ディレクトリ型キャッシュコヒーレンスプロトコルの性能評価

細見岳生[†,☆] 森眞一郎[†]
中島浩[†] 富田眞治[†]

分散共有メモリ型並列計算機において，ネットワークトラフィックの削減およびメモリアクセスレイテンシの削減/隠蔽は必須である．そのため，それらに重大な影響を持つコヒーレンスプロトコルに何を採用するかがシステムを設計するうえで重要な問題となってくる．本研究では，ディレクトリ方式の分散共有メモリ型並列計算機を対象に，Invalidate型，Update型，およびUpdateの改良であるCompetitive型の3つのキャッシュコヒーレンスプロトコルの性能評価を行った．評価には実行駆動型のシミュレータを用い，そのうえで4つのアプリケーションを動作させ，読み出しや書き込みにともなうネットワークトラフィック，レイテンシ等を測定した．その結果，Invalidate型とUpdate型，特にUpdate型は，トラフィック/レイテンシともアプリケーションの性質に大きく依存しその振舞いを変化させるが，Competitive型は，どのアプリケーションでも良好な性能を発揮することが明らかとなった．また，キャッシュからの要求を受けてメモリが発行するメッセージの数が，トラフィック全体に大きな影響を持つことが明らかとなった．

Performance Evaluation of Cache Coherence Protocols Based on a Directory Scheme

TAKEO HOSOMI,[†,☆] SHIN-ICHIRO MORI,[†] HIROSHI NAKASHIMA[†] and SHINJI TOMITA[†]

It is essential for distributed shared memory multiprocessors to lighten network traffic and tolerate memory access latency. In such systems, it becomes important that what cache coherence protocol is more prominent than other protocols. In this paper, we show the performance evaluation results of various cache coherency protocols for directory based distributed shared memory multiprocessors. We have evaluated and compared two major protocols, ***invalidate*** and ***update***, together with an improved update-type protocol, *competitive*. We measure network traffic and memory access latency caused by load and store using an execution driven simulator. As a result, it becomes clear that the competitive protocol stably gives good performance in any applications both on traffic and latency, while the performance of the update protocol sensitively fluctuates depending on the characteristics of applications. And messages which are issued by memory when it receive request messages from cache are influential in network traffic.

1. はじめに

分散共有メモリ型並列計算機において，ネットワークトラフィックの削減およびメモリアクセスレイテンシの削減/隠蔽は必須である．そのため，それらに重大な影響を持つコヒーレンス制御方式およびコヒーレンスプロトコルに何を採用するかがシステムを設計するうえで重要な問題となってくる．

従来は，共有バスを前提としたスヌープ方式[1]が採用されてきた．しかし，共有バスのシステムでは，プロセッサ台数の増加とともに，バスの飽和によるアクセスレイテンシの増加が重大な問題となった．この問題により，数十台以上のプロセッサからなる並列計算機では，相互結合網としてバス以外のネットワークが用いられてきている．また，コヒーレンス制御方式も，放送機構を持たないネットワークで実現可能なディレクトリ方式[2]が採用されている．

ディレクトリ方式では，スヌープ方式同様ネットワークトラフィックの軽減が重要な問題となるほか，隠蔽が困難なロードミス時のレイテンシの削減がとりわけ重要な問題となる．

† 京都大学工学部
Faculty of Engineering, Kyoto University
☆ 現在，NEC C&C研究所
Presently with NEC Corporation C&C Research Laboratories

コヒーレンスプロトコルはInvalidate型とUpdate型に大別される[1]．この両プロトコルの違いは，共有しているデータに対してプロセッサが書き込みを行ったときの処理にある．Invalidate型は，プロセッサ書き込み時に，他プロセッサのキャッシュコピーを無効化することでコヒーレンシを維持する．一方Update型は，プロセッサの書き込んだデータを他プロセッサのキャッシュコピーにも反映することでコヒーレンシを維持する．

従来はInvalidate型が主に用いられ，Update型が用いられることは少なかった．それは，Update型における冗長な更新，つまりプロセッサが必要としなくなったキャッシュコピーに対する更新が，ネットワークトラフィックの増加を招くことに原因がある．しかし，Update型では共有データが他のプロセッサにより更新されてもそのブロックのコピーがキャッシュにとどまるため，Invalidate型と比べてキャッシュのミスヒット率を低く押えることができる．したがって，キャッシュミスのペナルティが大きいディレクトリ方式ではUpdate型の方が有利になる可能性がある．

この両プロトコルに関する定量的な性能評価は，Invalidate型に関するものが多く[3),4)]，Update型についてはスヌープ方式に関するものはあるものの[1),5)]，ディレクトリ方式については評価されていない．また，Update型を改良し冗長な更新を減少させるCompetitive型[6)]も提案されているが，スヌープ方式では有効に機能しないという評価結果が文献7)で報告されている．また，ディレクトリ方式については未評価である．

そこで本研究は，ディレクトリ方式を用いた分散共有メモリ型並列計算機を対象に，Invalidate/Update/Competitiveの3つのコヒーレンスプロトコルの比較評価を行った．評価には実行駆動型のシミュレータを用い，その上で4つのアプリケーションを動作させ，読み出しや書き込みにともなうネットワークトラフィック，レイテンシ等を測定した．その結果，Invalidate型とUpdate型，特にUpdate型は，アプリケーションの性質に大きく依存しその振舞いを変化させるが，Competitive型はアプリケーションの性質によらず安定的に良好な性能を発揮することが明らかになった．

以降，2章では，評価対象の3つのコヒーレンスプロトコルについて説明する．3章ではシミュレーション対象の計算機モデル，ワークロードについて説明する．4章では，ネットワークトラフィック，レイテンシそれぞれの評価結果を示す．最後に5章でまとめ，今後の課題について述べる．

2. コヒーレンスプロトコル

本章では，評価対象である以下の3種のコヒーレンスプロトコルをについて述べる．

(1) Invalidate (Illinois方式[8)])
(2) Update (Firefly方式[9)])
(3) Compeitive (Firefly方式を改良)

以降プロセッサがロード/ストアを行ったときの各プロトコルの動作を概説する．

2.1 ロード時の動作

3プロトコルともプロセッサがロードを行ったときの動作は同一である．キャッシュにヒットした場合キャッシュからデータを読み出し処理は完了する．

キャッシュにミスした場合の動作を**図1**に示す．まずメモリに対して読み出し要求が発行される (Read Req. メッセージ)．読み出し要求を受けたメモリは最新のデータがメモリに存在するかどうかをチェックし，存在すればデータをリプライする (Data メッセージ)．存在しなければ，最新のデータを保持するキャッシュに対して書き戻し要求を発行し (Write Back Req. メッセージ)，キャッシュにデータの書き戻しを行わせる (Write Back メッセージ)．そして，データをリプライする (Data メッセージ)．

今，プロセッサのロードに対して，Read Req. メッセージが発行される割合，すなわちロード時のキャッシュミスヒット率を**読み出し要求率**と定義する．また，メモリが読み出し要求を受けたときにキャッシュに対して書き戻し要求を発行しなければならない割合を**書き戻し要求率**と定義する．

以上から，プロセッサのロードに起因するメッセージの数およびレイテンシは，以下の2項目に大きく依存する．

(1) 読み出し要求率

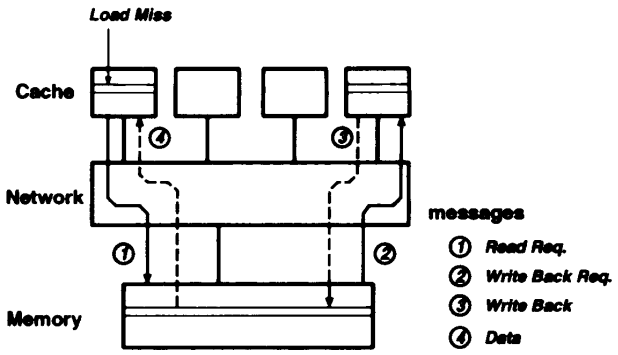

図1 ロードミス時のコヒーレンスプロトコルの動作
Fig. 1 Behavior of cache coherence protocols on load miss.

(2) 書き戻し要求率

このうち，書き戻し要求率はディレクトリ方式固有の項目である．

2.2 ストア時の動作

私有しているブロック，すなわち他のプロセッサがコピーを保持していないブロック，に対してプロセッサがストアを行った場合の処理も3プロトコル共通しており，キャッシュにデータを書き込むことで処理は完了する．3プロトコルで動作を異にするのは，キャッシュミス時，および共有しているブロックにヒットした場合である．

図2 に共有しているブロックにヒットした場合の各プロトコルの動作を示す．3プロトコルともメモリに対して書き込み要求を発行する（Write Req. メッセージ）．Update/Competitive 型の場合は書き込んだデータも同時に送られる．ただし，Update/Competitive 型については，一般にキャッシュに数エントリのライトバッファが設けられ，同一ブロックに対するストアのマージが行われる．そのため，書き込み要求はプロセッサのストアごとに発行されるのではなく，ライトバッファでマージされた複数のストアに対して1つ発行される．書き込み要求を受けたメモリは，コピーを保持するすべてのキャッシュに対して，Invalidate 型の場合は無効化要求を発行し（Invalidate メッセージ），Update/Competitive 型の場合は更新要求を発行する（Update メッセージ）．上記メッセージを受けたキャッシュは，Invalidate 型ではコピーは無効化され，Update 型ではコピーの更新が行われる．また Competitive 型では，ある条件に基づきコピーの更新を行うか無効化するかが決定される．この条件については次節で詳しく述べる．メモリはキャッシュコピーに対する処理が終了するのを待ち（Ack メッセージ），全コピーに対する処理が完了すると，キャッシュに対して処理完了を通知する（Write Ack メッセージ）．

キャッシュミスした場合の動作は共有ブロックに対して書き込みを行ったときの動作とほぼ同じであり，処理の最後にメモリからキャッシュに対してブロックデータがリプライされる．

今，プロセッサのストアに対して書き込み要求が発行される割合を**書き込み要求率**と定義する．また，書き込み要求を受けたメモリが無効化/更新要求を伝えるキャッシュコピーの平均数を**平均ライト分配数**と定義する．

以上から，プロセッサのストアに起因するメッセージの数およびレイテンシは，以下の2項目に大きく依存する．

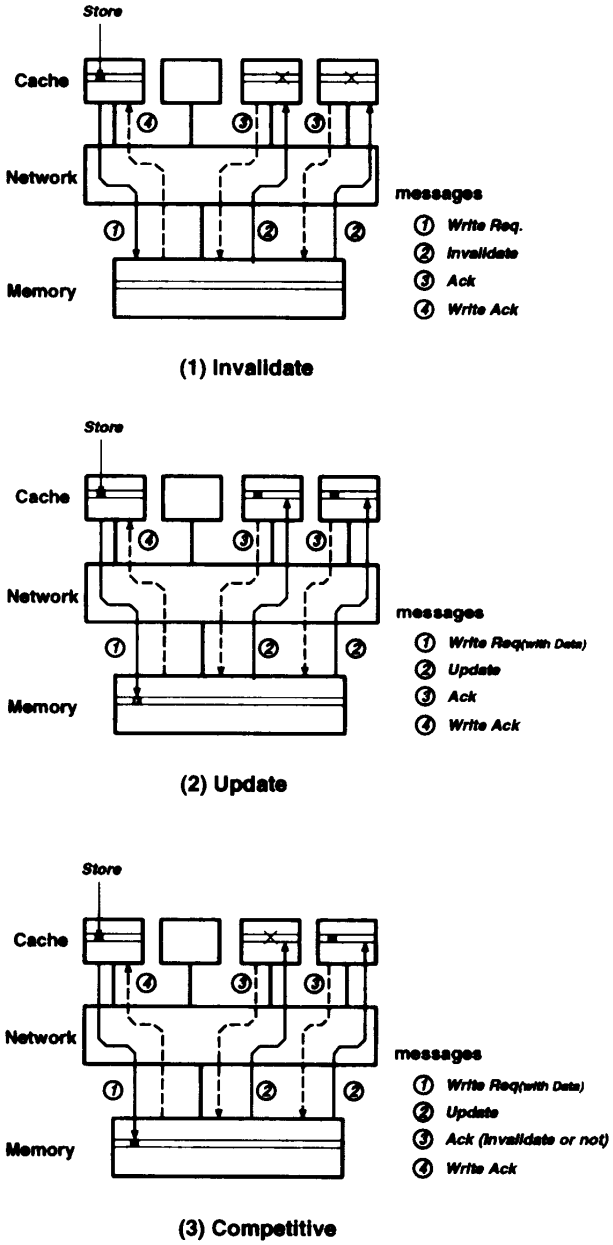

図2 共有データに対するストア時のコヒーレンスプロトコルの動作

Fig. 2 Behavior of cache coherence protocols on storing to shared data.

(1) 書き込み要求率

(2) 平均ライト分配数

このうち，平均ライト分配数はディレクトリ方式固有の項目である．

2.3 Competitive 型の動作

Competitive 型は，冗長な更新を削減するために Update 型を改良したものであり，ブロックに対する自プロセッサのアクセス頻度と他プロセッサの更新頻度から，キャッシュコピーを無効化するか更新するかを決定する．

今回の評価では無効化の基準として，プロセッサが最後にアクセスしてから受け取ったUpdateメッセージの数を採用し，この数が一定値（更新回数の上限値）以上になると無効化を行うことにした．具体的にはキャッシュブロックごとに数ビットのカウンタを設け，以下のような制御を行う．

- ラインアロケートしたときのカウンタの初期値は0とする．
- キャッシュを所有するプロセッサからのアクセスが発生するとカウンタを0にする．
- Updateメッセージを受けるとカウンタに1を加える．このとき，カウンタの値が更新回数の上限値未満であれば更新を行い，上限値以上であれば無効化を行う．

また，Ackメッセージに無効化したか否かの情報を付加し，メモリはその情報をもとにディレクトリを更新する．

3. 評価の方法

本章では，評価に用いたシミュレータおよびワークロードについて述べる．

3.1 シミュレータ概要

シミュレーションには実行駆動型のシミュレータを用い，実際にプロセッサ部で命令レベルのシミュレーションを行った．また，メモリオーダリングモデルはWeak[10]とした．

評価の対象とする計算機のモデルを**図3**に示す．以下にこの計算機モデルの構成を示す．また，各部でのレイテンシを**表1**にまとめる．

[プロセッサ] 全命令1クロックで動作．メモリからの命令フェッチはシミュレートせず，キャッシングも行わない．

[キャッシュ] 1階層のみであり，構成はダイレクトマップ，サイズは1Mバイト，ブロックサイズは32バイト．Update/Competitive型には，同一ラインに対する書き込みのマージを行うライト・バッファを2エントリ用意．

[メモリ] フルマップのディレクトリをブロック（32バイト）単位に保持．

[ネットワーク] 双方向（リンク共有）のトーラスネットワークで，フローコントロール方式はstore-and-forward．

3.2 ワークロード

本研究では，以下に示す4つのアプリケーションプログラムをワークロードとして用い，プロセッサ数は32でシミュレーションを行った．

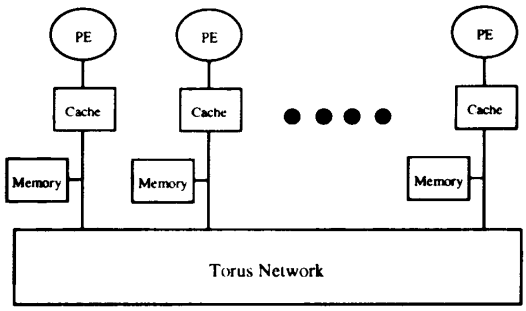

図3 シミュレーション対象の計算機モデル
Fig. 3 Multiprocessor Model for simulation.

表1 シミュレーションのパラメータ
Table 1 Parameters for simulator.

プロセッサ	1命令の実行	1 (cycle)
キャッシュ	ラインアクセス	5 (cycle)
	ワードアクセス	1 (cycle)
	タグアクセス	1 (cycle)
ネットワーク	ラインデータなし (1hop)	8 (cycle)
	ラインデータ付き (1hop)	24 (cycle)
メモリ	ラインアクセス	14 (cycle)
	ワードアクセス	10 (cycle)
	ディレクトリアクセス	10 (cycle)

- WATER — 液体状態の水分子の挙動をニュートン方程式を用いてシミュレーションするプログラム．シミュレーションは125分子，2タイムステップ行い，2タイムステップ目の実行のデータを収集した．
- Barnes-Hut — Barnes-Hut法を用いて銀河の動きをシミュレーションするプログラム．シミュレーションは1024分子，2タイムステップ行い，2タイムステップ目の実行のデータを収集した．
- MP3D — モンテカルロ法を用いた流体力学シミュレーションプログラム．シミュレーションは16384分子，平らな板が存在する $16 \times 16 \times 8$ の空間で，2タイムステップ行い，2タイムステップ目の実行のデータを収集した．
- SOR — Even-Odd法による差分方程式の求解プログラム．シミュレーションは 512×512 のグリッドで，2タイムステップ行い，2タイムステップ目の実行のデータを収集した．

これら4種類のワークロードのうち，WATER, Barnes-Hut, MP3D, はSPLASH[11]に属するものである．SPLASHは，共有メモリ型並列計算機を評価するために様々な科学技術計算の分野から選ばれた実用的なアプリケーション群であり，多くのシステムの評価に用いられている．もう1つのワークロードSORもSPLASHに属するプログラムOCEANで用いられ

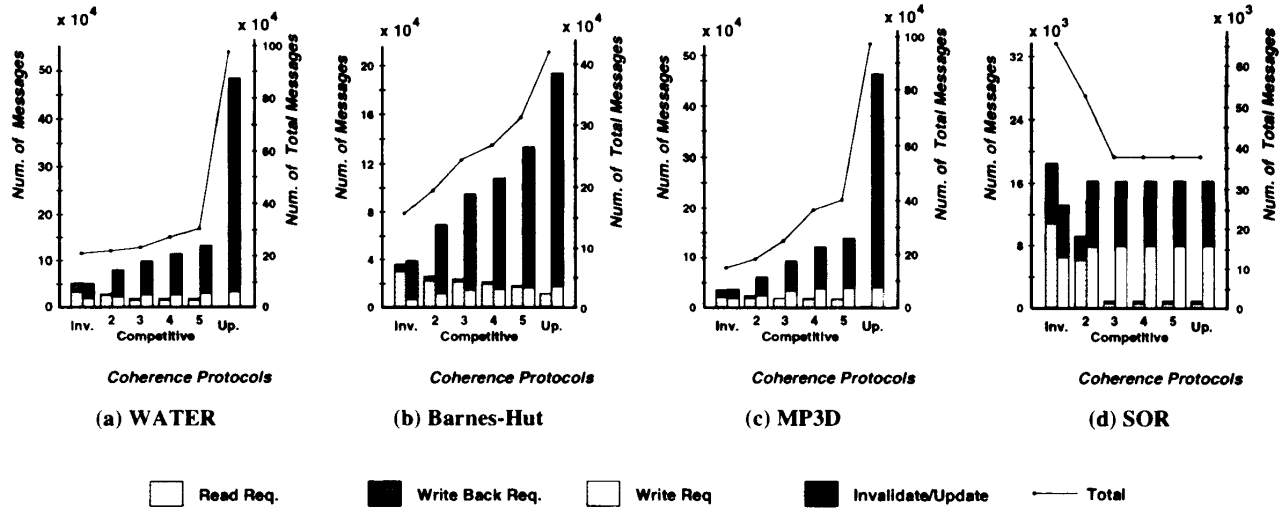

図4 ネットワークトラフィック
Fig. 4 Network Traffics.

ているアルゴリズムである.

4. シミュレーション結果

前述の4つのワークロードを用いて, Invalidate/Update/Competitiveの各プロトコルの性能に関わる様々なデータを, シミュレーションによって収集した. なお, Competitive型については, 更新回数の上限値を2～5の範囲で変化させ, それぞれについてのデータを収集した. 本章で示す図表中の2～5の値は, この上限値を表している.

4.1 ネットワークトラフィック

図4に各ワークロードごとのネットワークトラフィックを示す. この図に示された総メッセージ数から, SORではInvalidateプロトコルのトラフィックが多く, その他ではUpdateプロトコルが突出していることが分かる. すなわちネットワークトラフィックの観点からは, SORではUpdate, その他ではInvalidateが有利であることが分かる.

一方, Competitiveプロトコルでは, 両者のうちの有利なものに近い値となっており, ワークロードの性質によらず安定的に良い性能が得られることが明らかになった. すなわち, Invalidateが不利なものについてはプロセッサのロードに起因するメッセージの数が, Updateが不利なものではプロセッサのストアに起因するメッセージの数が, それぞれ大幅に減少するため, どのワークロードに対しても良い結果が得られている.

以下, ロード/ストアそれぞれに起因するメッセージの数について, これらの結果をさらに細かく分析する.

4.1.1 ロードに起因するメッセージの数

図4に, プロセッサのロードに起因するRead Req. メッセージおよびWrite Back Req. メッセージの数を示す. また, 各ワークロードの読み出し要求率および書き戻し要求率を求めた結果を**表2**, **表3**に示す.

表2 読み出し要求率
Table 2 Read request ratio.

	Inv.	Competitive 2	3	4	5	Up.
WATER	0.18	0.13	0.09	0.09	0.09	0.003
Barnes-Hut	0.23	0.17	0.16	0.15	0.13	0.09
MP3D	5.28	5.02	4.87	4.72	4.57	0.61
SOR	0.35	0.18	0.03	0.03	0.03	0.03

表3 書き戻し要求率
Table 3 Write back request ratio.

	Inv.	Competitive 2	3	4	5	Up.
WATER	57.2	15.5	17.0	13.3	9.3	0
Barnes-Hut	19.9	14.1	9.9	8.3	4.6	0.6
MP3D	91.6	30.5	2.4	1.6	0.4	1.1
SOR	69.3	50.4	47.3	47.3	47.3	47.3

これらの図表から, Updateプロトコルでのロードに起因するメッセージの数が, きわめて少ないことが分かる. Invalidateプロトコルを基準にして, Updateプロトコルでは Read Req. と Write Back Req. メッセージの総数がどれだけ削減できたかを見ると, 最大ではWATERの99%, 最小でもBarnes-Hutの68%となった. 以下これらのメッセージ数の削減の要因を, 読み出し要求率および書き戻し要求率の変化で見る. また, その原因となっている両プロトコルの動作の違いも解析する. ただし, 各プロトコルの動作については, ある1つのメモリブロックおよびそのキャ

ッシュコピーに着目して説明する．

Invalidate プロトコルを基準として，Update プロトコルではどれだけ読み出し要求率が削減できたかを見ると，最大では WATER の 98%，最小でも Barnes-Hut の 61%となった．この効果が Read Req. メッセージの削減となって表れている．

Invalidate プロトコルでは，共有データに対するストアが行われると，他キャッシュコピーが無効化される．一方 Update プロトコルでは，その更新が他キャッシュコピーにも反映され無効化されることはない．そのため，共有データに対するストアが発生した直後に，以前キャッシュコピーを保持していた他のプロセッサがロードを行った場合，そのロードは Invalidate プロトコルではキャッシュミスとなるが，Update プロトコルではキャッシュヒットとなる．この差が，読み出し要求率の差の原因である．

また，書き戻し要求率で見ると，Invalidate プロトコルでは，最低の Barnes-Hut でも 20%程度，その他は 50%以上と非常に大きな値をとっている．一方，Update プロトコルでは，SOR 以外のワークロードで数%以下の非常に低い値となっている．この効果が，Write Back Req. メッセージの削減となって表れている．

Invalidate プロトコルでは，共有データに対するストアが行われると，メモリには有効なデータが存在せず，ストアが行われたキャッシュにのみ有効なデータが存在する状態となる．一方，Update プロトコルでは，共有データに対する更新はメモリにも反映されるのでメモリにも有効なデータが存在する．そのため，共有データに対するストアが発生した直後にキャッシュコピーを保持していないプロセッサがロードを行った場合，読み出し要求を受けたメモリは，Invalidate プロトコルではメモリに有効なデータが存在せず有効なデータを持つキャッシュに書き戻しを要求しなければならないが，Update プロトコルではその必要がない．この差が，書き戻し要求率の差の原因である．

以上より，Update プロトコルは，単にキャッシュのヒット率を高めるだけではなく，キャッシュミス時の一連のコヒーレンス制御で必要なメッセージ数を削減する効果もあることが明らかになった．

一方，Competitive プロトコルでも，図 4 から，Invalidate プロトコルと比べ，ロードに起因するメッセージの数は大きく削減されていることが分かる．表 2，表 3 より，Competitive プロトコルでも，Update プロトコルと同様に，読み出し要求率と書き戻し要求率を削減することにより，それぞれ Read Req. および Write Back Req. メッセージを削減していることが分かる．

Competitive プロトコルでは，共有データに対するストアが行われると，あるキャッシュコピーは Update プロトコル同様更新され，あるキャッシュコピーは Invalidate プロトコル同様無効化される．更新回数の上限値が低いほど無効化される割合が高く，上限値が高いほど更新される割合が高い．そのため，共有データに対するストアが発生した直後に，以前キャッシュコピーを保持していた他のプロセッサがロードを行った場合，そのロードは，当該キャッシュコピーが更新されていればキャッシュヒットとなるが，無効化されていればキャッシュミスとなる．この Competitive プロトコルの Invaldiate プロトコルと Update プロトコルの中間的な動作により，読み出し要求率に関して Competitive プロトコルは Invalidate プロトコルと Update プロトコルの中間的な値をとる．また，更新回数の上限値が低いほど Invalidate プロトコルに近い値をとる．

また，Competitive プロトコルでは，あるプロセッサが共有データに対してストアを行った結果，Invalidate プロトコル同様他のすべてのキャッシュコピーが無効化される状況が発生しうる．このとき，データの共有状態は解消され，ストアを行ったプロセッサのみがデータを私有する状態となる．そのため，それ以降のストアはメモリには反映されずメモリに有効なデータが存在しなくなる．したがって，Competitive プロトコルの書き戻し要求率が，Update プロトコルよりも高くなり，Invalidate プロトコルよりも低くなる．また，更新回数の上限値が低いほど書き戻し要求率が高くなっているのは，キャッシュコピーが無効化される可能性が高く，共有状態が解消される可能性が高いからである．

また，興味深いことに MP3D ではメッセージの削減がほぼ書き戻し要求率を削減することによってのみ得られている．読み出し要求率は 5 ～ 19% のみの削減にとどまっているのに対して，書き戻し要求率の削減は 67 ～ 100% となっている．

以上より，Update や Competitive プロトコルを用いることによる，プロセッサのロードに起因するメッセージ数の削減効果は，以下の 2 点から得られていることが明らかとなった．

(a) 読み出し要求率の減少

(b) 書き戻し要求率の減少

また，Competitive プロトコルでは，(a) の効果が少ないワークロードでも，(b) の効果が大きく現れ，メッ

セージ数が削減されることが明らかとなった．すなわち，スヌープ方式でのトラフィックの削減効果が，(a)の効果のみによるものであったのに対し，ディレクトリ方式では（b）の効果がさらに加わる．したがって，ディレクトリ方式ではUpdateやCompetitiveプロトコルを用いることによるメッセージ数の削減効果が増すことが分かる．

また，Competitiveプロトコルについては，SOR以外のワークロードでUpdateプロトコルよりもその削減率が劣っているが，これは後述するライトメッセージの大幅な削減で補われる．したがってこれらのワークロードでも，一定の削減率が得られたことは大きな意味を持つ．

4.1.2 ストアに起因するメッセージの数

図4に，プロセッサのストアに起因するWrite Req.およびInvalidate/Updateメッセージの数を示す．また，各ワークロードの書き込み要求率および平均ライト分配数を求めた結果を**表4**，**表5**に示す．

WATER, Barnes-Hut, MP3Dでは，図4より，プロセッサのストアに起因するWrite Req.およびInvalidate/Updateメッセージの数が，Updateプロトコルではプロトコルに比べて大幅に増加していることが分かる．Invalidateプロトコルを基準とすると，両メッセージを合わせた数の増加は，MP3Dの場合が最高で12.5倍，最低のBarnes-Hutでも5.0倍となっている．これは，この3つのワークロードのメモリアクセスが，共有データを多くのプロセッサが交互にアクセスし，更新を行う，というようなパターンを持つことによる．すなわち，Updateプロトコルで問題となる冗長な更新が，多く発生するワークロードであることによる．

一方SORでは，ストアに起因するメッセージの数の増加が1.2倍程度にとどまっている．これは，SORのメモリアクセスが，主に2プロセッサ間でデータを共有し，一方が更新したデータを他方がアクセスする，といったパターンを持つことによる．すなわち，冗長な更新があまり発生せずUpdateプロトコルに適したアプリケーションであることによる．

以下，まずWATER, Barnes-Hut, MP3Dの冗長な更新が多く発生するワークロードについて，その後冗長な更新があまり発生しないSORについて解析した結果を述べる．

(1) 冗長な更新が多く発生する場合

WATER, Barnes-Hut, MP3Dの3つのワークロードをUpdateプロトコルで動作させた場合に，プロセッサのストアに起因するメッセージの数が増加した原因を，書き込み要求率および平均ライト分配数の変化で見る．また，その原因となっている両プロトコルの動作の違いも解析する．ただし，各プロトコルの動作については，ある1つのメモリブロックおよびそのキャッシュコピーに着目して説明する．

Invalidateプロトコルを基準として，Updateプロトコルではどれだけ書き込み要求率が増加しているかを見ると，最大ではBarnes-Hutの2.7倍，最小でもWATERの1.8倍となった．これがWrite Req.メッセージ数の増加となって表れている．

Invalidateプロトコルでは共有状態にあるデータも，ストアが一度行われると他キャッシュのコピーが無効化されデータを私有している状態となる．再びこのデータが共有状態となるのは，他プロセッサがロードを行ってからとなる．一方Updateプロトコルでは，ストアを行っても更新が他のキャッシュコピーに反映され，共有状態が解消されることはない．そのため，Invalidateプロトコルでは私有状態で発生したストアでも，Updateプロトコルでは共有状態で発生したストアとなりうる．この差が，書き込み要求率の差の原因である．

また，平均ライト分配数を見ると，Updateプロトコルではすべて10を超えている．すなわち，メモリが，受けたWrite Req.メッセージの数にして10倍以上のUpdateメッセージを発行している．一方Invalidateプロトコルでは，最大のBarnes-Hutでも4.8にとどまっており，他のワークロードでは2以下となっている．

Invalidateプロトコルでは，ストアが行われる度に共有状態が解消される．そのため，キャッシュコピー

表4 書き込み要求率
Table 4 Write request ratio.

	Inv.	Competitive 2	3	4	5	Up.
WATER	0.23	0.27	0.31	0.34	0.37	0.41
Barnes-Hut	0.09	0.16	0.20	0.21	0.23	0.24
MP3D	9.1	12.5	17.1	20.0	20.4	20.6
SOR	1.26	1.55	1.55	1.55	1.55	1.55

表5 平均ライト分配数
Table 5 Average write distribution.

	Inv.	Competitive 2	3	4	5	Up.
WATER	1.7	2.5	2.8	3.2	3.4	13.2
Barnes-Hut	4.8	4.8	5.4	5.4	6.8	10.0
MP3D	1.0	1.5	1.8	2.1	2.5	10.3
SOR	1.0	1.0	1.0	1.0	1.0	1.0

を持つプロセッサは，以前共有状態が解消されてから再び共有状態となりあるプロセッサがストアを行うまでに，アクセスを行ったプロセッサに限られる．一方，Updateプロトコルではストアを行っても共有状態が解消されることはない．そのため，キャッシュコピーを持つプロセッサは，プログラムが起動されてからその時点までにそのデータをアクセスしたことがあるプロセッサほとんどすべてとなる．このキャッシュコピーを持つプロセッサの違いが，平均ライト分配数の差の原因である．

以上をまとめると，Updateプロトコルでのプロセッサのストアに起因するメッセージの増加の原因は

(a) 書き込み要求率の増加

(b) 平均ライト分配数の増加

の2点であるが，特に(b)の影響が大きい．スヌープ方式では(a)のみがメッセージ増加の割合を決定するのに対し，ディレクトリ方式では(b)の強い影響が加わり，Updateプロトコルでのメッセージ数を大幅に増加させていることが明らかになった．

一方，Competitiveプロトコルでは，Updateプロトコルに比べてストアに起因するメッセージの数が大きく削減され，その効果は更新回数の上限値を小さく設定するほど大きい．冗長な更新を削減するためにUpdateプロトコルに対して加えた改良が，有効に機能していることが分かる．

この効果の要因をさらに詳しく見ると，どの3つのワークロードでも，書き込み要求率はInvalidateとUpdateの中間的な値をとっており，更新回数の上限値が小さいほどInvalidateプロトコルに近く，更新回数の上限値が大きい場合はUpdateプロトコルに近い値になっている．一方，Updateプロトコルと比較した平均ライト分配数の減少はどのワークロードでも大きく，WATERの場合で74～81%となっている．

Competitiveプロトコルでは，前節で述べたようにUpdateプロトコルと異なり共有状態が解消されることがあるため，書き込み要求率が削減されている．しかし，Competitiveプロトコルの主な作用はキャッシュコピーの無効化にある．そのため，キャッシュコピーを保持するプロセッサ数の減少の効果の方がより大きく表れ，平均ライト分配数の削減効果が大きく出ている．

以上より，Competitiveプロトコルの効果の多くの部分は，Updateプロトコルにおけるストアに起因するメッセージ数の増加に関する前述の2つの原因のうち，(b)の平均ライト分配数を削減することによって得られるものであることが分かる．スヌープ方式では(a)のみを抑止するためCompetitiveプロトコルの効果は小さいが，ディレクトリ方式では(b)をも抑止するため大きな効果が得られることが明らかになった．

(2) 冗長な更新があまり発生しない場合

一方，SORでのストアに起因するメッセージ，すなわちWrite Req.メッセージ，Invalidateメッセージ，およびUpdateメッセージ，の数の増加は，Invalidateプロトコルを基準として，Competitive/Update両プロトコルとも1.2倍にとどまっている．冗長な更新が多く発生するワークロードでストアに起因するメッセージ数の増加の原因であった平均ライト分配数が，Updateプロトコルでも変化していないためである．

4.2 メモリアクセスレイテンシ

ロードやストアに起因するメモリアクセスレイテンシの変化を評価するため，それぞれに起因しプロセッサがストールした時間を測定した．**図5**に，各ワークロードのプロセッサストール時間および実行時間を示す．プロセッサストール時間については全プロセッサの平均をとった結果を示した．

実行時間を見ると，ネットワークトラフィック同様に，冗長な更新があまり発生しないSORではUpdateプロトコルが，冗長な更新が多く発生する他のアプリケーションではInvalidateプロトコルの方が良い性能を示している．一方．Competitiveプロトコルは，更新回数の上限値を2，3程度におさえた場合，どのワークロードでも，安定して良い性能を示しており，Invalidateプロトコルを基準にして，2～15%の性能向上が得られている．

この性能向上は，メモリアクセスレイテンシを短縮することによって得られたものである．以降，メモリアクセスレイテンシが，各プロトコルでどのように変化しているかを解析する．

MP3D以外のワークロードでは，ロードに起因するストール時間は，Invalidate，Update，Competitiveと，読み出し要求率と書き戻し要求率の減少に合わせて，軽減されている．MP3Dでは，Competitiveプロトコルで更新回数の上限値を4，5に設定した場合，キャッシュミスの削減とは逆にストール時間は増加している．MP3Dが他のワークロードと振舞いを異にする原因は，ネットワークに対する負荷が異なることにある．これは，読み出し要求率や書き込み要求率が他のワークロードと比較して非常に高い値をとっていることから確認できる．すなわち，MP3Dでは，ネットワークでのメッセージの輻輳が頻繁に発生するため，1つのキャッシュミスを処理するのに要する時間が，ネットワークトラフィックの増加に従い大幅に増加したものと考えられる．

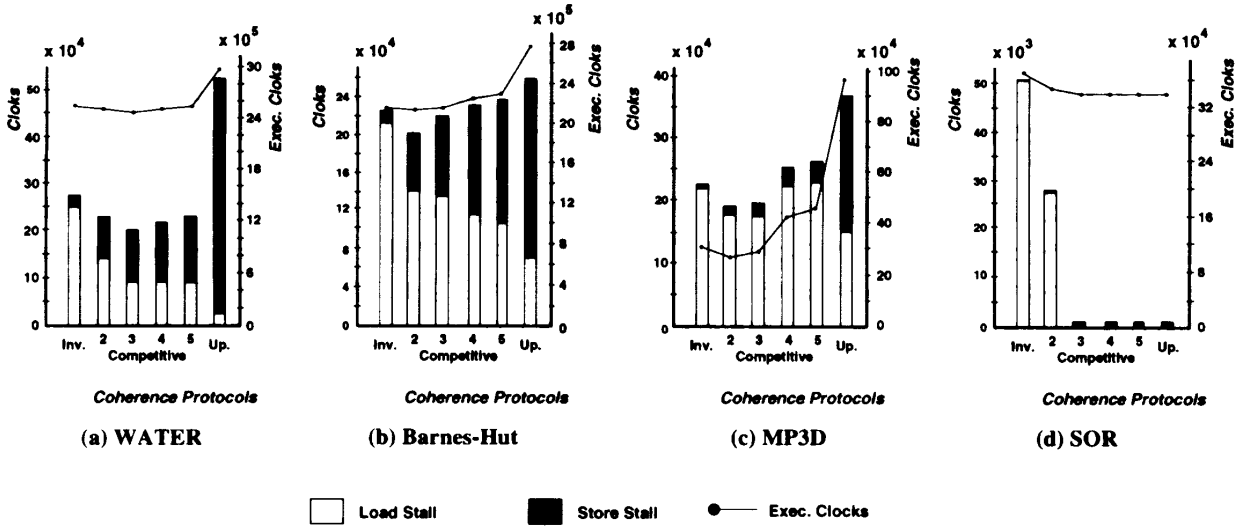

図5 メモリアクセスレイテンシと実行時間
Fig. 5 memory access latency and execution time.

一方，ストアに起因するストール時間を見ると，Update，Competitive 両プロトコルとも，SOR 以外のワークロードで，Invalidate プロトコルと比較して，書き込み要求率，平均ライト分配数の増加に合わせて，大幅に増大している．これは，ネットワークトラフィックの増加によるメッセージの輻輳に原因があるほか，平均ライト分配数が大幅に増加したことにより，メモリで多くのメッセージを生成する必要が生じ，そのための処理に要する時間が増加することにも原因がある．

5. まとめおよび今後の課題

ディレクトリ方式の分散共有メモリ型並列計算機を対象に，Invalidate/Update/Competitive の 3 つのコヒーレンスプロトコルの性能評価を行った．評価には実行駆動型のシミュレータを用い，その上で 4 つのアプリケーションを動作させ，読み出しや書き込みにともなうネットワークトラフィック，レイテンシを測定した．

その結果，ネットワークトラフィックについては，キャッシュからの要求を受けてメモリが発行するメッセージの数が，読み出しでも書き込みでも大きな影響を持つことが明らかとなった．そのため，スヌープ方式以上にディレクトリ方式では，Update プロトコルと Invalidate プロトコルの差が生じ，Update プロトコルでネットワークトラフィックが大幅に増加する．また，スヌープ方式で有効に機能しなかった Competitive プロトコルが，ディレクトリ方式では有効に機能し，Invalidate プロトコルと比べネットワークトラフィック全体の増加をおさえながら，読み出しに起因するメッセージの数を大きく削減することが明らかとなった．

メモリアクセスレイテンシについて，Update プロトコルはアプリケーションの性質によりその振舞いを大きく変化させることが明らかとなった．また，Competitive プロトコルがアクセスレイテンシ短縮に非常に有効であり，更新回数の上限値を 2，3 に設定した場合，どのような性質を持ったアプリケーションでも良好な性能を発揮することが明らかとなった．

今後は，プリフェッチ等のプロセッサストール時間を削減するための手法と組み合わせたときに，上記の結果がどのように変化するかや，ネットワークの性能が与える影響を評価していく．また，より多くのワークロードを用いてさらに評価を進める．

謝辞 日頃ご討論いただいた富田研究室の諸氏に感謝いたします．また，貴重なご意見をいただいたコンピュータ・システム研究部の諸氏に感謝いたします．なお，本研究は，一部文部省科学研究費（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) Archibald, J. and Baer, J.-L.: Cache Coherence Protocols: Evaluation Using Multiprocessor Simulation Model, *ACM Trans. Computer Systems*, Vol.4, No.4, pp.273–298 (1986).
2) Censier, L.M. and Feautrier, P.: A New Solution to Coherence Problems in Multicache Systems, *IEEE Trans. Computers*, Vol.27, No.12, pp.1112–1118 (1978).
3) Gupta, A. and Weber, W.-D.: Cache Invalidation Patterns in Shared-Memory Multipro-

cessors, *IEEE Trans. Computers*, Vol.41, No.7, pp. 794–810 (1992).
4) Agarwal, A., Simoni, R., Hennessy, J. and Horowitz, M.: An Evaluation of Directory Schemes for Cache Coherence, *Proc. 15th Int'l. Symp. Computer Architecture*, Vol.16, No.2, pp.280–289 (1988).
5) Eggers, S.J. and Katz, R.H.: A Characterization of Sharing in Parallel Programs and its Aplication to Coherency Protocol Evaluation, *Proc. 15th Int'l. Symp. Computer Architecture*, Vol.16, No.2, pp.373–382 (1988).
6) Karlin, A.R., Manasse, M.S., Rudolph, L. and Sleator, D.D.: Competitive Snoopy Caching, *Proc. 27th Annual Symp. Foundations of Computer Science*, pp.244–254 (1986).
7) Eggers, S.J. and Katz, R.H.: Evaluating the Performance of Four Snooping Cache Coherency Protocols, *Proc. 16th Int'l. Symp. Computer Architecture*, Vol.17, No.3, pp.2–15 (1989).
8) Papamarcos, M.S. and Patel, J.H.: A Low Overhead Coherence Solution for Multiprocessors with Private Cache Memories, *Proc. 11th Int'l Symp. Computer Architecture*, Vol.12, No.3, pp.348–354 (1984).
9) Thacker C.P., Stewart, L.C., Edwin, H. and Satterthwaite, J.: Firefly: A Multiprocessor Workstation, *IEEE Trans. Computers*, Vol.37, No.8, pp.909–920 (1984).
10) Adve, S.V. and Hill, M.D.: Weak Ordering–A New Definition, *Proc. 17th Int'l. Symp. Computer Architecture*, Vol.18, No.2, pp.2–14 (1990).
11) Singh, J.P., Weber, W.-D. and Gupta, A.: SPLASH: Stanford Parallel Applications for Shared-Memory, Technical Report CSL-TR-91-469, Stanford Univ. (1991).

(平成6年 9 月 16 日受付)
(平成7年 11 月 2 日採録)

細見 岳生（正会員）

昭和44年生．平成4年京都大学工学部情報工学科卒業．平成6年京都大学大学院工学研究科情報工学専攻修士課程修了．同年NEC入社．並列計算機アーキテクチャに関する研究に従事．

森 眞一郎（正会員）

1963年生．1987年熊本大学工学部電子工学科卒業．1989年九州大学大学院総合理工学研究科情報システム学専攻修士課程修了．1992年同大学院博士後期課程単位取得退学．同年京都大学工学部情報工学教室助手．1995年同大学助教授，現在に至る．工学博士．並列処理，計算機アーキテクチャの研究に従事．IEEE-CS，ACM 各会員．

中島 浩（正会員）

昭和31年生．昭和56年京都大学大学院工学研究科情報工学専攻修士課程修了．同年三菱電機（株）入社．推論マシンの研究開発に従事．平成4年より京都大学工学部助教授．並列計算機のアーキテクチャ，プログラミング言語の実装方式に関する研究に従事．工学博士．昭和63年元岡賞，平成5年坂井記念特別賞受賞．

富田 眞治（正会員）

1945年生．1968年京都大学工学部電子工学科卒業．1973年同大学院博士課程修了．工学博士．同年京都大学工学部情報工学教室助手．1978年同助教授．1986年九州大学大学院総合理工学研究科教授，1991年京都大学工学部情報工学科教授，現在に至る．本会理事．計算機アーキテクチャ，並列処理システム等に興味を持つ．著書「並列計算機構成論」「計算機システム工学」「並列処理マシン」等．電子情報通信学会，IEEE，ACM 各会員．