卒業論文

題目

メモリアクセスの局所性を用いた
小規模トランザクショナルメモリの
アドレス競合検出法

指導教員

佐々木 敬泰 助教

2018 年

三重大学 工学部 情報工学科
コンピュータアーキテクチャ研究室

今村 圭吾 (414810)

内容梗概

近年，共有メモリ型マルチプロセッサが広く普及し，並列処理に利用されている．一般的に，並列プログラムを実行させた際，実行順序が非決定的であるため，並行する複数のプロセスの間で，実行順序に依存せず，共有メモリ上のデータの一貫性を保持する排他制御が必要となる．従来，排他制御の手法としてロック手法が用いられてきたが，実質的に逐次実行となり，性能低下を招くという問題がある．そこで上記の問題を解決する排他制御の手法として，トランザクショナルメモリ (Transactional Memory:TM) が注目されている．TM では，投機的にメモリアクセスを行い，共有資源へのアクセス先の競合を検出した場合ロールバックすることで排他制御を実現している．これにより，ロックによって起こる上記の問題を解決することができる．特に，ハードウェアにより実装するハードウェアトランザクショナルメモリ (HTM) は，多数の研究，開発が行われている．しかし，これらはプロセッサコアだけでなくキャッシュにも拡張を施すため，回路規模やプロセッサの設計変更が大きく，組み込み向けプロセッサへの適用が困難である．この問題を解決する手法として，当研究室では，扱うトランザクション処理を限定する代わりに，回路規模の増大を最小限に防ぎ，組み込みプロセッサに適用可能な小規模 HTM が提案されている．この手法では，アクセスアドレスをハッシュ関数によって処理，識別することでメモリアクセスの競合検出を行っている．しかし，異なるアドレス間でハッシュが衝突することにより，アクセス競合の誤検出が頻発し処理性能が低下する問題がある．この問題の原因として，先行研究の手法は，メモリアクセスの局所性の影響を受け，アクセス頻度の高いアドレスに対し，検出精度が高くないという事が挙げられる．本研究では，一般的なプログラムでの，メモリアクセスの局所性によるアクセス傾向の観点から，ハッシュ関数を改良する．更に，改良した関数のハードウェアによる実装手法を考案し，上記の HTM に適用する．これにより，アドレスの誤検出を改善し，処理性能の向上を実現する．評価として，トランザクションの実行サイクル数を，最大 18%削減することに成功した．

Abstract

Shared memory multicore processors are widely used in parallel processing. Generally, when a parallel program is executed, execution order is nondetermistic. So, exclusive control that maintaines consistency of data on shared memory depends on execution order among parallel processes is required. Conventionally, Lock has been used as a method of exclusive control, but it has problem of performance degradation because of execution is practically serial processing. As, another method of exclusive control for solving the problem of Lock, Transactional memory (TM) is proposed. It speculatively access the memory, implements exclusive control by rolling back in case of conflict of the access destination to the shared resource is detected. This solves the above problem caused by locking. Especialy, the Hardware Transactional Memory (HTM) implemented TM by hardware is being conducted numerous research and development. However, they extend not only to the processor core but also to the cache, so their circit is too large and can't use embedded processor. As a method to solve this problem, a small scale HTM applicable to embedded processors has been proposed, instead of limiting transaction processing. In this method, Conflict detection of memory access is performed by access address process by hash function. But, due to confliction of hashes between different addresses, false detection is frequently caused and processing performance is descend. This problem is caused by detection accuracy is not high in high access frequency addresses. This paper improves hash function from access tendency due to locality of memory access in a generally programs. Furthermore, this function is applied to HTM by devising hardware. According to our evalution results, the number of execution cycles of the transaction was reduced to 18% in average.

目 次

図 目 次

表 目 次

1　はじめに

並列処理の需要から共有メモリ型マルチコアプロセッサが広く普及している．並列処理ではプログラムの実行順序が非決定的となるため，共有データの状態が不定となる．そのため，複数のプロセスの実行順序に依存せずにメモリ上の共有データの一貫性を保持する排他制御が必要となる．一般的に排他制御に用いられるロック手法では，競合の有無にかかわらずアクセスを制限する．近年，排他制御手法として，トランザクショナルメモリ (Transactional Memory:TM)[1] が注目されている．TM では，投機的にメモリアクセスを行い，共有資源へのアクセス先の競合を検出した場合ロールバックすることで排他制御を実現している．ロックと異なり，競合が起きない場合は並列処理が行われるため，ロックに比べ処理性能を向上させることができる．特に，ハードウェアにより実装するハードウェアトランザクショナルメモリ (HTM) は，Intel 社の TSX[2] をはじめとして多数の研究，開発 [3, 4, 5, 6, 9, 7, 8, 10] が行われている．しかし，これらはプロセッサコアだけでなくキャッシュにも拡張を施すため，回路規模やプロセッサの設計変更が大きく，組み込み向けプロセッサへの適用が困難である．この問題を解決する手法として櫻田が，扱うトランザクション処理を限定する代わりに，回路規模の増大を最小限に防ぎ，

組み込みプロセッサに適用可能な小規模 HTM[11] を提案している．この手法では，トランザクション間のメモリアクセスの競合検出を，アクセスアドレスをハッシュ関数によってハッシュ値に変換し，そのハッシュを用いた演算を行うことで実現している．しかし，ハッシュを用いた競合検出では，異なるアドレス間でハッシュが衝突することから，アクセス競合の誤検出が発生する場合がある．この結果，本来発生しないロールバックが起こることから，性能が低下する．先行研究ではこの誤検出が頻繁に発生することから，性能が大きく低下している．その原因として，メモリアクセスの局所性の影響を受け，アクセス頻度の高いアドレスに対し，検出精度が高くない事が挙げられる．このようなアドレスに対し検出精度が低い場合，誤検出が頻発し，性能が大幅に低下する．本研究では，メモリアクセスの局所性の影響により，アクセス頻度の高いアドレスに対し検出精度が改善する競合検出手法を提案し，性能向上を図る．更に，改良した手法のハードウェアによる実装手法を考案し，上記の HTM に適用する．以降，本稿は次のように構成する．まず，次章でトランザクショナルメモリの概要について，第 3 章で櫻田の提案する手法での，アクセス先の競合検出方法とその問題点を述べる．第 4 章では提案手法の詳細な解説を行い，第 5 章で評価，第 6 章で結論を述べる．

2　トランザクショナルメモリ

TM では，共有メモリ内のデータに対する読み込み，変更，書き込みの一連の動作を一つのトランザクションと定義する．トランザクション内の共有データの整合性は，

1. メモリアクセスの監視による競合検知

2. 処理結果を反映するコミット

3. 処理結果の破棄，トランザクションの再実行を行うアボート

の 3 つの処理によって保証される．TM の動作を図 2.1 を用いて説明する．図 2.1 では，thread1，thread2 が並列実行されており，各スレッドは，共有データ a を読み込み，演算し，書き戻すトランザクション Tx.A，Tx.B を実行する．トランザクション開始宣言には XBEGIN 命令を用いる．これにより，Tx.A は時刻 t0，Tx.B は時刻 t1 でトランザクションを開始する．次に，Tx.A は時刻 t4 でトランザクション終了命令 XEND を実行する．この時，他のトランザクションによりアドレス a は変更されていないため，Tx.A の処理結果をメモリに反映させる（コミット）．同様に，時刻 t5 で Tx.B が XEND を実行する．この時，アドレス a の値は時刻 t2 でのロード後，時刻 t3 で Tx.A により変更されているため，Tx.B では競合

の発生が検知される．この場合，Tx.B は処理結果を破棄，実行前の状態を復元し，始めから再実行する（アボート）．このように，TM ではトランザクションを投機的に実行し，複数トランザクション間のアクセス競合を検知する．図 2.1 では，アクセスが競合する場合を例示したが，Tx.A と Tx.B にアクセス競合がない場合は同時にコミットできる．そのため，メモリへのアクセス制限によって同時処理を行えないロックに比べ，より効率的な排他制御を実現できる．以下に，TM の実現方法であるソフトウェアトランザクショナルメモリ (STM) と，ハードウェアトランザクショナルメモリ (HTM) について述べる．

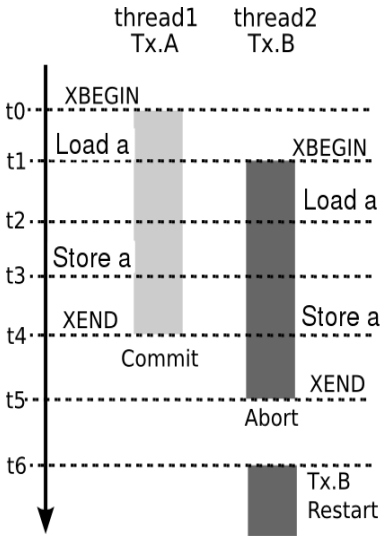

図 2.1: トランザクション処理

2.1 トランザクショナルメモリの実現方法

2.1.1 ソフトウェアトランザクショナルメモリ

ソフトウェアトランザクショナルメモリ (STM) は，TM の機能をソフトウェアのみで実現する方法である．STM は，TM を実現する上で必要となるハードウェアを追加する必要がなく，回路規模の増加を抑えることが可能である．しかし，ソフトウェア側で処理を行うために，発生するオーバーヘッドが増大するという問題がある．

2.1.2 ハードウェアトランザクショナルメモリ

ハードウェアトランザクショナルメモリ（HTM）は TM をハードウェア上で実装するため，オーバーヘッドの増加を最小限に留めることが可能である．HTM の分野では多数の研究がなされており，更に，Intel 社の Haswell[12] や，IBM 社の POWER 8[13] 等，既にいくつかの商用プロセッサで実現されている．しかし，これらの既存の HTM は，TM の機能を実現する上で，プロセッサコアだけでなくキャッシュメモリにも回路拡張を及ぼすため，回路規模が大幅に増大するという問題がある．このため，回路規模が小規模であることが求められる，組み込み向け用途に適していないという問題がある．そこで櫻田は，組み込み用途に適した規

模 HTM[11] を提案している．次章で，この HTM の実現方法及び動作について詳しく解説する．

3　先行研究

前章で述べた，櫻田が提案する HTM は，トランザクションサイズや TM の機能を限定する代わりに，プロセッサコアのみの拡張に留めることで小面積化を図っている．この手法は，図 3.2 によって HTM として実現される．先行研究ではトランザクション間のアクセス先の競合検出を，アドレスをハッシュ値に変換し，演算を行うことで実現している．あるコア内で行われたトランザクションがメモリを参照した時，式 (1) に示すハッシュ関数により，アドレスをハッシュ値に変換する．hash_length はハッシュの bit 長である．この式を用いることで，ハッシュ値は高々1 ビットが 1 となるビット列となる．ハッシュの bit 長は任意に設定できるが、ここでは説明のため 32 ビットとする．この場合，図 3.2 の競合検出用バスのバス幅も 32 ビットとなる．

$$hash(addr) = 1 << (addr \bmod hash\_length) \tag{1}$$

生成されたハッシュ値は，各コア内に存在する，ハッシュ格納レジスタである TxReg に格納される．このとき，メモリアクセスを行ったコアは

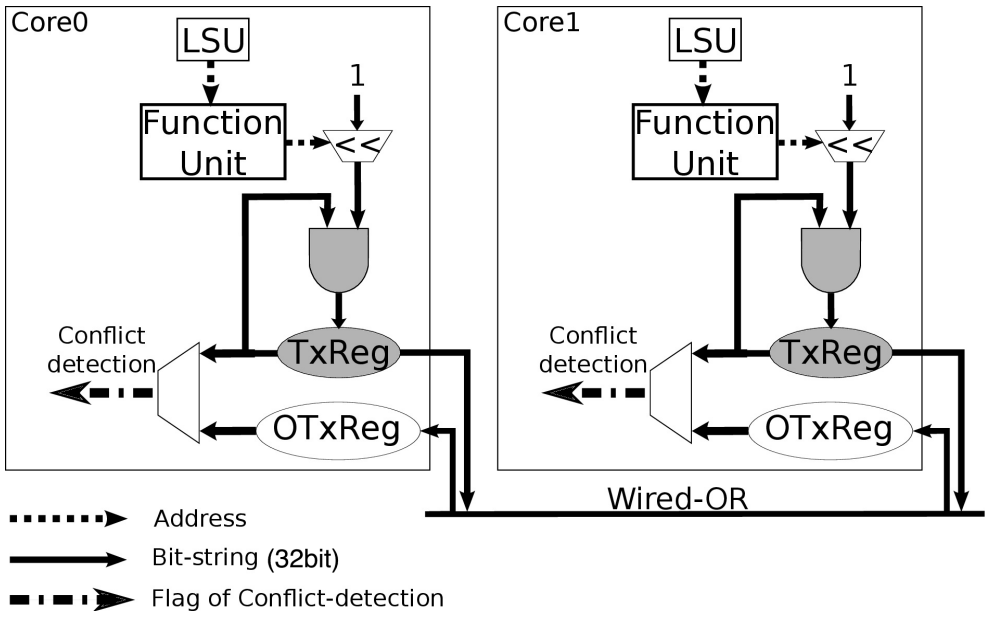

図 3.2: 競合検出回路

TxReg の値と自身の生成したハッシュ値の論理和をとり，TxReg に更新することで，自身がアクセスしたアドレスの情報を持つ．このように，アクセスしたアドレスの情報を，1 本のレジスタで管理することにより，回路規模の増大を抑えている．また，競合検出を行う際に，自身以外のコアがアクセスしたアドレスの情報を得る必要がある．そのため，あるコアは，他のコアの TxReg の値を，Wired-OR ネットワークを用い取得することで，自身以外のコアがアクセスしたアドレス情報を得ることができる．この値を，レジスタ OTxReg 内に格納する．

競合検出は，TxReg と OTxReg 同士の論理和によって行われる．例と

して，Core-A，Core-B の 2 コアでの競合検出過程を説明する．また，この例では説明の簡単化のためハッシュの bit 長を 5 とする．まず，図 3.3 のように，Core-A，Core-B で別のアドレスにアクセスした際について解説する．この場合では，Core-A 側で生成されるハッシュ値と，Core-B 側で生成されるハッシュ値が異なる値となる．これら 2 つのハッシュの論理和演算を行った際，演算結果は 0 となる．このような場合は，競合が起きていないと判断することができる．次に，まず，図 3.4 のように，Core-A，Core-B で同じアドレスにアクセスした際について解説する．この場合では，Core-A 側で生成されるハッシュ値と，Core-B 側で生成されるハッシュ値も同じ値となる．これら 2 つのハッシュの論理和演算を行った際，ハッシュのいずれかの桁が 1 となり，演算結果は 0 とならない．このような場合は，競合が起きていると判断することができる．

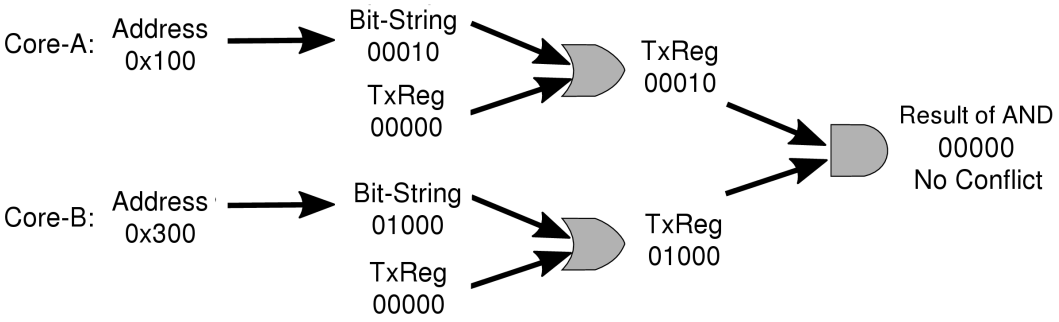

図 3.3: 競合検出：競合なし

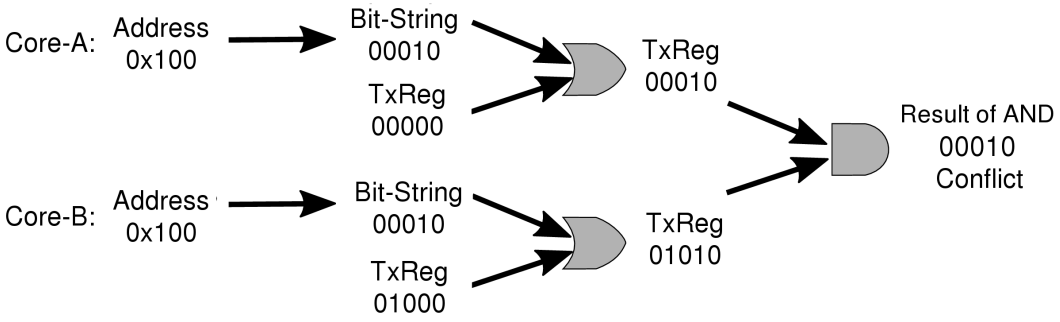

図 3.4: 競合検出：競合あり

3.1 問題点

先行研究の HTM ではハッシュ値を用いて競合検出を行うが，異なるアドレス間でも同じハッシュが生成されることがあり，アクセス先の競合を誤検出する場合がある．例として，以下の図 3.5 のように，Core-A がメモリの 0x100 番地，Core-B がメモリの 0x600 番地を参照したとする．本来，この 2 つのアドレスは異なるため参照先の競合は起きない．しかし，図 3.3，図 3.4 と同様にこれらのアドレス値を 5 で割った場合，割った余りが同じになるため，異なるアドレス間で同じハッシュ値が生成される．このような場合，異なるアドレスへのアクセスにもかかわらず，アクセス先が競合したと誤って判断される．

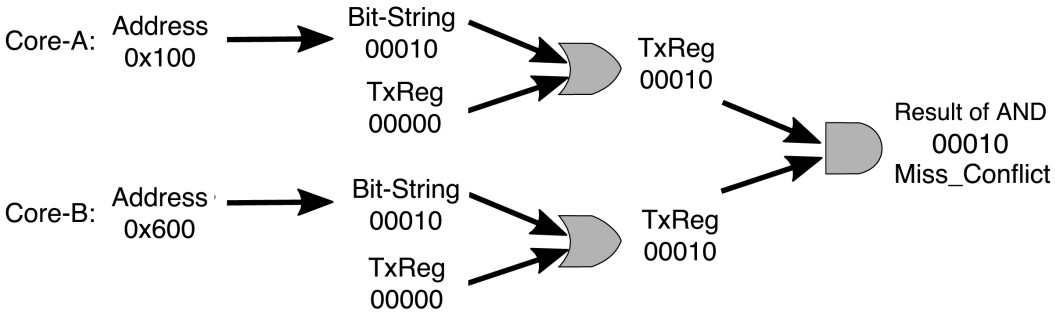

図 3.5: 競合検出：誤検出

この結果，本来起きないアボートが発生し，不要なトランザクション処理を行うことになり，性能が低下する．先行研究の競合検出方法では，この誤検出が頻発するという問題がある．この誤検出の頻発の原因として，アクセス頻度の高いアドレスに対し競合検出の精度が低い事が挙げられる．一般的にプログラムには，参照する領域に偏りがあるメモリアクセスの局所性という性質がある．例として，同じアドレスに短時間の間で複数回アクセスが発生する時間的局所性や，アクセスしたアドレスの周辺の領域にアクセスが偏る，空間的局所性というものが挙げられる．メモリアクセスの局所性の影響を受けた，アクセス頻度の多いアドレスに対し競合検出の精度が低い場合，誤検出を起こす可能性が高いアドレスに頻繁にアクセスすることになり，誤検出が頻発する．結果，発生するアボートの回数も大きく増加し，トランザクション処理の大きな妨げ

になると考えられる．しかし，先行研究での競合検出手法は，このようなメモリアクセスの局所性により発生するアクセスの偏りを考慮していない．そのため，上述の誤検出の頻発が発生する．

4　提案手法

本研究では，メモリアクセスの局所性を考慮した競合検出を行うことで，先行研究の問題点を解決する．メモリを一定サイズの領域分割し，それぞれの領域に対してのアクセスの回数から，局所性の影響を受けたアクセス頻度の高いメモリ領域を判断する．さらに，アクセス頻度の低い領域と，アクセス頻度の高い領域で，競合検出の際に異なるハッシュを使用する．このように，2つのハッシュをアクセス頻度に応じて切り替えて使用することで，アクセス頻度の高いアドレス領域に対する競合検出精度を改善し，性能向上を図る．提案手法は，図4.6に示すハードウェアによって実現できる．以下で手法，ハードウェアの動作について解説する．

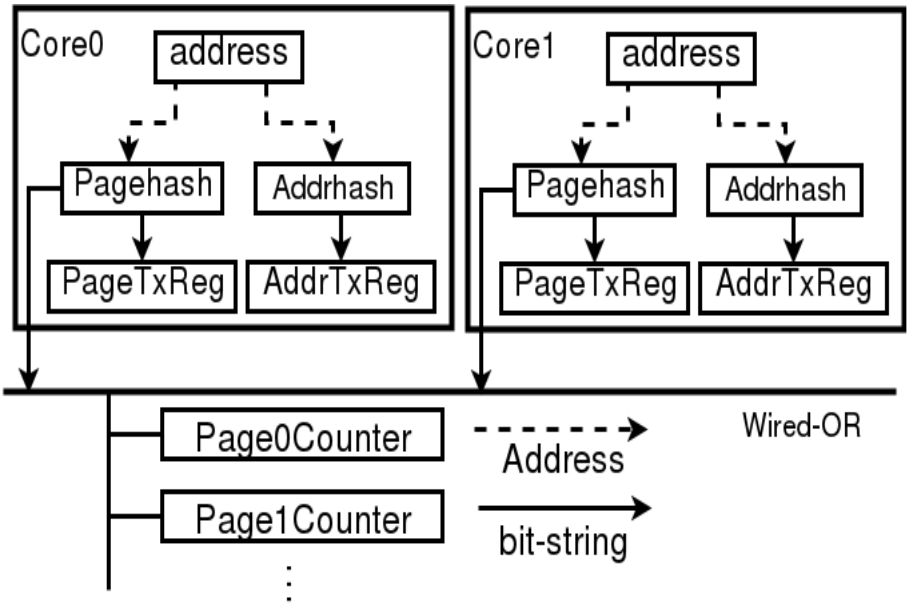

図 4.6: 提案手法ハードウェア

4.1 領域単位でのハッシュ生成

本手法では，上述のメモリの領域分割を，式 (2) により行い，分割した領域に対して番号を割り振る．この時，式 (2) の size は分割する領域の大きさ，page_num は領域について割り当てる番号の種類である．例として，分割領域のサイズが 3，領域番号の種類が 3 種類以上だとすると，式 (2) を用いることで，図 4.7 の上段のようなアドレス値に対し，図 4.7 の中段のように，一定サイズの領域に分割し，番号を割り振ることができる．更に領域番号から式 (3) によってハッシュを生成する．本研究ではこ

のハッシュを pagehash と呼ぶ.

このような領域分割を行った際，図 4.7 の色付きの領域のような，メモリアクセスの局所性の影響を受け，アクセス頻度の高い領域が存在すると考えられる．また，pagehash のみによる競合検出を行った場合，アクセス頻度の高い領域内のアドレス間で，ハッシュ値が衝突することから誤検出が頻発する問題がある．そこで，アクセス頻度の高い領域での競合検出に，高頻度アクセス領域用ハッシュとして，先行研究の式 (1) と同じハッシュを利用する．本研究ではこのハッシュを addrhash と呼ぶ．これにより，このハッシュの bit 長が 3 であるとすると，図 4.7 の下段の様なハッシュが生成される．addrhash は，図 4.7 の色付きの領域のような，アクセス頻度の高いアドレス領域内の競合検出にのみ使用でき，アクセス頻度の低い他の領域は競合検出に使用できない．pagehash，addrhash の 2 つのハッシュを，アクセス頻度に応じて切り替えて使用することにより，競合検出を行う．次節でアクセス頻度の判定について説明する．

$$page(addr) = (addr/size) \bmod page\_num \tag{2}$$

$$pagehash(addr) = 1 << page(addr) \tag{3}$$

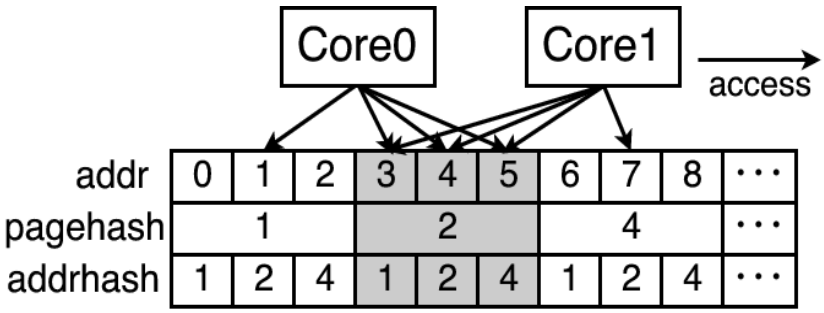

図 4.7: 提案手法のハッシュ割り当て

4.2 高頻度アクセス領域の決定

上記の addrhash による競合検出が可能である，アクセス頻度の高い領域の決定方法について説明する．決定するにあたり，アクセスの回数をカウントするためのカウンタを使用する．このカウンタは，分割するページ数分存在し，全てのページに対し 1 つずつ対応するように用意される．メモリに対しアクセスが発生した際，上記の式 (2) からアクセスするページ番号，式 (3) から pagehash が生成される．この時に，生成された pagehash の 1 となっているビットの桁を確認することで，どのページに対してアクセスが起きたか判断することが可能である．例として，生成された pagehash の下位 2bit 目が 1 となった場合を説明する．式 (3) での左シフトの量が 1 であることが生成された pagehash から分かるため，式 (2) より，アクセスしたアドレスに与えられたページ番号が 1 であるこ

とが判断できる．

あるページにアクセスが発生したと確認した際，アクセスしたページに対応するカウンタの値をインクリメントする．この時，各コアで生成された pagehash を，図 4.6 のように，全コア間で Wired-OR ネットワークによって纏めることで，全てコアの間でアクセスしたページの情報を得ることができる．この情報を基にページへのカウントを行うことで，全コア間でカウンタを共有して使用することができ，各コアにカウンタを置く場合と比較して，回路規模の増加を防ぐことが可能である．ページに対するアクセスのカウントを繰り返し，あるカウンタが閾値に到達した時，カウンタに対応するページはアクセス頻度が高いと判断し，addrhash を使用した競合検出が可能となる．

4.3 生成ハッシュの切り替え

2 つのハッシュの切り替えについて説明する．提案手法では図 4.6 のように，アクセス先の情報を持ったハッシュを格納するために，PageTxReg と AddrTxReg といった，2 つのレジスタを使用する．各コアは生成した pagehash，addrhash をそれぞれ，PageTxReg，AddrTxReg に反映させる．この時，アクセスしたページに対するアクセス頻度と，そのページに対する addrhash の生成状況によって，反映させるハッシュの種類を切

り替える．以下の 3 つのケースの間で切り替わる．

1. アクセスしたページが，アクセス頻度が高いと判断されていない
2. アクセスしたページはアクセス頻度が高いと判断されたが，まだ addrhash が生成されていない
3. アクセスしたページがアクセス頻度が高いと判断され，addrhash が生成されている

上から順にケース 1，2，3 として，それぞれのケースで生成されるハッシュを示したものが表 4.3 となる．この表では，○となっているものが，そのケースで反映させるハッシュであることを示す．それぞれのケースについて解説する．

ケース 1

アクセスしたページは，アクセス頻度が高いと判断されていないため，addrhash 側で競合検出を行うことができない．そのため，page-hash のみを反映させ，addrhash を反映させない．

ケース 2

アクセス頻度が高いページと判断されているため，addrhash による競合検出を行うことが可能である．しかし，このアクセスより前

に，同じページに対してアクセスした他のコアは，このページに対するアクセスの情報を pagehash 側のみでしか保管していない．そのため，このケースで addrhash のみを反映させたとすると，同じページにアクセスしているコアが存在するにも関わらず，競合が検出されない事態が発生する．この時，起きるべきアボートが発生せず，正しい排他制御が行われない可能性がある．そこで，このケースでは，pagehash と addrhash の両方を，アクセス先の情報として反映させる．これにより，このアクセスよりも前のアクセス，後のアクセスどちらの場合とも競合検出が可能となり，適切な排他制御を行うことができる．

ケース 3

上のケースと同様，アクセス頻度が高いページと判断されているため，addrhash による競合検出を行うことが可能である．また，このアクセスより以前に既に addrhash が生成されているため，addrhash 側のみで適切な排他制御を行うことが可能である．そのため，このケースでは，pagehash を生成し，ページに対するアクセスをカウントした後，pagehash の値を反映させず，addrhash のみをアクセス先の情報として反映させる．

	生成ハッシュ	
	pagehash	addrhash
ケース 1	○	
ケース 2	○	○
ケース 3		○

表 4.1: 生成されるハッシュの種類

競合の判定は先行研究と同様にハッシュの論理和を取ることによって行う．PageTxReg 側と AddrTxReg 側でそれぞれ競合検出を行い，どちらでも競合が起きていないと分かれば，競合が発生していないと判断する．逆に，どちらかで競合が確認できた場合，競合が発生したと判断し，トランザクションを再実行する．このように，アクセス頻度の低いページは pagehash 側，アクセス頻度の高いページは addrhash 側で競合検出を行うことが可能となり，先行研究と比較して，アクセス頻度の高い領域に対して競合検出の精度を高めることができる．それにより，本来発生しないアボートの回数を減らすことが可能となり，性能向上を実現できると考えられる．

5　性能評価

先行研究の手法及び提案手法を，C++言語で記述された，マルチコアプロセッサを想定する機能シミュレータ上に適用し，評価を行う．評価指標として，ベンチマークプログラムを実行した際の，先行研究と比較した，提案手法のトランザクションの実行サイクル数の減少率を用いる．

5.1　評価環境

実行環境を以下の表 5.2 に示す．ベンチマークプログラムは，先行研究でも用いられているマスタースレーブ型の並列プログラムを使用する．これは，複数のスレッドを並列実行させ，ある 1 つスレッドは常に 1 つのアドレスに対しアクセスし，他のスレッドは設定した確率でそのアドレスに対してアクセスする．アクセスしない場合は，別のアドレスに対してアクセスを行う．このとき，設定する確率を競合発生率とする．このベンチマークのパラメータを，競合発生率を 5%から 95%の間で 5%刻み，並列実行するスレッド数を 2，4，8，16 の 4 通り，トランザクションサイズを小，中，大の 3 通りの間で変化させ，評価を行った．トランザクションサイズとは，このベンチマークにおけるトランザクション内の命令数を表す．トランザクションサイズが小なら命令数が 1 のもの，中なら 5

のもの，大なら 10 のものとして定義した．先行研究の手法のシミュレーションにおいて，式 (1) によって与えられるハッシュのビット長を 32 とした．また，提案手法のシミュレーションにおいては，第 4 の，pagehash，addrhash のビット長を共に 16 ビットとした．これにより，2 つの手法は共にハッシュのサイズが 32 となり，等しくなる．以下に評価方法の表 5.3 を記載する．

表 5.2: 実行環境

CPU	Core i7-2600
動作周波数	3.4GHz
メモリ	16GB

表 5.3: 評価方法

評価指標	トランザクションのサイクル減少率
評価環境	ソフトウェアシミュレータ
ベンチマークのパラメータ	
競合確率	5% 〜 95%
スレッド数	2，4，8，16
トランザクションサイズ	小，中，大

5.2 評価結果

評価結果のグラフを，トランザクションサイズ小，中，大の順に，以下の図 5.8，図 5.9，図 5.10 に示す．横軸の conflict_rate が競合発生率，縦

軸の cycle_decrease がトランザクションのサイクル減少率を表す．縦軸が 0 に近いほど性能が先行研究と変わらず，値が大きいほど先行研究と比べ性能が向上している．

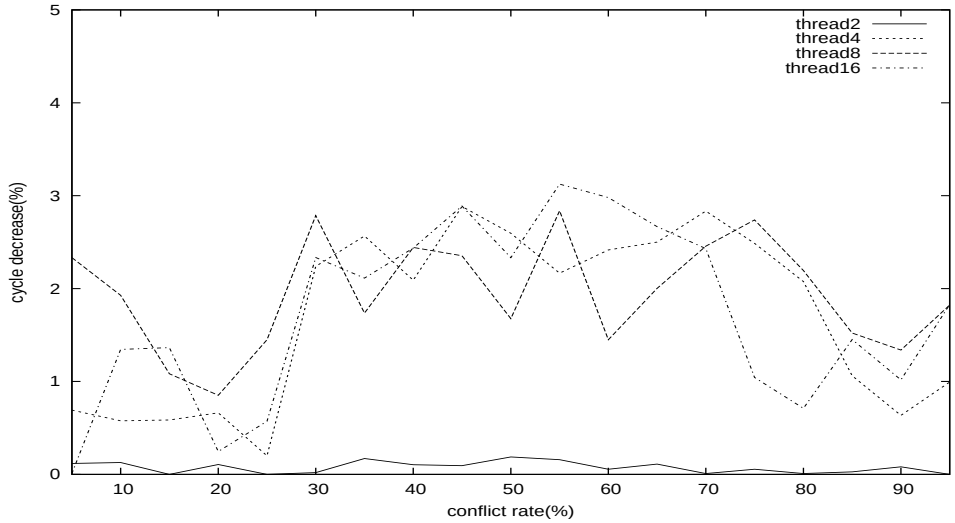

図 5.8: トランザクションサイズ小

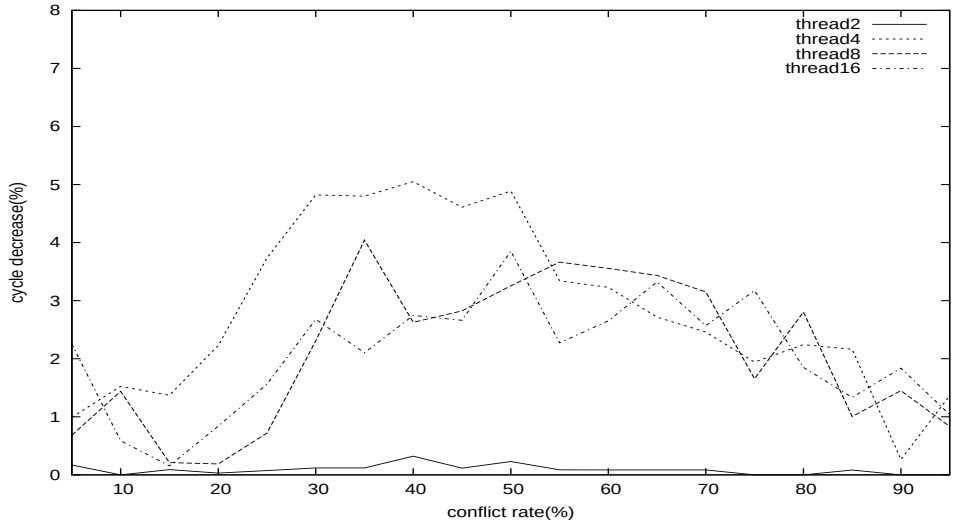

図 5.9: トランザクションサイズ中

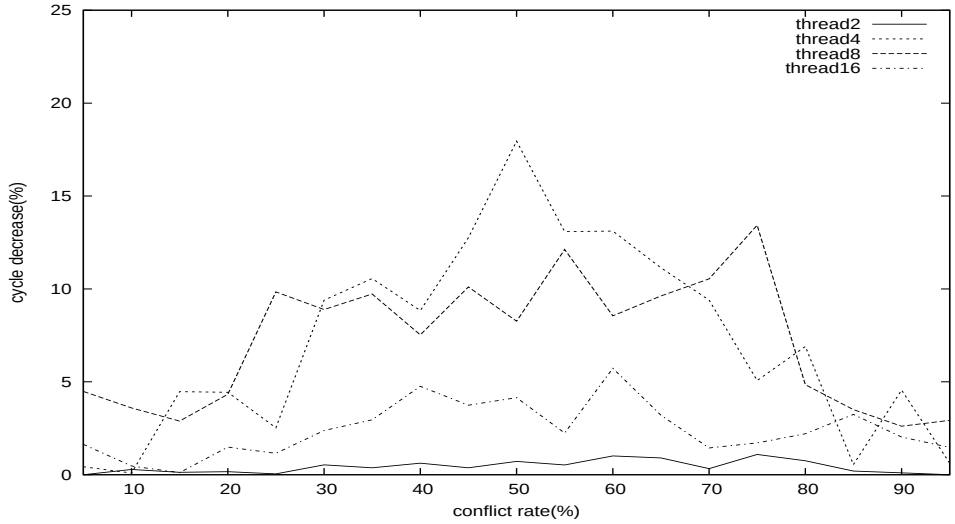

図 5.10: トランザクションサイズ大

5.3 考察

評価結果のグラフから，全ての場合において，先行研究と比べ，同等以上の性能が確認することができた．また，グラフから，競合発生率が高い数値になるほど，先行研究と比べ性能が高くなる傾向が確認できた．特に，トランザクションのサイズが大きいほど，競合検出方法の改善の恩恵が大きく，トランザクションのサイクル減少率が増える傾向がみられた．また，サイクル減少率が大きくなるケースは，競合発生率が高い場合に多くみられることがグラフから確認できる．このベンチマークにおいて，競合発生率が高くなるほど，単一アドレスへのアクセスが多くなり，アクセスの局所性が顕著に現れる．このような場合において減少率が大きいことから，提案手法が先行研究と比べ，局所性によるメモリアクセスの偏りへの対処に有効であると考えられる．反対に，競合発生率が低い場合，減少率が他の場合に比べ少ないという事が確認できる．これは，ベンチマークを動作させた際のアクセスの局所性が低いため，提案手法が有効に働いていないためと考える．

6　おわりに

本研究では，櫻田の提案する小規模 HTM における，アクセス先アドレスの競合の誤検出が頻発する問題を，アクセスの局所性を考慮した競合検出を行うことによって改善した．この結果，先行研究に比べ，平均 2.5%，最大 18%，トランザクションの実行サイクルの削減に成功した．今後の展望としては，実際に提案手法をハードウェアとして実装し，面積評価，動作検証，電力評価を行うことが挙げられる．

謝辞

本研究を行うにあたり，多数のご指導を頂きました近藤利夫教授，佐々木敬泰助教，並びに深澤研究員に深く感謝いたします．

参考文献

[1] M.Herlihy， J. Eliot and B. Moss.: Transactional Memory: Architectural Support for Lock-Free Data Structures.

[2] Richard M. Yoo et al.: Performance Evaluation of Intel$^{(R)}$ Transactional Synchronization Extensions for High-Performance Computing. Proceedings of the International Conference on High Performance Computing， Networking， Storage and Analysis， 17-22 Nov. 2013

[3] L.Hammond，B.D.Carlstrom，V. Wong，M.Chen，C.Kozyrakis，and K.Olukotun.Transactional coherence and consistency:Simplifying parallel hardware and software.Mico ' s TopPicks，IEEE Micro，24(6)，nov/dec 2004.

[4] L.Hammond， V.Wong， M.Chen， B.D.Carlstrom， J.D.Davis，B.Hertzberg，M.K.Prabhu，H.Wijaya，C.Kozyrakis，and K. Oluk otun. Transactional memory coherence and consistency. In Proceedings of the 31st Annual International Symposium on Computer Architecture ， page 102. IEEE Computer Society ， Jun 2004.

[5] A.McDonald, J.Chung, H.Chafi, C.Cao Minh, B.D.Carlstrom, L.Hammond, C.Kozyrakis, and K.Olukotun. Characterization of tcc on chip-multiprocessors. In Proceedings of the 14th International Confer ence on Parallel Architectures and Compilation Techniques , Sept 2005.

[6] J. Chung, H. Chafi, C. C. Minh, A. McDonald, B. D. Carlstrom, C. Kozyrakis, and K. Oluk otun. The common case transactional behavior of multithreaded programs. In HPCA '06: Proceedings of the 12th International Symposium on High-P erformance Computer Architectur e , Washington, DC, USA, 2006. IEEE Computer Society.

[7] C. Scott Ananian, Krste Asanovic, Bradley C. Kuszmaul, Charles E. Leiserson, and Sean Lie. Unbounded Transactional Memory. In Proc. of the 11th International Symposium on High-Performance Computer Architec-ture, 2005.

[8] R.Rajwar, M.S.LAM, and K.Lai. Virtulizing transactional memory.SIGARCH Comput.Archit.News, 33(2):494-505, 2005

[9] . E. Moore, J. Bobba, M. J. Mora van, M. D. Hill, and D. A.Wood. Logtm: Log-based transactional memory . In HPCA ' 06:Proceedings of the 12th International Symposium on High-P erfor-mance Computer Architectur e , Washington, DC, USA, 2006. IEEE Computer Society

[10] Christian Jacobi, Timothy Slegal et. al. Transactional Memory Architecture and implementation for IBM System z, 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture

[11] Takahiro Sakurada Keisuke Sasaki Yuki Fukazawa Tosio Kondo Efcient implementation method of a compact HTM into processor cores. Swopp 2016 August.

[12] Per Hammarlund et al.: HASWELL:THEFOURTH-GENERATION INTEL CORE PROCESSOR. IEEE Micro (Volume: 34, Issue: 2, Mar.-Apr. 2014)

[13] B. Sinharoy et al.: IBM POWER8 processor core microarchitecture. IBM Journal of Research and Development (Volume: 59, Issue: 1, Jan.-Feb. 2015)