1L-5

# 投機的メモリアクセスを実現するシステムソフトウェアの検討

松野　穣 † 大津　金光 † 大川　猛 † 横田　隆史 † 馬場　敬信 †
† 宇都宮大学工学部情報工学科

## 1　はじめに

近年マルチコアプロセッサの普及により, スレッドレベル並列性の活用による高速化が重要になっている. 特にシングルプログラムの高速化においては, プログラム内の様々な依存関係により並列実行が阻害される場面も多く高性能化への障害となっている. この問題に対してプログラムの実行内容を事前に予測して投機的に処理する投機的マルチスレッド実行が有効である.

この投機的な手法は本来, ハードウェアサポート [1] の下で行うのが一般的であるが, 広く普及しているプロセッサの多くはそのような機能を持っておらず, 投機的マルチスレッド手法の実現が難しい.

そこで本稿では, 特別なハードウェアを使わずにソフトウェアによる投機的メモリアクセスの実現方法を検討する.

## 2　投機的マルチスレッド実行モデル

本研究で前提とする投機的マルチスレッド実行モデルでは, 事前のプロファイリングで最も実行する割合の高い実行経路 (パス) の情報を取得し, そのパスに最適化したコードを投機的に実行する. 並列処理される各スレッド間にはプログラムオーダに基づいた親子関係が存在し, 先行スレッドから順に実行結果をコミットすることを保証しなくてはならない. 投機に失敗したスレッドはその実行内容を破棄し, 回復処理で実行前の状態に戻したあと非投機コードを実行する. この時, 後続スレッドが存在するならば, それらの実行も同様に破棄して再実行を行わせる.

この投機的マルチスレッド実行の動作を図 1 を用いて説明する. この図は 4 つのスレッドの投機的マルチスレッド実行の動作を表し, 左端が先頭スレッド, 他は後続スレッドである. 親子関係が保証されるため, 左端のスレッドの実行結果がコミットされた後, 2 番目のスレッドはコミット処理に入る. この左側 2 つのスレッド実行は共に投機成功の例である. 一方で 3 番目のスレッドは投機に失敗した場合で, 回復処理を行うと共に後続スレッドにも実行のロールバック指令を出す. 3 番目のスレッドが非投機コードを実行してコミットした後, 後続スレッドが実行結果をコミットする.

この投機的マルチスレッド実行モデルをハードウェアサポートなしで実現するために必要な要件は (1) 親子関係の保証, (2) 投機失敗時の回復処理, (3) 後続スレッドへのロールバック指令の 3 点が挙げられる.

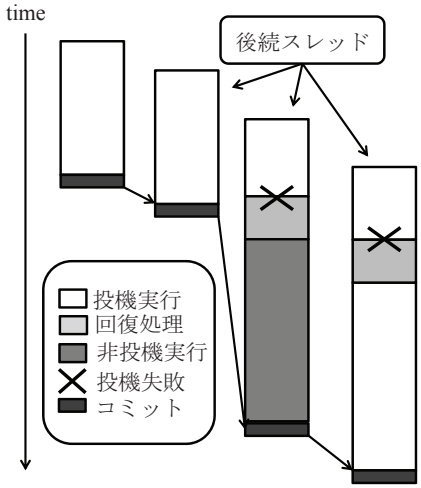


図 1: 投機的マルチスレッド実行の動作

## 3　ソフトウェアトランザクショナルメモリ (STM)

投機的マルチスレッド実行の要件を満たすための方法として, 本研究ではトランザクショナルメモリに着目した. STM はトランザクショナルメモリをソフトウェアのみで実行するアクセス手法である. これは共有変数へのアクセスをトランザクションとみなしてアクセス時に並列実行中の他スレッドの操作を止めず, 実行開始時の記録を取得して処理を続行する. トランザクションの処理が完了して結果をコミットする時に他スレッドからのストアなどの有無を調べ, 競合していればトランザクションをロールバックして再実行する. この機能を用いて, 投機的メモリアクセス実現を図る.

## 4　STM と投機的メモリアクセスの相違

STM で投機的マルチスレッド実行を実現するにはいくつかの課題がある. 投機的マルチスレッド実行では各スレッドの実行結果のコミット順を保証するため, 図 2 のように必ず先行スレッドの実行結果からコミットされる. だが STM はスレッド間の親子関係の概念を持たないため, トランザクションの競合やオーバーヘッドによって後で処理すべきスレッドの実行結果を先にコミットする可能性がある. 図 2 がその例であり, STM ではプログラムの正しい実行を保証できない.

そこで課題の解決のためにスレッド間で依存する共有変数へのアクセス時にスレッドの親子関係に基づいたバージョン管理を行う. なお, いくつかの STM の実装が存在するが, 文献 [2] および予備的な性能評価を行った結果より, 本研究では TinySTM[3] をベースとして使用する.

Consideration of Software Speculative Memory Access
†Yutaka Matsuno, Kanemitsu Ootsu, Takeshi Ohkawa, Takashi Yokota and Takanobu Baba
Department of Information Science, Faculty of Engineering, Utsunomiya University (†)

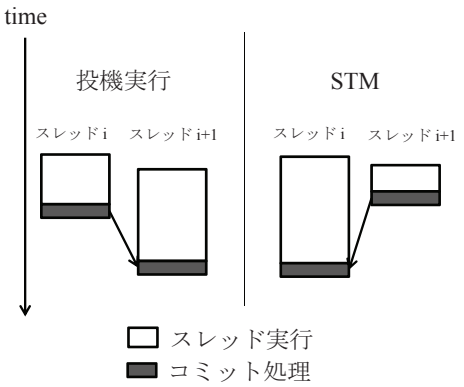


図 2: 投機実行と STM による並列実行の動作比較

## 5 バージョン管理のデータ構造

バージョン管理用に各スレッドに以下の構造体データを共有変数ごとに用意する. 図 3 のようにスレッド i+1 が既に共有変数 A へアクセスしていたスレッド i から A の値を読み込むことで正しい実行を保証する.

| | |
|---|---|
| thread_id | : スレッド番号 |
| valid_f | : データ値の有効性 |
| write_f | : 書き込みの有無 |
| read_f | : 読み込みの有無 |
| rollback_f | : 再実行の必要性 |
| dep_thread_id | : 読み込み先のスレッド番号 |
| thread_value | : データ値 |

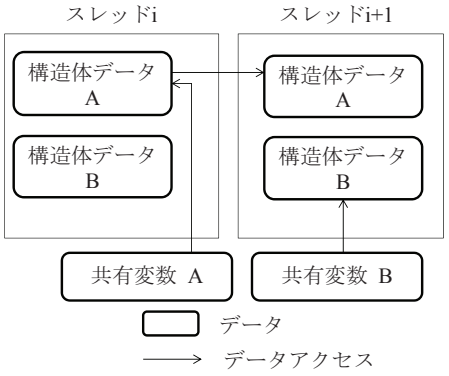


図 3: 共有変数と各スレッドの構造体データの関係

thread_id はスレッドの生成順の通し番号であり, 親子関係を把握するために使う. valid_f は後述の thread_value の有効性を示し, データロード時に用いる. write_f はトランザクション中の共有変数へ書き込みの有無を示すフラグであり, これもデータロード時に使われる. read_f は読み込みの有無を示すフラグであり, 依存違反検知で必要となる. rollback_f はそのスレッドの投機失敗を示すフラグである. dep_thread_id はデータロード時のデータ取得先のスレッド番号であり, 依存違反検知に用いる. thread_value はこのスレッドが共有変数に書き込むデータ値である.

## 6 スレッドの動作

各トランザクションの動作を述べる.

### 6.1 データロード

スレッドは thread_id を用いて自分より先行するスレッドの通し番号を探す. そのスレッドの valid_f と write_f を確認し, thread_value を読み込むと同時に自分の dep_thread_id にデータ取得先の thread_id を記録して read_f をセットする. 有効なデータ値がない場合は共有変数から読み込み, 同様の処理を行う.

### 6.2 データストア

スレッドはデータ値を自分の thread_value に書き込む. この時に write_f と valid_f をセットする. その後, 後続スレッドの読み込み依存違反を検知するため, 自分の thread_id と後続スレッドの read_if, dep_thread_id を比較参照する. 検知した場合は対象のスレッド以降の全てにおいて rollback_f をセットする.

### 6.3 ロールバック

実行中のスレッドが自分の rollback_f を確認し, 必要なら回復処理を行う. 他スレッドからのアクセス防止のために valid_f を更新後, トランザクション開始点へ戻って各種データを初期化後, 再実行する.

### 6.4 コミット

thread_id を参照することで, 先行スレッドから順にコミット処理を行う. write_f が立っている場合は thread_value を管理している共有変数へ反映する.

## 7 おわりに

本稿では, 投機的メモリアクセスを実現するための手段として, STM にバージョン管理の機能を持たせた投機的メモリアクセスを実現するシステムソフトウェアを検討した. 今後, 本システムの実装と性能評価を予定している.

**謝辞**

本研究は，一部日本学術振興会科学研究費補助金 (基盤研究 (C)24500055，同 (C)24500054) の援助による.

**参考文献**

[1] M.Franklin, G.S. Sohi, “ARB: A Hardware Mechanism for Dynamic Reordering of Memory Reference”, IEEE Transactions on Computers, Vol.45, No.5, pp.552-571, 1996.

[2] Gulfam Abbas，Naveed Asif, “Performance Tradeoffs in Software Transactional Memory”, Master Thesis Computer Science, School of Computing Blekinge Institute of Technology Sweden, No:MCS-2010-28, May 2010

[3] Pascal Felber, Christof Fetzer, Torvald Riegel, “Dynamic Performance Tuning of Word-Based Software Transactional Memory”, In PPoPP ’08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM, 2008.