C-023

トランザクショナルメモリを実現するスヌープキャッシュの遅延更新プロトコル
A Lazy-Updating Protocol of a Snoop Cache to Implement the Transactional Memory

一井 世界† 布目 淳‡ 平田 博章‡ 柴山 潔‡
Sekai Ichii Atsushi Nunome Hiroaki Hirata Kiyoshi Shibayama

1. はじめに

並行プログラミングにおける排他制御のための機構としてトランザクショナルメモリ[1]（Transactional Memory; 以下，TM とする）が注目されており、これをハードウェアで実現するものをハードウェア TM（Hardware TM; 以下，HTM とする）と呼ぶ．本稿では，メモリアクセス量の多いトランザクション（不可分に操作すべき一連のメモリアクセスの系列）に対応可能で，かつそれを低オーバヘッドで実現する HTM を提案する．

2. TM

TM では，トランザクションを投機的に実行する．トランザクションの終了時点で，他のトランザクションとアクセス競合が生じていなければ，実行結果をメモリに反映させる（**コミット**）．一方，アクセス競合が生じていれば，値の一貫性を保つため，アクセス競合を起こしたトランザクションの実行を中止し，その実行結果を破棄する（**アボート**）．トランザクション間のアクセス競合検出のポリシーには，以下の 2 種類があり，いずれの場合も，競合を検出した時点でいずれかのトランザクションをアボートすることにより，競合を解決する．

- **楽観的競合検出**: コミット時に競合検出を行う．
- **悲観的競合検出**: メモリアクセス時に競合検出を行う．

また，トランザクションの実行中は，それをアボートする場合に備えて，データの更新前の値を保存する．従って，トランザクションの実行中は，1 個のデータに対して，更新前の値と更新後の値の 2 種のバージョンが存在する．TM においてデータのバージョンを管理する方法は，以下の 2 種類に分けられる．

- **Eager Versioning**: メモリの値を書き換え，更新前の値を別領域に保存する方式.
- **Lazy Versioning**: メモリの値を書き換えず，更新後の値を別領域に保存する方式.

楽観的競合検出と Lazy Versioning を組み合わせて用いる方式に TCC[2]がある．TCC では，トランザクション実行中に更新した値を 1 次キャッシュに保存する．アボート時には，キャッシュをフラッシュするだけなので時間はかからないが，コミット時には，トランザクション中に更新したすべてのデータに関するアクセス競合の検出を行うので，時間的なオーバヘッドが生じる．

3. システム概要

本稿で提案するシステムの構成を図 1 に示す．トランザクション中に更新したデータを各 1 次キャッシュに記憶し，バージョン管理については，Lazy Versioning を用いる．従来の TM とは異なり，競合の検出と解決を分割して処理する．あるトランザクションがコミットする時に，競合関係にあるトランザクションをすべてアボートする点で，本方式は楽観的である．しかし，競合の検出をコミットまで待たず，メモリアクセスの各時点で検出しておくことにより，コミット時のオーバヘッドの発生を防ぐ.

†京都工芸繊維大学大学院工芸科学研究科情報工学専攻
‡京都工芸繊維大学大学院工芸科学研究科情報工学部門
Dept. of Information Science, Kyoto Institute of Technology

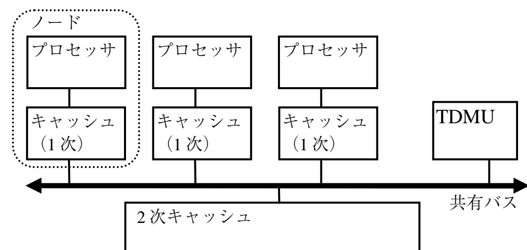

図 1　システム構成

トランザクション中に更新したコミット前のデータ（以下，**仮データ**とする）が 1 次キャッシュからあふれる場合に対応するために，本方式では，共有バス上で状況に応じてあたかも 1 次キャッシュまたは 2 次キャッシュであるかのように振る舞う**トランザクショナルデータ管理ユニット**（Transactional Data Management Unit; 以下 **TDMU** とする）を設ける．1 次キャッシュから追い出される仮データを，本来のメモリアドレスとは異なる領域に記憶し，共有バス上ではその本来のアドレスに記憶しているかのようにみせかける． 1 次キャッシュから追い出された仮データは，アクセス競合の検出対象としなければいけないので，その処理も TDMU が担当する．

LTM[3]では，1 次キャッシュがメモリ上に自身の拡張領域を設ける機能を持たせているが，本方式では，1 次キャッシュがこのような機構を備える必要はない．スヌープキャッシュプロトコルを巧みに利用して，仮想記憶管理にも関連するやや複雑なメモリ領域管理機能を TDMU に集約する.

4. 競合検出機構

トランザクション間のアクセス競合を検出するために、1 次キャッシュに以下の状態ビットを設ける.

- **W (Written)**ビット: ライン内のバイト領域ごとに設ける．トランザクションの実行中にそのバイト領域に書き込みを行ったことを示す.
- **FR (First Read)**ビット: ラインごとに設ける．ライン内の少なくとも 1 個の変数に対して，トランザクションにおける最初のアクセスが，読み出しであることを示す.
- **TS (Transactional Share)**ビット: キャッシュのラインごとに，各ノードに対応付けて設ける．トランザクションを実行中の他のノードで，このラインのデータを更新したことを示す.
- **C (Conflict)**ビット: 1 次キャッシュ全体で，各ノードに対応付けて設ける．そのノードで実行中のトランザクシ

Copyright © 2013 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

ョンとの間で，アクセス競合が生じていることを示す．そのトランザクションがコミットすると，自身はアボートする．

トランザクションを実行中のノードでは，プロセッサが 1 次キャッシュに書き込みを行う際に，W ビットをセットする．また，W ビットがセットされている領域から読み出す場合は，自らが生成した値を読み出すため，他のトランザクションとは競合しない（**メモリリネーミング**）．

W ビットがセットされていない領域から読み出す場合は，FR ビットをセットする．このとき，いずれかの TS ビットがセットされていれば，アクセス競合が発生するので，競合するトランザクションを実行中のノードに対応する C ビットをセットする．

バススヌーピングによって，トランザクションを実行している他のノードが書き込みを行ったことを検出した場合，そのノードに対応する TS ビットをセットする．また，このラインの FR ビットがすでにセットされていれば，書き込みを行ったノードに対応する C ビットもセットする．

あるノードがトランザクションをコミットするとき，共有バスを通じて全ノードに通知する．各 1 次キャッシュはこれをスヌープし，コミットしたノードに対応する C ビットがセットされていれば，実行中のトランザクションをアボートする．

以上のように，メモリアクセスの各時点でアクセス競合の発生を検出して C ビットに記憶しておくことで，コミット時の処理を高速に行うことができる．

5. 確定データにおけるバージョン管理

本稿では，コミット前の仮データに対して，それをコミットした後のデータを**確定データ**と呼ぶことにする．

例えば，2 個のノード A，B でそれぞれトランザクションを実行する場合を考える．まず，ノード A がライン L に対して書き込みを行うと，ライン L の更新領域に対応する W ビットをセットする．次に，ノード B で，同じライン L 内の異なる領域に書き込みを行うと，その更新領域に対応する W ビットをセットする．このとき，スヌープ処理により，ノード A，B のライン L の TS ビットをそれぞれセットする．その後，ノード A がコミットすると，A はライン L の更新領域を確定データとする．このとき，ノード B のライン L に対してノード A が更新した値を反映させる手段が必要となるが，同様の共有ラインが複数存在することもあるので，そのすべてのラインについて確定データを反映させるのには時間がかかる．また，更新した値を反映させる手段を設けないならば，実質的なアクセス競合が発生していない (false sharing) にもかかわらず，ノード B で実行しているトランザクションをアボートしなければならなくなる．

そこで，本方式では，コミット時点で確定データを他ノードのキャッシュに反映させるのではなく，後にそのデータへのアクセスが発生した時点で反映させる．従って，この例では，ノード A に確定データを置いたまま，ノード B ではそのまま仮データを使用する．後に，ノード B がコミットすると，2 種類の確定データが存在することになる．**2.** で述べたデータの値に関するバージョン管理とは別に，本方式ではキャッシュライン単位の確定データにおいても複数のバージョンが存在する．このようなバージョンの管理を行うために，1 次キャッシュに以下の状態ビットを設ける．

- **D (Dirty)**ビット：ライン内のバイト領域ごとに設ける．そのバイト領域に書き込みを行ったことを示す．W ビットがトランザクション中の書き込みを示すのに対し，D ビットは確定データとしての書き込みを示す．
- **P (Predecessor)**ビット：ラインごとに各ノードに対応付けて設ける．P ビットがセットされているノードのラインに対して，自ノードのラインのバージョンが 1 世代古いことを示す．
- **O (Obsolete)**ビット：ラインごとに設ける．W ビットがセットされていないバイト領域が無効であることを示す．

先の例では，ノード A がコミットすると，1 次キャッシュ内のすべての W ビットについて，それがセットされていれば，対応する D ビットをセットする（W ビットはクリアする）．一方，コミット通知を受けたノード B では，TS ビットがセットされているラインに対して，O ビットをセットする（TS ビットはクリアする）．

その後，ノード B がコミットすると，コミット通知を受けたノード A では，自身が保持するラインに対して，さらに新しいバージョンが作成されたかどうかをチェックする．すなわち，P ビットがセットされていない最新バージョンのラインに対して，ノード B がトランザクションの実行中に書き込みを行ったことを示す TS ビットがセットされていれば，P ビットをセットする（コミットしたノード B のラインが最新バージョンとなる）．

このように，P ビットを用いて確定データのバージョンの新旧を管理する．本方式では，トランザクションの実行中は，書き込みを行った時点ではなく，コミットする時点でその書き込みを確定する．コミット時のオーバヘッドを削減するためには，トランザクションを実行していないノードであっても，トランザクションを実行しているノードのメモリアクセスを監視していなければならない．

P ビットがセットされていない最新バージョンの確定データにおいて，D ビットがセットされていない領域の値は最新であるとは限らない．よって，このようなデータに対してアクセス要求が発生した時点で，古い値を含まないラインにまとめる．P ビットと D ビットを用いてラインのデータをマージし，2 次キャッシュ（メモリ）に書き戻す．

6. むすび

本稿では，低オーバヘッドの HTM を実現するためのスヌープキャッシュプロトコルを提案し，その概要について述べた．今後は，詳細な設計と性能評価を行う予定である．

謝辞

本研究の一部は日本学術振興会科学研究費補助金（基盤研究 （C） 25330058 ）の補助による．

参考文献

[1] M. Herlihy, J. Eliot, B. Moss, “Transactional Memory: Architectural Support for Lock-Free Data Structures,” Proceedings of the 20th Annual International Symposium on Computer Architecture, pp. 289-300 (1993).

[2] L. Hammond, V. Wong, M. Chen, B. D. Carlstrom, J. D. Davis, B. Hertzberg, M. K. Prabhu, H. Wijaya, C. Kozyrakis, K. Olukotun, “Transactional Memory Coherence and Consistency,” Proceedings of the 31st Annual Internatoinal Symposium on Computer Architecture, pp. 102-113 (2004).

[3] C. S Ananian, K. Asancovic, B. C. Kuszmaul, C. E. Leiserson, S. Lie, “Unbounded Transactional Memory,” Proceedings of the 11th International Symposium on High-Performance Computer Architecture, pp. 316-327 (2005).

Copyright © 2013 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.