計算機アーキテクチャ 113−20
（1995. 8. 24）

Weak Consistencyを利用した遅延キャッシュ・コヒーレンシ・プロトコル

中村 秀一　下山 朋彦　福井 俊之
濱口 一正　長 健二朗　柴山 茂樹

キヤノン(株) 情報メディア研究所

〒211 川崎市幸区鹿島田890-12

E-mail:{shuzo, shimo, fukui, hamaguti, kjc, bashi}@cis.canon.co.jp

緩いメモリ・コンシステンシ・モデルを利用した新しい遅延キャッシュ・コヒーレンシ・プロトコルを提案する．本プロトコルを採用したキャッシュ・メモリはプロセッサから要求されたメモリ・アクセス群に対するコヒーレンシ・トランザクションをリオーダリング限界まで遅延させることが可能である．この遅延によりメモリ・アクセス群に対してのコヒーレンシ・トランザクションを統合して発行することができ，トランザクション総数を従来のプロトコルより削減することが可能となる．本報告では遅延キャッシュ・コヒーレンシ・プロトコルを用いたシステムのシミュレーションによる性能評価を行い，シミュレーション結果から遅延キャッシュ・コヒーレンシ・プロトコルの特性について考察する．

A Deferred Cache Coherency Protocol Exploiting Weak Consistency

Shuichi Nakamura　Tomohiko Shimoyama　Toshiyuki Fukui
Kazumasa Hamaguchi　Kenjiro Cho and Shigeki Shibayama

Media Technology Laboratory
CANON INC.
Kawasaki-shi, Kanagawa 211 Japan

E-mail:{shuzo, shimo, fukui, hamaguti, kjc, bashi}@cis.canon.co.jp

We propose a new deferred cache coherency protocol exploiting the Weak Consistency. In our deferred cache coherency protocol, several coherency transactions for requested memory accesses from the processor can be postponed until the coherency trigger point.

These postponed transactions can be combined into one coherency transaction, resulting in a considerable reduction of the amount of coherency transactions compared to traditional protocols. We discuss the features of our protocol by means of simulation.

1 はじめに

高性能なコンピュータ・システムにおいて，参照の局所性を利用し，メモリ・アクセス・レイテンシを削減するキャッシュ・メモリは不可欠の要素となっている．

分散共有メモリのようなメモリ・アクセス・レイテンシの大きな共有メモリ型マルチプロセッサ・システムにおいて，1回のトランザクションで転送されるデータ・ブロックを大きくすることは，適度な範囲ではメモリ・アクセスのオーバヘッドの低減，プリフェッチ効果によるキャッシュ・ヒット率の向上という点で性能向上に寄与すると考えられる．しかしながら，動的なメモリ割り当てや，静的なメモリ領域確保の大部分はキャッシュ・メモリに関するパラメータを考慮しておらず，転送されるデータ・ブロックを大きくすることで false sharing[1] を生じやすくなり，場合によってはコヒーレンシ・トランザクション[1] のピンポン現象 (スラッシング) といった問題を引き起こす．

本研究ではコヒーレンシ・トランザクションのスラッシング問題に着目して，キャッシュ・コヒーレンシ・プロトコルの改良によってスラッシングを抑制する遅延キャッシュ・コヒーレンシ・プロトコルを開発した．遅延キャッシュ・コヒーレンシ・プロトコルでは，緩いメモリ・コンシステンシ・モデルを利用したメモリ・アクセス・リオーダリング[2]によるコヒーレンシ・トランザクションの統合を行い，キャッシュ・メモリの発行するトランザクション数を削減することが可能となる．

本稿では，2章でメモリ・コンシステンシ・モデルについて簡単に説明し，3章で緩いメモリ・コンシステンシ・モデルを利用した新しいキャッシュ・コヒーレンシ・プロトコル (遅延キャッシュ・コヒーレンシ・プロトコル) を提案し，その実装例について説明する．4章で遅延キャッシュ・コヒーレンシ・プロトコルを適用したシステムのシミュレーションを行い，性能を評価した結果について報告する．5章で遅延キャッシュ・コヒーレンシ・プロトコルの特徴について議論する．

[1]複数のキャッシュ・メモリに存在するデータ・ブロックのコピーの一貫性を保持するためのトランザクション．

[2]本稿ではメモリ・アクセスの発行，完了順序を動的に入れ替えることを意味する．

2 メモリ・コンシステンシ・モデル

一般的なマルチプロセッサ・システムにおいて，プロセッサ・ストール時間を短縮するため，依存関係と緊急度に応じてメモリ・アクセス・リオーダリングが行われる．メモリ・アクセス・リオーダリングにおいて Read アクセスを優先して Write アクセスを延期するといった単純な手法を行った場合，共有変数の排他制御に関するロック獲得の Write アクセスが延期され，クリティカル・セクション内の Read アクセスが先に複数のプロセッサで完了してしまうといった問題が生じる．

メモリ・コンシステンシ・モデル [2][3][4][5] は，マルチプロセッサ・システムにおける各メモリ・アクセス間の発行・完了[4] の順序関係を定義する．定義された各メモリ・アクセス間の発行・完了の順序関係を遵守することにより，前述した共有変数に対する排他制御問題の発生を抑制する．性能向上のためにメモリ・アクセス・リオーダリングを許したメモリ・コンシステンシ・モデルは緩いメモリ・コンシステンシ・モデル [3][4][5] と呼ばれる．

緩いメモリ・コンシステンシ・モデルの一種である Weak Consistency[3] は Dubois らによって提案されている．Weak Consistency では，メモリ・アクセスは「通常のメモリ・アクセス」と「同期アクセス」の2種類に分類される．通常のメモリ・アクセス間では発行順に完了する必要はなく，リオーダリング可能であるが，通常のメモリ・アクセスと同期アクセス間では発行順に完了しなければならない (図1)．すなわち，先行して発行された複数の通常のメモリ・アクセスは，メモリ・アクセス群として扱われ，メモリ・アクセス群の完了は同期アクセスによって保証される．

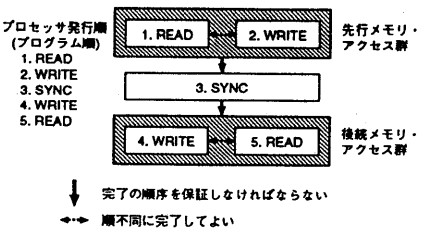

図1: Weak Consistency における順序保証

[4]本稿では「完了」は「システム (プロセッサやメモリ) 中に反映される」ことを意味するものとする．

3 遅延キャッシュ・コヒーレンシ・プロトコル

本節ではWeak Consistencyを利用したキャッシュ・コヒーレンシ・プロトコル(遅延キャッシュ・コヒーレンシ・プロトコル)を提案し，実装上の選択肢について説明する．

3.1 遅延キャッシュ・コヒーレンシ・プロトコル

従来のキャッシュ・コヒーレンシ・プロトコルでは，更新されたデータ・ブロックがシステム中で一意であることを保証するために，コヒーレンシ・トランザクションが必要なメモリ・アクセスが発行された場合，コヒーレンシ・トランザクションが他のメモリ・トランザクションに優先してその都度発行される．

本稿で提案する遅延キャッシュ・コヒーレンシ・プロトコルでは，メモリ・アクセス・リオーダリングを応用して，コヒーレンシ・トランザクションの発行を遅延させることを特徴とする．通常のメモリ・アクセスが

1. メモリ・アクセス発行
2. [必要ならば]
   コヒーレンシ・トランザクション発行

2.′ [2. が実行された場合]
   コヒーレンシ・トランザクション完了

3. メモリ・アクセス完了

の順で行われるものとすると，Weak Consistencyで許されるメモリ・アクセス・リオーダリングの定義によると，図2において(1)と(2)のメモリ・アクセスの発行順及び完了順は順不同である．したがって，Weak Consistencyの保証下において，コヒーレンシ・トランザクションの発行は，最大，同期アクセスの完了時点まで延期させることが可能である．

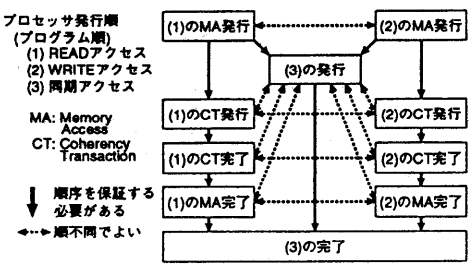

図2: コヒーレンシ・トランザクションと同期アクセスの完了順序

遅延キャッシュ・コヒーレンシ・プロトコルでは，コヒーレンシ・トランザクションの発行を遅延することで，遅延された複数のコヒーレンシ・トランザクションのうち，統合可能なものを1つのコヒーレンシ・トランザクションとして統合して発行することができ，コヒーレンシ・トランザクションの総数を削減することが可能となる．コヒーレンシ・トランザクションの総数を削減することは，トランザクション・レイテンシの大きなシステムにおいて，性能向上をもたらすものと予想される．

ここで，遅延キャッシュ・コヒーレンシ・プロトコルを実装する上で考慮しなければならない選択肢は，

(a) コヒーレンシ・トランザクション発行契機
(b) コヒーレンシ・トランザクション遅延方法

である．

コヒーレンシ・トランザクション発行契機に関しては図3の(1)〜(3)の場合が考えられ，それぞれバスの使用率やハードウェア実装の容易さといったことが設計上考慮されなければならない．コヒーレンシ・トランザクション遅延方法に関しては，コヒーレンシ・トランザクションが遅延されたセクタに関する情報を保持する手段を用意し，コヒーレンシ・トランザクション発行契機においてこの保持された情報を用いてコヒーレンシ・トランザクションの遅延を検出しコヒーレンシ・トランザクションを発行する手法が考えられる．コヒーレンシ・トランザクション遅延情報保持手段としては，

(A) キャッシュ・メモリ外部の記憶手段(バッファ等)に保持する
(B) キャッシュ・メモリ内部の記憶手段(状態フラグ等)に保持する

等が考えられ，記憶保持手段のハードウェア・コストや，遅延情報検出機構の実装の容易さといった設計上のトレードオフを考慮して実装する必要がある．

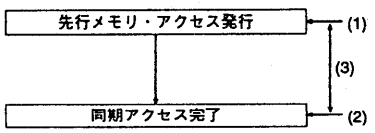

図3: コヒーレンシ・トランザクション発行契機

3.2 遅延キャッシュ・コヒーレンシ・プロトコルの実装例

3.1 で述べた設計上の選択肢をふまえ，遅延キャッシュ・コヒーレンシ・プロトコルの一例を実装した．本稿で実装した遅延キャッシュ・コヒーレンシ・プロトコル (以下遅延プロトコル) は，同期アクセス発行に主記憶を更新し[5]，コヒーレンシ・トランザクション発行契機は，図3の (3) に属するプロセッサから同期アクセスが発行される時点とした．なお，遅延プロトコルはスヌープをベースとした無効化型のプロトコルを採用した．コヒーレンシ・トランザクション遅延方法は，(B) のキャッシュ・メモリ内のセクタの状態を管理する状態フラグを利用して同期アクセス発行時点で状態フラグを検索し，Modified(図3.2参照) と検出されたセクタに対してコヒーレンシ・トランザクションを発行するという手法をとる．これらの手法は，追加するハードウェアが少ないことを主な理由に選択した[3]．

図 3.2 に遅延プロトコルにおけるセクタの状態遷移図を示す．

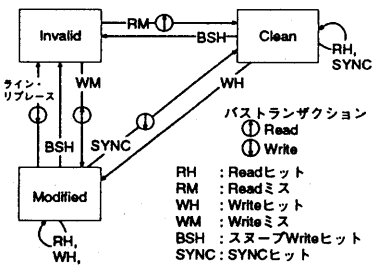

図 4: 遅延プロトコルの状態遷移図

次に，本稿で実装した遅延プロトコルを採用したキャッシュ・メモリの簡単な構成例を図5に示す．従来の一般的なキャッシュ・メモリと比較して，キャッシュ・ライン (以下ライン) 中のセクタ数が多い．(図5では1セクタ=2バイト，1ライン=32 セクタ)

[3]追加が必要なハードウェアは状態フラグを検索するための機構 (状態フラグの Modified の論理 OR)，及び制御論理のみ．

[5]同期アクセス時に自キャッシュのみにデータ・ブロックを保持している (オーナーシップを持つ) 場合に主記憶を更新しない遅延キャッシュ・コヒーレンシ・プロトコルも考えられるが，本稿で詳細については述べない．

[†]ロック獲得には特別な命令 (メモリ・アクセス完了時に成功する Write アクセス) を使用するものとするが，本稿で詳細については述べない．

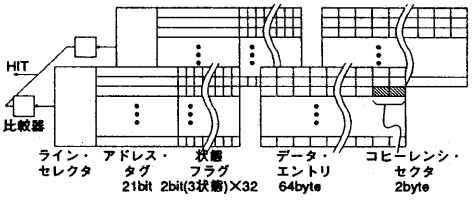

図 5: キャッシュ・メモリのハードウェア構成

図6にラインの制御例を示す．

同期アクセス発行時，状態フラグがModifiedであるラインのアドレス・タグ，状態フラグ，及び該当するセクタの内容を外部バスに転送する．主記憶メモリはこれらの内容により，主記憶のデータ・エントリを更新する．他のキャッシュ・メモリは外部バスに転送されたアドレス・タグ，及び状態フラグの内容をスヌープして該当するセクタを無効化する．複数のライン中に状態フラグがModifiedで存在する場合，どのラインからライト・バックしてもよいが，同期アクセスは全Modifiedラインのライト・バックの完了を待たなければならない．

また，1ライン中に無効化されたセクタと更新されたセクタが共存するラインが存在し得るため，キャッシュ・ミス時にアドレス・タグが一致し，かつ，状態フラグがModifiedでセットされているラインがある場合，リプレース時にはこのセットを選択してリプレースしなければならない[6]．ライン・リプレース時に状態フラグがModifiedでセットされているセクタは上書きしない．

以上，実装した遅延キャッシュ・コヒーレンシ・プロトコルの一例について説明した．

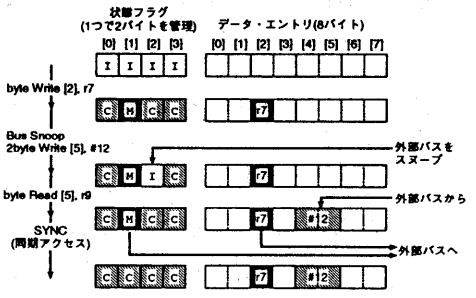

図 6: キャッシュ・ラインの制御例

[6]このセットを選択しない場合，2つのセットが同時に1つのアドレスに対応するといった問題が生じる．

4 性能評価

遅延キャッシュ・コヒーレンシ・プロトコルを採用したシステムに対して，シミュレーションを行い，性能を評価した．シミュレーションの目的は，以下の点を確認することである．

1. スラッシング発生の原因を明らかにすること，及び遅延キャッシュ・コヒーレンシ・プロトコルによってコヒーレンシ・トランザクションの総数が削減されること，
2. キャッシュ・コヒーレンシ・プロトコルとメモリ・アクセス・レイテンシ，キャッシュ・ライン・サイズとの相関を明らかにすること．

シミュレートしたシステム・モデルは，UMA方式のシステムとした．また，キャッシュ・コヒーレンシ・プロトコルには3.2節で説明した遅延プロトコルを採用した．また，市販のプロセッサで一般に使用されているMESIキャッシュ・コヒーレンシ・プロトコル(以下MESIプロトコル)を選んで性能を比較した．評価に用いたシステム・モデルを図4に，システム・パラメータを表4に示す．

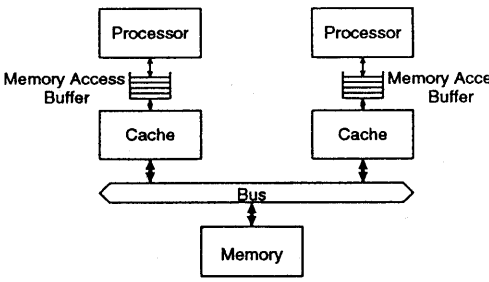

図7: シミュレートしたシステム・モデル

表1: システム・パラメータ

メモリ・アクセス・バッファ		
エントリ数	16‡	
キャッシュ・メモリ		
セット数	8‡	
ライン数	64‡	
ライン・サイズ(=L)	8〜128	byte
セクタ・サイズ(MESI)	L	byte
セクタ・サイズ(遅延)	1	byte
アクセス・レイテンシ		
メモリ・アクセス・バッファ	1	cycle
キャッシュ・メモリ	1	cycle
主記憶(=D)	4〜200	cycle

‡モデル・プログラムの性質上，これらの容量が原因で性能が低下することはない．

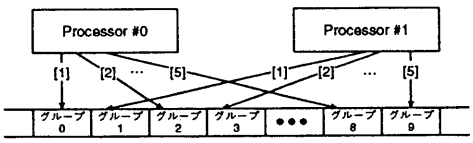

図8: 評価に用いたモデル・プログラム

評価に用いたモデル・プログラムを図8に示す．

(1) プロセッサ# 0は主記憶メモリ中の偶数グループ内のメモリの値をReadし，その値をインクリメントしてWriteすることを順次行う．プロセッサ# 1は奇数グループに対して同様のことを行う．

(2) 同一グループ内に対して(1)を8回行う．

(3) 同期アクセスにより，(1)，(2)で発行されたメモリ・アクセスを完了する．

(4) メモリ・アクセス以外の命令をm(=8, 200, 400)回実行し，次グループに対して同様に(1)から繰り返す．

以上，(1)〜(4)を5回行う．なお，各グループはそれぞれ主記憶メモリ内で連続したアドレスとした．また，モデル・プログラム自身は命令キャッシュ中にキャッシングされているものとした．

通常，データは構造体等を使用して，主記憶内のあるメモリ領域にまとめて確保される．メモリ領域の割り当てはライブラリやOSによって動的に行われる場合(malloc等)と，プログラマによって静的に指定される場合(構造体宣言等)があるが，複数のプロセッサで各々確保したメモリ領域が同一キャッシュ・ライン中に割り当てられた場合，false sharingが生じる(図9)．本モデル・プログラムはfalse sharingを生じるようなメモリ割り当てを行う例を想定し，キャッシュ・ライン・サイズ，及びその他のパラメータの影響を調べ，スラッシングの原因を探ることを目的としたものである．

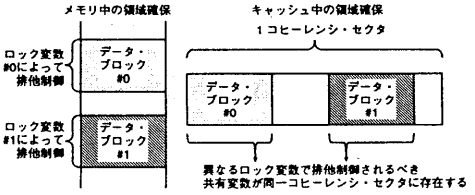

図9: false sharing

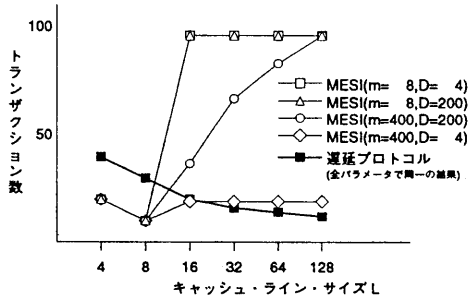

図 10: トランザクション数

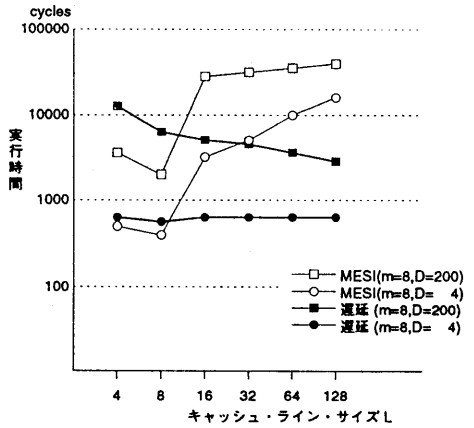

図 11: キャッシュ・ライン・サイズとの相関

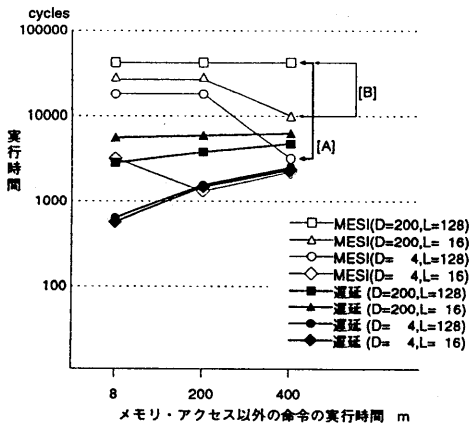

図 12: メモリ・アクセス以外の命令の実行時間との相関

5 考察

5.1 キャッシュ・ライン・サイズ, メモリ・アクセス・レイテンシとの相関

図 10はライン・サイズを変化させた場合のキャッシュ・メモリの発行するトランザクション数を比較した結果である.

ライン・サイズ 4,8 では false sharing が発生しない. 双方のプロトコルでコールド・ミスに伴うトランザクションが発生し, false sharing によるトランザクションは発生しない. プロトコルによる差は, 同期アクセス時に主記憶に対して行うライト・バックの是非のみである[4]. ライン・サイズ 16 以上では false sharing が発生する. MESI プロトコルでは, false sharing の発生によってトランザクション数が増加する. メモリ・アクセス間隔によって増加の割合は異なるが, トランザクション数は MESI プロトコルの方が多くなる. 遅延プロトコルでは, false sharing の発生によってもトランザクション数は増えず, ライン・サイズの拡大に伴うコールド・ミスの減少により, トランザクション数は減少する結果となっている. したがって, 遅延プロトコルを採用することにより, MESI プロトコルと比較して, トランザクション数を少なくすることができ, かつ, ライン・サイズの拡大によりさらにトランザクション数を削減することが可能である.

図 11はライン・サイズを変化させた場合の実行時間を比較した結果である. MESI プロトコルで, ライン全体をライト・バックしているため, ライン・サイズの拡大に伴い実行時間はさらに長くなる. 遅延プロトコルではライン・サイズの拡大に伴うトランザクション数の減少により, 実行時間が短縮される. 遅延プロトコルではメモリ・アクセス・レイテンシが大きくなるほど, ライン・サイズ拡大の効果が顕著になっており, 今後プロセッサと主記憶の速度差がさらに拡大すると予想され, 遅延キャッシュ・コヒーレンシ・プロトコルを使用することは性能向上の有効な手段になると思われる.

[4] 今回実装した遅延プロトコルにデータ・ブロックのオーナシップがないため, 同期アクセス時にライト・バックする必要がある. データ・ブロックのオーナシップを導入したプロトコルを実装し, 同期アクセス時, 及びその後他のプロセッサからのアクセス要求が発行されるまで主記憶にライト・バックしないようにするとこの差はなくなる.

5.2 スラッシング発生原因

図12は，false sharingが発生する(ライン・サイズ16以上)場合でMESIプロトコルにおいてメモリ・アクセス以外の命令の実行時間を変化させたときの実行時間を比較した結果である．メモリ・アクセス以外の命令の実行時間が短い場合，図10からもわかるように，トランザクション数が激増し，遅延プロトコルと比較して著しい性能低下を示す．メモリ・アクセス以外の命令の実行時間が長い場合，MESIプロトコルにおいてもスラッシングを生じていない場合がある．これは，図13に示されるようにfalse sharingが発生するキャッシュ・ラインに対するメモリ・アクセス期間が時間的に共有されていないためである．このような場合，遅延プロトコルではもちろん，MESIプロトコルにおいてもスラッシングが発生しない．

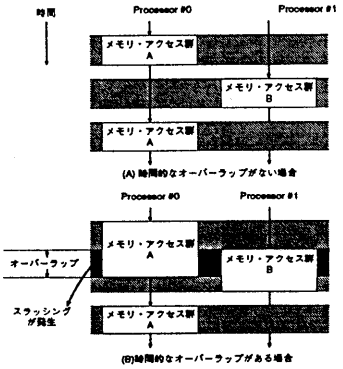

図13: false sharingが発生するキャッシュ・ラインの参照間隔

また，メモリ・アクセス・レイテンシ，及びライン・サイズが大きくなると，1回のメモリ・アクセス実行時間が長くなってメモリ・アクセス以外の命令の実行時間が相対的に小さくなり，false sharingが発生するキャッシュ・ラインに対するメモリ・アクセス期間の共有が起こる．図12の[A]でMESIプロトコルを同一ライン・サイズ(128)や図12の[B]でMESIプロトコルの同一メモリ・アクセス・レイテンシ(200)で比較するとメモリ・アクセス・レイテンシが大きくなったり，ライン・サイズが大きくなるとこのメモリ・アクセス期間の共有が起こり，スラッシングが生じて性能が低下していることがわかる．

5.3 スラッシング抑制対策

MESIプロトコルを使用する場合，複数のプロセッサからのメモリ・アクセスが時間的にオーバーラップすることが予想されるメモリ領域に対して，メモリ領域割り当てに細心の注意を払う必要がある．例えば，スピンロックで参照されるロック変数は，集中した時間帯で頻繁にメモリ・アクセスがなされるため，メモリ領域を動的にmalloc等で割り当てる場合，スラッシングを生じないよう，false sharingしないメモリ領域に割り当てるべきである．ただし，このようなメモリ割り当ては，キャッシュ・ラインにロック変数のみを割り当てることとなり，メモリのフラグメンテーションが生じる．このようなメモリ領域割り当てのフラグメンテーションにより，キャッシュの容量性ミスが誘発されることはWooらによって報告されている[9]．

また，MESIプロトコルにおいても，コヒーレンシ・セクタを多数持つことにより，スラッシングを抑制することは可能である．しかしながら，MESIプロトコルでは，このような多数のコヒーレンシ・セクタの導入によっても，コヒーレンシ・トランザクションを統合して発行することができないため，バス・トラフィックの不用意な増加を招く危険性がある．

遅延キャッシュ・コヒーレンシ・プロトコルでは，このようなメモリ割り当てをしなくともスラッシングは発生しない．ただし，遅延キャッシュ・コヒーレンシ・プロトコルを使用する場合において，スラッシングによる著しい性能低下はないが，コヒーレンシ・セクタは多数必要である．コヒーレンシ・セクタを少数しか用意しない遅延キャッシュ・コヒーレンシ・プロトコルも考えられるが，この場合，false sharing発生による性能低下の影響は小さいながらも依然として問題となるため，false sharing発生によるキャッシュ・ミスとハードウェア量のトレードオフをシステム設計者は考慮しなければならない．

以上，false sharingによるスラッシング抑制対策について述べた．メモリ・アクセス・レイテンシが大きくなることや，ライン・サイズが大きくなること等，現在基準としているトランザクション・コストよりもトランザクション・コストが大きくなった場合には，MESIプロト

コルではスラッシングによる著しい性能低下が生じる可能性が高く，スラッシングを抑制するために払うコストが高くなる．これに対して，遅延キャッシュ・コヒーレンシ・プロトコルを採用した場合，スラッシングによる性能低下を生じる可能性は低いため，スラッシングを抑制するために払うコストはMESIプロトコルと比較して小さくて済むもの思われる．

6 おわりに

緩いメモリ・コンシステンシ・モデルに対応した新しいキャッシュ・コヒーレンシ・プロトコルについて提案し，シミュレーション結果から得たデータについて考察した．

遅延キャッシュ・コヒーレンシ・プロトコルでは，Weak Consistencyによる一貫性保証の下で同期アクセスによってグループ化されるメモリ・アクセス群のコヒーレンシ・トランザクションが統合されて発行される．このことにより，コヒーレンシ・トランザクションの総数を削減することが可能となり，コヒーレンシ・トランザクションのコストが大きい場合においても，従来のキャッシュ・コヒーレンシ・プロトコルと比較して著しい性能向上が期待できる．

今後，オーナシップや他のより緩いメモリ・コンシステンシ・モデルへの対応等を考慮した，より洗練された遅延キャッシュ・コヒーレンシ・プロトコルについて検討し，実際の並列アプリケーション・プログラムによる性能測定等を行って，定量的な性能評価を行う予定である．

謝辞

日頃より有益な御意見を頂くキヤノン(株)情報メディア研究所の鈴木 茂夫氏，数藤 義明氏，並びに情報メディア研究所の諸氏に深く感謝致します．

参考文献

[1] Michel Dubois, "The Detection and Elimination of Useless Misses in Multiprocessors," In Proceedings of the 20th Annual International Symposium on Computer Architecture, pp. 88-97, May 1993.

[2] Leslie Lamport, "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs," IEEE Transactions on Computers, vol. C-28,No.9, pp. 690-691, September 1979.

[3] Michel Dubois, et al., "Memory Access Buffering In Multiprocessors," In Proceedings of the 13th Annual International Symposium on Computer Architecture, pp. 434-442, June 1986.

[4] Kourosh Gharachhorloo, et al., "Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors," In Proceedings of the 17th Annual International Symposium on Computer Architecture, pp. 15-26, May 1990.

[5] Pete Keleher, et al., "Lazy Release Consistency for Software Distributed Shared Memory," In Proceedings of the 19th Annual International Symposium on Computer Architecture, pp. 13-21, May 1992.

[6] 高橋雅史ほか, "複数のプロセッサによる共有を考慮したメモリアクセスバッファの構成," 情処研報, ARC107-29, pp. 225-232, 1994年7月

[7] Leonidsa I. Kontothanassis, et al., "Lazy Release Consistency for Hardware-Coherent Multiprocessors," TR547, Competer Science Department, University of Rochester, December 1994.

[8] Michel Dubois, et al., "Delayed Consistency and its Effect on the Miss Rate of Parallel Programs," In Proceedings of Supercomputing'91, pp. 197-206, Albuquerque, NM, November 1991.

[9] Steven Cameron Woo, et al., "The SPLASH-2 Programs: Characterization and Methodological Considerations," In Proceedings of the 22nd Annual International Symposium on Computer Architecture, pp. 24-36, June 1995.