計算機アーキテクチャ 102－6
（1993．10．21）

キャッシュ・オンリ・メモリ・アーキテクチャ向き ディレクトリ機構の提案

福田 宗弘†, 村田 浩樹††, 清水 茂則††

†Department of Information and Computer Science
University of California, Irvine

††日本アイ・ビー・エム（株） 東京基礎研究所

キャッシュ・オンリ・メモリ・アーキテクチャ（以下ＣＯＭＡ）は分散共有メモリ・システムの一種で, 各ノードのローカル・メモリを一種のレベル３キャッシュとして用いることで, 大きな共有メモリ領域を提供しようとするものである．しかし一般に階層ディレクトリ機構を用いるためメモリ遅延が長くなるという欠点を持つ．さらにメモリ空間をキャッシュとして使いきってしまった場合, 新たなメモリ空間を得るためにＯＳのサポートが必要となる．本論文では, ＣＯＭＡのメモリ・アクセス遅延を減少させるための非階層ディレクトリ機構と, ＯＳのサポートを不要とするためのデータ割付プロトコルの２つを提案する．

Flat Directory Scheme for Cache-Only Memory Architecture

Munehiro Fukuda†, Hiroki Murata††, and Shigenori Shimizu††

†Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717, U.S.A.

††Tokyo Research Laboratory, IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi, Kanagawa 242, Japan
e-mail: murata@trl.ibm.co.jp

Cache-only memory architecture(COMA), a distributed shared-memory system, uses each node's local memory as a kind of third-level cache to provide a large capacity for shared data. However it has the drawback of long memory access latency because of its hierarchical directory scheme. In addition, the operating systems (OSs) must be involved in directory tag invalidations upon reclaiming a new caching area. We aim to reduce the memory access latencies without increasing the directory size or the number of inter-node messages, and to reduce OS involvement. For these purposes, we propose a right-sized flat directory scheme and a caching-space allocation protocol.

1 はじめに

密結合型マルチプロセッサ・システムは, プロセッサ間で同一メモリ空間を共有しており, データ転送が暗黙のうちになされるため並列処理に広く用いられている. しかし, それらはプロセッサ間接続がネックとなりやすく, これを避けるために複雑で高価なネットワークを必要とする場合がある. また新しいCPUが作られるたびに, 設計をし直さなくてはならないことが多いという欠点もある.

近年は分散共有メモリ・システム (DSM)[5] が注目されている. DSMはネットワーク接続されたワークステーション・クラスタ上に共有メモリ・モデルを構成するもので, ハードウェア設計に大きな負担をかけることなく, 並列度を向上させることができると期待されている. Fiber Channel Standard(FCS)[7] や Scalable Coherent Interface (SCI)[1] などの高速な通信媒体の標準化, 商業化によって, ノード間のキャッシュ・コヒーレンシをハードウェアによって維持することが実用的になりつつある.

ここで2種のハードウェア指向DSMシステムについて述べる. 1つは cache-coherent nonuniform memory architecture (cache-coherent NUMA) を用いたもの, 他方は cache-only memory architecture (COMA) を用いたものである. これらは一般的に分散ディレクトリ機構を用いることで, キャッシュ・コヒーレンス維持のためのメッセージをデータを共有するノードのみに送る. これによってブロード・キャストした場合よりもメッセージ数を減らしている. これらのアーキテクチャの主な違いは,cache-coherent NUMA が各ノードのローカル・メモリを通常の主記憶として利用するのに対し,COMA はキャッシュとして用いる点である. これらの得失についてはP. Stenström らによって, 文献[6] で解析的に述べられている.

Cache-coherent NUMA を用いた例としては,Stanford's DASH マルチ・プロセッサ [3] がある.Cache-coherent NUMA システムでは, 各ノードに非階層ディレクトリと他のノードからデータをキャッシュするための特別のローカル・キャッシュが設けられる. 非階層ディレクトリは自ノードのデータをキャッシュしているノードを指す. このため cache-coherent NUMA システムはデータ検索が容易で早く, 特定のネットワーク・トポロジに依存しない. しかし他ノードからのデータのキャッシュ量がローカル・キャッシュの大きさに制限される. COMA を用いた例には Swedish Institute of Computer Science's Data Diffusion Machine (DDM)[2] がある. COMA を採用した計算機では, データは物理アドレスとは関係なく配置され, ノード間を移動する. システムのどこかに存在する必要なデータを見つけ出すために, 各ノードは, まず少数でクラスタとしてネットワーク接続される. 各クラスタはメモリ・アクセスがクラスタ内で済むか済まないかを判別するディレクトリを介して上位のネットワークに接続され, 規模の大きなクラスタを構成する. その大クラスタがさらにディレクトリを介して上位のネットワークへ, というように階層化して接続される. したがって,COMA は cache-coherent NUMA とは逆に, メモリ・アクセス遅延は長いが, キャッシュできるデータ量が大きいという利点を持つ.

Flat COMA は両アーキテクチャの利点, すなわち短いメモリ・アクセス遅延と大容量キャッシュをを合わせ持つものとして P. Stenström によって文献 [6] で提案されたが, 実現する上で重大な問題がある. Flat COMA は COMA の階層ディレクトリを NUMA 型の非階層ディレクトリで置き換えるが, その実現方法である. 単純な実現方法としては K. Li が文献 [4] で提案した fixed distributed manager algorithm によるものがある. このアルゴリズムでは, 各ノードは,DSM 空間中の予め割り当てられた範囲のデータについて, それらを所有するノード (オーナ・ノード) を記録する owner management table を持つ. このためデータをアクセスする場合, そのデータのオーナ・ノードを記録しているノードを介してアクセスするため, 直接アクセスできる場合に比べメッセージが1つ増え遅延が長くなる. 同様に K. Li が提案した dynamic distributed manager algorithm によるものでは, 各ノードが全ての DSM 空間のデータのオーナ・ノードをディレクトリに記録するため直接オーナ・ノードに到達できる. しかしこのアルゴリズムには2つの欠点がある. 1つは全てのノードが同時にディレクトリを更新するわけではないので, 実際のオーナがすでに変わっているのに, メッセージがフォワードされる場合がある. もう1つはノード数が増加した場合にディレクトリが巨大なものになるという点である. 適正な規模のディレクトリで, メッセージ数が少なくアクセス遅

延を短くできるアルゴリズムが望ましい．もう一つのCOMAの重大な問題点は，空きメモリがなくなった場合に,OSが使用されないと考えられるデータをディスクに書き出して，メモリに空きを作らなくてはならない点である．ディレクトリなどをハードウェアで構成した場合，これはOSのメモリ・マネジャ部分にハードウェア依存のコードを増加させることになり，これも少ない方が望ましい．

本論文では,flat COMAのための，適正な規模のディレクトリを実現するオーナーシップ管理機構を提案する．また空きメモリの欠乏を防ぐキャッシュ・スペース割当プロトコルについて考察を試みる．オーナーシップ管理機構はfixed distributed manager algorithmとdynamic distributed manager algorithmを統合したもので，メッセージ数とメモリ・アクセス遅延を減少させる．各ノードは2つのテーブル:dynamic directory tableとfixed owner tableを持つ．前者は，自ノードに持っているデータのコヒーレンシ維持と，オーナの記録に使われる．後者は自ノードに予め割り当てられたDSM空間のデータのオーナを記録する．キャッシュ・スペース割当プロトコルについては，自ノードのローカル・メモリに置かれる共有データの比率をハードウェアで調節することで空きメモリの欠乏を防ぐ方法について考察する．

ここで以後の議論に備えて，語句を定義する．

リクエスタ: データを要求するノード
マネージャ: DSM空間の予め指定された範囲のデータのオーナシップ情報を記録するノード
オーナ: データのオーナシップを持つノード
シェアラ: データのオーナシップは持たないがデータを共有しているノード
ネイバ: ノードIDが自ノードのものより1大きいノード
データ要求メッセージ: リクエスタからオーナへのリモート・アクセス
データ供給メッセージ: オーナからリクエスタへにデータを返す
確認メッセージ: 新オーナからマネージャへのオーナシップ更新情報
無効化メッセージ: オーナからシェアラへの共有データ無効化
要求遅延: ローカル・プロセッサの要求から確認メッセージまでの時間

2章では，本論文で考えるflat COMAの構成について述べる．3章では,flat COMAのための，適正な規模のディレクトリを実現するオーナーシップ管理機構について述べる．4章では，3章で述べたオーナシップ管理機構の性能について簡単な評価をおこなう．5章では空きメモリの欠乏を防ぐキャッシュ・エリア割当プロトコルについて考察を試み，6章で簡単なまとめを述べる．

2 Flat COMAシステムの構成

我々の想定するflat COMAシステムは，図1に示すように，各ワークステーションが通信ボードに非階層ディレクトリを持ち適当なネットワークに接続される．ワークステーションはローカル・メモリをレベル3・キャッシュとして扱うので，このレベル3・キャッシュがデータをリプレイスする先は存在しない．あえてリプレイスするとすれば他のノードとなる．プロセッサ／キャッシュがデータをローカル・メモリに要求すると，通信ボードはそのデータがローカル・メモリに存在するかをチェックし，存在すればローカル・メモリからデータを供給する．データがローカル・メモリに存在しなかった場合，通信ボードは，そのデータの最新のオーナへデータ要求メッセージを発行し，データ供給メッセージを待ち受ける．リモート・アクセス中のデッド・ロックを避けるために，各ノードのシステム・バスはスプリット・バス転送機能を持つ．通信ボードの主要構成要素はdynamic directory table,fixed owner table, directory controller,link control chipである．Dynamic directory tableの持つ機能は,DSMアドレスから物理アドレスへの変換と，ローカル・メモリにあるデータのコヒーレンシ／オーナシップの維持である．Fixed owner tableは，予め割り当てられた範囲のDSM空間のデータのオーナを記憶する．Directory controllerは，これら2つのテーブルを更新／維持する．Link control chipは，ネットワークとのメッセージのやり取りを行なう．プロセッサ／キャッシュが要求したデータがローカル・メモリに存在しない場合，データ要求メッセージがlink control chipを介してネットワークに送出される.Dynamic directory tableに要求されたデータの有効なエントリがあれば，そこに記録されているオーナにメッ

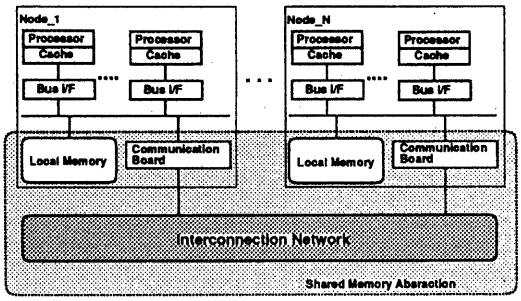

図 1: システム構成

セージが直接送られる. 有効なエントリがなければ, データの fixed owner table を持つマネージャ経由で送られる. このような仕組みで, 階層ディレクトリを置き換えることができる.

3 オーナシップ管理機構

COMA は, ソフトウェア・サポートによってノード間でページを共有する virtual shared memory (VSM) と, データの位置を物理アドレスから分離するという点で非常に良く似ている. したがって,VSM システムでオーナに捜し出すために提案された dynamic distaributed manager algorithm は, 理論的には COMA に適用することができる. このアルゴリズムでは, 各ノードのページ・テーブルが, すべての DSM データのオーナシップとコヒーレンシについての情報を持っている. ノードはページ・テーブルのオーナシップ・フィールドを参照して, 直接オーナにデータ要求メッセージを発行する. 該当データがすでに他のノードに転送されていた場合, データ要求メッセージもそのノードへ転送される. このメッセージ転送の繰り返しは, 転送に関わるノード全てにリクエスタの ID を知らせることで有限回に制限できる. VSM と COMA の違いは,DSM 空間をアクセスする際に用いるアドレスの違いにある. VSM は仮想アドレスを用いて DSM 空間をアクセスするが, COMA は物理アドレスを用いる. このため VSM と COMA では, データ・アクセスの大きさが異なる. VSM は, データをページ単位で扱い, ディレクトリとしてページ・テーブルを用いるが, COMA は, データをキャッシュ・ライン単位で扱い, ディレクトリ・テーブルを用いて制御する.Dynamic distributed manager algorithm を COMA に適用すると, VSM と比較して非常に大きなディレクトリ・テーブルが必要となる. 例えば,32 ノードからなる COMA システムで, 各ノードに 64Mbyte のローカル・メモリを持ち, キャッシュ・ライン長が 64byte で, ディレクトリの各エントリに 2byte 必要だとすると, ディレクトリ・テーブルとして 64Mbyte 必要になる.

我々は各ノードが,dynamic distributed manager algorithm のように全 DSM 空間に対応するディレクトリ・テーブルではなく, ローカル・メモリ空間に対応するディレクトリ・テーブルを持つべきであると考える. この場合, ディレクトリ・サイズはシステムの拡張とは関係なく, ローカル・メモリ・サイズによって決まる. 前述の条件で, その大きさは 2Mbyte となり, システム中のメモリ量によらず一定である. 我々はこのディレクトリを dynamic directory table と呼ぶ. このテーブルの構造は, 表 1に示すようにローカル・メモリに関係付けられた一種のフル・アソシアティブ・タグ・メモリである. データが一度ローカル・メモリに読み込まれると,dynamic directory table は, そのデータの物理アドレス, コヒーレンシ情報, と最新のオーナシップを記録する. このテーブルは, データが他のノードのライトで無効化された後も, 新たなデータが同じアドレスに読み込まれるまでは関連するオーナシップ情報を残しておく.

Dynamic directory table に, 要求されたデータのオーナシップ情報がない場合, 我々が fixed owner table と呼ぶもう一つのテーブルが必要になる. この状況はデータがローカル・メモリに取り込まれたことがないか, ローカル・メモリの同じアドレスに別のデータが取り込まれてオーナシップ情報が上書きされた場合に起こる. (我々はこの状況を cold cache miss と呼ぶ. これに対応して, ローカル・メモリにデータは残っていないが,dynamic directory table にオーナシップ情報が残っている状況を hot cache miss と呼ぶ.) 全てのノードは fixed owner table を持ち,DSM 空間の予め割り当てられた範囲のデータのオーナシップを記憶している. 各ノードに割り当てられる範囲の広さはそのノードのローカル・メモリの広さであり, 全てのノードのローカル・メモリが同じ大きさならば,DSM 空間をノード数で割った大きさとなる. Cold cache miss が発生すると, データ要求メッセージは, まずマネージャに送られ, そこ

表 1: ノード♯3 の Dynamic directory table

DSM Line	Phy. Line	Status	Owner	Fwd Link	Bwd Link
100	0	invalid	5	—	—
101	1	exclusive	3	—	—
302	2	exclusive	3		
	3				
—	999,998	invalid	—	—	—
59	999,999	shared	10	2	4

DSM Line: DSM Line Number
Phy. Line: Physical Line Number
Status: Coherency Status
Owner: Owner Node ID
Fwd Link: Forward Link to Sharing Node
Bwd Link: Backward Link to Sharing Node

から fixed owner table のエントリに記録されているオーナへと転送される. Fixed owner table を用いたオーナの捜し出しは, マネージャを介するため, メッセージ数が増加する. しかし我々は cold cache miss 数は hot cache miss 数に比べて非常に少ないと考えている. このオーナシップ管理アルゴリズムを semi-dynamic distributed manager algorithm と呼ぶことにする.

図 2は, リクエスタ, マネージャ, オーナの 3 者の間のメッセージの流れを二種類示している. 一方は hot cache miss の場合のもの, 他方は cold cache miss の場合のものである.DSM アドレス♯10 へのメモリ・アクセスは hot cache miss の場合のもので, リクエスタの dynamic directory table に, オーナ・ノード ID♯3 が記録されている. リクエスタはデータ要求メッセージをオーナ・ノードに直接送る. オーナは,DSM アドレス♯10 を dynamic directory table を用いてローカル・メモリの物理アドレス♯i に変換し, そこに置かれているデータ A をリクエスタに送り返す. DSM アドレス♯15 へのメモリ・アクセスは cold cache miss の場合のものである. リクエスタは, まずデータ要求メッセージをマネージャに送る. マネージャは fixed owner table を参照して, オーナ・ノードの ID,♯3 を発見し, メッセージをオーナへ転送する. 以降は hot cache miss 時と同様にして, データ B がオーナによって供給される.

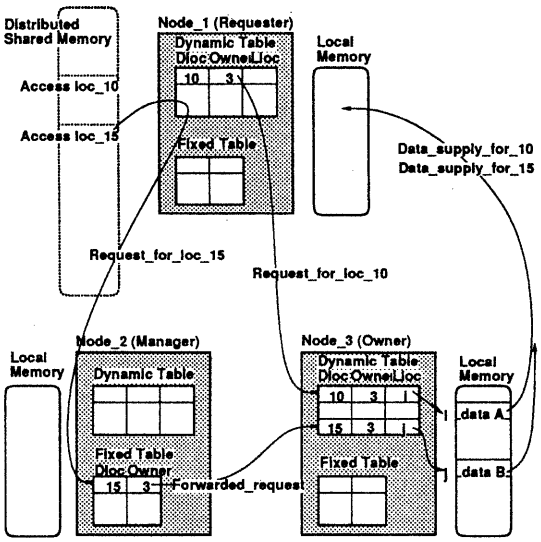

図 2: リクエスタ, マネージャ, オーナの 3 者の間のメッセージの流れ

4 Semi-dynamic distributed manager algorithm の性能

K. Li は文献 [4] において,fixed distributed manager algorithm と dynamic distributed manager algorithm のメッセージ・コストを, 以下の式 1, および, 式 2によって示している. ここで, 式中に現れる各変数は, 次の意味を持っている.

c_s : メッセージ送信コスト
c_r : メッセージ受信コスト
$c_{total}(i)$: プロセッサ i での page fault/cache miss 全体のコスト
$c_{inv}(i)$: プロセッサ i での無効化の総コスト
$f_r(i)$: プロセッサ i での read page fault/read cache miss の総数
$f_w(i)$: プロセッサ i での write page fault/write cache miss の総数
$r_p(i)$: プロセッサ i が受信したデータ要求の総数
$r_{inv}(i)$: プロセッサ i が受信した無効化要求の総数
$r_{loc}(i)$: プロセッサ i が受信したページ検索要求メッセージの総数

$s_{fwd}(i)$: プロセッサ i が転送した検索メッセージの総数

4.1 fixed distributed manager algorithm

$$\begin{aligned} c_{total}(i) &= c_{req1} + r_{loc}(i)(c_s + c_r) \\ &\quad + r_p(i)(c_s + c_r) \\ &\quad + c_{inv}(i) + r_{inv}(i)c_r \end{aligned} \tag{1}$$

但し,

$c_{req1} = 2(f_r(i) + f_w(i))(c_s + c_r)$

データを要求するのに必要なマネージャとオーナの間で交わされるメッセージのコスト

$r_{loc}(i) \approx \frac{1}{N}\sum_{j=1}^{N}(f_r(j) + f_w(j))$

データを検索するのに必要なリクエスタとマネージャの間で交わされるメッセージのコスト

4.2 dynamic distributed manager algorithm

$$\begin{aligned} c_{total}(i) &= c_{req2} + f_r(i) \times c_s + r_p(i) \times c_s \\ &\quad + c_{inv}(i) + r_{inv} \times c_r \end{aligned} \tag{2}$$

但し,

$c_{req2} = (f_r(i) + f_w(i))(c_s + c_r)$

データを要求するのに必要なマネージャとオーナの間で交わされるメッセージのコスト

$$\begin{aligned} \sum_{i=1}^{N} f_f(i) &\leq \sum_{i=1}^{N} s_{fwd}(i) \\ &\leq c(\log N)\sum_{i=1}^{N}(f_r(i) + f_w(i)) \end{aligned}$$

$f_f(i)$ は, ノード i において, データ要求メッセージを受信したが, 自分のローカル・メモリにデータが存在しておらず, オーナとして登録されているノードへ, そのメッセージを転送する総数

概略的に見て, 無効化に必要なコストを除いたコストは, dynamic distributed manager algorithm の方が fixed distributed manager algorithm の $\frac{1}{2}$ になることがわかる. semi-dynamic distributed manager algorithm のメッセージ・コストは, 以下の式3によって示される.

4.3 semi-dynamic distributed manager algorithm

$$\begin{aligned} C_{total}(i) &= (1 - p_f(i)) \\ &\quad \times (c_{req1} + (r_{loc}(i) + r_p(i))(c_s + c_r)) \\ &\quad + p_f(i)(c_{req2} + f_f(i)c_s + r_p(i)c_s) \\ &\quad + c_{inv}(i) + r_{inv}(i)c_r \end{aligned} \tag{3}$$

式3において,$p_f(i)$ は,hot cache miss の確率である. この確率が1に近付くほど, 式3の第2項の成分が大きくなり, その性能は dynamic distributed manager algorithm と変わらなくなる. 例えば, キャッシュ・ライン長が 128byte のシステム上で 1Mbyte の子プロセスを実行することを想定する. 単純に考えて, まず,8000 回の cold cache miss が発生する. しかし他の子プロセスと一箇所以上の共有領域を持ち, 互いにこの領域に 8000 回の書き込み操作を行なったとすれば,$p_f(i)$ の値は 0.5 となる.

以上のように, semi-dynamic distributed manager algorithm はシステム性能を dynamic distributed manager algorithm に近付けることができる.

5 キャッシュ・スペース割当プロトコル

COMA はライト・バック用の記憶は持たないので, ローカル・メモリ空間が満杯に近くなると, データ (DSM 空間) のスラッシングが発生する. キャッシュ・スペースを再利用し, スラッシングによる性能の低下を抑えるために OS による directory tag の操作を伴う DSM 空間中のデータの整理が必要となる. ユーザ・プロセスが解放した DSM 空間を特定し, 対応する fixed owner table を持つマネージャにテーブルを更新させるメッセージを送る必要がある. 以上の処理には directory controller と同様な

機能, すなわちDSMアドレスの物理アドレスへの変換, タグ情報の更新, そしてメッセージのやり取りが, OSに必要となる.

このハードウェア依存でタイミング・クリティカルな操作をOSに作り込むのを避けるために, 全DSM空間をローカル・メモリにマッピングして, かつ, 他ノードのデータをキャッシュする余裕を各ノードのローカル・メモリに残すようにハードウェアで自動的に制御するためのプロトコルについて考察する. DSM空間をローカル・メモリにマッピングする場合, ノード間で共有されるデータがあるため, DSM空間の容量はローカル・メモリの全容量よりも少なくなる. しかも共有データの量によって, ローカル・メモリにマッピングできるDSM空間が変化するため, ハードウェアによる制御が困難になる. そこで, 各ノードが共有データ量の上限を定めることで, DSM空間のローカル・メモリへのマッピングをdirectory controllerが維持できるようになる. 全DSM空間の大きさは, 以下の3つの定数と式4から求められる.

N: システム中のノード数
L_i: ノード$\sharp$iのローカル・メモリ・サイズ
S_i: ノード$\sharp$iの共有データ量の上限

$$\sum_{i=1}^{N} L_i(1-S_i) + \max_{i=1}^{N}(L_i \times S_i) \qquad (4)$$

このDSM空間のサイズは各ローカル・メモリにキャッシュ・スペースを確保するために定義される. したがってOSは空きスペースを作り出すためにディレクトリ・タグを操作する必要がなくなる. 各ノードはデータ・コヒーレンシとキャッシュ・スペースを, 以下の6つのローカル・レジスタを用いて式5を満たすように制御する.

$NODES$: システム中のノード数. 上でNとして与えられた.

$LSIZE$: ローカル・メモリのライン数. 上で与えられたLiに関係する.

$SDEGREE$: ノードの共有データの比率. 上でSiとして与えられた.

$VCNT$: ローカル・メモリ中の有効なライン数

$SCNT$: ローカル・メモリ中の共有ライン数

$DSIZE$: DSM空間中の総ライン数. 式4で計算された.

$$\begin{aligned} VCNT &\leq LSIZE \\ SCNT &\leq LSIZE \times SDEGREE \end{aligned} \qquad (5)$$

NODES,LSIZE, とSDEGREEは初期化の後は更新されないとする. するとDSIZEも式4で求められた後は更新されない. VCNTとSCNTは以下のデータ・コヒーレンシ・プロトコルに従って, 自動的に更新される. Directory controllerは, このプロトコルを全てのリモート・アクセスに適用する. データ・コヒーレンシ・プロトコルは次の三状況で異なる. (1) 共有ラインが共有データ量の上限を越えている場合 (リモート・ノードからデータが要求された場合に発生する.), (2) 新しいデータを置くスペースがある場合, (3) 新しいデータを置くスペースがない場合. 図3は, 各状況下で新しいデータをローカル・スペースに置く場合のデータ・コヒーレンシとキャッシュ・スペースを示している.

case 1: $SCNT \geq LSIZE \times SDEGREE$
ローカル・メモリ中の共有ライン数が上限を越えている場合である. Directory controllerは共有ラインを上書きしてラインを再利用する候補として選び出す. 候補のオーナが自ノードならば, 他の共有ノードにオーナシップを移す.Controllerは自ノードを共有ノードのチェインから外し, 新しいデータを上書きする. 新しい共有ノードのチェインに自ノードを入れる.

case 2: $(SCNT < LSIZE \times SDEGREE) \bigcap (VCNT < LSIZE)$
ローカル・メモリに無効ラインがある場合.Directory controllerは新しいデータを無効ラインに上書きし, 共有ノードのチェインに自ノードを入れる.

case 3: $(SCNT < LSIZE \times SDEGREE) \bigcap (VCNT = LSIZE)$
空きスペースがないが, 共有ライン数は上限を下回っている状態. Directory controllerは専有ラインを選び出し, オーナシップと共に他ノードに送り出す. そのラインに新しいデータを上

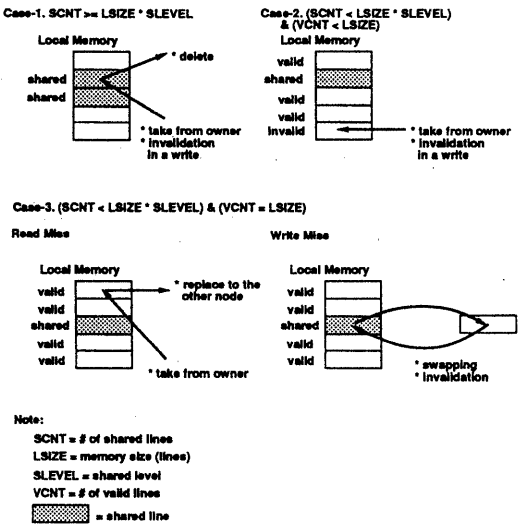

図 3: データ・コヒーレンシと共有データ量の上限の制御

書きし, 共有ノードのチェインに自ノードを入れる.

Directory controller は上で述べた 6 つのローカル・レジスタを調べてこれら三状況のいづれであるかを判断し, リモート・データを要求する前に空きスペースを確保する.

6 おわりに

本論文では, 非階層ディレクトリを用いて flat COMA システムを実現するオーナシップ管理機構,semi-dynamic distributed manager algorithm を提案し, 簡単な評価を行なった.Semi-dynamic distributed manager algorithm は, ディレクトリの大きさが fixed distributed manager algorithm と同様にローカル・メモリの大きさによって決まり, かつ, システム性能を dynamic distributed manager algorithm に近付けることができる. また,COMA システムのデータ・スラッシングを防止するためのキャッシュ・スペース割当プロトコルについて考察した.

Semi-dynamic distaributed manager algorithm, キャッシュ・スペース割当プロトコルについて, より詳細なモデル化, 性能評価を行なうことが今後の課題である.

参考文献

[1] Gustavson, D. B. The Scalable Coherent Interface and Related Standards Projects. *IEEE Micro,* Vol. 12, No. 1, February 1992, pages 10-22.

[2] Hagersten, E., Landin, A., and Haridi, S. DDM - A Cache-Only Memory Architecture. *IEEE Computer,* pages 44-54, September 1990.

[3] Lenoski, D., Laudon, J., Gharachorloo, K., Gupta, A., Hennesy, J., Horowitz, M., and Lam, M. The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor. In *Proceedings of the 17th Annual International Symposium on Computer Architecture,* pages 148-159, May 1990.

[4] Li, K. and Hudak, P. Memory Coherence in Shared Virtual Memory Systems. *ACM Transactions on Computer Systems,* Vol. 7, No. 4, November 1989, pages 321-359.

[5] Nitzberg, B. and Lo, V. Distributed Shared Memory: A Survey of Issues and Algorithms. *IEEE Computer,* Vol. 24, No. 8, August 1991, pages 52-60.

[6] Stenström, P., Joe, T., and Gupta, A. Comparative Performance Evaluation of Cache-Coherent NUMA and COMA Architectures. In *Proceedings of the 19th Annual International Symposium on Computer Architecture,* pages 80-81, May, 1992.

[7] X3T9.3 Task Group. Fibre Channel Physical and Signaling Interface (FC-PH). Rev. 2.2, January 1992.