（1994. 7. 23）

シミュレーションを用いた疑似フルマップ方式の定量的評価

佐藤充[*1], 三吉貴史[*2], 松本尚[*2], 平木敬[*2], 田中英彦[*1]
[*1]東京大学工学部, [*2]東京大学理学部
東京都文京区本郷 7-3-1

概要

分散共有メモリマシンでは、メモリアクセスのレイテンシを隠蔽するためディレクトリキャッシュを用いることが多い。本稿では、疑似フルマップ方式と呼ばれるキャッシュ方式をシミュレータを用いて評価した結果を報告する。シミュレーションの結果、相互結合網のマルチキャスト機能を有効に利用することによって、メモリトランザクションのレイテンシを小さくし、総パケット数を少なくすることができることがわかった。

Quantitative Evaluation of Pseudo-Fullmap Directory using Simulation

Mitsuru SATO[*1], Takashi MIYOSHI[*2], Takashi MATSUMOTO[*2],
Kei HIRAKI[*2], Hidehiko TANAKA[*1]
[*1]Faculty of Engineering, University of Tokyo,
[*2]Faculty of Science, University of Tokyo
7-3-1 Hongo, Bunkyo-Ku, Tokyo

Abstract

In Distributed Shared Memory System(DSMS), directory cache is often used to hide the remote memory access latency. In this paper, we report result of the evaluation about directory cache system called "Pseudo-Fullmap Directory" using simulator. We find that using multicast feature of the network effectively causes latency of memory transaction to be reduced, and total number of packets to be fewer than full map directory system.

1 はじめに

分散共有メモリマシンでは、遠隔ノードが保有するメモリを自由にアクセスすることができる。一般に遠隔ノードとの通信はプロセッサの速度やローカルメモリのアクセス速度に比べて低速であるので、処理のボトルネックとなりやすい。また、大規模な並列計算機になると、データや処理が分散され、頻繁に遠隔ノードのメモリに対するアクセスが生じることが予想される。このため、分散共有メモリマシンでは、遠隔ノードに対するメモリアクセスのレイテンシをいかに小さくするかということが重要な問題となる。

遠隔ノードに対するメモリアクセスのレイテンシを小さくするには、物理的にデータ転送を高速にする方法に加えて、レイテンシを隠蔽する方法がある。このレイテンシの隠蔽方法には、キャッシュ、プリフェッチ、コンテクストスイッチ等の手法が用いられている [1]。

最近アクセスされたデータのコピーを、プロセッサの近くの高速なメモリに保存しておくキャッシュ方式は、データアクセスにローカリティがあると、遠隔メモリアクセスのレイテンシを大幅に減少することができる。しかし、複数のプロセッサが並列に動作している並列計算機では、メモリトランザクションにおけるキャッシュのコンシステンシ維持が重要な問題となる。

本発表では、そのコンシステンシ保持方式のひとつであるディレクトリ方式 [2] をとりあげ、特にその中でも疑似フルマップ [3] と呼ばれる方式を用いたシステムについてのシミュレーションの結果を報告する。

2 ディレクトリ方式

並列計算機におけるキャッシュシステムでは、あるノードがコピーを持っているメモリに対する書き込みが生じた場合、場合によってはコピーと元のデータの間に整合性がなくなる可能性がある。このようにキャッシュ同士やキャッシュとメモリ間の不整合が生じるのを避けるため、システムは何らかのキャッシュコンシステンシ維持方式を用いる必要がある。

ディレクトリ方式は、主メモリのそばにディレクトリと呼ばれる小量のメモリを備え、コピーを持っているノードを管理する方式である。一般的にディレクトリ方式では、メモリトランザクションが生じるとディレクトリを参照して、そのメモリのコピーを持っているノードを調べ、それぞれのノードに対してメモリトランザクションが生じたことを通知する。

ディレクトリ方式では、コピーを持っているノードの情報の、ディレクトリでの表現方法によって分類することができる [2]。最も単純な実装であるフルマップ方式では、ディレクトリ1ビットに1ノードを対応させ、ディレクトリ・ビットの1/0によってコピーのある/なしを判別する。しかし、ディレクトリをこのような構成にすると、ノード数が増えれば増えるほどディレクトリ・ビットが必要になる。そのため、大規模な並列計算機では、スケーラビリティのあるディレクトリの構成が必要とされる。

2.1 疑似フルマップ

疑似フルマップ方式とは、大規模な並列計算機でもディレクトリを実現するために必要なメモリ量が巨大にならないようにした、スケーラビリティを重視したディレクトリ方式である。疑似フルマップでは、階層化放送機構をもった相互結合網 (例えば Tree Network など) を必要とする。また協調動作するスレッドはなるべく近傍のプロセッサ群にスケジューリングするというプロセッサ資源管理方針も仮定している。

疑似フルマップ方式は

- 近傍フルマップ
- 遠隔直接指定
- 階層化マップ

などの複数のマップを組み合わせて、ディレクトリキャッシュを実現する。

コピーを持つノード (以後、コピーノードと呼ぶ) が少なく、それらコピーノードが元のメモリを持ったノード (以後ホームノードと呼ぶ) の近傍に集められている場合は、近傍フルマップを用いる。近傍フルマップでは、ある範囲内にある近傍ノードをそれぞれディレクトリの1ビットに対応させ、フルマップと同様に管理する。

コピーノードが、近傍フルマップで表されるノード数を越えた場合や、近傍フルマップで表現できない遠方のノードがコピーを所有した場合は、階層化マップを用いてコピーノードを管理する。

階層化マップは、階層化放送機構の構造と対応したマップを用いる。ノードをどのようにマップするか

という方法には、近傍詳細型 (Local Precise Remote Approximation: LPRA)、遠隔詳細型 (Local Approximation Remote Precise: LARP)、同一マップ型 (Single Map: SM) 等が考えられている [4]。

この中から、ここでは近傍詳細型を例にとって説明する。

近傍詳細型階層化マップでは、通信距離が近いノードに関しては共有ページの存在を細かく管理し、通信距離が離れるにつれて粗く (放送機能の階層単位で) 管理する。つまり、遠くのノードへ通信する場合は、階層化放送機構でそのノードと通信可能になる階層まで遡り、そこから目的地を含む部分木に向かってメッセージをブロードキャストする (図 1, 2)。

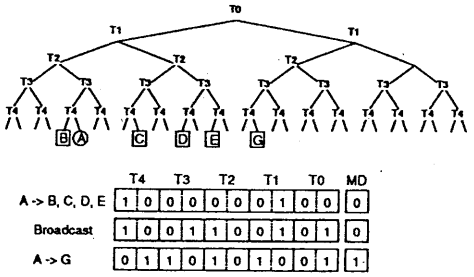


図 1: 階層化マップ (近傍詳細型)

このようなマップを用いることによって、すべてのノードをフルマップで管理するのに比べて、ディレクトリに必要なメモリ量を削減することができる。また、階層化放送網の階層単位で管理しているので、ブロードキャストを用いたリミテッド・ディレクトリ方式 (Dir$_i$B[5]) などに比べて無駄パケットの数を少なくすることができる。

また、コピーを持つノードが、遠方の (つまり、近傍フルマップで表現できる範囲外の) ノードただひとつである場合、この階層化マップの表現を利用して、コピーノードを直接指定することができる (遠方直接指定)。

さらに、階層化マップを用いることによって、Acknowledge(Ack) のコンバイニングを行なうことができる。

フルマップでは、メモリトランザクションはホームノードから各コピーノードに対して 1 対 1 で通信され、その結果の Acknowledge(Ack) は、逆にコピーノードからホームノードへ返信される。ところがこの方式では、コピーノードが多数存在した場合、多数の Ack の返信がホームノードに集中してしまい、相互結合網に対する負荷の増大、レイテンシの増大につながる。特に大規模な並列計算機では、このようにコピーノードが多数存在する可能性が高くなり、キャッシュのコンシステンシ管理のオーバーヘッドが急激に増大することが予想される。

スケーラビリティを重視した疑似フルマップでは、この問題を回避するため、階層化マップを用いた通信では Ack のコンバイニングを行なう。

階層化放送機構を介した通信の Ack は元のメッセージが送信された経路を逆にだどって返信される。部分木へのブロードキャストであった場合は各階層で Ack のコンバイニングがなされる。末端のすぐ上のノードでは、すべての子どもから Ack または Dack(Dummy Acknowledge) が返った時点で、Ack をさらに上位の親に返す。これを元の部分木のルートノードまで繰り返していく。部分木のルートノードまでこの操作が終了すると、自分より上層の部分木への送信がなされた場合は、上層から Ack が返送されるのを待ち、Ack を元のメッセージ送信ノード方向の下層のノードに返送する。上層への送信がなされていない場合は、上層からの Ack を待つことなしに、Ack を下層のノードへ返送する (図 2)。

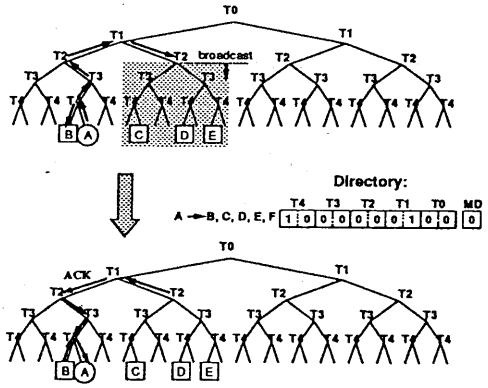


図 2: Ack のコンバイニング

このように途中で Ack を回収して行くことによって、ホームノードでの Ack の集中を避けることができる。この場合、ホームノードに対しては、ただひと

つの Ack が返信されることになる。

3 シミュレーション

疑似フルマップ方式の利点、問題点を定量的に評価するため、並列計算機のシステム・シミュレータを作成し、その上でプログラムを実行し、挙動を調べた。

3.1 シミュレータ

シミュレータは、相互結合網により複数のノードが接続された構成をとる。

3.1.1 相互結合網

疑似フルマップを実現するためには、対象とするシステムのネットワークは階層化放送機構を持っている必要がある。今回のシミュレーションではネットワークとして4進木+mesh ネットワークを用いた。

3.1.2 ノード構成

シミュレータの各ノードは Router, Network Interface Processor(NIP), Access Pattern Generator(APG), Local Memory から成る (図3)。

それぞれのブロックの機能と役割は以下のようになっている。

APG

APG ではアプリケーションを実行し、然るべきタイミングで NIP にノード間にまたがるメモリトランザクション要求を出すユニットである。APG はアセンブラで書かれた処理コードを読み、メモリの内容を参照し、NIP にパケット生成命令を発行する。APG はまた、メモリアクセスごとに invalidate/update を切替える機能 [6] を備えている。さらに、APG では Memory fence を張る/張らないに応じて、Acknowledge を待つ/待たないを選択することができる。

NIP

NIP は

1. パケットの生成
2. ホームにおけるメモリトランザクションの単一性の保証
3. 中継ノードにおける Ack の収集
4. メモリ属性の更新

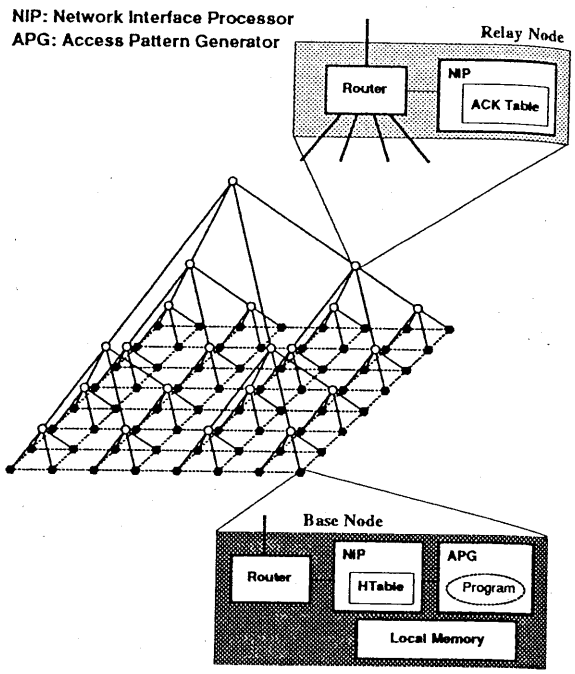


図 3: シミュレータのノード構成

等を行なうユニットである。

メモリトランザクションが発行されると、ホームノードの NIP において調停が行なわれ (あらかじめ単一性を保証する必要がないことが分かっている場合はこの限りではない)、実行される。階層化マップを用いた invalidate/update が実行されると、中継ノードの NIP は自分の ACK Table にエントリを生成し、定められた数の下位ノードから Ack が返ってくるのを待つ。そして、ACK Table のエントリが 0 になると、あらかじめ登録されていた元ノードに対して Ack を発行する。このようにして階層網の各レベルにおいて Ack を回収し、1つの Ack を上位へ返す。そうして最終的には invalidate/update を発行したノードにただひとつの Ack が返ってくることになる。この時点で application(APG) は自分の発行した invalidate/update が終了したことを知り、次の動作に入る (non-blocking プロトコルの場合 Ack を待たずに次の動作に入る)。

Router

ネットワーク・ルーターは上位ノードに対して1、下位ノードに対して4、NIP に対して1の合計6ポートを

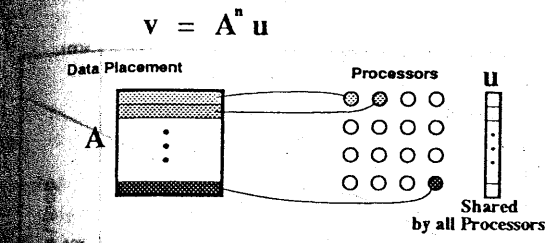


図 4: 行列・ベクトル積

持つ。また、デッドロック回避のため channel を 2 つ設けているので、それに応じてバッファも input/output 各 2 つずつ持っている。

Router は

- directory を直接解釈することによって、Router 上での multicast を実現
- あるレベルのノードの下位に接続しているすべてのノードに同じパケットを送る broadcast 機能
- base mesh における 4 方向 local multicast 機能

を備えている。さらにパケットの順序制御のため、内部カウンタを用い、Router 内部でのパケットの追い越しが起こらないようルーティング制御を行なっている。

3.2 アプリケーション

測定には、以下の 3 つのアプリケーションを用いた。

行列ベクトル積

行列とベクトルのかけ算で、連立 1 次方程式の解を求めたり、有限要素法を行なったりする場合に用いられる。典型的な数値演算アプリケーションの例である。

並列化は以下のような方式を用いた。まず各ノードはそれぞれ行を担当し、自分の担当行のデータを保有する。ベクトルは全員が共有して保持する (図 4) 。定常状態では、自分の担当行と共有ベクトルのベクトル積をとり、共有ベクトルの担当部分をアップデートする。そして全員が共有ベクトルをアップデートするのを待ち、次のループに入る。

したがって、1 ループに 1 回、全員が共有データに対する書き込みを行なうことになる。この時の invalidate/update の方法として、以下の 4 つの方式を実装した。ここで「単純な invalidate/update」は、ホームノードによる調停を受けずに、コピーノードを直接 invalidate/update する方式のことである。

- 単純な invalidate(invalidate)
- 単純な update(update)
- 各プロセッサの計算開始場所をずらし、update を用いてレイテンシの隠蔽を行なう方式 (opt-1)
- 各プロセッサの計算開始場所をずらし、local multicast による隣接プロセッサ間通信を用いて、通信距離を最小にする方式 (opt-2)

Red-Black SOR

Successive Over-Relaxation は、周囲の状況から自分の値を計算する計算である。SOR では全体を Red Phase と Black Phase の 2 つの Phase に分けて計算することができる。

SOR では図 5のようにデータを配置する。このため、このプログラムでは自分の周囲 4 ノードに対する invalidate/update しか生じない。

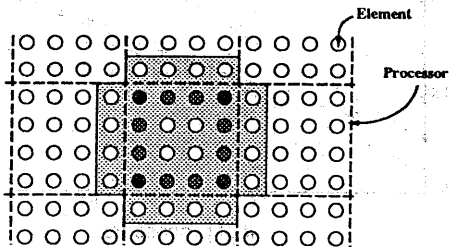


図 5: Red-Black SOR

FFT

FFT では、各ノードに一定数の要素を割り当て[1]、1 対 1 通信を用いて他のノードと通信し、計算を進める。したがって、このプログラムでは 1 対 1 通信のみで計算が実行される。

これらのアプリケーションは、それぞれ階層化マップ、近傍フルマップ、遠隔直接指定の例となっている。

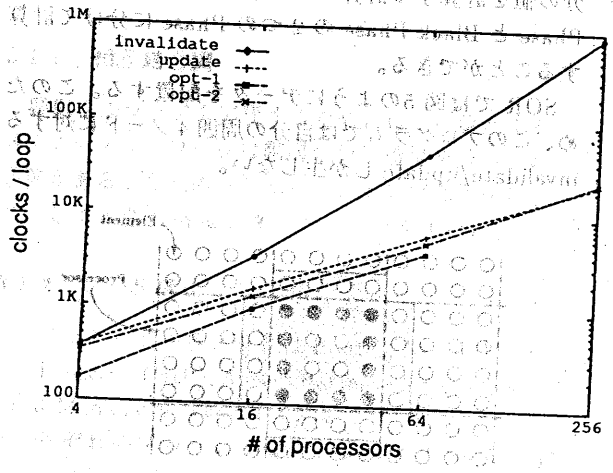
4 測定

4.1 アプリケーション実行速度

測定は、ノード数 2 ~ 256 の範囲で行なった。

[1]今回は 4 要素

データ配置はプログラム実行前にあらかじめ決定されている。各ノードのローカルメモリは、与えられたデータ配置によって初期化されている。APGに与えられるプログラムは、定常状態になった時のループである。こうすることによって、一般にプログラムを実行した時に行なわれるさまざまな初期化処理を省略し、プログラムがループを繰り返しているという定常状態を実現することができる。本測定では、このループ1回を実行するクロック数を測定した(図6, 7, 8)。

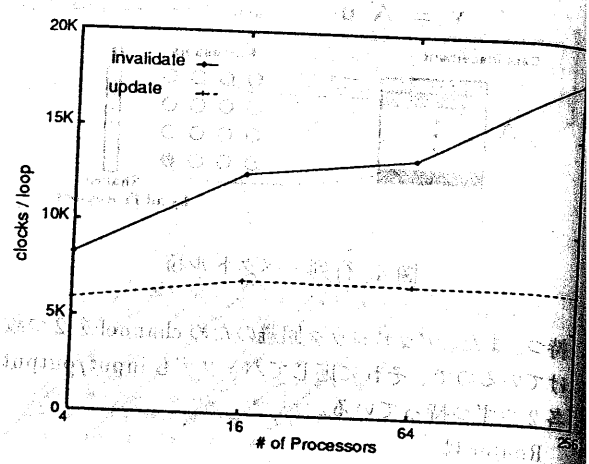


invalidate: キャッシュのコンシステンシ管理に invalidate を使用
update: キャッシュのコンシステンシ管理に update を使用
op-1: base mesh を用いて通信量を削減
op-2: 計算の順序を入れ換え、レイテンシの隠蔽を行なう

図6: 行列ベクトル積

図6, 7, 8から、次のことがわかる。

- 図6より、invalidateよりは単純updateを用いた最適化の方が、アプリケーションを高速に実行できることがわかる。特にinvalidateを用いた場合とupdateを用いた場合の差は大きく、256ノードで約100倍にもなる。また図7においても、invalidateに比べてupdateの方が3倍ほどの速度が得られた。このことから、行列ベクトル積やSORなどでは、updateを用いることの利点がかなり大きいことがわかる。このように、アプリケーションに応じたキャッシュプロトコルを用いることによって、アプリケーションの実行速度を大幅に改善できる可能性が示されたと言える。



update: キャッシュのコンシステンシ管理に update を使用
invalidate: キャッシュのコンシステンシ管理に invalidate を使用

図7: Red-Black SOR

- 今回の測定では、各ノードに割り当てるデータの数を一定にしているので、図6等のグラフの傾きは、ノードが増えることによるオーバーヘッドの増加を表している。アプリケーションに応じた適切なプロトコルを選択することによって、全体のクロック数だけでなく、ノードが増えることによるオーバヘッドをも削減することができる。

4.2 Acknowledgeの収集

疑似フルマップ、特に階層化マップを用いた場合の利点、問題点をはっきりさせるために、シミュレータ上にフルマップ・ディレクトリを実装して、行列ベクトル積を実行し、その結果を疑似フルマップでの結果と比較した。行列ベクトル積の、1回のループを実行するのにかかるクロック数、総パケット数を比較したのが表1である。ここでは、プロトコルとして単純updateを用いている。したがって、総パケット数はupdateを実行するために必要なパケット数と等しい。

表1から、疑似フルマップの階層化マップのように効率的にマルチキャストを用いることによって、全体

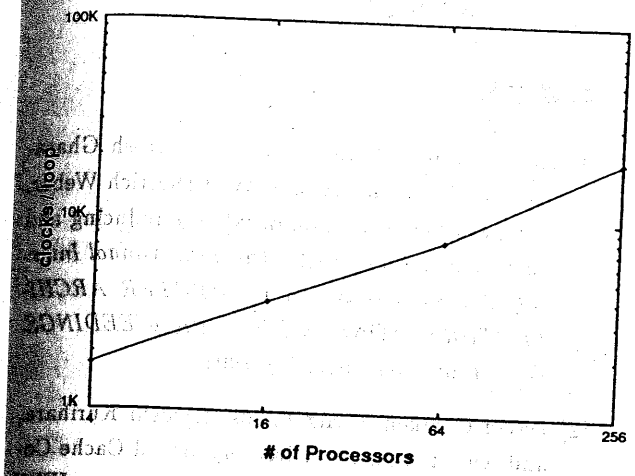


図 8: FFT

表 1: フルマップとの比較

台数	Fullmap		Pseudo	
	clock	packet	clock	packet
2×2	299	24	376	16
4×4	3285	480	1560	256
8×8	49368	8064	6033	4096
16×16	801109	130560	23378	65536

の実行速度も向上し、パケット数も少なくなることがわかる。また、その効果はノード数が多くなればなるほど顕著になる。

この原因は2つあげられる。第一に、階層化マップを用いることにより相互結合網のマルチキャスト機能を用いた通信が可能になり、同じ通信でも全体のパケット数を減らすことができる。そのため、相互結合網に対する負荷が小さくなり、通信のレイテンシが下がるのである。ただし、これは本シミュレータのように、マルチキャストパケットと1対1パケットが同じルーティング規則にのっとって通信を行なう場合のことである。1対1通信として非常にステップ数の少ないルーティングをサポートしている相互結合網の場合には、パケット数の増加に対するレイテンシの増大と、マルチキャストを行なうためのレイテンシの増大との比較になる。しかし一般的に階層化放送機構を用いると、マルチキャストは非常に低コストで実現できるので、そのような1対1通信の実現は非常に難しいであろう。

第二に、疑似フルマップ (階層化マップ) では、途中の中継ノードで Acknowledge(Ack) のコンバイニングを行なっているということである。このため、フルマップのようにホームノードでパケットの集中が起きることがなく、閉塞によるレイテンシの増大を避けることができる。

階層化マップにおける Ack のコンバイニングの様子をより詳しく見るために、Router 内での、Router から NIP へ向かうポート (図 9参照) に対するパケットの待ち行列長を調べた。その結果が図 10である。ただし図 10 は行列ベクトル積、16 × 16 ノード、単純 update の場合である。

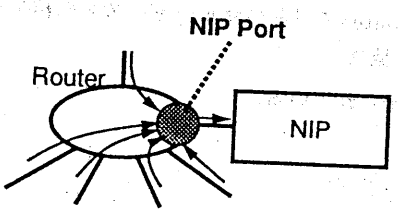


図 9: Router 内 NIP ポート

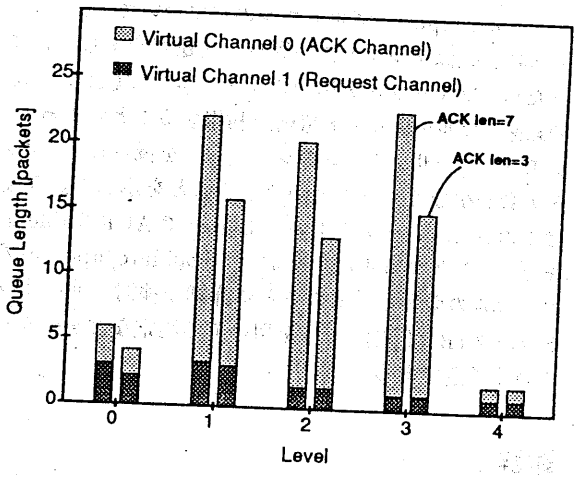


図 10: NIP ポートに対する待ち行列長

図 10からわかるように、NIP に向かうポートは、レ

ベル3またはレベル2のRouterでで混雑している。これは次のような原因によるものである。

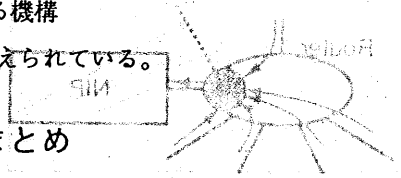
階層化マップでは中継ノードにおいて複数のAckをまとめて、1つのAckとして上位ノードに送る必要がある。このため、invalidate/updateのアドレスと必要なAckの数を保持するテーブルが必要となる。本システムでは、このテーブル(ACK Table)をNIPが保有している。そのため、invalidate/updateメッセージやACKは必ず中継ノードのNIPを通過する必要があり、そこで混雑が生じる。

この混雑は、フルマップのBase Levelほどではないが、階層化マップを用いた時のシステムのボトルネックになる可能性がある。そのため、このレベル2,3における混雑を解消する何らかの措置を施す必要がある。

現在、この混雑を解消する方法として、

- RouterでのACK Tableのキャッシング
- Routerで以前のマルチキャストの結果を記憶する機構

等が考えられている。



5 まとめ

分散共有メモリシステムにおける疑似フルマップの性能を、シミュレータを用いて測定した結果を報告した。特に、疑似フルマップを用いた場合のinvalidateプロトコルとupdateプロトコルの差について評価した。さらに、疑似フルマップの利点、問題点を明らかにするため、フルマップを用いたシステムと比較した。その結果、マルチキャスト機構を利用することによって、メモリトランザクションのレイテンシを減らし、パケットの数を少なくすることができることを示した。今後はこのシミュレータを用い、RouterでACK Tableをキャッシングする手法の検討、invalidate/updateプロトコルの切替え手法のさらに詳細な検討、他のネットワーク(RDT[7])など)を用いての測定などを行なっていく予定である。

謝辞

本研究の一部は、文部省科学研究費(重点領域研究(1)課題番号04235130「超並列原理に基づく情報処理基本体系」)による。また日頃御討論いただいてい同重点領域研究の関係者に感謝します。

参考文献

[1] Anoop Gupta, John Hennessy, Kourosh Ghara chorloo, Todd Mowry, and Wolf-Dietrich Webe Comparative evaluation of latency reducing an tolerating techniques. In *The 18th Annual Inter national Symposium on COMPUTER ARCHI TECTURE CONFERENCE PROCEEDINGS* Vol. 19, pp. 254–263, May 1991.

[2] David Chaiken, Craig Fields, Kiyoshi Kurihara and Anant Agarwal. Directory-Based Cache Co herence in Large-Scale Multiprocessor. *IEEE Computer*, Vol. 23, No. 6, pp. 49–58, June 1990.

[3] 松本尚, 平木敬. 超並列計算機上の共有メモリアーキテクチャ. 電子情報通信学会技術研究報告コンピュータシステム研究会 CPSY, No. 92–26, pp. 47–55, August 1992.

[4] 工藤知宏, 松本尚, 平木敬, 西村克信, 楊愚魯, 天野英晴. 超並列計算機でのコヒーレンシ維持のためのマルチキャスト手法の 検討. 情報処理学会研究会報告 計算機アーキテクチャ, July 1994.

[5] Anant Agarwal, Richard Simoni, John Henessy and Mark Horowitz. An Evaluation of Directory Schemes for Cache Coherence. In *The 15th Annual International Symposium on COMPUTER ARCHITECTURE CONFERENCE PROCEEDINGS*, pp. 280–289, 1988.

[6] 松本尚. 細粒度並列実行支援機構. 情報処理学会計算機アーキテクチャ研究会報告, Vol. 12, No. 77, pp. 91–98, July 1989.

[7] 楊愚魯, 天野英晴. 超並列向きのプロセッサ結合網の提案. 情報処理学会計算機アーキテクチャ研究会報告, No. 96-20, October 1992.