RC-001

スレッドレベル並列投機実行のためのメモリリネーミング機構
A Memory Renaming Mechanism for a Thread-Level Speculation on a Multiprocessor

平田 博章† 藤井 崇弘‡ 藤 皓平‡ 森田 清隆‡* 布目 淳† 柴山 潔†
Hiroaki Hirata Takahiro Fujii Kouhei Fuji Kiyotaka Morita Atsushi Nunome Kiyoshi Shibayama

1. はじめに

マルチコアプロセッサが商用のマイクロプロセッサとして市販される一方で，プログラムからスレッドレベルの並列性を抽出するのは困難な現状にある．そこで本稿では，プログラムから粗粒度の並列性を抽出し，スレッドとして投機的に並列実行を行うマルチプロセッサアーキテクチャを提案する．

2. スレッドの粒度とメモリデータの依存関係

スレッドレベル並列処理では，メモリ上のデータの使用に関して依存解析を行い，ハザードの発生を回避するとともに並列性を抽出することが重要である．メモリ上のデータに対する依存解析機構の先がけ的な提案として ARB (Address Resolution Buffer)[1] や SVC (Speculative Versioning Cache)[2]があり，スレッドレベルの並列処理に利用することが可能である．また，TM (Transactional Memory)を投機スレッド実行の管理に適用するアプローチ[3]も考えられる．しかし，これらはいずれも，1 個のスレッドがアクセスするメモリデータ量を勘案すれば，粒度が粗く，動的な実行命令数が非常に多いスレッドを並列処理単位とする場合には向いていないものと考えられる．

ループを対象としてスレッドレベルの並列性を抽出する場合，最内ループを並列化の対象とするのが一般的であり，実際にベクトルコンピュータや命令スケジューリングなどにおいて大きな効果を上げてきた．基本ブロックや手続き（関数），ループ（多重ループ）などの制御構造に基づいて，マルチグレインの並列投機実行を行うアーキテクチャも提案されているが，例えば，Cascadi[4]ではメモリアクセスに関する並列性について十分な検討がなされていないため，性能向上比も小さい．また，Pinot[5]では SVC を使用しており，投機実行中のスレッドのデータをキャッシュから追い出さなければならなくなった時点でそのスレッドの実行を中断・待機する．従って，原理的には粒度の粗さに制限はないものの，実質的には非常に粗い粒度の並列処理は想定していないものと受け取れる．このほか，スレッドレベルの並列投機実行を行う初期の提案に Hydra[6]があり，2 次キャッシュの前段に，ARB と同等の機能および実装上の複雑さを有する書き込みバッファを配置している．この書き込みバッファの容量によってスレッドの規模が制限される点で，あまり粗い粒度の並列処理は想定していないものといえる．

一方，アルゴリズムを念頭に置いて人手で並列化を行う場合は，最外ループまたは最外ループに近い内側のループに着目することが少なくない．そこで，本研究ではそのようなループから非常に粗い粒度のスレッドを生成して投機的に並列実行する方式を提案する．変数の使いまわしによってスレッド間に多数の逆依存関係が生じることが予想されるため，真の依存関係であるフロー依存以外の依存関係を除去しなければ，スレッドの投機実行開始直後からハザードが発生し，並列処理効果はほとんど望めない．そこで，スレッドがアクセスする各変数に対して，投機実行によって生成されるデータの複数のバージョンをキャッシュ上に保存することにより，メモリリネーミングを実現する．SVC や Hydra でも同様にメモリリネーミングの機能が実現されているが，本方式ではそれらに比べてより粗い粒度の並列処理を行うため，キャッシュあふれに対する対策は必須である．

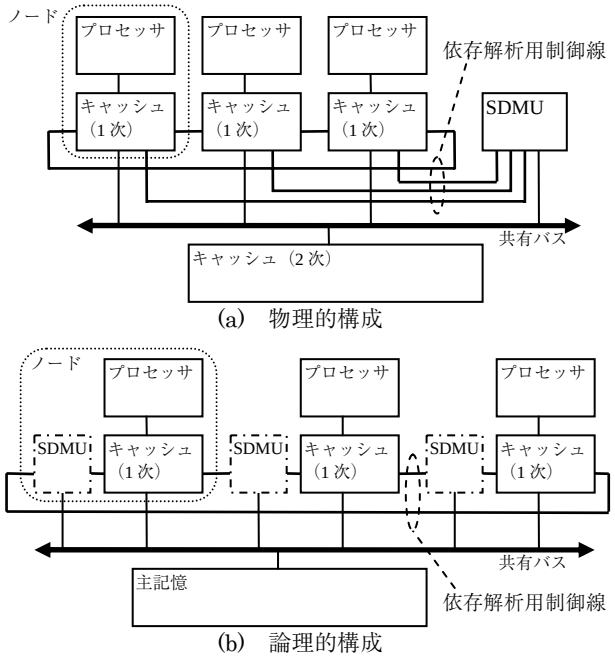

図 1 システム構成

3. メモリリネーミング機構

3.1 システム概要

本稿で提案するメモリリネーミング機構を用いたマルチプロセッサシステムの構成を図 1に示す．共有バスで接続した各プロセッサ上でスレッドを実行する．並列化対象のループに対して，その各イタレーションをスレッドとして実行する．$i-1$ 番目までのイタレーションの実行が完了しているものとすると，i 番目のイタレーションを実行するスレッドを**確定スレッド**と呼び，これと並列に，$i+1$ 番目以降のイタレーションを投機的に実行するスレッドを**投機スレッド**と呼ぶことにする．確定スレッドが終了した時点で，その次のイタレーションを実行する投機スレッドが確定スレッドとなる（コミット）．また，

†京都工芸繊維大学大学院工芸科学研究科情報工学部門
‡京都工芸繊維大学大学院工芸科学研究科情報工学専攻
Dept. of Information Science, Kyoto Institute of Technology
＊現在，三菱電機株式会社 Mitsubishi Electric Corp.

投機スレッドと確定スレッドが生成したデータを，それぞれ，**投機データ**，**確定データ**と呼ぶことにする．

スヌープキャッシュプロトコルを利用してスレッド間の依存解析を行うための専用機構を 1 次キャッシュに設け，依存関係を破壊するハザードを検出した場合には，それに関連する投機スレッドのみをアボートして，イタレーションの最初から再実行する．各プロセッサの 1 次キャッシュにおいて，スレッド間で共有する変数の別バージョンを格納することで，メモリ上のデータに対してリネーミングを行い，スレッド間の逆依存と出力依存を除去する．

SVC ではコピーバック方式，Hydra ではライトスルー方式の 1 次キャッシュをそれぞれ用いており，また，いずれも投機データを格納する 1 次キャッシュのあふれには対応していない．本方式の 1 次キャッシュはコピーバック方式を採用し，共有バス側からはその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動作する**投機データ管理ユニット(SDMU; Speculative Data Management Unit)**を設けることで，キャッシュあふれに対応する．SDMU は，1 次キャッシュから追い出される投機データのメモリアドレスを別のアドレスにマッピングして 2 次キャッシュ（またはそれ以降の階層のメモリ）に記憶し，共有バス上では SDMU 内に記憶してかのように振舞う．2 次キャッシュ以降の階層のメモリに置かれる投機データは，アプリケーションプログラムからは別のアドレスのデータとして可視となるが，逆に，投機データをバックアップするための領域をアプリケーションのアドレス空間内にあらかじめ確保しておかなければならない．

1 次キャッシュから追い出された投機データについてもスレッド間の依存解析の対象としなければならないので，その処理を SDMU が担当する．従って，SDMU は，物理的には図 1(a)に示すように 1 台であるが、論理的には，図 1(b)に示すように各 1 次キャッシュと仮想的に対を構成し，1 次キャッシュの容量を拡張する機能を提供する．

3.2 依存解析機構

3.2.1 アクセス状態ビット

メモリ上のデータに対するスレッド間の依存解析を行うために，1 次キャッシュの各ラインごとに以下のアクセス状態ビットを設ける．

- **W(Written)ビット**： ライン内の各バイト領域に対応付けて設け，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**： ライン中の少なくとも 1 個の変数に対して，そのスレッドにおける最初のアクセスが読み出しであることを示す．
- **O(Obsolete)ビット**： ライン中のまだ書き込みを行っていない変数のメモリアドレスに対して，先行スレッドが書き込みを行ったことを示す．

プロセッサが各自の 1 次キャッシュに書き込みを行うとき、対応する W ビットをセットする．また，W ビットがセットされていない領域から読み出す場合，FR ビットをセットする．O ビットについては，自プロセッサからのアクセスではなく，先行スレッドを実行しているプロセッサが書き込みを行ったことをバススヌーピングによって検出し，ハザードが発生していないことを確認する．ハザードが検出されず，かつ，その書き込みアドレスに対応する W ビットがセットされていなければ，O ビットをセットする．

3.2.2 メモリ間依存解析

各 1 次キャッシュは，プロセッサ側からリード要求とライト要求を，また，共有バス側からロード要求と依存解析要求を受け付け，アクセス状態ビットを用いて各ラインの状態を管理する．その状態遷移を表 1に示す．

プロセッサからのライト要求（表 1の「自 Write」）に対してキャッシュにヒットする場合，キャッシュにデータを書き込むのと同時に，対応する W ビットをセットする．また，書き込みアドレスとデータサイズを共有バスに出力し，他のキャッシュに対して依存解析要求を行う．キャッシュ間をリング状に接続する制御線を用いて，先行スレッドを実行するノードから後続スレッドを実行するノードへと信号を伝播することにより，各ノードでハザード検出を行う必要があるか否かを判断する．

一方，キャッシュミスの場合は，まず，共有バスを用いてロード要求を行う．ロード要求はラインデータの最新バージョンを得るためのもので，論理的にキャッシュ間をリング状に接続するロード要求専用の制御線を用いて，このロード要求に応答すべきキャッシュを決定する．応答責任の優先順位は，過去にキャッシュから追い出したデータを管理している SDMU が最も高く，次に，直近の先行スレッドを実行するノードから確定スレッドを実行しているノードの順に低くなり，最後は 2 次キャッシュとなる．最新バージョンのコピーをキャッシュにロードした後の処理は，キャッシュヒットの場合と同様であり，W ビットをセットするとともに依存解析要求を行う．

プロセッサからのリード要求（表 1の「自 Read」）に対してキャッシュにヒットする場合，アクセス対象のデータに対応する W ビットがセットされていれば，それは自スレッドで生成した値であり，先行スレッドとの依存関係は存在しない（メモリリネーミング）．W ビットがセットされていなければ，O ビットの値によって次のように動作する．まず，O ビットがセットされていない場合は，そのデータは最新のバージョンであるので，読み出すと同時に FR ビットをセットする．一方，O ビットがセットされている場合は，データをキャッシュにロードした後に先行スレッドがその値を書き換えたことを示しているので，ラインをロードし直さなければならない．そこで，共有バスにロード要求を行って最新バージョンのデータを取得し， W ビットがセットされていないバイト位置のみを更新して，O ビットをリセットするとともに FR ビットをセットする．

一方，キャッシュミスの場合は，共有バスを用いてロード要求を行い，その後はキャッシュヒット時と同様に動作する．

以上のように，あるキャッシュにおいてミスヒットが発生すると，共有バスを介してロード要求を他のノードのキャッシュに伝える．それぞれのノードのキャッシュは，当該データを保持しているかどうかを検査するのと並行して，応答責任の有無を調べる．リング状の専用線において，後続スレッドを実行する隣接ノードからの入力が応答要求を示していて，かつ，ロード要求に対してキャッシュヒットならば，先行スレッドを実行する隣接

ノードへの応答要求信号の伝播を遮断する．もちろん，確定スレッドを実行するノードでも応答要求信号の伝播を遮断する．

また、ライト要求に起因して共有バスに出力される依存解析要求（表 1の「先行 Write」）についても，同様に，当該データを保持しているかどうかを検査するのと並行して，リング状の専用線を用いて，ただしロード要求の場合とは逆方向に，解析要求を伝播する。解析要求の伝播を遮断するのは，依存解析要求に対してキャッシュがヒットし，かつ，書き込みアドレス領域に対応する W ビットがセットされている場合（この場合，先行スレッドによる書き込みは自ノード以降の後続スレッドには影響しない）である． W ビットがセットされていない場合は，何らかのハザードが生じている可能性がある．このとき，FR ビットがセットされていれば、先行スレッドによって書き換えられる前の値を用いて投機実行していることになるので、スレッド実行をアボートしてスレッドの最初から再実行する．一方，FR ビットがセットされていなければ，O ビットをセットする．先行スレッドが書き換えた値を自スレッドで実際に使用するかどうかわからないため、この段階ではハザードとして検出することは行わず，後にそのデータにアクセスする時点まで判断を遅らせる．ただし，FR ビットと O ビットはライン単位で設けているので，同じライン内において先行スレッドが書き換えた領域と異なる領域をリードする場合でも部分更新が発生し，また，先行スレッドが書き換えた領域を自スレッドが書き換える場合でも，O ビットをリセットすることはできない．

表 1　アクセス状態の遷移（ライン単位）

現状態 (W's, FR, O)	次状態 (W's, FR, O) 自 Read	自 Write	先行 Write
(∀0, 0, 0)[注1]	(∀0, 1, 0)	(∃1, 0, 0)	—
(∀0, 1, 0)	(∀0, 1, 0)	(∃1, 1, 0)	ABORT
(∃1, 0, 0)	(∃1, 0, 0), (∃1, 1, 0)	(∃1, 0, 0)	(∃1, 0, 0), (∃1, 0, 1)
(∃1, 1, 0)	(∃1, 1, 0)	(∃1, 1, 0)	(∃1, 1, 0), ABORT
(∃1, 0, 1)	(∃1, 0, 1), (∃1, 1, 0)	(∃1, 0, 1)	(∃1, 0, 1)

注1) 初期ミスによってキャッシュにロード（アロケート）した直後の内部初期状態であり，実際にはこのような状態は存在しない．

3.3 バージョン管理機構

確定スレッドが終了すると，隣接するプロセッサで実行中の投機スレッドをコミットし，確定スレッドとして実行を継続する．コミットされたデータをキャッシュに残したまま，順次，投機スレッドがコミットされてゆく状況では，確定データについても複数のバージョンが存在することになる．本方式では，投機データ間のバージョンの前後関係はリング接続の制御線を用いることで固定しているが，確定データについては，投機実行中からアクセスしていたかコミット後に初めてアクセスしたかにより，必ずしも順序を固定できない．

そこで，本方式では，最新の確定バージョンを保持することに責任を持つキャッシュが必ず 1 個だけ存在するように制御する．そのため，1 次キャッシュのラインごとに以下の状態ビットを設ける．

- **L(Loaded)ビット**：　各プロセッサ（ノード）に対応付けて設け，後続の投機スレッドを実行するノードのキャッシュがこのラインをロードしたことを示す．
- **P(Predecessor)ビット**：　複数存在する確定データの中で，最新バージョンではないことを示す．

他のノードからロードされるとき，L ビットをセットする．しかし，その後，ロードしたノードで実行中の投機スレッドがアボートする場合もあり得る．このとき，そのノードは共有バスを用いてアボートしたことを全ノードに通知する．それを捕らえて，各キャッシュでは，L ビットを一斉にリセットする．また，投機スレッドをコミットする場合も，同様に共有バスを用いて通知するので，各キャッシュでは，コミットしたノードに対応する L ビットがセットされているラインに対して，その P ビットをセットする．

SVC では明示的なポインタを用いてバージョンの前後関係を管理するが，本方式では，L ビットと P ビットを用いることで同様の機能を実現する．しかし，本質的な違いは，SVC ではワード単位のみでバージョンを管理するのに対し，本方式ではライン単位でかつバイト領域へのアクセスを考慮して管理する点から生じる．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（ライン）において，W ビットがセットされていない領域は最新であるとは限らない．従って，例えば後続の確定スレッドがこのようなデータに初めてアクセスする場合（キャッシュミスの場合はもちろん，キャッシュに存在する場合もキャッシュミスと同様に扱う）には，最新のデータのみからなるラインにまとめなければならない．そこで，L ビットと P ビットを用いて共有バス上で最新バージョンのラインから古いバージョンのラインの順に再帰的に検索し，W ビットを参照しながらマージする．高々ノード数分の連続バスサイクルを要するが，マージデータが完成した時点でそれより古いバージョンのラインを一斉に無効化することにより時間短縮を図る．その後，マージされたラインのデータに対するアクセスは，2 次キャッシュ（メモリ）アクセスと同様に扱うことができる．

以上のように，確定データであっても，最新データにマージする前の状態とそれ以外の状態（マージ後またはメモリから読み出した最初のバージョンのコピーしか存在しない状態）とを区別して管理する．

3.4 投機データ管理ユニット（SDMU）

1 次キャッシュのライン置換において追い出さなければならない投機データは，そのラインの状態ビットも含めて SDMU に転送してバックアップする．SDMU 内においてそれらのバックアップデータを管理するためのデータ構造を図 2に示す．ライン管理テーブルはダイレクトマッピング方式のキャッシュと同様の検索機能を備え，マッピングコンフリクトが発生する場合は連結リストを用いて管理する．ライン管理テーブルの各エントリはそのアドレスに対応するすべてのノードの 1 次キャッシュの状態ビットとデータを記憶する．ただし，ラインデータ自体は 2 次キャッシュに保存する．1 次キャッシュからラインを追い出す場合は，SDMU は 2 次キャッシュであるかの

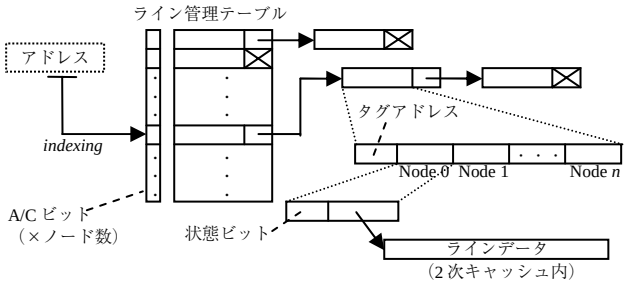

図 3　SDMU におけるライン管理

ように振る舞い，また，そのラインデータを再び 1 次キャッシュにロードする場合は，他のノードの 1 次キャッシュのように振る舞うことで，スヌープキャッシュプロトコルに対する変更を最小限に留める．また，SDMU は 2 次キャッシュではないので，同一の投機データは，1 次キャッシュと SDMU のいずれか一方にのみ存在し，同時に両方に格納されることはない．

スレッドのコミットやアボートについては，その都度，共有バスを用いて SDMU にも通知する．このとき，SDMU で管理しているすべてのデータに対して無効化や確定処理を行うと，SDMU が性能上のボトルネックとなりかねない．そこで，これらの処理をいずれかの 1 次キャッシュのライン置換が発生するまで遅らせて時間的に分散して処理できるよう，ライン管理テーブルの各エントリに，各ノードに対応させて，アボートの通知があったことを示す **A(Abort)ビット**とコミットの通知があったことを示す **C(Commit)ビット**を設ける．それぞれの通知があった場合にそのノードに関するこれらのビットを一斉にセットし，ライン置換等の処理を行う際に例えば A ビットがセットされていれば，連結リストをたどりながら該当ノードのラインを無効化した後，A ビットをリセットする．なお，コミットやアボートの通知を検知した際，すでに A/C ビットのいずれかがセットされている場合には，変更は行わない．その理由は，例えば A ビットがセットされている状態でコミットの通知を受けた場合，A ビットがセットされてから新たにラインが追加されてはいないので（もしも追加されていれば無効化の処理を行って A ビットもリセットしているはずである），連結リスト中に確定すべきラインデータは存在しないからである．

SDMU の機能を実現するには，アボート/コミットの遅延処理や，連結リストノードと投機データ自体を記憶するための記憶領域の管理機能が必要であり，これらすべてをハードウェアのみで実現するのは得策ではない．従って，部分的にソフトウェアを用いて SDMU を実現する．また，連結リストをたどる点や，同時に複数のノード分の処理を担当する（例えば，図 1の例では，ある 1 次キャッシュからのロード要求に対して，3 台の 1 次キャッシュに相当する処理を物理的には 1 台で行う）点で，共有バスからの要求に対して応答に複数サイクルの遅延を要する場合も生じる．このため，少なくとも確定スレッドの実行性能を阻害しないよう，SDMU 内で行う処理の時間的隠蔽手法を検討中である．

4. 評価

PowerPC[7]の命令レベルシミュレータを作成し，SPEC CPU2006[8]ベンチマーク（入力には test データセットを使用）を対象に，外側のループの各イタレーションの全実行命令を，(a) 逆依存および出力依存によるハザードが生じる命令，(b) フロー依存によるハザードが生じる命令，(c) 依存関係によるハザードが生じない命令（ロード/ストア命令以外の命令も含む），に分類してそれぞれの割合を調べた結果を図 3に示す．これより，逆依存/出力依存によるハザードが多く（最大約 43%）生じることが分かり，メモリリネーミングによってそれらを取り除くことにより，並列性抽出の機会が劇的に増加するものと期待できる．

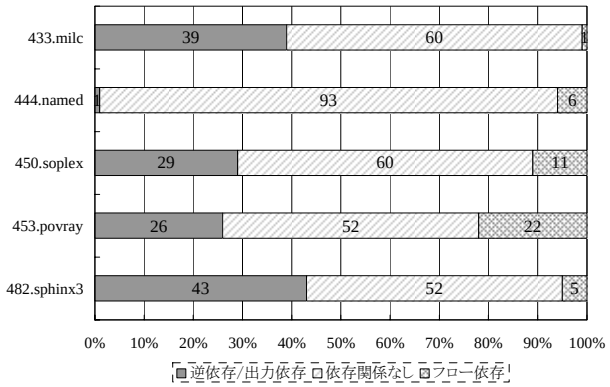

図 2　依存関係に関する命令の分類

5. むすび

本稿では，並列投機スレッド実行のためのメモリリネーミング機構の概要について述べた．今後は，本機構の詳細設計とともにフロー依存の影響を最小化するための方策を検討し，システム全体の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）22500046）の補助による．

参考文献

[1] M. Franklin and G.S. Sohi, “ARB: A Hardware Mechanism for Dynamic Reordering of Memory References,” IEEE Trans. on Computers, vol. 45, No. 5, pp. 552–571 (1996).
[2] T.N. Vijaykumar, S. Gopal, J.E. Smith, G. Sohi, “Speculative Versioning Cache,” IEEE Trans. on Parallel and Distributed Systems, Vol. 12, No. 12, pp. 1305-1317 (2001).
[3] J. Chung, H. Chafi, C.C. Minh, A. McDonald, B. Carlstrom, C. Kozyrakis, K. Olukotun, “The Common Case Transactional Behavior of Multithreaded Programs,” Proc. of the 12th Annual Int’l Symposium on High-Performance Computer Architecture, pp. 266–277 (2006).
[4] D.A. Zier, B. Lee, “Performance Evaluation of Dynamic Speculative Multithreading with the Cascadia Architecture,” IEEE Trans. on Parallel and Distributed Systems, Vol. 21, No. 1, pp. 47-59 (2010).
[5] T. Ohsawa, M. Takagi, S. Kawahara, S. Matsushita, “Pinot: Speculative Multi-threading Processor Architecture Exploiting Parallelism over a Wide Range of Granularities,” Proc. of the 38th Annual Int’l Symposium on Microarchitecture, pp. 81-92 (2005).
[6] L. Hammond, B.A. Hubbert, M. Siu, M.K. Prabhu, M. Chen, K. Olukolun, “The Stanford Hydra CMP,” IEEE Micro, Vol. 20, No. 2, pp. 71-84 (2000).
[7] Freescale Semiconductor, “Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture” (2001).
[8] Standard Performance Evaluation Corporation, “SPEC CPU2006”, http://www.spec.org/cpu2006/