計算機アーキテクチャ　99－16
(1993. 3. 12)

ソフトウェアとハードウェアのコヒーレンス制御を融合したキャッシュ・コヒーレンス制御

細見岳生†，　森 眞一郎†，　中島浩†，　富田眞治†
†：京都大学 工学部

あらまし　共有メモリ方式の並列計算機において，キャッシュ・コヒーレンス制御のオーバーヘッドを軽減することは必要不可欠となっている．本稿では，false sharing時のPing Pong現象に着目し，この問題を解決したSelf-Invalidate型のソフトウェア・コヒーレンス制御とディレクトリ型のハードウェア・コヒーレンス制御を融合したコヒーレンス制御方式を提案する．複数のプロセッサが同一ラインに同時に書き込みを行うことを可能とし，false sharingが発生したラインに関するコヒーレンス制御を行うタイミングをプログラムで明示的に指定する．これにより，false sharingが発生してもPing Pong現象を引き起こさないコヒーレンス制御方式を実現した．また本稿ではシミュレータを用いた性能評価を行い，従来のコヒーレンス制御方式と比較して良好な結果が得られたことも示す．

和文キーワード　共有メモリ型並列計算機，キャッシュ，ハードウェア・コヒーレンス制御，ソフトウェア・コヒーレンス制御

A Hardware Cache Coherence Scheme with the Assistance of Software

Takeo HOSOMI†, Shin-ichiro MORI†, Hiroshi NAKASIMA†,　Shinji TOMITA†

†: Department of Information Science
Faculty of Engineering, Kyoto University
Yoshida-hon-machi, Sakyo-ku, Kyoto 606-01 Japan

E-mail: {hosomi,moris,nakasima,tomita}@kuis.kyuto-u.ac.jp

Abstract　It is essential to reduce the overhead of cache coherence in shared memory multiprocessor systems. For this purpose, we propose a new cache coherence scheme which cooperates the software based self-invalidation scheme with the directory scheme. In this paper, we focus to solve the cache ping-pong problem in practice. This sheme allows more than one processors to update same line simultaneously enusuring by software that no data conflict occures inside the line. And this scheme relys on the software to inform the hardware when to invalidate the falsely shared lines. We show some simulation results that show the performance of the new scheme compared with the ordinary directory scheme.

英文 key words　shared memory multiprosessor system, cache, software coherence scheme, hardware coherence scheme

1 はじめに

共有メモリ方式の並列計算機において高いパフォーマンスを得るためには，キャッシュを用いてメモリ・アクセスやネットワークのレイテンシを隠蔽することが必須となっている．しかし，キャッシュが分散して複数存在することにより，メモリ，複数のキャッシュ間でデータのコンシステンシを保つための制御，つまりキャッシュ・コヒーレンス制御が必要となっている．

従来のコヒーレンス制御方式において共有されている同一ライン内の異なるデータに対して複数のプロセッサが書き込みを行った場合（false sharing が生じた場合），Ping Pong 現象が発生しキャッシュの目的であるメモリ・アクセスやネットワークのレイテンシを隠蔽できなくなっていた．

本稿では，ライン単位のコヒーレンス制御を行うディレクトリとデータ単位のコヒーレンス制御を行うソフトウェアを組み合わせることにより，false sharing が発生しても Ping Pong 現象を引き起こさないコヒーレンス制御方式を提案する．

まず 2 章で前提および背景について述べた後，3 章で新しいコヒーレンス制御方式の提案を行い，4 章でその性能評価を行う．

2 前提および背景

2.1 前提

1. プログラムにおいて，プロセッサを跨る依存関係が存在する場合，排他制御や順序制御を行うための同期操作が行わる．また，この同期操作と通常のメモリ・アクセスとはハードウェアで識別可能であるとする．そのような同期操作として図 2 に示すように依存関係のソース側の命令の後に End-of-Section，デスティネーション側の命令の前に Start-of-Section が挿入されているものとする．

2. キャッシュのコヒーレンス制御を行う単位であるラインは複数のワードよりなるものとする．

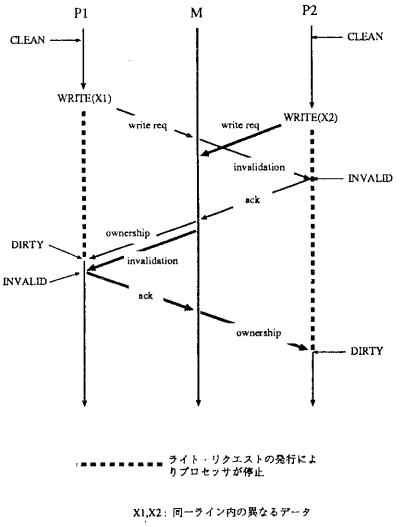


図 1: false sharing による Ping Pong 現象

3. メモリにはライン単位にフルマップのディレクトリが存在する．

2.2 フォールス・シェアリング

ソフトウェアはある意味のあるデータ集合に対して排他制御や順序制御を行う．しかしながら，このデータ・サイズがキャッシュのライン・サイズとは一致しているとは限らない．そのため，共有されているライン内の異なるデータに対して複数のプロセッサが同時に書き込みを行うという状況（false sharing（以降 FS））が発生し得る．従来のコヒーレンス制御方式においては，キャッシュのあるラインに対して書き込みを行う場合は，当該ラインを保持する他のキャッシュ・ラインを無効化してから書き込みを行っていた．そのため，共有されているラインの異なるデータに対して複数のプロセッサが書き込みを行った場合，図 1に示すようにネットワークにメッセージが飛び交い（Ping Pong 現象），書き込みが終了するまでプロセッサが停止するためキャッシュの目的であるメモリ・アクセスやネットワークのレイテンシを隠蔽できなくなっていた．

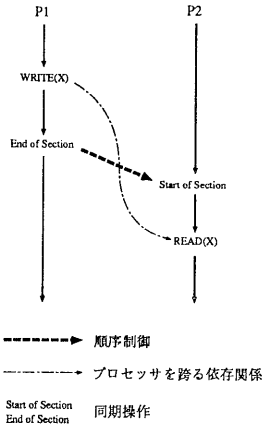


図 2: プログラムに対する条件

この問題を解決する方法として，従来はコンパイラでFSを減少させる方法[2][3][4]や，ソフトウェア・コヒーレンス制御とハードウェア・コヒーレンス制御との協調により，FSが発生してもPing Pong現象を引き起こさない方式[1]が提案されていた．前者はコンパイラで静的に解析を行い，FSが発生しないようなデータや，プロセッサへの処理の割り当てを行うものである．しかし，コンパイラにより十分な解析ができない場合や動的な要因が介在する場合，FSを解消することができずPing Pong現象を引き起こしていた．後者の方式はディレクトリを実行時情報収集のための手段としてのみ利用し，コヒーレンス制御は同期操作時にソフトウェアが明示的に行っていた．しかし，同期操作時にキャッシュのラインをソフトウェアですべて検索し，書き込みが発生したラインを選択的にメモリに書き戻さなければならないため，この処理が非常に重たいという問題点が存在した．

本稿では，ライン単位のコヒーレンス制御を行うディレクトリとデータ単位のコヒーレンス制御を行うソフトウェアを組み合わせPing Pong現象を解決し，後者で問題であった同期操作時の処理をメモリ更新アルゴリズムを動的に切替えることで解決したコヒーレンス制御方式を提案する．

3 Self-Invalidate型ハードウェア・キャッシュ・コヒーレンス制御方式

3.1 提案するコヒーレンス制御方式の特徴

本稿で提案するSelf-Invalidate型キャッシュ・コヒーレンス制御方式は以下の特徴を持っている．

1. 書き込みが発生したことを通知するが，その時点でインバリデーションを行わない

   あるキャッシュで書き込みが発生した場合，ディレクトリを用いて他のキャッシュ・コピーに対して書き込みが発生したことは通知するが（以降**staling**を行うと呼ぶ）その時点で無効化は行わない．そのため，他のプロセッサで書き込みが発生したラインであってもキャッシュに対するアクセスはヒットしFSによるPing Pong現象を解決できる．

2. ソフトウェアによりインバリデーションのタイミングを指定する．

   stalingされたキャッシュ・ラインには当該ラインに対する更新が全て反映されている訳ではないので，コヒーレンスを維持するためにある時点で無効化しなければならない．本方式では，無効化するタイミングをソフトウェアによって明示的に指定することでコヒーレンスを維持する．

3. メモリ更新方式を動的に切替える．

   FSが発生しているラインについてはwrite through方式のメモリ更新アルゴリズムを採用し，FSが発生していない場合はwrite back方式のメモリ更新アルゴリズムを採用する．これにより，同期操作時にキャッシュを検索してメモリに書き戻しを行わなくても済むようにし，かつネットワーク・トラフィックの増加を防ぐ．

4. ライトでオーナーシップを要求せず，プロセッサを停止させない．

   ソフトウェアによりデータ単位で排他制御がなされている．そのため，実行時にオーナーシップを獲得する必要はなく，プロセッサはブロッキン

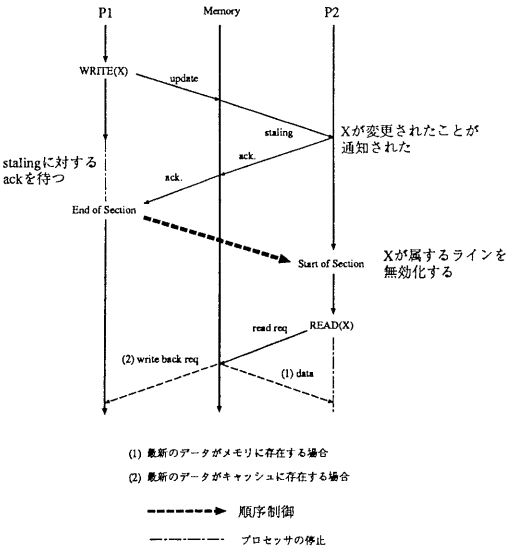


図 3: コヒーレンスの保証

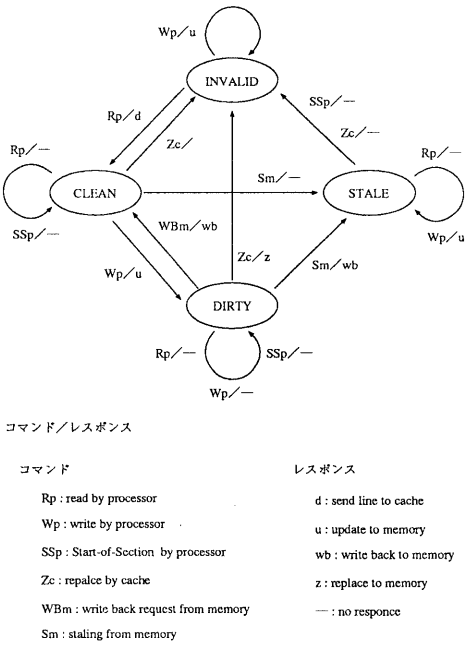


図 4: キャッシュの状態遷移図

グされない．また，このためキャッシュ・ラインが無効な状態での書き込みに対してライン・アロケーションを行わない方式を採用する（no write allocate）．

3.2 提案するコヒーレンス制御方式の実現

3.2.1 同期操作時の処理の実現

ソフトウェアの同期操作をハードウェアがそれを認識し以下の処理を行うことで，システムとしてコヒーレンスが維持されることを保証している（図3参照）．

Start-of-Section

staling されたラインを選択的に無効化する．

End-of-Section

当該同期操作が行われる前に発行された書き込みに対して，staling が完了していることを保証する．

また，これらの処理は以下のようにしてハードウェアとして実現する．

Start-of-Section

Start-of-Section 時に staling されたラインを選択的に無効化するために，キャッシュ上にライン単位に sync-bit を設け [5]，以下のように使用する．

- 状態遷移時に sync-bit をリセットする．
- Start-of-Section 時に全ラインの sync-bit を一斉にセットする[1]．

このようにすることで，staling されたラインでかつ sync-bit がセットされているラインは無効なラインであるとハードウェアで認識可能であり，選択的な無効化を高速に行うことができる．

[1] Clear Memory モジュールを用いる

表 1: メモリおよびキャッシュの状態の対応

メモリの状態	キャッシュの状態	意味	ディレクトリ	最新のデータ	書き込みを行ったキャッシュ（数）	メモリへの更新方式
CLEAN	INVALID	キャッシュ・ラインは無効である	0	メモリ	0 or many	write through
	CLEAN	キャッシュ・ラインには最新のデータが存在する	1		0	write back
	STALE	他のキャッシュが当該ラインを更新しており，キャッシュ・ラインは最新のデータではない	0		many	write through
MODIFIED	INVALID	キャッシュ・ラインは無効である	0	キャッシュ	1	write through
	DIRTY	当該キャッシュ・ラインにのみ最新のデータが存在する	1			write back
	STALE	他のキャッシュが当該ラインを更新しており，キャッシュ・ラインは最新のデータではない	0			write through

End-of-Section

End-of-Section 時は，staling の処理が完了していることを保証しなければならないので ack 管理の機構が必要となる[2].

3.2.2 キャッシュおよびメモリの動作

キャッシュには図4に示すように INVALID,CLEAN, STALE, DIRTY の 4 つの状態が存在し，メモリには図 5に示すように CLEAN, MODIFIED の 2 つの状態が存在する．メモリおよびキャッシュの状態の存在する組合せを表 1に示す．

この方式でのキャッシュおよびメモリの動作で，従来のライト・バック-インバリデート型と[3]異なる点は，STALE 状態でのキャッシュの動作，および書き込み発生時のキャッシュとメモリの動作である．ここでは STALE 状態でのキャッシュの動作について触れた後，書き込み発生時のキャッシュ，メモリの動作について説明する（図 4，図 5，および表 1，表 2参照）．

STALE 状態のキャッシュは以下のように動作する．

- ラインに対するアクセスはヒットとなる．
- リプレースの対象となった場合，そのラインは単に INVALID 状態に遷移する．
- Start-of-Section が発行されると，INVALID 状態に遷移する．

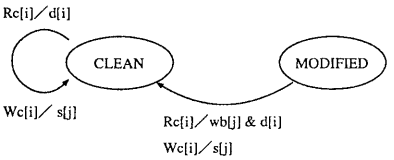
また，書き込みが発生した場合キャッシュ，メモリは以下のように動作する．

1. キャッシュが DIRTY であれば，キャッシュ・ラインに対して更新を行い完了する．キャッシュが INVALID であれば，ライン・アロケーションを行わず（no write allocation），メモリに対して update を発行する。キャッシュが CLEAN または STALE の場合，キャッシュ・ラインに対して更新を行うとともにメモリに対して update を発行する．

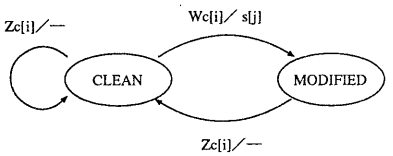
2. メモリは update を受けて，ディレクトリに登録されている他のキャッシュ・コピーに対して staling を行うとともに，staling 発行先のキャッシュに対応するディレクトリのビットを unset する．この時，update を発行したキャッシュに対応するディ

[2]従来の方式と同様であり本稿ではこの機構については触れない

[3]イリノイ・プロトコルで Valid-Exclusive と Shared を区別せず Clean とし，Invalid，Clean，Dirty の 3 状態からなる方式を想定した



(a) directory[i] = 0 の場合キャッシュiの状態はINVALIDまたはSTALEであり，図上の遷移しか発生しない



(b) directory[i] = 1 の場合キャッシュiの状態はCLEANまたはDIRTYであり，図上の遷移しか発生しない

コマンド／レスポンス

コマンド

Rc[i] : read request from cache i

Wc[i] : update from cache i

Zc[i] : repalce from cache i

レスポンス

wb[j] : write back request to cache j (directory[j] = 1)

d[i] : send line to cache i

s[j] : staling to cache j (j ≠ i & directory[j] = 1)

— : no responce

図 5: i番目のキャッシュからのコマンドによるメモリの状態遷移図

レクトリのビットがセットされているかいないかでFSを検出し，以下のように動作する．

- updateを発行したキャッシュに対応するディレクトリのビットがセットされている場合，そのラインに対して更新を行ったラインは当該キャッシュのみであり，FSが発生しておらず図5(b)に示される状態遷移を行う．この時，他のCLEAN状態のキャッシュ・コピーはstalingを受けるのだが，既に書き込みを行ってDIRTY状態に遷移している可能性がある．そのため，キャッシュ・ライン上にmodify-bitを設け，書き戻し時には書き込みが行われたワードのみを選択的に書き戻す必要が生じる．
- ディレクトリがセットされておらずメモリがMODIFIED状態の場合，FSがこの瞬間発生したと考えられる．よって，最新のデータを保持していたキャッシュに対して書き戻しを要求しメモリに反映させてからメモリに対する更新を行う．またCLEAN状態の場合，既にFSが発生しておりメモリが最新のデータを保持しているのでメモリに対して更新を行う．(図5(a)参照)．

表 2: i番目のキャッシュからのコマンドによるディレクトリの動作

read request	update	replace
set[i]	unset[j] (j ≠ i)	unset[i]

4 性能評価

この章では，前章で提案したSelf-Invalidate型キャッシュ・コヒーレンス制御方式の性能評価を行う．比較の対象を従来のライト・バック-インバリデート方式とし，シミュレータを用いて性能の比較を行う．

4.1 シミュレータおよびベンチマーク

シミュレータはプロセッサ部，キャッシュ部，ネットワーク部，メモリ部からなり，クロックに同期しながら動作する．プロセッサ部は，ベンチマーク・プログラムを実行しながらメモリ・アクセス命令を発行し，他の構成要素はそれを受けて動作する．ただし，命令は100%キャッシュ・ヒットするものとしている．

今回シミュレーションを行ったマシンのモデルを図

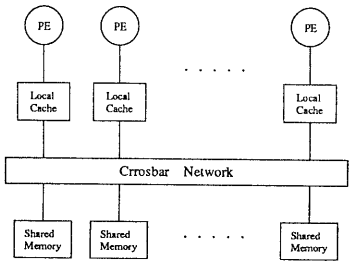


図 6: マシンのモデル

表 3: シミュレーションのパラメータ

プロセッサ	クロック (cycle)	1
キャッシュ	ライン・サイズ (B)	32
	ライン・アクセス (cycle)	2
	ワード・アクセス (cycle)	1
	タグ・アクセス (cycle)	1
ライト・バッファ	エントリー数	1
	サイズ (B)	32
	ライン・アクセス (cycle)	2
	ワード・アクセス (cycle)	1
ネットワーク	メッセージのみ (cycle)	40
	ラインを含む (cycle)	56
メモリ	ライン・アクセス (cycle)	20
	ワード・アクセス (cycle)	8
	ディレクトリ・アクセス (cycle)	8

6に示す．各プロセッサは，ローカルなキャッシュを保持し，プロセッサ数と同数のメモリ・モジュールとクロスバー・ネットワークを介して接続されている．また，同期機構としてハードウェア・バリア同期機構を用いた．

ベンチマーク・プログラムとしてはバブルソートを用い，問題の大きさを一定にし，プロセッサ台数を変化させることで false sharing の発生する割合を変化させた．

4.2 シミュレーション結果

シミュレーションの対象としては，

[WBI] 従来のイリノイ型のキャッシュ・コヒーレンス制御方式

表 4: false sharing が発生するラインの割合

台数	1	2	4	8	16	32	64
割合	0	0.008	0.023	0.055	0.12	0.24	0.49

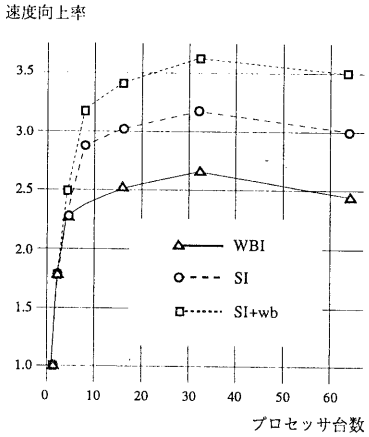


図 7: バブル・ソートの台数効果

[SI] 提案した Self-Invalidate 型キャッシュ・コヒーレンス制御方式（ライト・バッファなし）

[SI+wb] 提案した Self-Invalidate 型キャッシュ・コヒーレンス制御方式（ライト・バッファあり）

の 3 方式とした．

SI+wb で使用したライト・バッファには [7] で提案された方式を利用した．具体的には，同一ラインに対する連続した書き込みが発生した場合に，ワード単位にメッセージを流すのではなくライト・バッファ上でライン単位にまとめて，1 回のメッセージでメモリへの更新を行う方式である．

バブル・ソートにおけるソートの要素数を 1024 とし，プロセッサ台数を 1〜64 に変化させて false sharing の割合を変化させ（表 4参照），それぞれ表 3に示すパラメータでシミュレーションを行った．ネットワークに流れるメッセージの数，および SI+wb 方式で一台のプロセッサで実行した場合を基準とした台数効果を

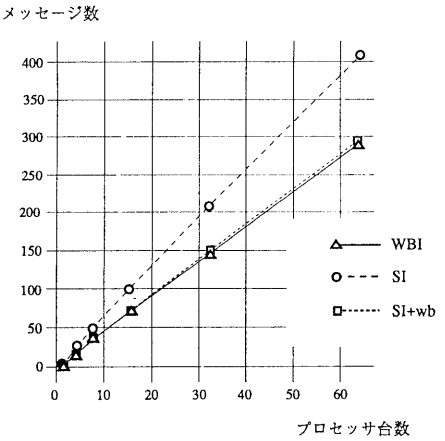


図 8: ネットワーク・トラフィック

測定した．結果を図 7，図 8に示す．

図7の結果より，WBI の方式と比較して SI，SI+wb の両方式とも速度向上が得られていることがわかる．しかし，図 8に示すようにライト・バッファを用いない SI の方式では WBI と比較してネットワーク・トラフィックが増加しているため，ネットワークの性能によっては性能低下の可能性も考えられる．しかし，ライト・バッファを用いた SI+wb 方式では WBI と変わらぬネットワーク・トラフィックを実現しているため，ネットワークの性能に依存せず WBI より高い性能を出すことが可能であると考えられる．

5 おわりに

本稿では，ライン単位のコヒーレンス制御を行うディレクトリとデータ単位のコヒーレンス制御を行うソフトウェアを組み合わせたコヒーレンス制御方式を提案した．

今回はバブル・ソートのプログラムをベンチマーク・プログラムとし，シミュレータを用いて従来方式と比較して性能向上が得られたことを示した．今後は，より多くのベンチマーク・プログラムを用いさらに比較を進める．

謝辞

日頃御討論頂く京都大学工学部情報工学教室富田研究室の諸氏に感謝致します．なお，本研究の一部は文部省科学研究費（重点領域研究（1）課題番号 04235103「超並列ハードウェア・アーキテクチャの研究」）による．

参考文献

[1] Y.-C. Chen and A.V.Veidenbaum:"A Software Coherence Scheme with the Assistance of Directories", *Technical Report 1106,CSDR,University of Illinois*,1991

[2] J.Torrellas, M.S.Lam, and J.L.Hennessy:"Shared data placement optimizations to reduce mulitprocessor cache miss rates" *Proc. 1990 Int'l. Conf. Parallel Processing*,volumeII,pp266-270,August 1990

[3] S.J.Eggers and T.E.Jeremiassen:"Eliminating False Sharing" *Proc. 1991 Int'l. Conf. Parallel Processing*,pp377-381,1991

[4] J.Fang and M.Lu:"A Solution of Cache Ping-Pong Problem in RISC based Parallel Processing Systems" *Proc. 1991 Int'l. Conf. Parallel Processing*,pp238-245,1991

[5] H.Cheng and A.V.Veidenbaum:"A Cache Coherence Scheme with Fast Selective Invalidation," *Proc. 15th Int'l. Symp. Computer Architecture*,pp299-307,Jun.1985

[6] J. Archibald and J.-L. Baer:"Cache Coherence Protocols:Evaluation Using Multiprocessor Simulation Mode", *ACM Trans. Computer Systems,Nov.1986,pp.273-298*

[7] Y.-C. Chen and A.V.Veidenbaum:"An Inproved Write Buffer Design for a Write-Through Cache", *Technical Report 1105,CSDR,University of Illinois*,1991

[8] Sarita V.Adve,Mark D.Hill:"Weak Ordering - A New Definition", *Proc. 17th Int'l. Symp. Computer Architecture*,pp214,May 1990

[9] 岩田ほか:" 統合型並列化コンパイラシステム—コンパイラ支援キャッシュ・コヒーレンス制御—", 情処研報,90-ARC-83-21(1990 年 7 月)