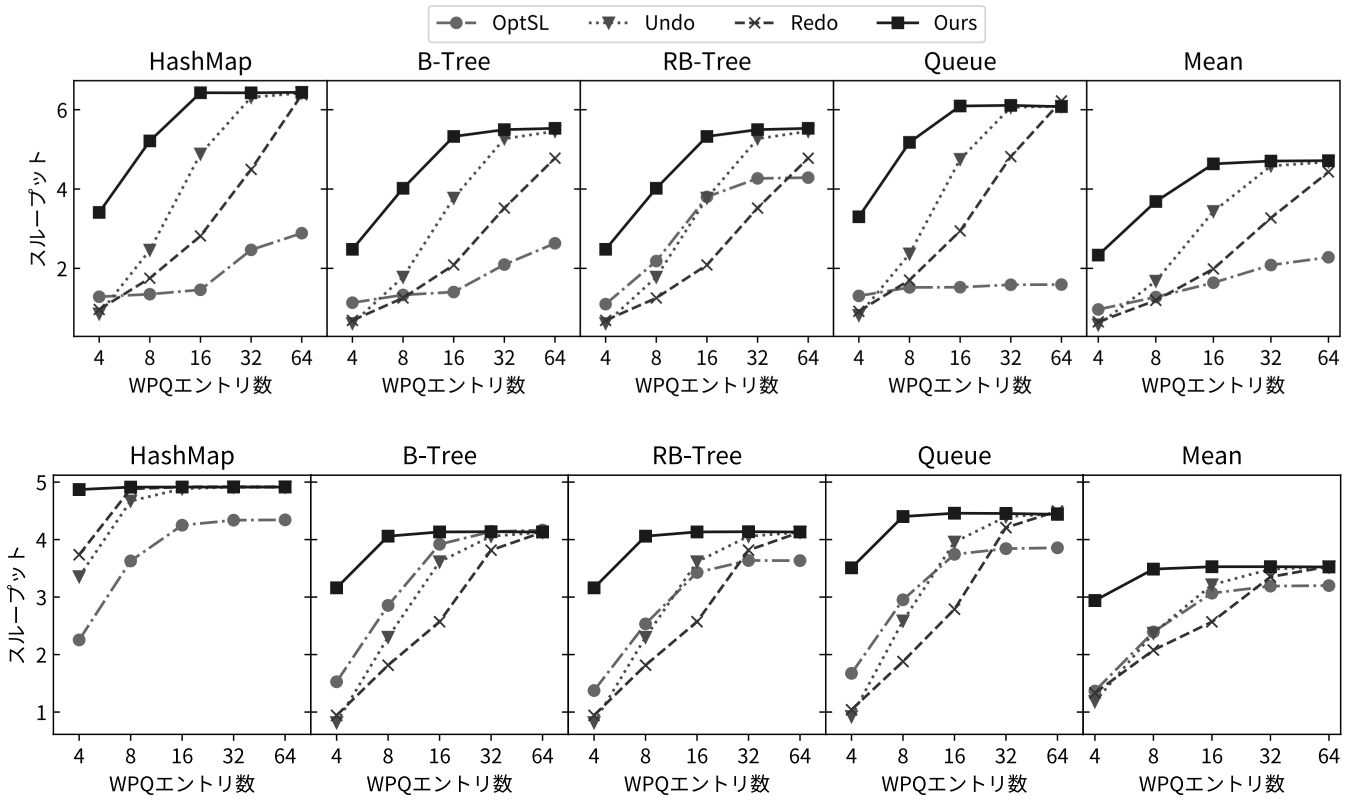

図 9 トランザクション・スループット．上段は小さいデータサイズ，下段は大きいデータサイズのワークロード．

Fig. 9 Transaction throughput for small (top) and large (bottom) workloads.

テクチャも redo 方式のハードウェア・ロギングなので同様の傾向が見られるものの，2 つの提案手法の導入の効果が上回り，`OptSL` から 2.8 倍，Redo から 3.2 倍の高速化を実現している．

WPQ ブロック時間

同図の下段に，WPQ がいっぱいになり新規リクエストをブロックした時間が総実行時間に占める割合を示している．この割合は小さいデータのトランザクションで顕著に大きく，`OptSL`/`Undo`/`Redo` では最大で 80%を超す．一方，2 つの提案手法により，提案アーキテクチャは WPQ ブロック時間の割合を著しく減少させている．

WPQ サイズの影響

図 9 は各ワークロードのトランザクション・スループットに対する WPQ サイズの影響を示している．上段が小さい（64 KB）データサイズ，下段が大きい（1 KB）ものである．予想通り，パフォーマンスは WPQ サイズが大きくなるにつれて飽和する様子が見られる．これは，WPQ サイズが大きくなれば WPQ がいっぱいになる頻度が下がり，BMT 更新のオーバーヘッドの影響が小さくなるためである．しかし，大きな WPQ サイズが現実的ではないことは再度触れておく [24]．実際，唯一の商用 NVM システムが WPQ サイズを 8 エントリに限定している [25] ことは，すでに述べたとおりである．提案アーキテクチャでは，現実的な範囲の WPQ サイズにおいて，パフォーマンスの低下の度合いが他と比べて小さく済んでいる．また，データサイズの影響を見ると，大きなデータに対してのほうが小さなデータに対してより，提案アーキテクチャがより小さな WPQ サイズで飽和パフォーマンスに近づくことも確認できる．これは，WPQ ブロック時間の割合で説明できるかもしれない．図 8 下段を見ると，データサイズが大きくなるほど割合が大きく減少しており，BMT の負荷が小さいといえる．

プロセス数の影響

プロセス数が違えば，メモリコントローラへの負荷も異なる．同時並行して実行するプロセス数を 1 から 8 まで変化させたときのスループットを**図 10** に示す．スループットの値は，ワークロードについて平均したのち `OptSL` に対して正規化している．この図から，単一プロセスであれば `Undo`/`Redo`/`Ours` のスループットは近いことがわかる．これは，WPQ への負荷が小さく，ハードウェア・ロギング間の違いが見えてこないためであろう．しかし，プロセス数が増えるに従い，提案アーキテクチャの 2 重の BMT 更新の緩和手法の効果が顕になり，相対パフォーマンスの増加につながる．

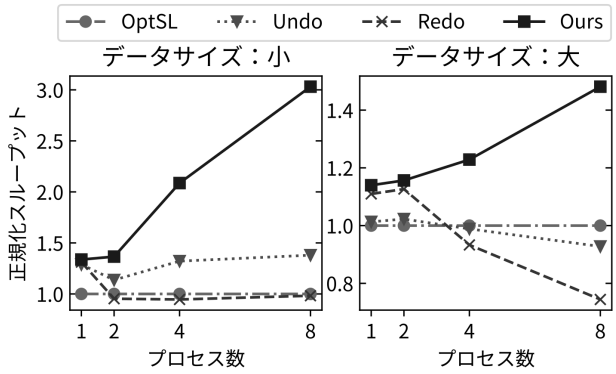

図 10 各データサイズについて，プロセス数がスループットに与える影響.

Fig. 10 Sensitivity of execution time to number of processes for small and large workloads.

7. 結論

BMT 更新は葉から根へ逐次的に行われるので非常に高コストであり，メモリコントローラの WPQ を埋め尽くし，セキュアな NVM でトランザクションを実行する主要なボトルネックとなっている．WPQ が埋まってしまう根本原因は，ライト・アヘッド・ロギングによって BMT 更新が 2 倍行われることである．本研究では，この 2 重化された BMT 更新を緩和するため，ハードウェア・ロギングのアーキテクチャを提案した．これは，各 BMT 更新タイプの特徴にもとづき冗長性を特定し，実現容易な機構によってそれを排除するアーキテクチャである．ログについては，BMT を分離することで 64%の BMT 更新レイテンシを削減し，上書き更新については WPQ への遅延挿入により 91%の BMT 更新の手間を隠蔽した．評価の結果，最大で 2.8 倍の性能向上を確認した．

謝辞 本研究の一部は，JSPS 科研費 19H04075，18H05288，および JST さきがけ JPMJPR18M9 の支援により行われたものである．

参考文献

[1] 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).

[2] 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).

[3] 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).

[4] 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).

[5] Chakrabarti, D. R., Boehm, H.-J. and Bhandari, K.: Atlas: Leveraging locks for non-volatile memory consistency, *ACM SIGPLAN Notices*, Vol. 49, No. 10, pp. 433–452 (2014).

[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* (2018).

[7] 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).

[8] 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).

[9] 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).

[10] 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).

[11] 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).

[12] Jeong, J., Hong, J., Maeng, S., Jung, C. and Kwon, Y.: Unbounded Hardware Transactional Memory for a Hybrid DRAM/NVM Memory System, *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 525–538 (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] 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).

[15] 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).

[16] 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).

[17] 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).

[18] Liu, S., Kolli, A., Ren, J. and Khan, S.: Crash consis-

tency in encrypted non-volatile main memory systems, *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 310–323 (2018).

[19] 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).

[20] Awad, A., Manadhata, P., Haber, S., Solihin, Y. and Horne, W.: Silent Shredder: Zero-Cost Shredding for Secure Non-Volatile Main Memory Controllers, *ACM SIGPLAN Notices*, Vol. 51, No. 4, pp. 263–276 (2016).

[21] 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).

[22] 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).

[23] 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).

[24] SNIA, S.: NVDIMM Messaging and FAQ (2014).

[25] 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).

[26] 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).

[27] 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).

[28] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S. et al.: The gem5 simulator, *ACM SIGARCH computer architecture news*, Vol. 39, No. 2, pp. 1–7 (2011).

[29] Zuo, P., Hua, Y. and Xie, Y.: SuperMem: Enabling application-transparent secure persistent memory with low overheads, *Proceedings of the 52rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 479–492 (2019).

[30] Chen, Z., Zhang, Y. and Xiao, N.: CacheTree: Reducing Integrity Verification Overhead of Secure Non-Volatile Memories, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020).