C-003

スレッドレベル並列化のためのメモリリネーミング
Memory Renaming for Thread-Level Parallelization

藤井 崇弘[†] 森田 清隆[†] 布目 淳[‡] 平田 博章 柴山 潔[‡]
Takahiro Fujii Kiyotaka Morita Atsushi Nunome Hiroaki Hirata Kiyoshi Shibayama

1. はじめに

マルチスレッドプロセッサや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，逐次実行を前提として記述されたプログラムから自動的にスレッドレベルの並列性を抽出することは依然として困難な状況にある．そこで，プログラム中のできるだけ外側のループを対象として，その各イタレーションをスレッドとして並列に実行するものと想定した場合にどのような要因が並列実行（並列性抽出）の障害となるかを調べた．その結果，スレッド間に依存関係を生じさせる変数の多くは，スレッド内での最初のアクセスが書き込みで始まっていることが判明した[1]．つまり，その変数の使用に関してスレッド間で依存関係が生じるが，変数の値そのものを後続のスレッドで使用するわけではなく，このような本質的でない依存関係のために並列実行の効果が期待できないものとなっている．

命令レベルでこのような依存関係が生じる場合，例えばスーパスカラプロセッサなどでは，レジスタリネーミングによって依存関係を除去している．概念的には同じアプローチでスレッド間の依存関係を除去することができるが，並列処理の粒度の違いから，メモリ上のデータに対してリネーミングを行う方式を開発しなければならない．また，その前提として，メモリ上のデータに対する依存解析を行う必要があり，この点については過去にも試み[2][3]がなされている．

本稿では，キャッシュを用いてメモリ上のデータのアクセスに関するスレッド間の依存解析を行うとともに，メモリ上のデータに対してリネーミングを行う方式を提案する．本稿で提案するメモリリネーミングのみではすべての種類の依存関係を取り除くことは不可能であり，完全なスレッドレベル並列化を達成するまでには至らないが，並列性抽出の可能性について報告する．

2. メモリリネーミング

2.1 メモリリネーミングの概要

逐次実行を前提として記述されたプログラムにおいて，できるだけ外側のループを対象に，その各イタレーションをスレッドとして投機的に並列実行する．実行環境としてマルチコアプロセッサを想定し，1 個のプロセッサコアで 1 個のスレッドを実行するものとする．プロセッサコアごとにデータキャッシュを備え，各データキャッシュの容量は十分に大きく，メインメモリへの追い出しが発生しないものと仮定する．

1 個の変数に対してそれぞれのキャッシュにその変数の別バージョンを格納することで，メモリリネーミングを実現する．スレッド間の依存関係およびメモリリネーミングの制御を行うために，データキャッシュ内に各変数へのアクセス履歴を表す情報を保存する．具体的には，変数単位で（実際にはアドレスごとに）アクセス履歴を保存するための V 状態ビットを，また，キャッシュのライン単位でアクセス履歴を保存する L 状態ビットを設ける．

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

2.1.1 V 状態ビット

変数へのアクセス履歴を保存するために 2 ビットの V 状態ビットを設ける．V 状態ビットは，以下に示す 2 種類のビットから成る．

- VFR ビット
  ある変数に対する最初のアクセスが読み出しであること（First Read）を示す．
- VW ビット
  ある変数に対して書き込みが行われたこと（Written）を示す．

また，キャッシュにアクセスする際に，以下に示す条件で V 状態ビットをセットする．

- VFR ビットをセットする条件
  VFR ビットと VW ビットがいずれもセットされていない状態で変数の値を読み出すとき．
- VW ビットをセットする条件
  変数に値を書き込むとき．

2.1.2 L 状態ビット

キャッシュのラインへのアクセス履歴を保存するために 2 ビットの L 状態ビットを設ける．L 状態ビットは，以下に示す 2 種類のビットから成る．

- LFR ビット
  初回のアクセスが読み出しである変数がライン中に 1 個以上存在することを示す．
- LW ビット
  ライン中のすべての変数に対して書き込みが行われたことを示す．

また，キャッシュにアクセスする際に，以下に示す条件で L 状態ビットをセットする．

- LFR ビットをセットする条件
  ライン中のすべての変数の VFR ビットと VW ビットがいずれもセットされていない状態で，ライン中の変数の値を読み出すとき．
- LW ビットをセットする条件
  変数に値を書き込むことで，ライン中のすべての変数の VW ビットがセットされ，かつ，すべての VFR ビットがリセットされているとき．

なお，キャッシュミスが発生した場合は，キャッシュに新たなラインを確保し，そのラインの L 状態ビットとそのラインに含まれる変数の V 状態ビットをリセットする．その後，上記の条件で状態ビットをセットする．

Copyright © 2011 by Information Processing Society of Japan and
The Instiute of Electronics, Information and Communication Engineers
All rights reserved.

2.2 スレッド間の依存解析

複数のスレッドを並列実行するときに，V 状態ビットと L 状態ビットを用いてメモリアクセスに関するスレッド間の依存解析を行う．説明を簡単化するため，ここでは 1 ラインに 1 変数のみが格納されるものとする．このとき V 状態ビットと L 状態ビットは同一視できるため，以下では V 状態ビットと L 状態ビットをまとめて状態ビット，VFR ビットと LFR ビットをまとめて FR ビット，VW ビットと LW ビットをまとめて W ビットとして説明する．

2.2.1 変数の値を読み出すときの処理

あるスレッド（スレッド番号を k とする）で変数 A の値を読み出すときの処理を図 1 に示す．ただし，スレッドの番号は最も先行しているスレッドから順に 0 番から割り当てるものとする．

1. 自スレッドの番号 k を i とし，スレッド i のキャッシュに保存されている変数 A の状態ビットを調べる．
   - FR ビットと W ビットのどちらか一方でもセットされている場合，手順 2 へ進む．
   - FR ビットと W ビットがセットされておらず，i が 0 でない場合，手順 3 へ進む．
   - FR ビットと W ビットがセットされておらず，i が 0 である場合，手順 4 へ進む．
2. スレッド i のキャッシュから変数 A の値を読み出し，スレッド k のキャッシュに保存されている変数 A の状態ビットを調べる．
   - FR ビットと W ビットがセットされていない場合，手順 5 へ進む．
   - FR ビットと W ビットのどちらか一方でもセットされている場合，終了する．
3. i の値を 1 だけ減らし，スレッド i のキャッシュに保存されている変数 A の状態ビットを調べる．
   - FR ビットと W ビットのどちらか一方でもセットされている場合，手順 2 へ進む．
   - FR ビットと W ビットがセットされておらず，i が 0 でない場合，手順 3 へ進む．
   - FR ビットと W ビットがセットされておらず，i が 0 である場合，手順 4 へ進む．
4. メインメモリから変数 A の値を読み出し，スレッド k のキャッシュに保存されている変数 A の状態ビットを調べる．
   - FR ビットと W ビットがセットされていない場合，手順 5 へ進む．
   - FR ビットと W ビットのどちらか一方でもセットされている場合，終了する．
5. スレッド k のキャッシュに保存されている変数 A に FR ビットをセットする．

2.2.2 変数に値を書き込むときの処理

あるスレッド k で変数 A の値を書き込むときの処理の流れを図 2 に示す．ただし，並列実行しているスレッドの数を n とし，それぞれのスレッドの番号を最も先行するスレッドから順に 0 番から $n-1$ 番とする．また，i の初期値を k とする．

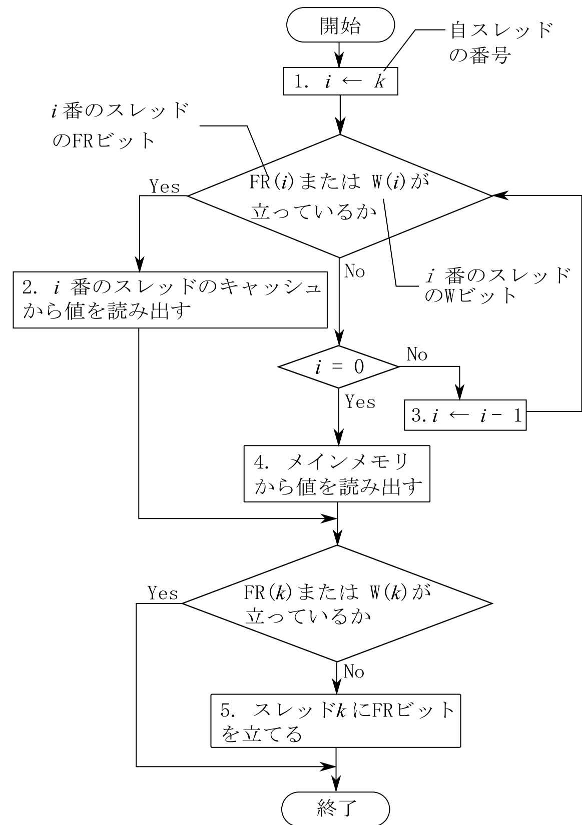

図1　変数を読み出すときの処理

1. 自スレッド i のキャッシュに変数 A の値を書き込み，変数 A の W ビットをセットする．
   - 自スレッド i が最後（$i=n-1$）のスレッドではない場合，手順 2 へ進む．
   - 自スレッド i が最後（$i=n-1$）のスレッドである場合，終了する．
2. i の値を 1 だけ増やし，スレッド i のキャッシュに保存されている変数 A の状態ビットを調べる．
   - FR ビットと W ビットがセットされておらず，スレッド i が最後（$i=n-1$）のスレッドではない場合，手順 2 へ進む．
   - FR ビットがセットされている場合，手順 3 へ進む．
3. 変数 A に関してスレッド k とスレッド i の間にフロー依存の関係が生じているため，スレッド i を破棄する．
   - スレッド i が最後（$i=n-1$）のスレッドではない場合，手順 2 へ進む．
   - スレッド i が最後（$i=n-1$）のスレッドである場合，終了する．

2.2.3 メモリリネーミングの実施例

例として，4 個のスレッドを並列実行している状況で，それぞれのスレッドが変数 A へアクセスした場合の処理の様子を図 3 を用いて説明する．図 3 において，時刻 1 以前では変数 A に対するアクセスが発生していないものとする．

Copyright © 2011 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

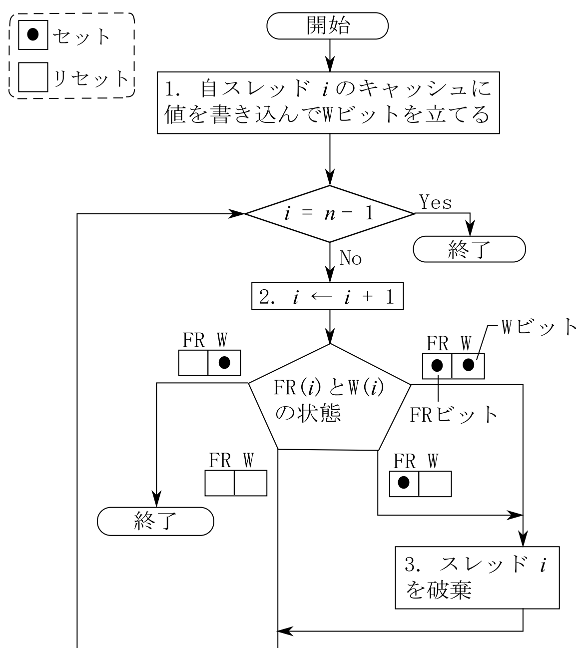

図2 各スレッドで変数を書き込むときの処理

時刻1) スレッド 3 が変数 A に対して読み出しを行うものとする．そのとき，まず，スレッド 3 のデータキャッシュ上の変数 A の状態ビットを調べる．スレッド 3 では初めての変数 A へのアクセスであり，FR ビットと W ビットがセットされていないため，メインメモリから変数 A の値を読み出すとともに，変数 A の FR ビットをセットする．

時刻2) スレッド 1 が変数 A に対して読み出しを行うものとする．時刻 1 におけるスレッド 3 での処理と同様に初めての読み出しアクセスであり，メインメモリから変数 A の値を読み出すとともに FR ビットをセットする．

時刻3) スレッド 2 が変数 A に対して書き込みを行うものとする．まず，スレッド 2 のキャッシュに値を書き込んで，W ビットをセットする．そして，スレッド 2 の後続スレッドであるスレッド 3 の状態ビットを調べる．スレッド 3 のキャッシュで変数 A の FR ビットがセットされているため，スレッド 3 を破棄する．

時刻4) スレッド 0 が変数 A に対して書き込みを行うものとする．まず，スレッド 0 のキャッシュに値を書き込んで，W ビットをセットする．そして，後続であるスレッド 1 のキャッシュにおいて変数 A の状態ビットを調べる．FR ビットがセットされているため，スレッド 1 を破棄する．また，スレッド 2 については W ビットがセットされているため，スレッド 2 は破棄しない．スレッド 2 の後続スレッド（スレッド 3）については状態ビットは調べない．

文献[3]等の従来の依存解析を行う場合，時刻 4 でスレッド 0 が変数 A へ書き込みを行うと，スレッド 2 との間

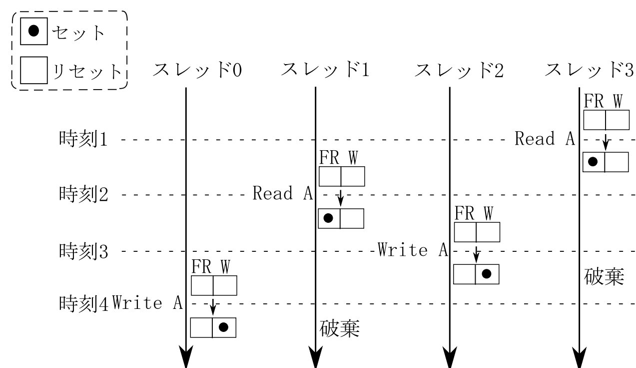

図３ ４スレッド実行時の変数アクセスの例

で出力依存によるハザードが生じるため，スレッド 2 も破棄する必要がある．しかし，メモリリネーミングを行うと，上記のように同時刻 4 でスレッド 2 を破棄する必要性を免れ，スレッド 2 の実行を続行することができる．

3. 性能評価

3.1 評価方法

PowerPC[4]の命令セットを対象とした命令レベルのシミュレーションを行い，メモリリネーミングの効果を調べる．評価用のテストプログラムには SPEC2006 ベンチマークプログラム[5]の中の 433.milc，444.namd，450.soplex，453.povray，482.sphinx3（入力データセットは test）を用いる．それぞれの評価用プログラムの主要部を構成する最外ループを並列化の対象とし，そのイタレーションをスレッドとして投機的に並列実行した場合に，アクセスするメモリ上のデータの依存関係に起因して生じるハザードの発生頻度を見積もる．

3.2 依存関係の除去効果

後続スレッドの投機実行を開始してから，依存関係によるハザードが生じるまでの実行命令数を初期連続実行命令数と呼ぶことにし，そのスレッドの全実行命令数に対する初期連続実行命令数の割合を初期連続実行率と定義する．

1 個のラインに 1 個の変数のみが含まれるものと仮定した場合の初期連続実行率を図 4 に示す．

メモリリネーミングを行わない場合は，初期連続実行率が高々0.3%程度であり，そのスレッドの実行開始直後に依存関係を破壊するアクセスが生じる．実際の並列実行方式によるが，この時点で先行スレッドとの同期を行うための待ちが生じるか，あるいはスレッドの実行を破棄して再実行を行うことになる．これに対してメモリリネーミングを行った場合は，最大 37.2%までスレッドの実行を進めることができ，依存関係による並列実行の阻害要因を大幅に削減できることが分かる．

図 4 の中には，433.milc と 453.povray のように，メモリリネーミングを行っても依存関係によるハザードが生じるまでの命令数が少数しか増加しないプログラムもある．これは，フロー依存によるハザードが発生する命令が，スレッドの実行開始直後に存在するためである．このよ

Copyright © 2011 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

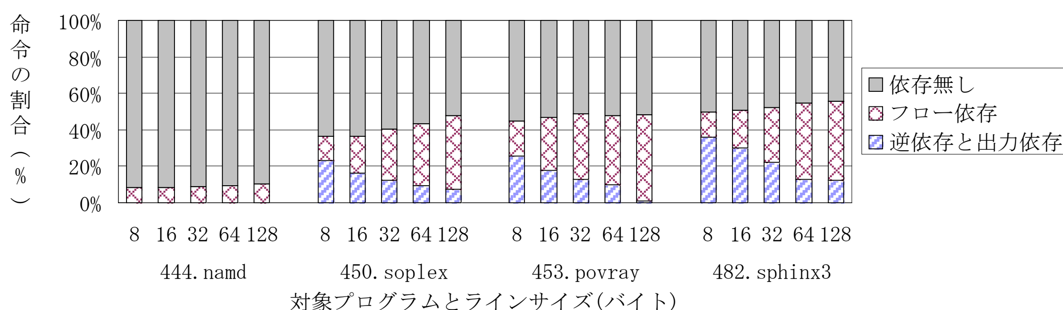

図 6　依存関係を生じる命令の割合のラインサイズによる違い

うに，初期連続実行命令数は，フロー依存によるハザードが発生する命令がスレッドのどの部分に存在するかによって異なる．

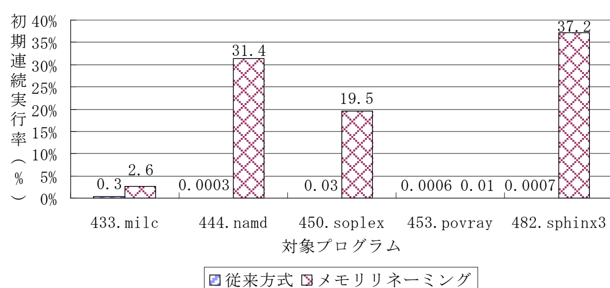

図 4　初期連続実行命令数（変数単位）

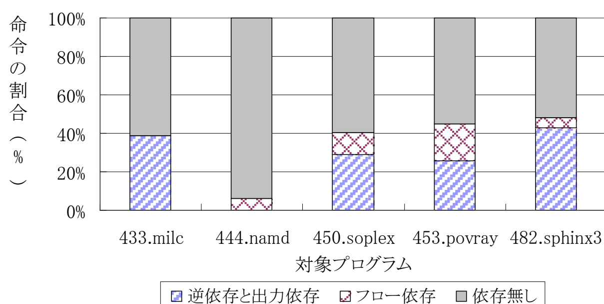

図 5　依存関係を生じる命令の割合（変数単位）

そこで，他のスレッドと依存関係を生じる命令がどれくらい存在するかを計測した結果を図 5 に示す．図 5 では，逆依存または出力依存によるハザードが発生する命令と，フロー依存によるハザードが発生する命令，依存関係によるハザードが発生しない命令，の 3 種に分類し，全実行命令数に占める割合で示した．

メモリリネーミングを行うことによって，逆依存と出力依存の依存関係を取り除くことができるので，444.namd を除いて，メモリリネーミングを行うことにより，依存関係によるハザードの発生回数のうち，その半数以上を除去できると考えられる．

3.3 ラインサイズによる影響

3.2 節では 1 個のラインに 1 個の変数のみが含まれるものと仮定したが，これは非現実的である．実際のキャッシュではライン単位で管理し，一般に，1 個のライン中に複数の変数を含む．このような現実的なキャッシュ構成においてメモリリネーミングの効果を調べる．ラインサイズを 8～128 バイトの間で変化させた場合に他のスレッドと依存関係を生じる命令がどれくらい存在するかを図 6 に示す．433.milc においては，ラインサイズによる変化が見られなかったため省略する．

1 ライン中にフロー依存を生じる変数が存在すると，同じライン中の他の変数が逆依存を生じるものであっても，そのラインはフロー依存を生じるものとして扱う．そのため，図 6 のようにラインサイズを大きくするとフロー依存によるハザードが発生する命令の割合が大きくなり，逆依存と出力依存によるハザードが発生する命令の割合が小さくなる．メモリリネーミングを行うことによって，並列性抽出の機会が増大するが，フロー依存をどのように扱うかがより重要な課題になる．

4. むすび

本稿では，プログラムのスレッドレベルの並列性抽出の可能性を向上させるためのメモリリネーミングを提案した．また，シミュレーションによる評価の結果，スレッド間の依存関係によるハザードの発生回数のうち，その半数以上をメモリリネーミングによって除去できることが分かった．

今後は，フロー依存への対応策とともに有限キャッシュでのメモリリネーミングの実現方式を検討してゆく．

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）21500053 および同 22500046）の補助による．

参考文献

[1]森田 清隆，布目 淳，平田 博章，柴山 潔，”スレッドレベル並列化のためのスレッド間依存関係の分類”，第 9 回情報科学技術フォーラム論文集, vol.1,pp.81-86 (2010).
[2]Manoj Franklin, Gurindar S. Sohi, "ARB: A Hardware Mechanism for Dynamic Reordering of Memory References," IEEE Transactions on Computers, vol.45, no.5, pp.552-571 (1996).
[3]Sridhar Gopal, T. N. Vijaykumar, James E. Smith, Gurindar S. Sohi, “Speculative Versioning Cache”, Proceedings of the International Symposium on High-Performance Computing, pp.195-205 (1998).
[4]Freescale Semiconductor, “Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture” (2001).
[5]Standard Performance Evaluation Corporation, “SPEC CPU2006”, http://www.spec.org/cpu2006/

Copyright © 2011 by Information Processing Society of Japan and
The Instiute of Electronics, Information and Communication Engineers
All rights reserved.