2H-2

最適なチェックポイントを選択するトランザクショナル・メモリ

伊藤悠二† 塩谷亮太† 五島正裕† 坂井修一†
†東京大学大学院情報理工学系研究科

1 はじめに

近年では，複数のプロセッサ・コアを1つのチップ上に集積したマルチコア・プロセッサが広く普及しており，共有メモリ型の並列プログラムを実行するためのインフラは既に整ったと言える．しかし，並列プログラムが普及したとは言い難い．この原因の一つに，ロックを用いた同期通信機構が挙げられる．そこで，ロックを用いない同期通信機構として，**トランザクショナル・メモリ (Transactional Memory)** と呼ばれる手法が提案されている [1, 2, 3].

プログラマは，ソース・コード上で不可分 (atomic) に実行された結果と同様の結果になることを保証した**トランザクション (transaction)** を指定する．実際には，トランザクションとして指定された部分は，不可分に実行されているかのように実行される．

トランザクショナル・メモリの多くの手法では，トランザクションを投機実行する．しかし，あるトランザクションと他のトランザクションが同じアドレスにアクセスすることがある．このとき，トランザクションは不可分に実行された場合と同じ結果とならないことがある．このようなアクセスを**競合**として検出し，いずれか一方のトランザクションを「なかったことにする」．この処理をロールバックと呼び，ロールバック先をチェックポイントと呼ぶ．ロールバックしたトランザクションは再実行を行う．一方，処理をすべて終了したトランザクションは，状態を確定する**コミット (commit)** を行う．

トランザクション中のスレッド・スイッチ トランザクションはプログラムによってその実行時間や数は様々である．しかし，実行時間の長いトランザクションは，スレッド・スイッチによって中断されることがあり得る．中断されるトランザクションをロールバックしてしまうと，このような長いトランザクションがコミットできなくなる．このような扱いではプログラマにとってトランザクションの実行時間を制限されることとなり，大きな負担となる．このため，トランザクショナル・メモリでは，トランザクション中のスレッド・スイッチに対応する必要がある．

スレッド・スイッチへの対応のために，後のスレッド・スイッチで再開されるまで中断トランザクションの不可分性を保たなければならない．中断トランザクションの不可分性を保つには，実行中のトランザクションと中断トランザクションとの競合検出する必要があり，以下の 3 つの処理を行う．

競合の有無の検出 実行中のトランザクションが中断トランザクションと競合していないか．

競合トランザクションの検索 実行中のトランザクションがどの中断トランザクションと競合したか．

競合の位置の検索 競合した中断トランザクション中のどのアクセスが競合の原因となったか．

既存手法では，競合の有無のみを検出する LogTM-SE[2] や，競合した中断トランザクションの検出までを行う FlexTM[3] が提案されている．しかし，これらの既存手法の問題点として，誤検出率の増加，競合トランザクションの検出のオーバヘッド，トランザクション途中のチェックポイントへの部分ロールバックできないことが挙げられる．

本稿では，中断トランザクションとの競合検出を行い，最適なロールバックを行うトランザクショナル・メモリを提案する．図1では，各手法での実行中トランザクションと中断トランザクションとの競合解決を表している．このように，上の問題点を削減し，効率的に中断トランザクションとの競合を解決できる．以下，2 章で提案手法の概要を述べる．

2 提案手法

2.1 トランザクションの実行

まず，通常のトランザクションの実行について述べる．

Transactional Memory Selecting the Optimal Chekpoint
Yuji Ito†, Ryota Shioya†, Masahiro Goshima†and Syuichi Sakai†
†Graduate School of Information Science and Technology, The University of Tokyo

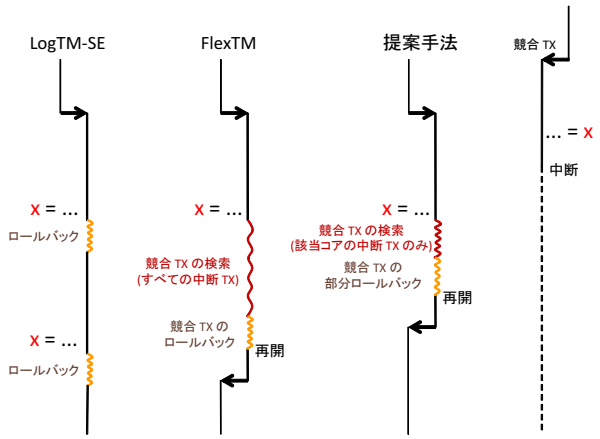

図 1: 提案手法と既存手法

競合検出については，アクセスしたアドレスを圧縮したシグネチャを用いる手法 [2, 3] で行う．シグネチャは，各コアごとにリード/ライトそれぞれ設け，現在実行されているトランザクションのアクセスごとに更新される．他トランザクションによってアクセスされたアドレスをキャッシュ・コヒーレンス・プロトコルより得て，自コアのリード/ライト・シグネチャと比較することで競合を検出する．チェックポインティング時や中断時には，シグネチャをアドレス空間上の別の領域に保存する．チェックポイントからの再開時，スレッド・スイッチによる復帰時には，保存してあったシグネチャを戻して以降の競合検出を行う．

トランザクションによる投機的な書き込みは，LogTM-SE[2] と同様に，ソフトウェア・ログを用いて書き換えられる前の値を保持しておく．ロールバック時には，このログに保存してあった値を用いてチェックポイントの状態に戻す．

2.2 中断トランザクションとの競合検出

競合の有無の検出 提案手法では，実行中のトランザクションのシグネチャとは別に，コアごとに中断トランザクションのシグネチャをまとめたシグネチャを保持しておく．このシグネチャをサスペンド・シグネチャと呼ぶ．スレッド・スイッチ時，あるコアで中断したトランザクションすべてのシグネチャをリード/ライトそれぞれ OR を取り，そのコアのサスペンド・シグネチャを更新する．実行中のトランザクションは，アクセスごとに実行中のコアのサスペンド・シグネチャをチェックしてそのコアで中断したトランザクションとの競合を検出する．また，キャッシュ・コヒーレンス・プロトコルによるキャッシュへの要求を用いて，他コアのシグネチャとサスペンド・シグネチャと比較し，他コアでの実行/中断トランザクションとの競合を検出する．

競合トランザクションの検索 競合の有無をサスペンド・シグネチャで検出後，そのコアで中断したトランザクションについてシグネチャをそれぞれ調べることで競合トランザクションを検出する．既存手法がすべての中断トランザクションを調べなければならなかったのに対し，提案手法ではそのコアで中断したトランザクションだけを調べればよい．このため，競合トランザクションの検出のオーバヘッドが少ない．

競合の位置の検索 競合トランザクションが検出されたら，そのトランザクションのチェックポイント時に保存してあったシグネチャを調べることで競合の位置を検出する．最も古い競合アクセスより前のチェックポイントにロールバックすることで，再実行される処理を最小限にできる．

3 まとめ

本稿では，中断トランザクションがある場合でも効率的にロールバックできる手法を提案した．トランザクションの実行時間や数に制限なく，中断トランザクションとの競合検出を行い，最適なロールバックを行うことで，トランザクションの実行効率を高める．

今後の課題として，本手法の評価をしていきたい．

参考文献

[1] M. Herlihy and J. E. B. Moss. Transactional Memory: Architectural Support for Lock-Free Data Structures. In *Proceedings of the 20th Annual International Symposium on Computer Architecture* (1993).

[2] Luke Yen, Jayaram Bobba, Michael R. Marty, Kevin E. Moore, Haris Volos, Mark D. Hill, Michael M. Swift and David A. Wood. LogTM-SE: Decoupling Hardware Transactional Memory from Caches. In *Proceedings of HPCA '07: Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture* (2007).

[3] Arrvindh Shriraman, Sandhya Dwarkadas and Michael L. Scott. Flexible Decoupled Transactional Memory Support. In *Proceedings of the 35th Annual International Symposium on Computer Architecture* (2008).