解 説

キャッシュメモリの一致性について†

浦 城 恒 雄††

1. はじめに

メモリシステムの階層化の考えはコンピュータの発明の当時からあったようで，Von Neumann の論文[1]にも示されている．プロセッサの性能に見合った十分な容量の高速メモリを実装することが経済的でないとき，小容量の高速高ビット価格のメモリと，大容量の低速低ビット価格のメモリを組み合わせて階層的にメモリシステムを構成することは有効な手段である．最初に実現したのは1950年代末に英国において開発された ATLAS[2] で，コアメモリと磁気ドラムの2レベルの階層をもち，特別なハードウェア機構とソフトウェアにより階層制御が行われた．

主メモリとそれより1桁以上高速なアクセスタイムをもつキャッシュ (cache) メモリをもち，階層制御を完全にハードウェアで行う方式を最初に実用化したのは，1968年に発表された IBM 360/85[3] である．

当時日立製作所で次期の大形機システムを検討中であった筆者らは，この方式を主メモリを共有したマルチプロセッサシステムに適用しようとした．キャッシュはその高速性を生かすために各プロセッサごとに固有にもたせる必要があるが，すべてのプロセッサが最新の共有データを使用できねばならない．メモリへのライト (write, 書込み) 動作はキャッシュに書き込むとともに主メモリにも書き込む方式を採用したが，他のプロセッサがその記憶場所を含むブロックをキャッシュに取り出しているときにはデータの不一致が発生する．この一致性 (consistency または coherency) の問題を解決するための方法を考案した．これは後にライトスルー無効化方式[4]と呼ばれるものの原型である．共有メモリバスを介して送られるアドレスおよび書込み信号を各プロセッサが監視し（スヌーピング方式の原型でもある），他のプロセッサによって変更されたブロックを保持するプロセッサはそのブロックを無効にすることにより一致性を保つ方式である．

1980年代に入ると非共有キャッシュをもった共有メモリマルチプロセッサは汎用大形機システムでの主流的技術となり，2レベルのキャッシュをもつシステムも開発された[5]．キャッシュの一致性を制御する方式もいろいろな改良が加えられてきた．

またここ10年来の半導体技術の進歩により，1チップ VLSI マイクロプロセッサも年々高性能化し，さらに RISC 技術の採用により拍車がかかっている．マイクロプロセッサの性能向上にともない，メモリの性能とのギャップが生じ，性能を十分引き出すためにキャッシュ方式を採用するようになった．

こうして低コスト高性能プロセッサができると，それらをマルチプロセッサ構成にすることにより，従来の大形機なみの性能をもつことも可能になり，また相当多数のマルチプロセッサ構成をとることにより，スーパコンピュータクラスのシステムも考えられるようになった．こうした背景のなかで，共有メモリマルチプロセッサにおけるキャッシュの一致性の問題は，ワークステーションからスーパコンピュータのクラスにいたる広い範囲での技術的課題となりつつある．

本稿では単一プロセッサのキャッシュについてはある程度の知識をもった読者を主対象とするが，まず用語の説明を兼ねてキャッシュの基本とライト制御について簡単に概説したあと，共有メモリマルチプロセッサにおけるキャッシュの一致性の問題を説明し，これを解決するための方法に関し，主としてハードウェアによる一致性の保証方法について，歴史的な改良の経過などを含めて解説する．

† Consistency of Cache Memory by Tsuneo URAKI (Hitachi Ltd.).

†† (株)日立製作所

2. キャッシュ制御

2.1 キャッシュ

主メモリおよびキャッシュはブロック (block) と呼ばれる単位 (通常 16 バイト～128 バイト) に論理的に分割され，主メモリとキャッシュとの対応をとるマッピング (mapping) 制御や，主メモリとキャッシュの間のデータ転送 (一部のライト動作を除き) もブロックを単位として行われる．転送データ幅とは必ずしも一致せず，分割転送されることが多い．

キャッシュが保持するデータは主メモリのコピーであるから，キャッシュのブロックに対応して，それが主メモリのどのブロックのコピーであるかを示す主メモリブロックアドレス (アドレスの上位部分) と，ブロックの状態を示すフラグ (最低限データの有効性の有無を示す 1 ビット) が必要であり，これらを総称してディレクトリィ (directory) と呼ぶ．ディレクトリィを用いてのマッピング制御にはいろいろな方式があるが，複数のブロックをグループ化してマッピングを行うセットアソシアティブ方式が最も普及している．また仮想アドレス方式が広く普及し，そのための変換と関連をもたせる場合もあるが，本稿ではまったく独立と考えて仮想アドレスには言及しない．

プロセッサがメモリのあるアドレスをアクセスしようとすると，そのアドレスを含むブロックがキャッシュに存在しているときは，キャッシュヒット (hit) と呼び，キャッシュを利用できる．キャッシュに存在していないときはキャッシュミス (miss) と呼び，そのブロックを主メモリからキャッシュに転送してから利用する．これをブロック転送という．このときキャッシュ上の他のブロックを追い出す (purge) 必要があり，これをブロック置換え (replacement) という．置換えのアルゴリズムとしては，同じブロックグループの中から無効なブロックを優先し，続いて最も長く利用されなかったブロックを置換えの対象とする LRU (least recently used) 方式が広く採用されている．

メモリの実効的なアクセスタイムを向上するためにはキャッシュミス率を下げるとともにミス発生時の処理時間を短くする必要がある．

2.2 ライト制御

メモリへのライト動作の制御に関して大別して二つの方式がある．

(1) ライトスルー (write-through) 方式

ライト動作により変更されたデータはつねに主メモリに書き込まれる．キャッシュにそのブロックが存在しておればキャッシュにも書き込まれる．キャッシュを採用した古典的なシステムの多くはこの方式であり，制御も比較的簡単である．しかしライト動作を必要とするストア系の命令の使用頻度が高いと主メモリの使用率 (単位時間当りのアクセス回数) が高まり，性能への影響がでる．

(2) ライトバック (write-back) 方式

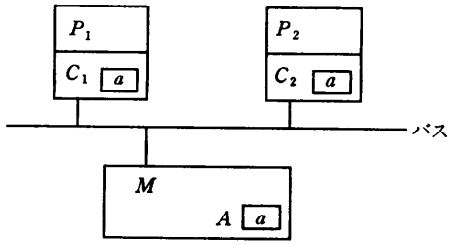
ライト動作はリード (read, 読出し) 動作と同じように扱われる．つまりキャッシュに存在しないときは主メモリからコピーされ，そのあとこのブロックへのアクセスはリード／ライトともにキャッシュを対象とする．しかしブロック置換えが発生すると追い出されるブロックを主メモリに戻す必要があり，これをライトバックあるいはコピーバック (copy-back) という．

キャッシュに存在している間にライト動作により変更されたかどうかを示す変更フラグをディレクトリィにもつことにより，変更のないブロックに対してライトバックを省くことができ，主メモリの使用率はライトスルー法に比べて低く押えることができる．

2.3 共有メモリマルチプロセッサとキャッシュの一致性

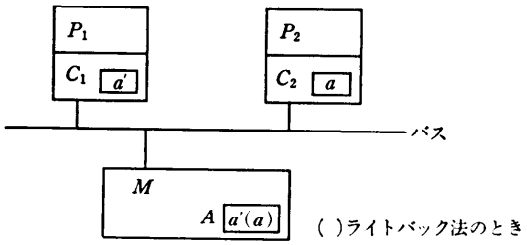
キャッシュが共有主メモリに附属した共有キャッシュでなく，各プロセッサに非共有で附属しているマルチプロセッサシステムでは，もし各プロセッサが単一プロセッサで必要なキャッシュ制御のみを行うと，主メモリのあるブロックは 2 個以上のキャッシュに同時に存在することがある．こういう状態で，ライト動作や，プロセスの移転，あるいは主メモリへの直接的な I/O 動作が起こるとデータの不一致が発生する (図-1)．

共有メモリマルチプロセッサシステムではメモリへのすべての参照に対して，どのプロセッサで変更されたにせよ最も最新のデータを得られることが，ソフトウェアからみたマルチプロセッサの論理モデルとして一番望ましいことであり，そのためにはキャッシュの一致性を論理的に保証する必要がある．



(a)

P_1 と P_2 が M 内の A をリードしたとき
2個のキャッシュに存在し，一致性は保たれている



(b)

P_1 が M 内の A に a' をライトしたとき
ライトスルー法のときは　C_2 のデータが陳腐で不一致
ライトバック法のときは　C_2 および M のデータが陳腐で不一致

P_1, P_2: プロセッサ　　M: 共有主メモリ
C_1, C_2: キャッシュ

図-1 主メモリとキャッシュの不一致

2.4 キャッシュの一致性の保証法

この問題を解決するためにはなんらかのハードウェア機構が必要であるが，ソフトウェアの制御と合わせて保証する方法と，ハードウェア制御主体で保証する方法に大別できる．

（1） ソフトウェア制御の方法

まず一致性を乱す要因を排除するために，共有の書込み可能データをキャッシュ不可 (non-cachable) とし，非共有あるいは共有の書込み不可データのみをキャッシュ可 (cachable) にすることにより，一致性を維持する方式がある．このためにはコンパイラなどが変数データをキャッシュ可か不可かを識別し，メモリのブロック単位にタグを付与し，ハードウェアでそれをチェックする必要がある．この方式は原理は簡単であるがユーザおよびコンパイラに対して，マルチプロセッサの透過性を失うとともに，言語のデータの共有性に関する指定能力との関係も生じる．またプロセス移転やI/O動作による不一致をさけるために，これらの実行の前や後にキャッシュの一部または全部を無効化する必要があり，性能の低下を引き起こす．

すべての共有書込み可能データをキャッシュ不可にしないで，これらを特別領域と考え，これを保護制御するためのロックなどだけキャッシュ不可とする方式もある．特別領域で変更したデータは，つぎのアクセスのときにキャッシュに古いデータが残らないようにキャッシュ上で無効にしておく．他のプロセッサも必ずロックの獲得を通して特別領域にシーケンシャルにアクセスすることにすれば一致性は保たれる．この方式はソフトウェアによる同期化ポイントでのみ一致性を保証するという考え方で，ライト制御にライトスルー方式を採用すると実現は比較的容易である．

一致性の問題を生じうるブロックに対して無効化命令をいかに効率的に発行し，ヒット率の劣化を防ぎ，かつプロセッサ数の増大にも対応できるようにすることが主要な課題で，そのための研究がいろいろ行われている[6]．

（2） ハードウェア制御の方法

すでに述べたように制御の容易さから，ライト制御にライトスルー方式，一致性制御には無効化方式がまず採用された．プロセッサの数が少なく(2〜4個)，主メモリの性能に比べてプロセッサの性能があまり速くないときは，ライト動作にはバッファをもたせていわゆる置いてきぼり制御を行い，一致制御にも若干のバッファをもたせることにより大幅な性能低下をさけることができた．むしろキャッシュが高価であることから容量が少なく，キャッシュミスによる性能低下のほうが問題が大きかったといえる．

しかしプロセッサ性能が向上するにつれ，まずライトスルー方式では共有メモリバスの使用率が高くなり，プロセッサ数をたとえば2個に限定するか，ライトバック方式による使用率の低減をはかる必要が生じた．しかしライトバック方式ではライト動作でキャッシュしか変更しないので，ある時間断面では主メモリと不一致になっている．こういう状態で，同じブロックを他のキャッシュ上に存在することを許していると一致性が保てなくなる．

ハードウェアの制御で一致性を維持するためには，あるプロセッサが他のキャッシュに存在するブロックにライト動作を行うと，他のキャッシュにあるデータは陳腐 (stale) になるので無効化 (invalidate) するか，新しいデータに更新 (update)

するかしなければならない．キャッシュミスが起こったとき，主メモリが最新のデータに更新されていないときは最新のデータを見付けてミスを起こしたキャッシュにブロック転送してやらねばならない．この二つの行動がキャッシュ一致性制御の基本である．

いろいろな制御方式が開発されてきたが，ディレクトリィ方式とスヌーピング方式の2種類に大別できる．

3. ディレクトリィ方式

3.1 Tang の方法[7]

1976年にTangが各プロセッサのキャッシュディレクトリィのすべてのコピーを主メモリ側に主ディレクトリィとしてもたせる方式を発表し，ディレクトリィ方式の出発点となった．キャッシュのブロックの状態としては従来の“無効／有効”に加えて“共有 (shared)／排他 (exclusive)”のフラグをもたせる．排他ブロックは1個のキャッシュにしか存在しないが，主メモリからブロック転送されたのち変更が行われ，主メモリと不一致の可能性があり，ブロック置換のときライトバックの必要なブロックである．共有ブロックは複数のキャッシュに存在しうるブロックであるが，主メモリとは一致している．

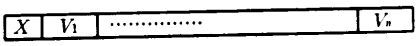
共有ブロックに対してライト動作が起こると，状態を排他にするとともに主ディレクトリィに報告して，他のキャッシュに共有ブロックとして存在しているかどうかを調べ，あれば無効化をしてからライト動作を実行する．

リード動作でキャッシュミスが起こるとやはり主ディレクトリィを調べ，もし他のキャッシュに排他ブロックが存在しておれば，そのブロックの状態を共有にするとともにデータを主メモリにライトバックさせてから要求元のキャッシュに転送する．要求元キャッシュの状態も共有となる．

ライト動作でキャッシュミスが起こると，やはり主ディレクトリィを調べ，他のキャッシュに排他ブロックが存在すれば，そのブロックを無効化するとともにデータを主メモリにライトバックさせてから要求元のキャッシュに転送する．もし他のキャッシュ（複数存在しうる）に共有ブロックとして存在すれば，そのすべてを無効化し，要求元キャッシュへは主メモリから転送する．いずれの場合も要求元キャッシュのブロック状態は排他となる．そのあと排他状態のブロックは主ディレクトリィに知らせることなくローカルに使用できる．

3.2 Censier & Feautrier の方法[8]

Tang の方法では，各キャッシュディレクトリィのコピーすべてを主ディレクトリィにもち，調べるときは各ディレクトリィをすべてサーチしなければならない．これに対し，主ディレクトリィは主メモリの各ブロックに対して1ビットの共有／排他ビットとキャッシュの数に等しい無効／有効ビットをもたせる方法が提案された（**図-2**）．排他ブロックはたかだか1個のキャッシュにしか存在しないので，Tang の方法と等しい状態情報をもつ上に，要求元キャッシュから与えられた主メモリのブロックアドレスにより直接的にブロックの状態情報にアクセスできる長所をもっている．



X: 共有／排他ビット
　0 共有　1 排他
V_i: 無効／有効ビット（キャッシュ C_i に対応）
　0 C_i にコピーが存在しない
　1 C_i にコピーが存在する

図-2 主ディレクトリィの構成例

3.3 ディレクトリィ方式の改良

Tang や Cencier & Feautrier の方法では主ディレクトリィをキャッシュ（つまりプロセッサ）の数に応じた構成をとる必要がある．これに対し，現実のマルチプロセッサの実行環境においては複数のキャッシュで共有ブロックが存在する率はあまり高くないとの前提に立ち，主ディレクトリィにおいて各キャッシュに対応した有効／無効ビットを廃してキャッシュの数に無関係にブロック当たり2ビットとし，コード化してつぎの4状態を示す方法[9]が提案された．

a）キャッシュに存在せず
b）1個のキャッシュに存在し，無変更
c）何個かのキャッシュに存在し，無変更
d）1個のキャッシュに存在し，変更

ライト動作はキャッシュにブロックを確保してから行うものとすると，b)の状態では　d)状態へ変えるのみでよいが，c)の状態でのライト動作や，　a)状態以外で起こったキャッシュミスに対してはそのブロックを含むキャッシュを示す情報をもたないので，無効化やライトバックの要求

を他のすべてのキャッシュに伝えるための全報知(broadcast)が必要となる.

しかしこれでは全報知不要というディレクトリ方式の利点が生かされないので,ブロックはシングルコピーに限定して c)状態をなくし,主ディレクトリに各ブロックに対応してそのブロックをもつキャッシュへのポインタを1個もたせることにより全報知を回避する方法[10]も提案されている.さらに拡張して複数のポインタを設け,その数だけのキャッシュに存在することを許す方法もある.一般にはライト動作が行われるブロックが他のキャッシュに含まれているのはごく少数のブロックに限られており,それだけ全報知は非能率と考えられ,少数のポインタに限定しても性能への影響はあまり大きくないと考えられる.

4. スヌーピング方式

4.1 Goodman の方法[11]

1983年に Goodman はキャッシュつきのシングルボードプロセッサを Multibus を用いて接続した共有メモリマルチプロセッサの構成法を検討し(**図-3**),バスの使用率を下げ,かつ各プロセッサで分散して制御できるキャッシュの一致性制御方法を提案した.ブロックが複数のキャッシュに存在する共有状態にあるとき,最初のライト動作のみライトスルー方式で行い,そのブロックへの2回目以降のライト動作はライトバック方式で行われることから,ライトワンス(write-once)法と命名された.また絶えずバス上の他のプロセッサのキャッシュの一致性に関連ある動作を監視することからスヌーピング(snooping)方式の元祖とされている.

ライトワンス法を説明するにあたり,まず共有バス上でのバス要求について分類する.

- リード　プロセッサのリード動作に対してキャッシュミスしたとき発生するブロックリード要求

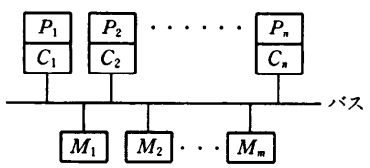


図-3 バス結合マルチプロセッサの構成

- 変更リード　プロセッサのライト動作に対してキャッシュミスしたとき発生する変更意図をもったブロックリード要求
- ライト　変更ブロックをライトバックするときのブロックライト要求
- 部分ライト　部分的なライトスルー制御をするときのブロック内部分ライト(バイト単位,語単位など)要求
- 無効化　一致性制御のため他のプロセッサのキャッシュのブロックを無効化する要求

またバス要求を出した側を要求キャッシュ(requesting cache),バスを監視して一致性制御のための処理の必要性を検出した側を検出キャッシュ(detected cache)と呼ぶ.

Goodman の方法ではキャッシュの各ブロックはつぎの4状態が存在する.状態遷移を**図-4** に示す.

a) 無　　効
b) 共　　有　主メモリとは一致
c) 排他一致　1度だけ変更されているが主メモリと一致
d) 排他不一致　2度以上変更され主メモリと不一致

プロセッサのライト動作は,キャッシュのブロックの状態が共有のときはキャッシュに書き込み,状態を排他一致にするとともに,部分ライトのバス要求をして主メモリにも書き込まれる.バス要求はバスを介して全報知され,検出キャッシュはそのブロックを無効化する.排他一致または不一致状態にあるときのライト動作はキャッシュのみを対象に行われ,ブロックの状態は排他不一

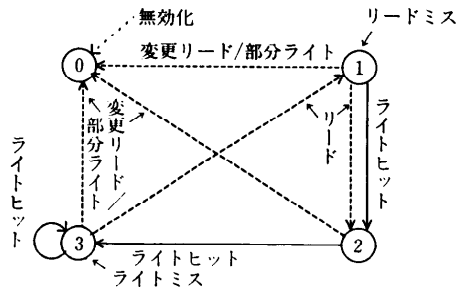


図-4 ライトワンス法の状態遷移図

致となる.

リードミスに対してはバスにリード要求を出し，主メモリからブロック転送を受けるとブロック状態は共有にする．しかし正当なデータがブロック転送されるためには，排他不一致状態にあるブロックを検出したキャッシュは，バス動作を中断させ，そのブロックを主メモリにライトバックしたあと動作を再開させ，検出キャッシュの状態は共有にする．排他一致状態にあるブロックを検出したときはバス動作の中断は行わず，検出キャッシュの状態は共有にする.

ライトミスに対してはバスに変更リード要求を出す．共有状態にあるブロックを検出したキャッシュはそのブロックを無効化する．排他不一致状態にあるときは，リードと同様にバス動作を中断させ，ライトバックしてから動作を再開させるが，検出キャッシュの状態は無効にする．主メモリからのブロック転送が終わると要求キャッシュの状態は排他不一致にする.

プロセッサのキャッシュへのアクセスへの影響を少なくしてバス監視を行うためには，ディレクトリィを2組設けることも提案された.

4.2 スヌーピング方式の改良

Goodman の提案を出発点として，スヌーピング方式を実際に適用するためにいろいろな改良が加えられた[12]．代表的なものに，Berkley 法[13]，Dragon 法[14]，Illinois 法[15]，Firefly 法[16]などがある．改良点に着目して主な点を述べる.

（1） 共有ラインの追加

リードまたは変更リードのバス要求に対し，同じブロックのコピーをもつ検出キャッシュはバス上に設けられた“共有ライン (shared line)”を立ち上げる．要求キャッシュは所定時間内にこのラインを検出しないときは，ブロック転送後このブロックの状態を排他にする．これにより，要求キャッシュにのみ存在するブロックに対するライト動作に対して，他のキャッシュに知らせるバス動作が不要となる．この方法は Illinois 法，Dragon 法，Firefly 法などで採用されている.

（2） キャッシュ間転送

不一致状態にあるブロックをもつ検出キャッシュは，要求キャッシュにコピーをバス経由で転送し，主メモリは更新しない方法である．これにより，要求キャッシュがブロックを追い出す前に変更したときに発生するバス動作を減らすことができる．ブロックの状態として共有でかつ主メモリと一致している状態（共有一致）と共有だけど主メモリと不一致の状態（共有不一致）を区別し，リードのバス要求に対しては，検出キャッシュは追い出されるときの主メモリへのライトバックの責任（所有権と呼ぶこともある）を維持するためにブロックの状態は共有不一致となり，要求キャッシュは共有一致状態となる.

Berkley 法，Dragon 法で採用された．Illinois 法では同じバス動作で要求キャッシュと主メモリの両方にブロック転送する方法を採用している.

（3） 部分ライトの廃止

ライトワンス法では，共有ブロックへの最初のライト動作に対してはバスに対して部分ライト要求をしたが，これを無効化要求に変更する．部分ライトは2バスサイクルかかるケースが多く，それに対し無効化は1サイクルで実行できる．したがってこの改良によりバスおよび主メモリ制御から部分ライトをなくし，バス使用率を減らすことができる．しかし1回切りのライト動作のあとブロック追出しされるときライトバックが不要であったものが，この改良では必要となり，バス使用率の増大の可能性もある．この改良は多くの方式で採用されている.

（4） ライト全報知 (write broadcast)

スヌーピング方式の基本は変更されるブロックは共有を許さず，共有ブロックの変更に対しては他のキャッシュのブロックを無効化するライト無効化方式である．ライト無効化方式による性能オーバヘッドとしては，

a）無効化要求の処理によって起こるプロセッサとスヌーピング制御の干渉

b）無効化によるキャッシュミス率の増大

が考えられる．a）はディレクトリィを2組もつことにより監視による干渉は減らすことはできるが，キャッシュやディレクトリィの状態変化に対しては競合をさけるためのロック制御が必要であり，無効化による干渉は無視できない．b）はデータを本質的に共有しているときは仕方がないにしても，共有はしないが，同一ブロック内の異なるデータを異なるプロセッサで変更することによるプロセッサ間ブロック競合が起こると，互いに無効化をし合うピンポン現象が起こる．ブロッ

クサイズが大きくなるとこの現象の起こる率が増え，問題によっては大きな影響がでる危険性がある．

この問題を解決するために，共有ブロックに対する変更を許すかわりに絶えず同一コピーを維持する方法が提案された．共有状態にあるブロックへのすべてのライト動作はバス上に全報知され，そのブロックをもつすべてのキャッシュのコピーが更新され，かつ主メモリも更新される．

しかしこの改良のみを採用するとライトスルー方式になってしまい，バスの使用率増大による性能低下が大きいが，改良(1)を併用して始めて実用的な価値がある．つまり共有データに対してライトスルー方式，非共有データに対してライトバック方式という形態が取れることに意味がある．

この方式をライト無効化方式に対してライト全報知方式という．Dragon 法や Firefly 法で採用された．しかし無効化の競合問題はさけられるものの，キャッシュの容量が増えると，以前のプロセスが利用したデータなどのように，もはや実質的には共有されていないがキャッシュに残っているブロックに対して，異なるプロセッサでプロセスが再開され，そこで変更されると，バス監視により更新を続けてやることになり，性能を落とす要因も存在する．

改良(3)と(4)を組み合わせたときは，共有ブロックへのすべてのライト動作は全報知され，キャッシュは更新されるが主メモリはされない．このときは改良(2)と同様にいずれかのキャッシュがブロック追出しのときライトバックをする責任を取る必要があり，通常要求キャッシュがその責任を取る．

(5)　リード全報知 (read broadcast)

ライト無効化方式の問題点を解決するもう一つの方法として提案された[17]．共有ブロックに対するライト動作をバスで検出すると，いったんはそのブロックを無効にする．しかしそのあとそのブロックに他のキャッシュからのリード要求を検出するとバスからそのデータを取り込むことにより再び有効化してミスの発生を減らす方法である．これによれば1回の無効化によって起こるキャッシュミスの発生を，全システムで最大1回にすることができる．

5. 両方式の適用領域と最近の動向

5.1 両方式の適用領域

ディレクトリ方式は汎用大形機システムに向いた方式と考えられ，いくつかのシステムで採用されてきた．主な理由は汎用大形機システムでは，共用バスによるマルチプロセッサ構成では十分な性能が得られないためである．

スヌーピング方式は全報知が容易にできるバス結合の共有メモリマルチプロセッサに適合しており，ミニコンピュータの領域（いわゆるスーパミニコンも含む）において採用されてきた．最近では高性能ワークステーションにおいても採用されつつある．方式の改良と評価については多くの報告がなされており，かなり出つくした感じもする．しかしいかに1個のプロセッサあたりのバス使用率を減らす努力をしても，プロセッサの個数に比例してバス使用率は増えるのでおのずから限界がある．プロセッサとバスの性能にもよるが，プロセッサの個数としては8個からせいぜい16個あたりに適用限界がある．スヌーピング方式は高速の全報知機構が不可欠であり，たとえバスの階層化などの工夫をしても限界がある．

一方ディレクトリ方式の大きな利点は共有ブロックのコピーをもつキャッシュの場所が分かっていることである．そのためすべての共有ブロックを見付けるための全報知が不要であり，無効化の必要なときはコピーをもつキャッシュに個別にメッセージを送ればよい．実際の動作環境では多くのキャッシュにコピーが存在して無効化をしなければならないケースは少ないと考えられる．また無効化メッセージは共有バスでなくても任意のネットワークを介して送ることが可能である．この全報知不要がキャッシュ一致性を保証したマルチプロセッサの規模に関する限界を破り，たとえば100台を越える大規模化を可能にする．また制御もスヌーピング方式に比べて簡単なことも長所である．主メモリの分散化とともに，主ディレクトリも分散化することも可能で，分散化により両方のバンド幅を軽減することが可能であり，大規模な共有メモリマルチプロセッサに対して有力な方式と考えられる．

5.2 標準バス仕様への組込み

最近，スヌーピング制御を標準バスの仕様に組み込む動きがある．ライトワンス法は Multibus の

上で動かすことを狙ったが，その後の改良は，たとえば共有ラインなどの追加を必要とする．またプロセッサの性能向上に対応するためにも既存のMultibus や VMEbus などは能力的にも不足している．これに対して IEEE の Futurebus[18] はキャッシュ付きのマルチプロセッサ構成を可能とする標準バスとして検討が進められ，今まで提案されてきた代表的なスヌーピング方式の制御法をそのままか，あるいは一部を修正して容易に実装可能になっており，スヌーピング方式の適用が可能な小中規模のマルチプロセッサ用標準バスとして提案された．キャッシュの各ブロックは，M（排他，変更），O（共有，変更），E（排他，無変更），S（共有，無変更），I（無効）5状態とし，プロセッサからのイベントに対する制御と，共有バスのスヌーピングによって検出されたイベントに対する制御を各状態に対応して定めることにより，一致性のプロトコルを規定している．

この Futurebus (IEEE 896.1-1987) をベースにさらに改良したものが，Futurebus＋ として検討されている[19]．この仕様案ではブロックの状態としてO（共有，変更）状態を許さず4状態であるが，要求と応答をバス動作上分離するスプリットバストランザクション方式を採用し，階層的にバス結合された大規模なシステムへの適用を可能にするような改良が加えられており，90年代の標準的なバス仕様として注目されている．

5.3 大規模システムへの適用

VLSI 技術の進歩により，今やキャッシュ内蔵の高性能プロセッサチップができるようになった．マルチプロセッサの数も8台は実用的になり，10数台規模のものも実用化が近い．一方まだ実験研究段階が多いが，少なくとも数十台から数百台，将来的には数千台を目標とする大規模マルチプロセッサの研究開発も進められている．このような場合も共有メモリの必要性はやはり高く，キャッシュの一致性をどのように保証するかは重要な課題になっている．

プロセッサ数の拡大にともない，ハードウェア制御が複雑になるので，たとえば IBM の RP3[20] とかニューヨーク大の Ultracomputer[21] などではソフトウェア的手段を取り，コンパイラに変数がキャッシュ可か不可を決めさせ，一致性はライトスルー方式の採用とプログラムによる共有キャッシュ可変数の無効化（プログラムの終了時）などにより実現している．ハードウェアによる一致性制御について代表例を紹介する．

5.3.1 Wisconsin 大 Multicube[22]

このシステムではキャッシュ付きプロセッサは格子状バスによって結合されている（図-5）．各プロセッサはライトスルー型の内蔵キャッシュとライトバック型のスヌープキャッシュの2階層キャッシュをもつ．ブロックの状態としては，下記の状態をもつ．

グローバル状態

a）無変更　主メモリと一致．他のキャッシュにも存在可能性あり．

b）変　更　主メモリは正しくなく，どこか1個のキャッシュに正しいデータ．

ローカル状態

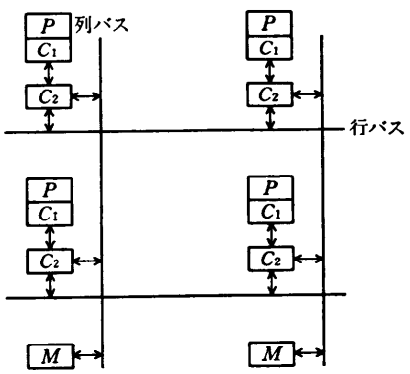
a）無　効

b）共　有　主メモリと一致．他のキャッシュにも存在可能性あり．

c）変　更　そのキャッシュに正しいデータ．

共有状態のブロックに対するライト動作に対して他のキャッシュのコピーの無効化が必要であるが，単一バスに比べ全報知が一度にはできないので，まず列方向に報知し，その列のキャッシュコントローラを介して行方向に報知する2ステップを取る．

変更状態のブロックから読み出す場合は，正しいデータをもつキャッシュを探す必要があるが，各キャッシュコントローラはその列に存在する変更ブロックのリストをすべてを記憶しておき，行方向に報知すれば正しいデータをもつキャッシュ



P: プロセッサ　　C_2: 第2レベルキャッシュ
C_1: 第1レベルキャッシュ　M: 主メモリモジュール

図-5 Multicube の構成

が特定でき，続いてそのキャッシュから，キャッシュコントローラを介して主メモリおよび要求元キャッシュにブロック転送される．

これはスヌーピング法の提案者であるGoodmanらにより開発されたもので，Illinois 法をベースに格子状バスへの適用拡大をはかったものといえる[23]．

5.3.2 Stanford 大 DASH

DASH システムの基本構成単位はクラスタと呼ばれ，4台のキャッシュつきプロセッサと分散主メモリがスヌーピングバスを介して接続されている．さらにクラスタ内には分散主メモリに対応するディレクトリィメモリとリモートアクセスキャッシュが含まれ，おのおの格子状のネットワークに接続され，クラスタ間の相互通信を可能にしている（**図-6**）．

キャッシュはプロセッサ内に2レベルあり，第1レベルは命令キャッシュとライトスルー型のデータキャッシュがあり，第2レベルはライトバック型のキャッシュである．クラスタ内では，他のプロセッサのキャッシュとリモートアクセスキャッシュが一緒になってリモートメモリに対するクラスタキャッシュになっている．クラスタ内ではスヌーピング方式の Illinois 法で一致性を保ち，クラスタ間では分散型ディレクトリィライト無効化方式で一致性を保っている．両方式の長所をいかして規模の大きいマルチプロセッサシステムを実現しようとしており注目に値する．

主メモリの各ブロックはつぎの3状態がある．

a）非キャッシュ　いずれのクラスタのキャッシュにも存在しない．

b）共有　1個以上のクラスタのキャッシュに存在するが変更されていない．

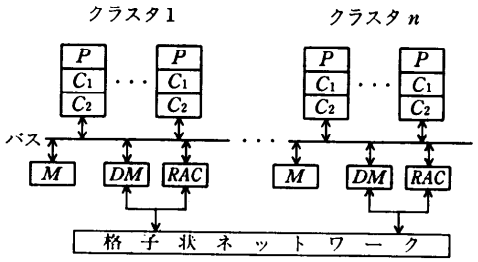


図-6 DASH の構成

c）変更　いずれかのクラスタのキャッシュに存在し，変更されている．

ディレクトリィは各ブロックに対し，状態情報および，キャッシュに取り込んでいるクラスタの番号を保持している．

DASH のメモリシステムは論理的には4つの階層的なレベルから成り立っている．

（1）　プロセッサ　レベル

各プロセッサがローカルにもつキャッシュ．

リード要求に対してはキャッシュに存在すれば読み出せるが，存在しないときは上位にデータ要求する．ライト要求に対しては，キャッシュに存在しかつ変更状態にあれば書き込めるが，そうでないときはこのブロックに対する所有権とデータを上位に要求する．

（2）　ローカルクラスタ　レベル

要求プロセッサの属するクラスタ内の他のプロセッサのキャッシュおよびリモートアクセスキャッシュ．

リード要求に対してはローカルクラス内に存在すれば処理できる．ライト要求に対してはローカルクラスタ内のいずれかのキャッシュに変更状態で存在すれば処理できる．ローカルクラスタ内で処理できないときは，さらに上位に要求する．

（3）　ホームクラスタ　レベル

与えられた主メモリブロックアドレスに対する分散主メモリとディレクトリィメモリを含むクラスタ．

ローカルクラスタとホームクラスタは一致している場合もあるが，一般にはネットワークを介してローカルクラスタからホームクラスタへ要求が伝えられる．下記の場合を除き要求はホームクラスタで処理され，ディレクトリィは必要な更新が行われる．

a）変更状態にあるときはさらに上位に要求を伝える．

b）ライト要求にともなう所有権要求に対して共有状態にあるときは，共有しているすべてのコピーを無効化する必要がある．ディレクトリィに従って必要なすべてのクラスタに無効化要求メッセージをネットワークを介して論理的にはポイント・ツー・ポイントで送られる．

（4）　リモートクラスタ　レベル

変更ブロックをキャッシュにもつクラスタ．

リード要求に対しては要求元クラスタに対してブロック転送するとともに，ホームクラスタの主メモリにライトバックしてディレクトリィも更新させる．ライト要求に対しては，要求元のクラスタにブロックデータおよび所有権を送るとともに，ディレクトリィを更新させる．

6. おわりに

20年以上も前から存在した課題であるキャッシュの一致性に関し，主としてハードウェアで制御する方法について歴史的経緯もふまえて解説をした．制御方式の説明が主体となり，その評価についてはほとんど定性的な説明にとどまった．

今後の課題としては，まずマルチプロセッサの本格的な普及の時代を迎えるにあたり，新しい利用動作環境をふまえた適切な評価方法の確立が重要と思われる．また大規模のマルチプロセッサシステムでは，次第にハードウェアの制御だけで完全な一致性を保証することがむずかしくなり（保証するための性能ペナルティが増える），ソフトウェアによる同期制御と組み合わせた有効な方法の開発が必要であろう．

参考文献

1) Von Neumann, J. et al.: Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, J. Von Neumann Collected Works, Vol. V, Oxford Pergamon Press (1963).
2) Fotheringhan, J.: Dynamic Storage Allocation in the ATLAS Computer Including an Automatic Use of a Backing Store, Comm. ACM, Vol. 4, pp. 435 (1961).
3) Conti, C. J. et al.: Structural Aspects of the System 360/85—General Organization, IBM Syst. J., Vol. 7, No. 1, pp. 2-24 (1968).
4) 日本国特許 943867 出願 1968. 5. 31 公告 1974. 3. 20 発明者 浦城恒雄 小高俊彦 出願人 日立製作所.
5) 小高俊彦他：演算パイプラインや3階層記憶により高速化を図った M-680/682 の処理方式，日経エレクトロニクス，No. 382, pp. 228-267 (1985).
6) Stenstrom, P.: A Survey of Cache Coherence Schemes for Multiprocessors, IEEE Computer, Vol. 23, No. 6, pp. 12-24 (1990).
7) Tang, C. K.: Cache System Design in the Tightly Coupled Multiprocessor System, AFIPS Proc., Vol. 45, pp. 749-753 (1976).
8) Censier, L. M. and Feautrier, P.: A New Solution to Coherence Problems in Multicache Systems, IEEE Trans. Comput., C-27, No. 12, pp. 1112-1118 (1978).
9) Archibald, J. and Baer, J. L.: An Economical Solution to the Cache Coherence Problem, Proc. 12th ISCA, pp. 355-362 (1985).
10) Agarwal, A. et al.: An Evaluation of Directory Schemes for Cache Coherence, Proc. 15th ISCA, pp. 280-289 (1988).
11) Goodman, J. R.: Using Cache Memory to Reduce Processor-Memory Traffic, Proc. 10th ISCA, pp. 124-131 (1983).
12) Vernon, M. K. et al.: An Accurate and Efficient Performance Analysis Technique for Multiprocessor Snooping Cache-Consistency Protocols, Proc. 15th ISCA, pp. 308-315 (1988).
13) Katz, R. H. et al.: Implementing a Cache Consistency Protocol, Proc. 12th ISCA, pp. 276-283 (1985).
14) McCreight, E. M.: The Dragon Computer System: An Early Overiview, Technical Report, Xerox PARC (1984).
15) Papamarcos, M. et al.: A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories, Proc. 11th ISCA, pp. 348-354 (1984).
16) Thacker, C. P. et al.: Firefly: A Multiprocessor Workstation, IEEE Trans. Comput., C-37, No. 8, pp. 909-920 (1988).
17) Segall, Z. et al.: Dynamic Decentralized Cache Schemes for an MIMP Parallel Processor, Proc. 11th ISCA, pp. 340-347 (1984).
18) Sweazey, P. et al.: A Class of Compatible Cache Consistency Protocols and Their Support by the IEEE Futurebus, Proc. 13th ISCA, pp. 414-423 (1986).
19) Futurebus +, IEEE Std. P 896. 1 Logical Layer Specifications Draft 8. 2, pp. 148-182, IEEE Computer Society Press (1990).
20) Pfister, G. F. et al.: The IBM Research Parallel Processor Prototype (RP 3): Introduction and Architecture, Proc. ICPP, pp. 764-771 (1985).
21) Rudolph, L. et al.: Issues Related to MIMD Shared-Memory Computers, The NYU Ultracomputer Approach, Proc. 12th ISCA, pp. 124-131 (1985).
22) Goodman, J. R. et al.: The Wisconsin Multicube: A New Large Scale Cache-Coherent Multiprocessor, Proc. 15th ISCA, pp. 422-431 (1988).
23) Lenoski, D. et al.: Design of Scalable Shared-Memory Multiprocessors: The DASH Approach, Proc. IEEE Compcon 90, pp. 62-67 (1990).

(平成2年8月24日受付)

浦城 恒雄（正会員）

1936年生．1959年東京大学理学部物理学科卒業．同年(株)日立製作所入社．以来同社戸塚工場および神奈川工場において，大型コンピュータからミニコンピュータまでの各種コンピュータの開発やコンピュータシステムの製品計画に従事．現在同社コンピュータ事業部次長．当学会情報規格調査会理事．