Self-Cleanup Cache を採用した NCC-NUMA アーキテクチャの評価

森 眞一郎,　福島 直人,　五島 正裕,　中島 浩,　富田 眞治

京都大学大学院 工学研究科

本稿では, Self-Cleanup Cache を備えた NCC-NUMA アーキテクチャの性能を, 無効化型のキャッシュ一貫性制御方式を採用した場合の実行速度, ネットワーク・トラフィックの観点からシミュレーションにより評価した.

その結果, ソフトウェアによる簡単な最適化を行うことで, NCC-NUMA が CC-NUMA に比肩する性能をだせること, また場合によっては CC-NUMA よりも良い性能が得られることを確認した. さらに, アプリケーション・プログラムの十分な静的解析ができた場合, CC-NUMA に対する NCC-NUMA の相対性能はネットワークのレイテンシに比例して大きくなることを確認した.

An Evaluation of an NCC-NUMA Architecture configured with Self-Cleanup Cache

Shin-ichiro MORI,　Naoto FUKUSHIMA,　Masahiro GOSHIMA,
Hiroshi NAKASHIMA,　Shinji TOMITA

Graduate School of Engineering, Kyoto University

This paper reports the result of an evaluation of an NCC-NUMA Architecture configured with Self-Cleanup Cache. In this evaluation, a self-invalidation based cache coherence strategy is assumed. Some simulation works are made with “WATER” code from SPLASH code suits to compare the execution time and network traffic of several implementation models of the NCC-NUMA architecture.

As a result of these simulation works, we could confirm that the NCC-NUMA architecture shows the comparable performance to the CC-NUMA architecture if some simple software optimizations are applied to the application code. Furthermore, we also confirmed that the NCC-NUMA architecture shows better performance than the CC-NUMA and the relative performance of NCC-NUMA to CC-NUMA architectures is proportional to the network latency if complete analysis of the code is possible.

1. はじめに

共有メモリベースの並列処理環境では, キャッシュ・システムの性能がシステム全体性能の要となっている.

今, NonUniform Memory Access(NUMA) 型の比較的大規模な並列計算機を考えると, キャッシュシステムの性能は, キャッシュの一貫性制御をいかに効率的に実現するかにかかっているといっても過言ではない.

このような NUMA アーキテクチャは, キャッシュの一貫性制御をコヒーレンスディレクトリ等のハードウェアを用いて積極的に支援する Cache Coherent-NUMA(CC-NUMA) アーキテクチャと, 特別なハードウェアを設けずに, ソフトウェアで一貫性制御を行う Non Cache Coherent-NUMA(NCC-NUMA) アーキテクチャに分られる.

従来, CC-NUMA 型の並列計算機ハードウェアに関しては, 我々を含め多くの研究者により研究・開発がなされてきたが, NCC-NUMA に関しては最近になって一部の研究者がようやく本格的に注目しはじめたという状況である. またこれらの研究も, その多くが, ライトスルー型のキャッシュを前提としたものである.

そこで我々は, ライトバック型キャッシュを用いた NCC-NUMA の潜在的な能力を検証するため, 一貫性制御に関してごく簡単なハードウェア支援のみを行う NCC-NUMA アーキテクチャを仮定し, それが従来の CC-NUMA に対してどれだけの性能を発揮できるかを, 1つの並列アプリケーション・プログラムを用いたシミュレーションにより評価した.

以下, 2章では研究の背景として, NCC-NUMA アーキテクチャでのソフトウェアによる一貫性制御方式と, それを WB キャッシュで可能にする Self-Cleanup Cache について述べる. 3章では評価に用いた応用プログラムについて簡単に述べた後, シミュレーション対象とした NCC-NUMA アーキテクチャ・モデルとシミュレーション環境について述べる. 4章でシミュレーション結果の考察をおこない, 5章で本論文のまとめを述べる.

2. 研究の背景

2.1 NCC-NUMA アーキテクチャでのキャッシュ・コヒーレンス制御

ハードウェアによる大域的なコヒーレンス制御を行わない NCC-NUMA アーキテクチャでは, ソフトウェアにより何らかの形でキャッシュの一貫性保持を行う必要がある. このような一貫性制御の方式として提案されているものは主に以下の3つである.

アクセス属性方式

初期化時以外に書き込みが起らないコード, データのみのキャッシングを許すことで, インコヒーレントな状態そのものを発生させない方式2).

ディレクトリ・エミュレーション方式

CC-NUMA アーキテクチャにおいてハードウェアで実装していたディレクトリをソフトウェアでエミュレーションする方法.

最も簡単な実装法としては, キャッシングの単位はハードウェア・キャッシュの「ライン」単位で行うが, 一貫性制御のためのディレクトリ情報を「ページ」を単位として管理し, 共有ページを allocate/free する際にディレクトリ情報の更新ならびにページ単位の一貫性制御を行う方法がある. これをベースにして, false sharing を軽減するために同一ページへの複数のプロセッサからの同時書き込みを許すための改良を行ったもの等が種々提案されている11),13). この方式はワークステーション・クラスタにおける共有メモリ環境の実現方式と親和性がよく, 最近多くの研究がなされている.

自己無効化方式

各プロセッサが実行するコード中にキャッシュ操作命令を挿入し, キャッシュ内の古いデータを必要に応じて明示的に無効化することで, 一貫性制御を行う. キャッシュ操作命令は, アプリケーションプログラム中にユーザが明示的に挿入するか, あるいは, コンパイラがコンパイル時に挿入する. 十分な静的解析が出来ない場合等は, 安全策のための無駄な無効化が発生する. 無効化処理の高速化, 効率化のための種々の提案がなされている4)~6)

なお本研究の評価対象システムでは, キャッシュ一貫性制御方式として, OSの実装方式を考慮する必要がなく, かつユーザレベルの最適化が容易な自己無効化方式を採用する.

2.2 Self-Cleanup Cache

ソフトウェアによる自己無効化型のキャッシュ一貫性制御では, コヒーレンスを維持すべきプログラムのある時点（以下, 同期点と呼ぶ.）でメモリには最新のデータが存在することが前提とされている. そのため, WT 型のメモリ更新アルゴリズムとの親和性が良かった.

いま, Write-Back 型のキャッシュにこの方式を適用することを考えると, 同期点ではまず自キャッシュ内にある最新データ (dirty line) のメモリへの書き戻しを行い（これによりメモリに最新のデータがあることを保証する）, その後で無効化処理を行わなければならない.

そのためには, 「Processor が行った Write アクセスのトレース」ならびに「アクセスされなくなった dirty line の速やかなメモリへの書き戻し」が必要である. またそのための Write-Back 処理はネットワーク・トラフィックの集中をさけるため「時間的に分散」している必要がある.

これを実現可能にするのが Self-Cleanup Cache

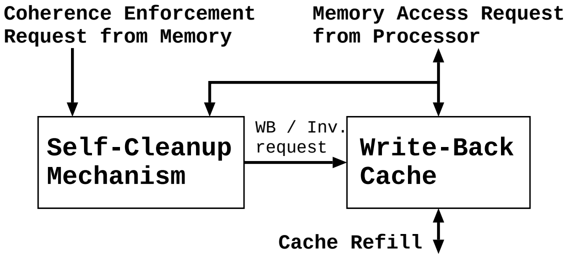


図 1　SCC の構成

(SCC) である．SCC は通常の WB キャッシュ部に自浄制御回路（Self-Cleanup Mechanism:SCM）を付加することで構成する（図 1参照）．

SCM は，キャッシュ内の dirty なラインの管理と，それらの dirty なラインのうち，どのラインを，いつ書き戻すかを制御する回路である．SCM が適切な時期に適切なラインに対する WB 要求を WB 型キャッシュ部に送ることで，上記 3 つの要件を充足可能としている．

これにより SCC は，WB 型キャッシュの特徴であるライト・アクセスのマージ機能を保ちつつ，可能な限りキャッシュおよびメモリをクリーンな状態に保つことで，システム全体としてのメモリ・アクセス・レイテンシの軽減を図ることができる．ある意味で，WB 型と WT 型の中間のメモリ更新アルゴリズムを採用するキャッシュとみなすこともできる．

なお本評価では, SCM の自浄制御方針として LRU 方式を採用し, 書き戻しのタイミングは，LRU エントリのリプレース時とした．

3. 評価の方法

3.1 評価プログラムの概要

ワークロードとしては，SPLASH[14] に属するアプリケーションから WATER を採用した．これは, 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である．シミュレーションは 256 分子で, 2 タイム・ステップ処理を行い，並列実行部分のデータを収集した．分子状態を示すデータをブロック分割し, それを静的にプロセッサに割り当て並列処理を行う．なお以下では, ある分子が割り当てられたプロセッサを, その分子に対する OwnerPE と呼ぶ.

1 ステップは複数のフェーズで構成されている．各プロセッサは担当する分子の分子内の原子の運動を計算し，次に，自分から一定距離内にある他の分子との相互作用を求め，次状態を決定する．

分子の運動に関するデータは 1 分子当たり 約 600Byte 必要であり, 共有メモリアクセスのほとんどが, このデータへのアクセスである．個々の分子状態は, 構造体で定義されておりその構造体メンバへのアクセスは表 1 に示す 3 つのパターンが存在する．

なお, ソフトウェアによるキャッシュ一貫性制御を行う際は, 何らかのハードウェア支援[3],[9] を設けない限り false sharing が発生すると正しい結果が得られない．本評価では, 分子状態を示すデータの基本構成要素をキャッシュ・ラインに整列するために PAD を挿入し, false sharing が起らないようソフトウェア的に対処した．このため, 1 分子当りのデータが 約 30% 増加している．また, true sharing データの一貫性制御を行う同期点にバリア同期命令を挿入している．

表 1　分子状態を表す構造体のアクセス・パターン

データの種類	空間的共有		時間的共有
	参照	定義	
Type I	○	×	×
Type II	×	×	×
Type III	○	○	○

○ :アクセス有り, × :アクセスなし
空間的共有 : OwnerPE 以外の PE からのアクセス状況
時間的共有 : 同一フェーズ内での, プロセッサを跨る定義参照関係の有無

3.2 シミュレーション対象モデル

以下の 4 つのキャッシュ・システム・モデルに対して，2 つのパラメータ, LRU エントリ数 (1,2,4,8,16,32, 自浄制御なし)，ならびに, ネットワーク ディレイ (100 クロック, 10 クロック (1 ホップあたり)) を変化させ，実行時間とその内訳☆（プロセッサ稼働時間，リード待ち時間，ライト待ち時間，同期待ち時間），ならびに総メッセージ数（キャッシュが発行する読み出し要求/書き込み要求数，メモリが発行する WB 要求/無効化要求数）を計測した．

DISC モデル

Directory base のハードウェア制御キャッシュに SCC を付加したモデル．DISC モデルでは，フル・マップのディレクトリをライン単位に保持しており，コヒーレンス・プロトコルとしては Illinois protocol (Dirty,Valid Exclusive,Shared,Invalid の 4 状態) を採用した．なお, DISC モデルでは, 対象プログラムとして オリジナルの WATER コード (orig) と false sharing 対策を行ったコード (noFS) の両方で評価を行った.

SISC モデル

Self Invalidation 型のソフトウエア・コヒーレンス制御と Self-Cleanup Cache を組み合わせたモデル．SISC モデルは，同期点でキャッシュ内の全ての dirty なラインをメモリへ書き戻した後，プライ

☆ 実行時間の内訳は，Read Stall と Write Stall の和が最も大きいプロセッサのデータである．

ベートなデータを除く全データを無効化する．この操作のために，キャッシュ･ライン毎に 1bit の タグを設けており，この タグ は一斉に クリア 可能であると仮定する☆．

SISCopt モデル

SISC モデルにおいて以下の 2 つのソフトウェア最適化を行ったもの.

(1) 同期点での無効化対象から自プロセッサのメモリからキャッシングした共有データを除外し, 必要に応じて明示的に無効化命令を挿入.

(2) 表 1 の **Type I** のデータに関して, **OwnerPE** 以外の **PE** がそのデータを最初に参照するフェーズにおいて, 当該データのコピーを各 **PE** が非共有データとして作成.

ExpSISC モデル

SISC モデルにおいて, プログラム中に明示的に無効化命令を挿入するモデル. SISC モデルが, 同期点で全共有データを無効化するのに対し, ESISC モデルでは, 明示的な無効化命令を発行しない限りキャッシュ内のデータの無効化は行われない. なお, 無向化命令 1 つで 1 キャッシュラインの無向化が可能である.

3.3 シミュレータの概要

シミュレーションには実行駆動型のシミュレータ10) 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☆☆．コンシステンシ モデルは Weak1) を仮定している.

また，評価の対象とする計算機のモデルとしては，図 2に示すものを仮定した．以下にこの計算機モデルの構成をまとめる．また，各部でのレイテンシは, 複数の並列計算機の設計データを参考として, 表 2に示す値とした．

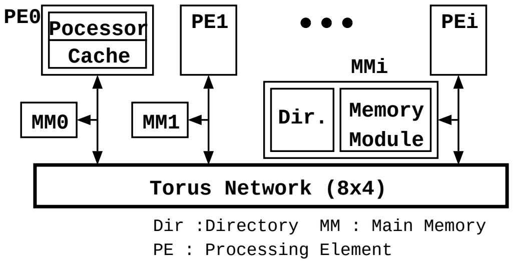


図 2　シミュレーション･モデル

表 2　シミュレータ各部のレイテンシ

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	4[cycle]
	ワード･アクセス	1[cycle]
	タグ･アクセス	1[cycle]
ネットワーク	データなし (1hop)	L[cycle]
	データ (32B) 付き (1hop)	L+4[cycle]
メモリ	ライン･アクセス	10[cycle]
	ワード･アクセス	4[cycle]
	ディレクトリ･アクセス	4[cycle]

L :ネットワークのレイテンシ. 本シミュレーションでは L=10 および 100 について評価

プロセッサ　全命令 1 クロックで動作. 命令は全て命令キャッシュにヒットするものとし, 命令フェッチがデータアクセスの妨げになることはない.

キャッシュ　キャッシュサイズ 1MB の 1 階層ライトバック･キャッシュであり, 構成は連想度 1 のダイレクト･マップ方式を採用する. また, ライト･ミス時にデータのフェッチを行うライト･アロケート型を採用する. ライン･サイズは 32 バイト固定である. 各キャッシュでは最大 4 つのライト･ミス処理 (書き込み権利の獲得も含む) と 1 つのリード･ミス処理を同時に実行可能である. キャッシュとネットワークの間にライトバッファは設けない．

メモリ　幅 64 ビットのメモリで, アクセスタイムは 4 クロック, 連続アクセス時のサイクルタイムは 2 クロックとする. DISC モデルでは, 各ライン対応のフルマップのディレクトリを設ける.

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, フロー制御は ストア･アンド･フォワード方式. データ幅は 64 ビット.

4. シミュレーション結果

シミュレーション結果を，図 3, 図 4, 表 3, および表 4に示す．図の横軸は SCM の LRU エントリ数である．ここで，WOSCM は SCM 制御なしのモデルである. NCC-NUMA モデルで WOSCM の場合は, 同期点でコンパイラが明示的に キャッシュ内の全 dirty line のライトバックを行うモデル15) であり, ソフトウェアによる情報管理オーバヘッドを 0 とした理想状態のシミュレーションを行っている.

ネットワーク･レイテンシが 10clk の場合は, 100clk の場合に比べてメモリアクセス時間が相対的に短くなるものの, 各モデル内でのメモリアクセス時間に関する特徴は 100clk の場合と同じであるので図は省略した.

表 3は各モデルの最短実行時間を示したものであり, それに対応するシステム構成時のネットワーク･トラフィックの状況を示したのが表 4である.

DISC モデルに関する考察

両 DISC モデルの比較をすると, false sharing 対策に伴ってデータサイズが大きくなることによるリード

☆ 文献4) で提案されている Fast Selective Invalidation 方式において，全ての共有データを *memory_read*，private データを *cache_read* としたものに相当する無効化方式

☆☆ Australia Adelaide 大学の Spa package V1.0 を参考に，データ分割 directive，並列処理プリミティブを指定可能としている．

ストール時間の増加は若干あるものの（WOSCM や L32 の場合）, 逆に false sharing によるピンポン減少がなくなったことによるミスヒットの減少が見られストール時間が短くなる現象が見られた（L2,L4,L8）.

しかし, noFS コードでは一貫性制御のためのバリア同期を挿入したことにより, DISC モデルにとっては無駄な待ち時間が激増し, 実行時間全体としては orig コードに比べて 23% 近く実行時間が増加している.

SISC モデルに関する考察

これらのモデルの比較では, SISC モデルが圧倒的に実行時間が遅くなることを予測していたが, シミュレーション結果では, ネットワーク・レイテンシが 10clk の場合, 全体的に SISC モデルが 5～10% 程度遅いものの, レイテンシが 100clk の場合は, 両者拮抗する性能が得られている.

また, 実行時間の内訳を見ると, SISC モデルでは無駄な無効化に伴うリードストール時間の増加が見られるが, 同期点ではネットワークを介したシステムワイドな一貫性制御が不要であるため, 同期待ち時間が短縮されている.

SISCopt モデルに関する考察

SISC モデルでは, 表 1の Type I のデータが, 参照フェーズでキャッシングされても, 次の同期点で無効化されてしまう. これに対して, SISCopt モデルでは 参照フェーズの最初にローカル・コピーを作るのでそれ以降の同期点でも無効化されずにキャッシュに残っている. そのため, リード待ち時間が SISC モデルの約半分に短縮された. その結果, DISC(noFS) モデルに比べても実行時間が短くなるという結果が得られた.

また, キャッシュ・ミスヒットも DISC モデルより少なく, 総メッセージ数は, DISC モデルの約半分にまで減少した.

ExpSISC モデルに関する考察

ExpSISC モデルでは, 表 1の Type I のデータに対して, 参照フェーズにおいて, そもそも明示的な無効化を行わないために, SISCopt モデルと同じ効果が得られている. また, SISC モデルにおける一貫性制御指示のための同期点が一部不要になるために, それにともなう同期待ち時間がなくなっている. 結果として, 短縮された同期時間に対応する分だけ実行時間が短縮されている.

また, ExpSISC モデルでは, false sharing 対策を行っていないコードを実行した DISC モデルよりも常に実行時間が短くなるという結果が得られた. また, ネットワークのレイテンシが大きいほどその効果も大きく, レイテンシが 100clk の場合, 17% も実行時間が短縮されることが確認できた.

これは, プログラムの静的解析が十分にできた場合, NCC-NUMA モデルが CC-NUMA モデルよりも優れた性能を発揮できることを示した例である.

表 3　各モデルの最短実行時間

システム モデル	実行時間	
	L = 10[clk]	L = 100[clk]
DISC(orig)	20,839,118	39,457,221
DISC(noFS)	25,480,394	51,177,143
SISC	26,292,081	50,887,769
SISCopt	25,284,585	42,028,001
ExpSISC	20,580,396	32,887,898

表 4　メッセージ数

システム モデル	メッセージの種類			
	CRead	MWB	CWrite	INV
DISC(orig)	226,887	256,750	567,050	757,940
DISC(noFS)	207,352	223,835	561,928	744,405
SISC	707,425	—	287,061	—
SISCopt	327,438	—	287,061	—
ExpSISC	317,371	—	287,061	—

CRead :キャッシュが発行する読み出し要求
MWB :CRead に対してメモリが発行する書き戻し要求
CWrite :キャッシュが発行する書き込み要求
INV :CWrite に対してメモリが発行する無効化要求
Latency = 100[Clk]

5. まとめ

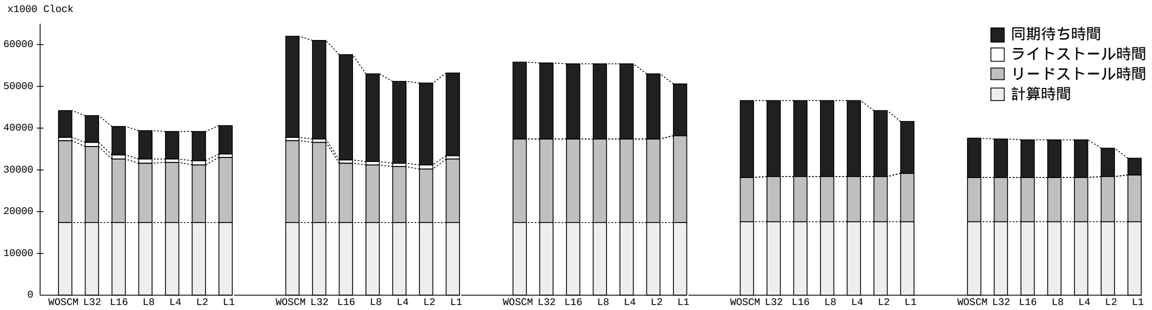
本稿では, Self-Cleanup Cache を備えた NCC-NUMA アーキテクチャの性能を, 無効化型のキャッシュ一貫性制御方式を採用する場合の実行速度, ネットワーク・トラフィックの点からシミュレーションにより評価した.

その結果, ソフトウェアによる簡単な最適化を行うことで, NCC-NUMA が CC-NUMA に比肩する性能をだせること, また場合によっては CC-NUMA よりも良い性能が得られることを確認した. さらに, 十分な静的解析ができた場合, ネットワークのレイテンシが大きいほど, NCC-NUMA の相対性能が良くなることも確認した.

謝辞

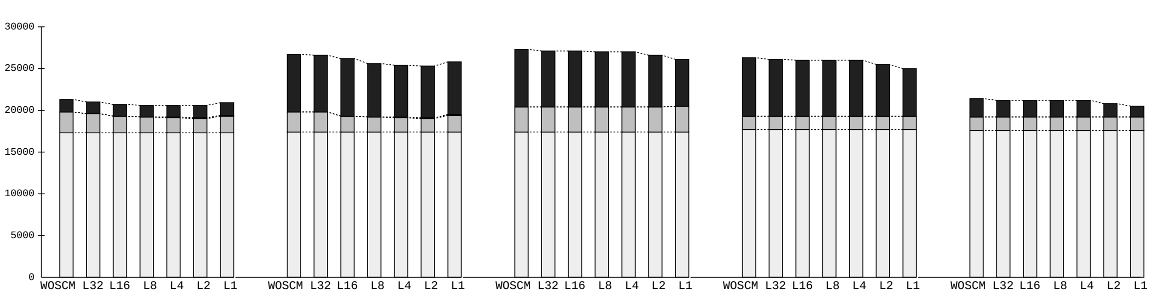
本研究で使用した並列システム・シミュレータは, 細見岳生氏（現 NEC C & C システム研究所）, 神嶌潔氏（現 Fuji Xerox）が 京都大学在学中に作成したものをベースにしている. 両氏に心から感謝の意を表する.

また, 日頃より有益な御意見を頂く, 富田研究室の諸氏に感謝致します.



(a) DISC (orig) *(b) DISC (noFS)* *(c) SISC* *(d) SISCopt* *(e) ExpSISC*

図 3　各システムモデルでの実行時間 (Latency=100[clk])



(a) DISC(orig) ***(b) DISC(noFS)*** ***(c) SISC*** ***(d) SISCopt*** ***(e) ExpSISC***

図 4　各システムモデルでの実行時間 (Latency=10[clk])

参考文献

1) Sarita V.Adve and Mark D.Hill, "WEAK ORDERING - A NEW DEFIFINITION AND SOME IMPLICATIONS," *Comuter Sciences Technical Report*, #902, Computer Sciences Department, University of Wisconsin-Madison, December 1989.

2) Brantley, W.C., McAuliffe, K. P., and Weiss, J., "RP3 Processor-Memory Element," *Proc. 1985 Int'l. Conf. on Parallel Processing*, pp.782–789, Aug. 1985.

3) Yung-Chin Chen and Alexander V. Veidenbaum, "A Software Coherence Scheme with the Assistance of Directories," *CSRD Report No. 1106*, pp.1–22, June 1991.

4) Cheong, H. and Veidenbaum, A. V., "A Cache Coherence Scheme with Fast-Selective-Invalidation," *Proc. 15th Int'l. Symp. Comput. Architect.*, pp. 299-307, June 1988.

5) Cheong, H. and Veidenbaum, A. V., "A Version Control Approach to Cache Coherence," *Proc. Int'l Conf. on Supercomputing '89*, pp. 322-330, June 1989.

6) L. Choi, H.-B. Lim, and P.-C. Yew, "Techniques for Compiler-Directed Cache Coherence," *IEEE Journal of Parallel & Distributed Technology*, pp. 23-34, Winter 1996.

7) Michel Dubois, Christoph Scheurich, and Faye A. Briggs, "Memory Access Buffering in Multiprocessors," *Proc. 13th Ann. Int'l. Symp. Comput. Architect.*, pp.434–442, 1986.

8) Kourosh Gharachorloo, Daniel Lenoski, James Laudon, Phillip Gibbons, Anoop Gupta, and John Hennessy, "Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors," *Proc. 17th Ann. Int'l. Symp. Comput. Architect.*, pp.15–26, May 1990.

9) 細見, 他：「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」, 情処研報 (ARC), Vol.93, No.20, pp.117–124, 1993 年 3 月

10) 細見, 他：「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」, 情報処理学会論文誌, Vol.37, No.2, pp.290–299, 1996 年 2 月

11) L. I. Kontothanassis and M. L. Scott, "Distributed Shared Memory for New Generation Networks," *Tech. Rep. 578, Dep. of Computer Science, Univ. of Rochester*, Mar., 1995.

12) 森, 他：「Self-Cleanup Cache の提案」, 情報処理学会論文誌, Vol.38, No.2, 1997 年 2 月

13) K. Petersen and K. Li, "Cache Coherence for Shared Memory Multiprocessors Based on Virtual Memory Support," *Proc. of Int'l Parallel Processing Symp.*, pp.49–55, Apr., 1993.

14) J. P. Singh, W. D. Weber and A. Gupta, " SPLASH: Stanford Parallel Applications for Shared-Memory," *Tech. Rep. CSL-TR-91-469,Stanford Univ*, Apr., 1991

15) Harjinder S. Sandhu, "Algothitms for dynamic software cache coherence," *J. of Parallel and Distributed Computing*, pp.142–157, Academic Press Inc., Sept. 1995.