木構造データ管理による分散共有データオブジェクトの実現方式†

山崎 剛†† 和田耕一†††

本論文では，分散システム上で共有データオブジェクトを管理するアルゴリズムについて述べている．独立性の高いプロセッサをネットワークで結合した分散システムは，ハードウェア構成の柔軟性の点で優れているが，プログラミングが難しい，等ソフトウェアに関する広範な難点を持つ．この問題を解決するため，プロセッサ間で共有されるデータオブジェクトの実現が試みられてきた．本アルゴリズムは，オブジェクトの配置状況を木構造のデータを利用して管理するものである．オブジェクトは一つの木の葉に配置されるか，コピーされて複数の葉に配置される．ノード間のメッセージ通信によってオブジェクトの配置状況を更新する．本アルゴリズムは，プロセッサ数に関する対数オーダの実行時間，小さなオブジェクトに適合する管理データ量，処理の有効な局所化と分散化，1対1のメッセージ通信のみによる実現，といった特徴を有しており，適用性の高いものである．本論文ではまず，このアルゴリズムについて，処理手順と正当性検証の概略を示し，次に細粒度の並列処理を試行した分散共有メモリシステムの管理に適用した例について，管理データ量と応答時間に関する考察を行っている．また，ディレクトリを使用したキャッシュ一貫性プロトコルとの比較についても言及している．

1. はじめに

独立性の高いプロセッサをネットワークで結合した分散システムは，ハードウェア構成の柔軟性の点で優れている．しかし，プログラミングが困難である，プログラムがハードウェアの性質に強く依存したものになりやすい，等のソフトウェアに関する広範な難点を持つ．この問題を解決するため，プロセッサ間で共有されるデータオブジェクトの実現が試みられてきた．このような試みの一例として，すべてのプロセッサからアクセスできる共有アドレス空間を提供する，分散共有メモリシステム[2),6),7)]が挙げられる．

本論文では，分散システムの上で，共有データオブジェクトの操作を実行するアルゴリズムについて述べる．管理対象のデータオブジェクトは，一つのプロセッサに保持されるか，コピーされて複数のプロセッサに配置される．本アルゴリズムは，各プロセッサの要求に対応した配置状況の更新を目的としている．

以下，2章で本アルゴリズムが前提とするデータ構造と，実現する処理について述べ，それに基づいて3章で処理手順を記述する．4章で正当性検証の概略を示し，5章で単純な実現を想定した性能の見積もりを行う．そして，6章で結論を述べて結びとする．

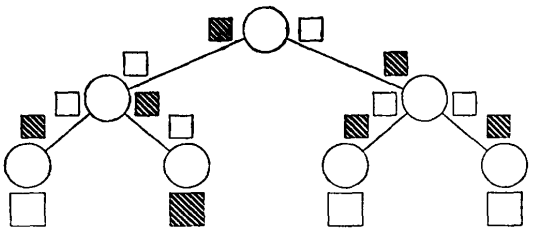
2. 前提と目標

本稿で記述するアルゴリズムの目的は，分散配置されるデータオブジェクトの配置状況の管理である．このアルゴリズムは，**図1**のような，木状のデータ構造を前提とする．以後本稿では，記述を簡略化するため2進木の場合を例示するが，これは本質的な制限ではない．アルゴリズムには，木の枝分かれ数を制限するような要因は存在しないため，任意の木構造に対して適用することができる．木のノードの中で任意の一つを根とする．図では，根を最も上のノードとして示す．それに対応して，根へ近づく方向を上方，根から遠ざかる方向を下方と呼ぶ．管理対象のオブジェクトは，木の一つ以上の葉に分散配置される．図において葉に付随する箱は，一つのオブジェクトの配置状況を示している．節と葉では，それぞれの枝ごとにビットを用意し，その枝に接続する部分木にオブジェクトが存在しているか否かを記録する．これを配置ビットと呼ぶ．図では配置ビットをノードから出る枝に添えた箱で示す．以上のデータをすべてのオブジェクトについて用意する．

このデータ構造を利用してオブジェクトの配置状態の操作を実現する．それぞれの操作は，葉においてオブジェクトを指定して起動され，ノード（節および葉）間の通信によって実行される．このような操作をプリミティブと呼ぶ．次のようなプリミティブを実現

† Tree-Based Management of Distributed Shared Data-Object by TAKESHI YAMAZAKI (The Doctoral Program of Engineering, Graduate School, University of Tsukuba) and KOICHI WADA (Institute of Information Sciences and Electronics, University of Tsukuba).

†† 筑波大学大学院工学研究科
††† 筑波大学電子・情報工学系



■：ページの配置されている葉、又はページに到達できるリンク

図 1 ページ管理データ構造
Fig. 1 Data structure for page management.

する．

1) コピーリクエスト (copy-request)
起動ノードにオブジェクトをコピーする．

2) パージ (purge)
起動ノードに配置されているオブジェクトを削除する．

3) インバリデート (invalidate)
起動ノード以外のすべての葉のノードが持つオブジェクトを削除する．

以下で示す手順は，メッセージが各ノードでオブジェクトごとに直列化され処理されることを仮定している．つまり，すべてのメッセージは，一度に一つずつ受け取られ処理される．通信路の性質としては，メッセージが有限時間で到着することのみを仮定する．メッセージの転送時間，転送順序に対する制約は存在しない．以下の処理では常に，到着したメッセージを即座に処理できるが，それぞれの葉ノードでは，プリミティブの起動を延期しなければならない場合がある．

ここで述べた木構造はあくまでもデータ構造であって，物理的な通信路の接続形態を限定するものではない．すべてのノードをプロセッサに割り当て，プロセッサ間通信によって処理を行うことにより任意の分散システム上で実現できる．

3. オブジェクト管理アルゴリズムの記述

3.1 クリークモデルによる処理手順の記述

同一のオブジェクトを操作するプリミティブがそれぞれの葉で同時かつ非同期的に起動されるのに対応するため，本システムでは，それぞれのプリミティブの実行ごとに木構造の一部をおおう排他的な管理領域を設ける．これをクリーク (clique) と呼ぶ．

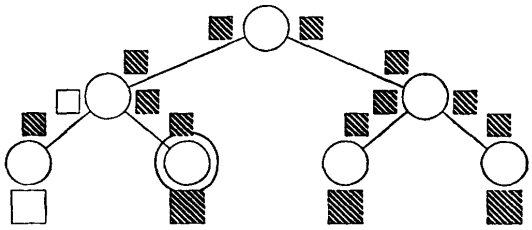
クリークは，ノードの接続した集合である．それぞれのクリークは，単一のプリミティブの実行に対応している．構造全体が一つの木であることを仮定しているので，クリークは任意の時点で一つの木構造を取る．

3.1.1 基本的な処理手順

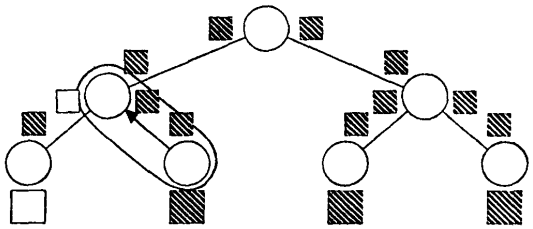
クリーク間の干渉がない場合の処理は以下のような手順で実行される（**図 2**）．

プリミティブが起動されると，起動点となった葉がクリークに取り込まれる（図 2 a）．新たにクリークに取り込まれたノードは，まず，自分がそのクリークの最大拡張点か否かをチェックする．そこが木の葉であるか，またはその先のどの部分木にも，プリミティブが操作対象とするデータが存在しない点であれば最大拡張点である．パージの最大拡張点は，最初に到着した分岐点または葉ノードで，コピーリクエストとインバリデートの最大拡張点は，葉ノードである．

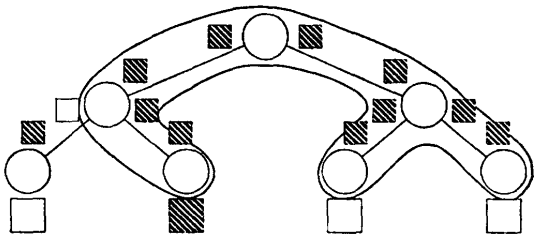
もし最大拡張点ならクリークの縮小を開始する．そうでなければ一つ以上の接続する枝に対してクリーク



(a) インバリデートプリミティブの起動
(a) Start of an invalidate primitive.



(b) クリークの拡張
(b) Enlargement of a clique.



(c) 最大拡張点でのページ操作
(c) Page operations at the largest enlargement point.

図 2 クリークによるプリミティブの実行
Fig. 2 Execution of a primitive with a clique.

の拡張を行う（図2b）．図では，インバリデートプリミティブを例としているため，オブジェクトを保持するすべての部分木に向かってクリークの拡張が行われる．コピーリクエストは，オブジェクトを保持する部分木の中の一つを選択し，そこに向かって拡張する．もし，下方の枝にオブジェクトを保持するものが存在したら，その中の任意の一つを，存在しなかったら上方の枝を選択する．パージは，オブジェクトを保持する部分木に向かってクリークを拡張する．

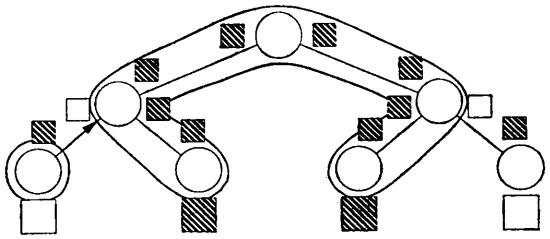
クリークの縮小は，端点のノードをクリークから解放することによって実行される．縮小時にもクリークの連続性は維持されなければならない．この制限より，分岐点の縮小は，すべての枝の縮小が完全に終了してから行われる．

オブジェクトの操作は，クリークの拡大が終了したノード（インバリデートの場合，図2c参照），またはプリミティブの実行を開始したノード（コピーリクエスト，パージ）で実行される．また，ノード上の配置ビットの操作は，クリークが縮小した枝のビットをリセットする（インバリデート）か，あるいはこれからクリークを縮小しようとする枝のビットをセット(コピーリクエスト）またはリセット（パージ）することにより実行される．

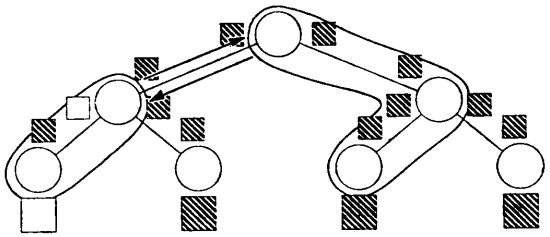
3.1.2 クリーク間の干渉

複数のクリークが存在する状況では，拡張しようとするクリークが他のクリークに妨げられる可能性がある．2種類のクリーク間干渉が生ずる．第1は，クリーク間の妨害が一方的に生ずる場合である．これをブロックと呼ぶ．もう一つは，二つのクリークが相互に相手を妨害する場合である．これを衝突と呼ぶ．以上を図3に示す．

前提とする構造が木であることから，巡回的なブロックは発生しない．したがって，ブロックのみが生じている限り，必ず拡張または縮小する端点が存在する．しかし，衝突は，すでにデッドロックが生じた状態なので，何らかの対策を行わなければならない．ここでは，クリーク間の優先順位を決定し一方のクリークを強制的に後退させることによって衝突の処理を行う．後退処理は，優先権を得たクリークが最大拡張点に到達するか，優先権を失ったクリークに妨害されなくなるまで実行される．もし，プリミティブが開始ノードまで後退してすべてのクリークを失うと，そのプリミティブは，開始ノードが解放されるのを待って再実行される．



（a）ブロック
（a）Block.



（b）衝　突
（b）Collision.

図3 クリーク間の干渉
Fig. 3 Interference between cliques.

3.2 メッセージ転送による処理手順の記述

3.2.1 メッセージの種類

プリミティブの実行に際しては，次のようなメッセージが転送される．すべてのメッセージには，対象とするオブジェクトのオブジェクト識別子が付加される．また，メッセージを受け取ったノードでは，それがどの枝から到着したかを判別できるものとする．

1）起動メッセージ

クリークの拡張を指示するメッセージであり，それぞれのプリミティブについて一つ存在している．

2）成功返答メッセージ

プリミティブが部分木について完了したことを伝達する．コピーリクエストに関する成功返答メッセージには，対象とするオブジェクトの内容が付加される．

3）失敗返答メッセージ

プリミティブの後退を知らせるメッセージ．

4）解放メッセージ

処理を完了したノードをクリークから解放する．

3.2.2 実行時管理データの構成

それぞれのノードではオブジェクトに対応したエントリを管理する．エントリは，対応するオブジェクトに関する処理の実行状況を保持するものであり，ノードにメッセージが到着するたびにアクセスされる．エントリの構造を次に示す．エントリの各フィールドの説明でフィールド名の次に中括弧にくくって示したの

は，それぞれのフィールドが取りうる値である．以下の説明において，nは木の枝分かれの数である．2分木では$n=2$となる．一つの節に接続する枝には，0からnまでの数値が振られているものとする．

1）節のエントリの構造

[failureflag]-[dir]-[op0..op$n$]-[wait0..wait$n$]

- failureflag {true, false}

処理中のプリミティブの実行が失敗したことを示す．

- dir {close, nil, 0..n}

現在処理中のプリミティブがやってきた方向を0..nで示す．nilはプリミティブが実行されていないときの値で，closeは返答メッセージの転送後，解放メッセージ待ちを示すためにとる値である．

- op0..op$n$ {nil, copy-request, purge, invalidate}

枝0..nから来て，実行中あるいはブロック中のプリミティブを示す．そのようなプリミティブが存在しない枝については nil となる．

- wait0..wait$n$ {nil, copy-request, purge, invalidate}

枝0..nに転送された起動メッセージで，それに対する返答をまだ受け取っていないものを示す．起動メッセージの転送が行われていない枝については nil となる．

2）葉のエントリの構造

[closeflag]-[op0]-[op1]-[wait0]

- closeflag {true, false}

解放メッセージ待ち状態であるか否かを示す．

葉は，一つの枝しか持たないが二つの op フィールド，op0 と op1 を持つ．op0 は枝から到着したプリミティブを示し，op1 はプロセッサから起動されたプリミティブを示す．

3.2.3 メッセージ転送手順とノードでの処理

本アルゴリズムの詳細な記述は，煩雑な場合分けを伴うため，2進木上での処理の進行の様子をいくつかの典型的な場合について図示する．**図4**は，**図5**で使用されるノードの図式表現の例を示している．ノードの中に描かれている矢印は，dir フィールドの値を示し，それぞれの枝に添えられたアルファベットは，op フィールドおよび wait フィールドを示す．op フィールドの値は，以下の略号で現す．ただし nil の場合は，値を記述しない．wait フィールドの値は，略号に下線を引いて現す．

c : copy-request

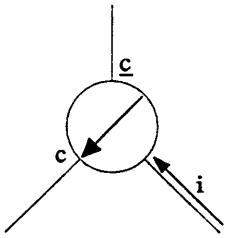


図4 ノードとメッセージの図式表現
Fig. 4 Schematic representation of a node and a message.

i : invalidate

p : purge

枝上の矢印は，転送中のメッセージを示す．メッセージの値は以下のとおりである．

c, i, p : 各種起動メッセージ

s, fail : 成功，失敗返答メッセージ

free : クリーク解放メッセージ

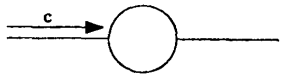
図5には，図4のような3本の枝を持つノード以外に2本の枝しか持たないノードも示されているが，これは，もう1本の枝を介してのやり取りがまったく生じない状況で使用する簡略表現である．

i）クリークの拡張と縮小

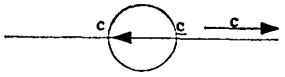
図5aに，クリークの拡張と縮小の様子を示す．起動メッセージが到着すると，まず，エントリと配置データをアクセスする．その結果，他のプリミティブとの干渉が検出されなければ，メッセージが到着した枝に対応する op フィールドに起動するプリミティブを登録し，dir フィールドの値を設定する．そして，必要な枝に起動メッセージを転送する．起動メッセージを転送した枝に対応する wait フィールドにプリミティブを登録する．この wait フィールドは，返答メッセージを受取るとクリアされる．クリークの縮小は，返答待ちがすべて解決された後，実行される．図では，左方から来たコピーリクエスト要求が干渉を生じずに処理される場合の全過程を示している．close 状態を設けて他のプリミティブの影響を排除することにより，メッセージの到着順が保証されないという仮定の元で，ノードのクリークからの安全な解放を保証できる．

ii）ブロックの処理

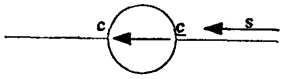
図5bに，ブロックの発生を示す．左下方から来たコピーリクエストが実行されている時に，右下からインバリデートが到着した場合を示している．到着したインバリデートは，op フィールドに登録されるが，



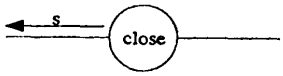
1) 起動メッセージの到着
1) Arrival of starting message.



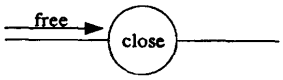
2) 起動処理，メッセージ転送
2) Begin process, message transfer.



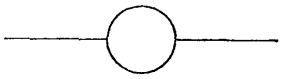
3) 返答の受け取り
3) Receive reply.



4) 返答の転送
4) Send reply.

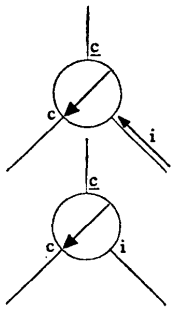


5) 解放メッセージの受け取り
5) Accept "free" message.

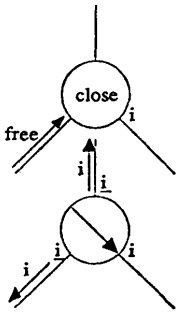


6) 処理終了
6) Process completion.

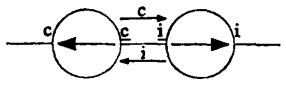
(a) クリークの実現
(a) Realization of the clique.



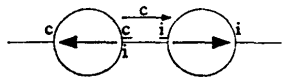
(b) ブロックの発生
(b) Blocking.



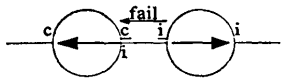
(c) ブロックされていた処理の起動
(c) Unblocking.



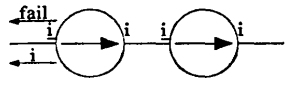
1) メッセージの交換
1) Message exchange.



2) 強いメッセージの到着，ブロック
2) Arrival of strong message.



3) 弱いメッセージの到着，失敗返答
3) Arrival of weak message.



4) 強い処理の進行，失敗返答の転送
4) Strong primitive proceeds.

(d) 衝突の発生と処理
(d) Clique collision and fall-back.

図 5 メッセージ転送によるプリミティブの実行
Fig. 5 Execution of primitives by message-passing.

ブロックを検出した時点でその処理は中断される．

図5cは，プリミティブのブロックからの解放を示す．close 状態にあるノードが解放される時に，op フィールドに登録されているプリミティブの中から一つを選択して実行する．選択，起動手順の詳細は後述．

ⅲ) 衝突の処理

図5dは，衝突の処理である．本システムでは，メッセージの処理が中断されないため，衝突は常に起動メッセージの交換として生ずる．起動メッセージの到着時，到着枝に対応する wait フィールドにプリミティブが登録されていたら，衝突の処理を行う．衝突の生じたノードでは，起動メッセージと wait フィールドのプリミティブの優先度を比較する．プリミティブの間には，任意の非循環的な優先度を設定することができる．同一の優先度を持つプリミティブの場合には，上方のノードを占めているプリミティブが優先される．もし起動メッセージが優先権を失えばそのメッセージに対する失敗返答をおこない，得ればプリミティブを op フィールドに登録して処理を終了する(ブロックの発生)．

失敗返答の受け取りは，その枝へ拡張したクリークが後退したことを示す．失敗返答を受け取ったノードでは，転送したすべての起動メッセージに対する返答が終了していたら，プリミティブの選択，起動手順を実行する．さもなくば，failureflag フィールドを

true にする．以後，処理が進行し，すべての返答が実行された時点で failureflag フィールドがチェックされる．true であることが検出されると，実行中のプリミティブに関する成功返答と終了処理（図5 a，3)，4)）を実行する代わりに，プリミティブの選択，起動手順を実行する．

iv）プリミティブの選択，起動手順

もし failureflag が true なら，false にセットする．op フィールドの値をチェックし，登録されている中で最も高い優先度を持つプリミティブを特定する．その方向に dir フィールドを設定し，プリミティブの起動メッセージ転送を実行する．もし転送しようとする枝の op フィールドにプリミティブが登録されていたら，それをクリアし，起動メッセージの転送を実行してから，その枝への失敗返答を行う．図5 d，4) は，このような処理の例である．

4. 正当性の検証

以上の処理の正当性検証の概略を示す．ここでは，始めに処理の一貫性の検証を行い，次にプリミティブが必ず終了することを示す．

4.1 処理の一貫性

プリミティブの実行がオブジェクトの位置を正しく反映しているオブジェクト配置データを参照して行われることを示す．クリークによる排他制御が行われるため，オブジェクト配置データの過度的な更新状態による一貫性の乱れがクリークの外部に反映されないことを示せば十分である．分岐を持たないプリミティブについては，縮小を開始したクリークがもはや他のクリークとの干渉を生じないことから，一貫性を保証できる．インバリデートについては，クリークの分岐点までの縮小が途中で干渉を発生せずに行われること，そして縮小が分岐点まで到達した時点で部分木に関する処理が終了し，オブジェクト配置データの一貫性が回復することから，一貫性を保証できる．

4.2 停止性の検証

任意の一つのクリークについて考える．そのクリークの一つの拡張端を見ると，その拡張端は，いつか必ず次の状況の一つに出合い停止することになる．

a．ブロックして停止する

b．他の拡張端と衝突し押し戻される

c．最大拡張点に到達する

ブロックが生じた場合，ブロック先の部分木について考える．考察の対象を部分木に限定することがプリミティブ数の減少をもたらすため，部分木に関する正当性を仮定するのは妥当である．したがってブロックしている拡張端は，必ず部分木中のプリミティブの端点と相互作用を生ずる．言い換えると次の状況のいずれかが必ず生ずる．

d．ブロック相手のクリークが縮小し進行可能になる

e．他の拡張端に押し戻される

dは通常の進行と同等であり，eはbと同等である．そこでbについて考える．この場合は衝突相手の拡張端の進行を追跡する．初期状態で存在するクリーク数が有限であること．拡張端が進行する限りその拡張端と他の拡張端の間にすでに確立した優先権が逆転しないこと，そしてプリミティブの優先度が非循環的に定義されていることから，以上の手順により最大拡張点に到達する拡張端を必ず発見することができる．

これにより，枝分かれのないプリミティブに対しては，即座に終了を保証することができるが枝分かれのあるプリミティブについては必ずしも保証できない．そこでクリーク全体について考えると，構造が木であることからクリーク同士の接触が必ず1カ所で生ずることが分かる．したがって拡張端での優先権の決定をクリーク同士の優先権の決定と同一視できる．これよりプリミティブの終了が保証される．

以上より，どのような状態からでも終了するプリミティブが存在することが証明された．これより，任意の初期状態からすべてのプリミティブが終了することを結論できる．

5. アルゴリズムの評価

本アルゴリズムの特性に関する考察を行う．本アルゴリズムは，データオブジェクト配置状況の管理枝法として一般性を保持しているため，幅広い応用に適用可能である．本章では，性能面での限界に関する考察を行うため，細粒度の並列処理を指向した分散共有メモリシステムについて検討する．

5.1 システムの構成

システム全体は，木構造をとる．考察の単純化のため，完全k進木に限定する．葉ノードにプロセッサとローカルメモリを置き，葉および節ノードに上記の手順を実行できるコントローラを置く．メモリ空間は，ページに分割されており，各ページの配置状況を本アルゴリズムで管理する．プロセッサのアクセスに応じてプリミティブを起動して，メモリ空間の一貫性を維

持する．

このようなシステムを仮定し，性能向上のため，次のような実装を行うものとする．

1）ノードキャッシュの付加

各節ノードに，ページを保持するキャッシュメモリを設ける．これをノードキャッシュと呼ぶ．各節のノードキャッシュは，コピーリクエストによりページが移動される際にデータを満たされ，ページの削除を配置ビットに反映する際に無効化される．コピーリクエストの起動時，ノードキャッシュがヒットしたら，そこからデータを読み出して処理を行う．これによりコピーリクエストの処理を高速化できる．

2）メッセージ処理の簡略化

このような実現では，メッセージの到着順保証が比較的容易であるため，3.2 節で述べた手順の一部を簡略化できる．例えば，処理が完了したノードを close 状態にせず，即座に解放することができる．

ここで実行される操作は，ページの定まった収納場所を前提としないことを除けば，並列キャッシュシステムの一貫性プロトコル[1),3)~5)]と同一視できる．したがって，本章では，主としてキャッシュ一貫性プロトコルとの比較を通じて性能評価を行う．

5.2 管理データ量の見積もり

アルゴリズムの実行に要する管理データ量は，システムの拡張性を制限する要因である．本節では，必要な管理データ量を算出し，システムの拡張性について検討する．

まず，各ノードで必要な管理データ量について検討する．前章までの説明では，実行時管理データ（エントリ）を各ページごとに用意して処理することを仮定した．しかし，あるページに関するエントリは，そのページを処理するプリミティブが実行されていない時には，有効な情報を保持していない．そのため，連想メモリ等を使用して動的なエントリ管理を行うことによって，使用メモリ量を抑えることができる．必要とされるエントリ数は，ページの総数でなく，各プロセッサの処理実行状況に依存するため，ページ数と独立にエントリ数を管理することは健全である．

エントリ数制限は，各ノードにおいて実行中のクリーク数を管理することによって実現できる．クリーク数を数えておき，それが一定数に達したら，クリークのどれかが縮小または後退するまで次の起動メッセージの処理を延期する．ここで 4.2 節で述べた優先権を，異なったページを処理するクリークの間にも適用する．あるクリークが別のクリークの縮小を待つという関係の発生を，後者の前者に対する優先権の成立と考える．デッドロックが発生しないことを保証するには，この関係を考慮に入れた優先権について循環的な優先権の決定が発生しなければ十分である．そのためには，次の二つの条件が満たされればよい．

a．上方から到着したクリークが下方から到着したクリークの縮小を待つことがない

b．優先順位の高いプリミティブが低いプリミティブを待つことがない

a を保証する方法の一つは，上方の枝から到着したクリークと下方の枝から到着したクリークに別々にクリーク数管理を適用することである．b を保証する方法の一つは，優先度の異なるプリミティブごとにクリーク数管理を行うことである．

以上のエントリ管理により，各ノードで必要なエントリ数を必要に応じて小さくおさえることができる．したがって，エントリ数は，システムの規模を規定する要因としては2次的なものとなる．以後の議論では，管理データとして，ページの配置状況を示すビットのみを考える．その量は，ノードに接続する枝の本数と同一のビット数である．ただし，以下の考察において N はプロセッサ数，M はプロセッサ当たりの記憶容量（ビット），k は木の枝分かれの数，p はページサイズ（ビット）である．

物理メモリへのページ割り当てを一意に決定し，ページが割り当てられている部分木に関する管理データのみを用意する．管理データ量が最大となるのは，部分木に含まれるノード数をすべてのページに関して合計したものが最大となる場合である．すべての部分木のすべての葉の合計数は，割り当て方法によらず一定であるため，それぞれの部分木で葉あたりのノード数を最大にすることによってノード総数を最大にすることができる．すなわち，すべてのページの割り当てが二つの葉に対して行われており，それらの葉を結ぶ道が必ず木の根を通るような場合，管理データ量は最大となる．したがって，最悪割り当てを行った場合，一つのページに対応する部分木を構成するために，葉ノードが二つ，根ノードが一つ，それ以外のノードが $2(\log_k N-1)$ 必要となる．葉には2ビット，根には k ビット，それ以外のノードには $k+1$ ビットのデータが必要であることから，このときのページ当たりのビット数は：

$$4+k+2(\log_k N-1)(k+1)$$

$=2(k+1)\log_k N+2$ (1)

割り当てられるページ数は物理メモリ量の2分の1となるため，管理データ総量は：

$NM((k+1)\log_k N-k/2+1)/p$ (2)

これより，管理データ量のオーダは，$O(N\log_k N)$となる．これは，システムの拡張性を確保できる大きさだと思われるが．実装を考えると，一つのノードへのデータの集中が問題になる可能性がある．部分木の割り当て方法から明らかなように，最悪割り当てを行った場合，根と葉以外では，あるノードはその下のノードのk倍の管理データを保持しなければならない．この問題は，複数の木をマッピングできる通信路を使用し，管理対象のページを木の間に分散させることによって緩和できる．

5.3 応答時間の見積もり

プリミティブ実行時，各ノードでは次のような処理が行われる．

a．処理するメッセージを決定
b．管理データのアクセス
c．管理操作実行
d．メッセージ転送，管理データ更新

それぞれのプリミティブは，この一連の操作（以後これをサイクルと呼ぶ）を以下に示す回数繰り返すことにより実行できる．ただし，hは，木の葉を0として数えたクリークの高さ，Pは一つのページをリンクを通して転送するために必要なメッセージ数である．ここでは，一つのメッセージで転送できるデータ量がメッセージの種類によらず一定であることを仮定しているため，Pはページの大きさに比例する．

コピーリクエスト：$2h+P$〜$4h+P$
インバリデート　：$4h+1$
パージ　　：$2h+1$〜$4h+1$

これらは，処理完了までに要するメッセージ転送回数の見積もりである．異なったノードで並列に実行されるものを含めた実行サイクルの総数は，ページの重複度に依存する．

これより，本システムの応答時間が，ページが配置されている部分木の高さによって決定することが導かれる．したがって，応答時間の最大値は，部分木に含まれるプロセッサ数の対数に比例し，プロセッサ総数に依存しない．

5.4 ディレクトリを使用したプロトコルとの比較

本方式の重要な特性である拡張性と並列性は，ディレクトリを使用したキャッシュ一貫性プロトコル[3)~5)](以後，ディレクトリ方式と呼ぶ）でも得られる．これは，ページの配置状況データ（ディレクトリ）をメモリモジュールで集中して保持し，メモリモジュールとの通信で管理を実行するものである．複数のメモリモジュールをネットワークで接続することにより，負荷分散を行う．以下，本方式とディレクトリ方式の比較を行う．

まず，本方式がディレクトリ方式の一般化となっていることを確認する．$k=N$と定めると，本方式は，ディレクトリ方式と同一視できる．この時，ディレクトリ方式のメモリモジュールに当るのは，ノードキャッシュである．そこでここではkの大きさに依存する性質について検討する．

小さなkの利点は，処理の局所化，分散化である．kが大きくなると，データオブジェクトの配置に局所性が成立しても，それを処理の局所性に反映するのが難しくなる．また，要求が特定のノードに集中する可能性も高くなる．それに対し大きなkの利点は，5.2節で検討したサイクル数が少なくてすむことである．

実際のハードウェア上でのkの選択は，通信路の形態に強く依存している．特に，ここで検討したシステムのように通信路上に木構造がマッピングできる場合には，通信路の枝分れ数と同一のkを選択するのが適切であると思われる．木構造をマッピングできる通信路としては，他にハイパキューブ型マルチステージネットワーク等が挙げられる．

6. おわりに

データオブジェクトの配置状況を管理するアルゴリズムを示し，その正当性検証，性能見積もりを行った．本アルゴリズムは，以下のような特徴を持つ．

- 配置状況に応じた対数オーダの実行時間
- 細粒度の処理にも適合する管理データ量
- 処理の有効な局所化，分散化

これらの特徴により，本アルゴリズムは，分散共有メモリシステムから分散ファイルシステムまで，幅広い応用に適用可能である．

現在，マルチステージネットワークを利用して分散共有メモリを実現するマルチプロセッサシステムを設計中である．このマルチステージネットワークは，配置管理アルゴリズムを実行できるノードによって構成されており，ノード間の負荷分散機能を持っている．

参考文献

1) Archibald, J. and Baer, J.: Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model, *ACM Trans. Comput. Syst.*, Vol. 4, No. 4, pp. 273-298 (1986).
2) Bisiani, R. and Ravishankar, M.: PLUS: A Distributed Shared-Memory System, *Proceedings The 17th Annual International Symposium on Computer Architecture*, pp. 115-124 (1990).
3) Censier, L. M. and Feautrier, P.: A New Solution to Coherence Problems in Multi-cache Systems, *IEEE Trans. Comput.*, Vol. C-27, No. 12, pp. 1112-1118 (1978).
4) Chaiken, D., Fields, C., Kurihara, K. and Agarwal, A.: Directory-Based Cache Coherence in Large-Scale Multiprocessors, *IEEE Comput.*, Vol. 23, No. 6, pp. 49-58 (1990).
5) O'Krafka, B. W. and Newton, A. R.: An Empirical Evaluation of Two Memory-Efficient Directory Methods, *Proceedings The 17th Annual International Symposium on Computer Architecture*, pp. 138-147 (1990).
6) Li, K. and Hudak, P.: Memory Coherence in Shared Virtual Memory Systems, *ACM Trans. Comput. Syst.*, Vol. 7, No. 4, pp. 321-359 (1989).
7) Warren, D. H. D. and Haridi, S.: Data Diffusion Machine—A Scalable Shared Virtual Memory Multiprocessor, *Proceedings of the International Conference on Fifth Generation Computer Systems 1988*, pp. 943-952 (1988).

(平成2年8月9日受付)
(平成3年4月9日採録)

山崎 剛 (学生会員)

昭和40年生. 平成2年筑波大学第3学群情報学類卒業. 現在, 同大学大学院博士課程に在学中. 並列処理に興味を持つ.

和田 耕一 (正会員)

昭和31年生. 昭和53年神戸大学電気工学科卒業. 昭和59年同大学院博士課程修了. 同年神戸大学自然科学研究科助手. 昭和62年筑波大学電子・情報工学系, 現助教授. 学術博士. 計算機アーキテクチャ, 並列処理システムに関する研究に従事. ACM, ソフトウェア科学会各会員.