キャッシュ制御機構内蔵型多段結合網:MINC

安川 英樹　舟橋 啓　西村 克信　塙 敏博　天野英晴

{yasukawa, funa, nisimura, hanawa, hunga}@aa.cs.keio.ac.jp

慶應義塾大学 理工学部

概要

MIN(Multistage Interconnection Network) は, 小規模なスイッチングエレメントを複数ステージ接続した結合網で, 比較的少量のハードウェアで大規模システムの実現が可能である. 一方, 最近の VLSI 技術の進歩により, スイッチングエレメント内に高い機能を与える事が可能となったため, キャッシュ自体やキャッシュディレクトリを搭載した MIN が提案されているが, これらはいずれも大量のメモリもしくは遅延を必要としてしまう. そこで本論文では, 縮約された階層ビットマップと少量の枝刈りバッファを利用する、キャッシュ制御機能を備えた多段結合網 MINC(MINwith Cache control mechanism) を提案し, ソフトウェアシミュレーションによる評価を行なった.

Multistage Interconnection Network with Cache control mechanism: MINC

Hideki Yasukawa　Akira Funahashi　Katsunobu Nisimura
Toshihiro Hanawa　Hideharu Amano
Keio University

Abstract

Multistage Interconnection Network (MIN) has been used for connecting processors and memory modules in a large scale multiprocessor. While multiple memory modules can be accessed simultaneously with a reasonable amount of hardware in such a machine, coherent cache is difficult to be implemented. Some MINs including cache or cache directory in their switching elements have been proposed. However, conventional approaches require a large amount of memory and a latency for accessing the cache or directory inside switching elements. Here, a novel MIN with a cache coherent mechanism called MINC (MIN with cache control mechanism) is proposed. By introducing reduction schemes of the hierarchical bit map directory and the directory cache inside the MIN, the coherence is managed quickly with a small amount of memory.

1　はじめに

多段結合網 (Multistage Interconnection Network: MIN) は, 2×2 から 8×8 程度の小規模なクロスバスイッチから成るスイッチングエレメントを複数ステージ接続した結合網で, 比較的少量のハードウェアで大規模なシステムを実現できる. プロセッサ-メモリ間接続に MIN を用いたマルチプロセッサは, 同時に多数のメモリモジュールをアクセスすることができ, バス結合型に比べてはるかに多くのプロセッサを結合することができる。近年の要素プロセッサの性能向上の結果、バスで結合可能なプロセッサ数は4程度に制限されることから、MIN 結合型マルチプロセッサは将来の中規模、大規模な汎用マルチプロセッサシステムとして広く利用される可能性を持っている。

しかし、MIN 結合型マルチプロセッサは, 特に規模が大きくなると, 共有メモリのアクセス時間が大きくなり, 性能低下の原因となる. ここで, 主記憶に対

するキャッシュをプロセッサ-MIN 間に設けることができればアクセス時間の低減に大きな効果が期待される. ところが, プロセッサ–MIN 間にキャッシュを設ける場合, その一致制御が問題となる.

MIN 結合型は同時に複数のプロセッサが共有メモリをアクセスするため, 共有バスにおけるスヌープ機構を用いることができない. また、CC(Cache Coherent)–NUMA で用いられているディレクトリ方式を用いるためには, スイッチングエレメントに対し特殊な機能を設ける必要がある. このため, 従来の MIN 結合型マルチプロセッサのキャッシュ制御としては, コンパイラ等による事前解析の結果を利用する方式が検討されてきた. 最も簡単なのは, 事前解析により一貫性を保持する必要がない変数のみをキャッシュする (あるいはローカルメモリに割り付ける) 方式であるが, これではキャッシュの効果が制限される. そこで, 並列 DO 文の境界毎に, 無効化・キャッシュ開始/終了などのキャッシュ制御命令をコンパイラが埋め込む方法 [11], 参照マーキング法 [4], 部分的にハードウェアの助けを借りる方法 [3] 等様々な方法が提案されている.

一方, 最近の LSI 集積度の向上により MIN のスイッチングエレメントに高い機能を与えることが容易になった. このため, スイッチングエレメント内部にキャッシュを組み込んだり, ディレクトリを組み込む方法の検討が行なわれてきた [9][10]. しかし, 従来の方法はスイッチングエレメント中にキャッシュ本体やディレクトリを組み込んだため, 全体の記憶量が膨大になり, エレメント毎に外部メモリをアクセスしなければならないオーバヘッドの点でも不利であった.

そこで, これらの問題点を解決するために, 超並列マシン JUMP-1[6] 用に開発された階層ディレクトリ縮約方式 [13][8] を MIN に導入した. さらに, MIN の性質を利用し, 縮約方式の問題点である無駄パケットを減らすための枝刈りバッファを提案する. 階層ディレクトリ縮約方式と枝刈りバッファの組合せにより、それぞれのスイッチングエレメントは外部メモリをアクセスする必要はなく, パケットのヘッダと枝刈りバッファの内容に従ったマルチキャスト機能のみで, 高速かつ効率の良いキャッシュ制御が可能となる. 本論文ではこのようなスイッチングエレメントを持つ多段接続網を MINC(MIN with Cache control mechanism)[12] と呼び, ソフトウェアシミュレータを用いて, 無駄パケット数, 結合網の混雑状態を評価した.

2 階層ディレクトリ方式

2.1 MIN の構造と階層ディレクトリ

スヌープ機構を用いないでキャッシュの一貫性を維持するためには, 共有関係を示すディレクトリを持つ必要がある. 通常, ディレクトリには, キャッシュラインを保持しているプロセッサが記録され, キャッシュに書き込みが起きた場合などには, このディレクトリを参照してラインを共有するプロセッサ番号を調べ, これらに対し無効化メッセージあるいは書き込みデータを送る.

キャッシュディレクトリの管理方式はすべてのプロセッサに対応する bit map を保持するフルマップ方式をはじめ, リミテッドディレクトリ方式 [1], チェインドディレクトリ方式 [7] など様々な方式が提案され, 特に CC-NUMA で用いられている.

これに対して, 結合網が階層構造を持つ場合に有効な方式が階層ビットマップディレクトリ方式である. ここでは, 木構造のネットワークの葉の位置にプロセッサが存在すると場合を想定する. 木の根からパケットを供給して, 同一のパケットを複数のプロセッサにマルチキャストする場合を考える. 木構造のネットワークであるから, 各節において宛先の葉が末端にある枝にのみパケットを送れば, 必要なプロセッサにのみパケットが届くことになる. なお, 単純な木構造では木の根付近のトラフィックが大きくなって隘路となってしまうため, 結合網は Fat-Tree を内包することが望ましい. Omega 網, Generalized-cube(G-Cube) 網を代表とする全ての MIN はメモリをルートとする Fat-Tree 構造を内蔵していると考えることができる. このため, MIN にディレクトリ管理方式を組み込む場合, 階層ビットマップディレクトリ方式は最も自然な方式である.

図 1は, 階層ビットマップディレクトリ方式を MIN 結合型システムに応用した場合を示す. 共有メモリ情報を下位 (プロセッサ側) のステージが持っていれば 1, そうでなければ 0 という 4bit のビットマップを各スイッチに与える事で, 送信先プロセッサ **D** を完全に指定できるが, フルマップよりも多くのビット数を必要としてしまう.

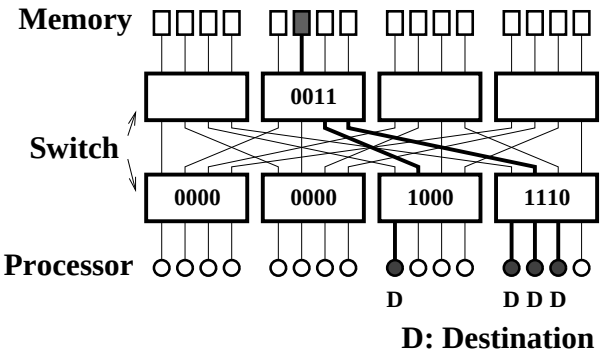

図 1: 階層ビットマップディレクトリ方式

2.2 従来のキャッシュ制御機構を組み込んだ MIN

図 1に示すように, 単純に MIN 上で階層ディレクトリを実現するためには中間ノードに相当するスイッチングエレメントにディレクトリおよび場合によっ

てはキャッシュ本体を備える必要がある.

Memory Hierarchy Network(MHN)[9] は, MIN の階層構造を利用し, 各スイッチングエレメント中にそのエレメントを根とするツリーの葉に当たるプロセッサの共有するデータに対するキャッシュを置く方式である. 無駄なコピーとデータ一貫性に関するロスを防ぐため, キャッシュのコピーはひとつに制限されている. 共有するプロセッサ数が少ないデータは MHN のプロセッサに近いステージに置かれるため, アクセス遅延は少ない. 多くのプロセッサが共有するキャッシュは MHN のメモリ側のステージに移動していく.

これに対し MIND(MIN with Directory)[10] は, 各スイッチングエレメント上にはキャッシュ自体は持たず, ディレクトリのみを置く方法である. MIN はそれぞれのメモリモジュールを頂点とするツリー構造として考えることができるので, これを利用して MIN 全体で階層的なディレクトリを実現する. キャッシュの無効化メッセージはディレクトリの内容が 1 の所のみに送られるため, 必要なプロセッサにのみ, マルチキャストの形で無効化メッセージを送ることができる. この方法をより効率化するため, MIN の各スイッチングエレメントをバスで構成する MBN(Multistage Bus Network)[2] も提案されている.

しかし, MHN は各エレメントにキャッシュおよびディレクトリを置くため, 全体として膨大な量のメモリを必要とする. さらに, MHN のディレクトリおよびキャッシュは大容量のメモリになり, スイッチングエレメント内に装備することは不可能であるため, チップ外に置く必要がある. パケットの転送は全てディレクトリをアクセスする必要があるので,MHN ではパケット転送の度に外部メモリのアクセスを必要とし, 転送遅延が大きくなる.

MIND はスイッチングエレメント内にキャッシュを持つ必要はないが, ディレクトリは持つ必要があり, 同様にチップ外メモリのアクセスに必要が生ずる. このため, MHN 同様全体としての記憶容量が大きい上, 各ステージでの外部メモリのアクセスによる遅延は, キャッシュがヒットしなかった場合にアクセス時間を大きく引き延ばす可能性がある.

3 MINC(MIN with Cache control mechanism)

3.1 MINC の基本構成

本論文では, 従来のキャッシュ制御機構付きの MIN の問題点を解決する新しい機構 MINC(MIN with Cache control mechanism) を提案する. MINC は通常のマルチプロセッサにおける MIN と同様, 図 2 に示すように, プロセッサからメモリに対してパケットを送る Forward MIN と, メモリからプロセッサにパケットを送る Backward MIN に分かれている. 従来の方式との相違は以下の点である.

- 階層ビットマップは各スイッチングエレメントに置かず, 共有メモリに縮約されたビットマップの形で置く. 無効化や更新データの放送時には, このビットマップをパケットのヘッダに格納し, このマップに従って Backward MIN 上でマルチキャストする.
- Backward MIN 上に枝刈りバッファを設け, キャッシュラインのロード時に登録し, ビットマップを縮約することにより発生する無駄パケットを防ぐ.

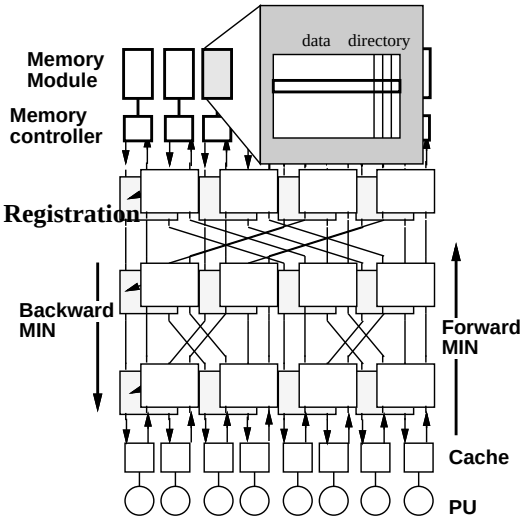

図 2: キャッシュ制御付き MIN の構成

以下, まずディレクトリ縮約方式を紹介し, これを用いた MINC の基本操作を述べた後, 枝刈りバッファについて解説する. なお, Forward MIN, Backward MIN のトポロジについては特に制限がないが, ここではプロセッサの位置による局所性を利用できるように, Baseline 網を想定する. また, 図 2では 2×2 のスイッチングエレメントを示すが, 実際は LSI チップでの実装を考え, 8×8 程度のサイズを想定している.

3.2 RHBD 方式

階層ビットマップ方式は, プロセッサ数が多くなるとディレクトリに必要なメモリ量が膨大になり, 実現が困難になる. また, ディレクトリはかなり大容量になることが予想されるので, スイッチングエレメントを構成するチップの外部に設ける必要がある. この場合, 階層毎にディレクトリを参照するには, 大きな時間を要する. そこでビット数を縮約する RHBD(Reduced Hierarchical Bit-map Directory) 方式を導入する. この方法は超並列マシン JUMP-1 のディレクトリ制御用に考案された方式 [13] [8] である. この方法は,

- ある節以下はブロードキャストとする.

- 複数の節で同一のビットマップを用いる.

のいずれか，もしくは両方の組み合わせによって，階層毎に n bit(n 進木で) のビットマップを一つだけ持つこととする. このようにすると, ディレクトリエントリ毎に必要な bit 数は, m 階層の n 進木において $m \times n$ bit となる. さらに, ビットマップは, パケットのヘッダ中に持つことができるので, 完全にエレメント内のみでマルチキャストが可能であり, 外部のディレクトリを参照する必要がない. 上記 2 つの方法の組合せにより 3 つのディレクトリ縮約方式が提案されている [8] が, ここでは, SM 法と LARP 法について検討した.

- **SM(Single Map) 法:**
  各階層毎に, その階層の全ての節の縮約前のビットマップの論理和をとり, その階層の全ての節で用いる.
- **LARP(Local Appropriate Remote Precise) 法:**
  根から送信元に至るパスをその他のパスと区別して扱い, ある節で (1) 送信元を含む枝, (2) 送信元を含まない枝 のうちのいずれかに対し両方共パケットが送られると, 送信元を含む枝の下の部分全体にパケットがブロードキャストされる. それ以外の枝では, 同一階層の全ての節で, それらの節の縮約前のビットマップの論理和を用いる.

図 3に三進木を用いた摸式図を示す. この図で **s** が送信元のクラスタ, **d** が本来の送り先, •が結果的にパケットが送り付けられるクラスタである. 従って, d の無い•の数が少ないほど無駄にパケットを受けとるクラスタ数が少ない事になる. この縮約方式を採り入れた階層ビットマップディレクトリ方式を縮約階層ビットマップディレクトリ方式 (RHBD: Reduced Hierarchical Bit-map Directory) と呼ぶ.

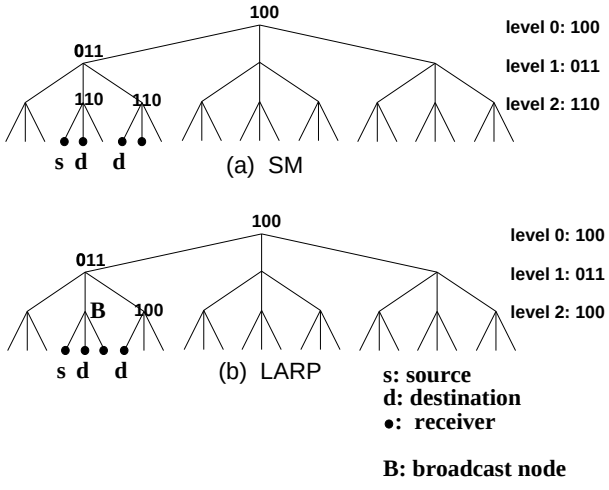

図 3: ディレクトリ縮約方式

縮約方式を用いた場合, MIN のエレメント内にディレクトリを持つ必要はなくなり, 共有メモリ上にのみ, 各ステージ (階層) で用いるビットマップを保持すれば良い. Backward MIN を通過するパケットのヘッダにこの各ステージで持ちいるビットマップを入れておき, Backward MIN 内ではこのビットマップに従ってパケットをマルチキャストする. この方式では無駄なパケットがプロセッサに到着してしまうが, 全くスイッチングエレメント内のディレクトリをアクセスする必要がなくなり, 記憶容量が節約されると共に, 無効化の実行速度も向上する.

3.3 MINC のキャッシュ制御プロトコル

まず,RHBD を導入した場合のキャッシュ制御の基本プロトコルに関して解説する. MINC では, ディレクトリが共有メモリに集中している分 MHN や MIND に比べてプロトコルは簡単である. ただし, ディレクトリを縮約したため, ラインのコピーを保持しているプロセッサ数をカウントするカウンタを共有メモリのディレクトリに設ける必要がある.

各プロセッサに接続されたキャッシュには, そのラインの有効/無効 (Valid:V / Invalid:I) および, コピーが他に存在する/存在しない (Shared:S / Private:P) のタグが設けられている. 提案するプロトコルは, これらの組合せにより I, P, S の 3 状態を持つライトスルー型であり, 図 4に示す状態遷移を行なう. MINC では無効化型 (図 4a), 更新型 (図 4b) 両方のプロトコルの利用が可能である.

まず, 有効なキャッシュに対しての読み出しアクセスがヒットすれば, 問題なくキャッシュからデータを読み込む. その他のアクセスは以下のように処理される.

1. 書き込みミス／ヒット: Forward MIN を用いて共有メモリにデータを送り, 共有メモリ上のディレクトリに従って Backward MIN を用いて, 無効化パケットまたは更新用のデータをマルチキャストする.

2. 読み出しミス: ミスしたラインに対応するセットに I 状態が存在しなければ, 3. に従って登録消去を行ない読み込み領域を確保する. 次に Forward MIN を用いてライン要求パケットを共有メモリに送る. ここで, 共有メモリコントローラは, 共有メモリ上のビットマップの必要な場所に 1 をセットし, カウンタをインクリメントする. そして,Backward MIN を用いて要求元のプロセッサのキャッシュに読み出したラインを送る. この時, ディレクトリ上のカウンタが 0, すなわち他に同一ラインを共有するプロセッサが存在しない場合, 共有メモリは読み出したラインを送るパケットのヘッダ中の Private bit をセットして送り出す. パケットを送り出した後, 共有メモリコントローラはカウンタをインクリメントする. 要求を出したプロセッサは, パケットを受け取りヘッダ中の Private bit がセットされていれば P 状態, そうでなければ S 状態になる.

さて, 共有メモリコントローラは, カウンタが 1 のラインに対するライン要求パケットを受け取ると, 共有化パケットを Backward MIN を用いてマルチキャストし, このラインを共有しているもう一つのキャッシュラインの状態を S に変更する.

3. 登録消去: ディレクトリを縮約した場合, ビットマップ中の 1 は複数のキャッシュのコピーを表現している場合がある. したがって, 簡単にこのビットをリセットすることはできない. ラインを捨てる場合は, Forward MIN を用いて登録消去メッセージを共有メモリに送り, ディレクトリ中のカウンタをデクリメントする. 共有メモリコントローラは, デクリメントの結果, カウンタが 0 になった時にのみビットマップ中の全ての bit を消去する.

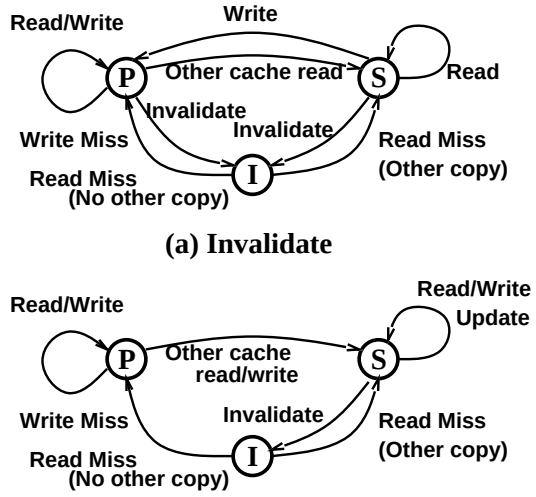

図 4: キャッシュの状態遷移

MINC は無効化型と更新型の両方を用いることができるが, 更新型を用いると共有データのマルチキャストやブロードキャストを効率良く行なうことができる. この場合, キャッシュのヒット率が上がらない場合でも,1 対 1 転送のみに機能が限定される従来の MIN に比べると高い性能を得ることが期待できる.

3.4 枝刈りバッファの導入

縮約方式は大きな利点を持つ一方, 無効化パケットが必要以外のプロセッサにも届いてしまう問題がある. このメッセージは届いたプロセッサのキャッシュコントローラで捨ててしまえばよいのでプロトコル上の実害はないが, ラインを共有するプロセッサ数が増えると, 無駄なパケットが増え Backward MIN の混雑が激しくなる. システムのサイズが大きく, データの共有関係の局所性が小さいと有効パケットの数百倍の無駄パケットが発生する可能性がある [8].

このため, Backward MIN 上の特定のステージのエレメントのみに, チップ内部に実装できる程度の簡単なフルアソシアティブな枝刈りバッファを設ける. 図 5は, LARP 型 MINC に枝刈りバッファを適応させたものである. 枝刈りバッファは以下のように動作する.

- Backward MIN により読み出したラインを要求元のプロセッサに転送する際に, 通過するエレメント上の枝刈りバッファ上にラインアドレスを登録し, 対応する入力の bit をセットする. 既にキャッシュ上にエントリが存在する場合は, パケットの行き先に相当する bit をセットする.

- Backward MIN を介して更新データあるいは無効化パケットを送る時, スイッチングエレメント内の枝刈りバッファを参照し, アドレスが一致すればそのディレクトリのビットマップを利用して, 1 に対応する出力に対してのみパケットを送る. 無効化型プロトコルを用いた場合は, 枝刈りバッファ上のエントリは, 無効化パケットを送った時点で削除する.

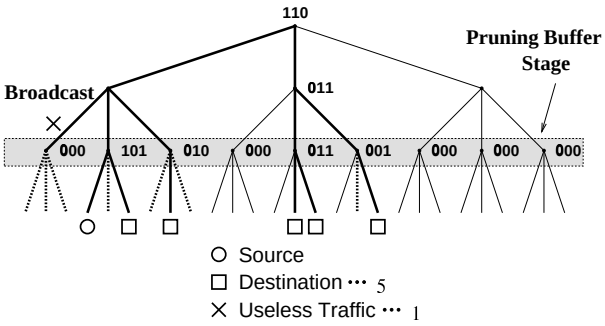

図 5: 枝刈りバッファ

無効化型プロトコルで枝刈りバッファは, スイッチングエレメント内に実装することが必要で, なおかつ高速なアクセスが要求される. このように容量の制限が厳しいため, 無効化プロトコルで頻繁に無効化が行なわれる場合を除いて, ディレクトリキャッシュの登録時にはバッファの領域がなくなる場合が頻繁に起きると考えられる. 枝刈りバッファの目的は無駄パケットの削減にあるので, このような場合は, 単純にどこかのラインを選んで捨ててしまって領域を確保する. この場合, どのラインを選ぶかは, 以下のアルゴリズムが考えられる.

- 最も 1 の数の多いビットマップを持つものを捨てる. これは 1 の数が少ないものの方が, 無駄パケット削減の効果が大きいからである.
- 単純な FIFO で先に登録されたものから順番に捨てていく.
- 通常のキャッシュ同様 LRU を用いる.

これらの選択は, 実際のアクセスに基づく評価が必要である.

4 評価

4.1 シミュレーションモデル

実際に MINC を設計する前に確率モデルに基づく簡単な評価を行ない, その有効性を確認した. 評価に用いたシミュレータでは, 確率モデルにより各プロセッサが共有メモリアクセスパケットを生成する. 各々のパケットのアドレスの決定方法は以下の 2 通りである.

- **Nonlocal** モード
  一様乱数を用いた決定法で, どのメモリモジュールにも等確率でアクセスをする.
- **Local** モード
  データの局所性を疑似的に実現する決定法. あるラインを共有するプロセッサが, 特定のプロセッサ番号をピークとした正規分布に基づいて分布していると仮定する. すなわち, あるラインは, プロセッサ番号が近いプロセッサ間で共有される確率が高く, あるプロセッサは, 特定のメモリモジュールに近い番号のメモリモジュールをアクセスする確率が高くなる.

MINC は, SSS 型 MIN[5] により制御することを想定した. すなわち, 全てのアクセスは共通のフレーム信号に同期して数 bit シリアルのパケットの形で入力され, ネットワーク通過中に衝突して目的地メモリモジュールに到達できなかったパケットは次のフレームで再送される. キャッシュコヒーレンスを維持するために発行されるマルチキャストパケットについても同様であるが, あるキャッシュラインに対するマルチキャストパケットの一部分が衝突した場合は, 衝突を起こしたパケットについてのみ再送する.

MINC は, 各スイッチングエレメントにバッファを設けて独立に転送する従来型の MIN にも用いることができるが, SSS 型 MIN ではパケットの入力がフレームに同期されるため, 共有メモリでのディレクトリの登録操作やマルチキャスト時の枝刈りバッファの制御が容易で高速な動作が期待できる.

4.2 トラフィック量の評価

まず,Backward MIN 上を流れる総パケット数を各縮約法について調べると共に枝刈りバッファの効果を評価した. ここで, 性能比較の基準としてトラフィック量を定義する. トラフィック量とは, キャッシュコヒーレンスを取るための無効化 (更新) パケットが Backward MIN を通過する総リンク数をカウントしたものである. 従って, トラフィック量が大きいと Backward MIN 上が混雑し, 無効化 (更新) パケット同士の衝突確率が高くなる. すなわち, トラフィック量はネットワークの混雑度を示す指標である.

データの共有が局所性を持つ Local モードにより SM 法,LARP 法のそれぞれ枝刈りバッファ(Pruning Buffer: P.B.) を搭載した場合とそうでない場合を評価した結果を図 6に示す. ここでは, 枝刈りバッファは最もプロセッサに近いステージに一段のみ搭載している. 比較のためマルチキャストを行なわず,1 対 1 転送を行なう場合のトラフィック量 (Fullmap) も併せて示す.

図 6は, プロセッサのキャッシュにコピーされるラインの数 (v), すなわち, あるラインを共有するプロセッサ数に対するトラフィック量を調べている. システム中のプロセッサ数 N は 4096, MIN は 8×8(sw) のスイッチングエレメントを 4 ステージ接続して, 4096 入出力としている.

キャッシュにコピーされるライン数が増大すると, それにつれてトラフィック量も増える. 交信の局所性を考慮しているにもかかわらず,LARP 法ではトラフィック量が増大し, 枝刈りバッファ(P.B.) を搭載した場合でも, 搭載していない SM 法に比べてトラフィック量が大きくなってしまう. 一方,SM 法は, 枝刈りバッファを搭載すると, トラフィック量は減少し,1 対 1 転送とほとんど差がないくらいに抑えることができている. 以上のシミュレーション結果により,MINC における縮約法は SM 法が有利で, 枝刈りバッファの搭載はトラフィック量の抑制に大きな効果があることがわかる.

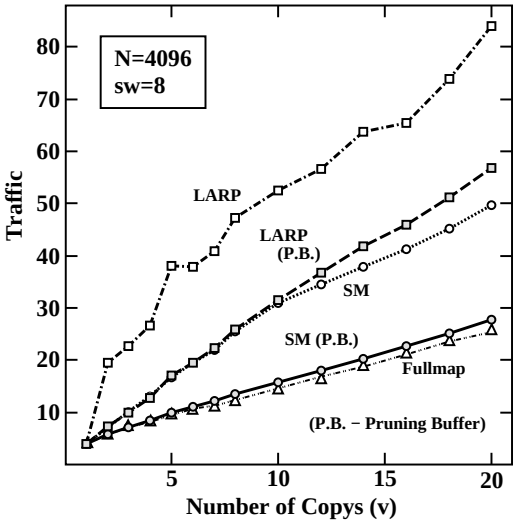

図 6: ディレクトリ縮約法の評価

4.3 結合網の混雑状況の評価

次に, 前述のシミュレーションモデルに基づき, MIN 上のパケットの衝突, それによる再送等の MIN の動作のダイナミックな状況をシミュレーションした結果を示す.

ここでのシミュレーションで用いたパラメータを表 1に示す.

ここで示す結果は, プロセッサ数 (N) を 256 とし,4×4 のスイッチングエレメント (sw=4) 3 ステー

表 1: シミュレータパラメータ

パケット生成パラメータ	
Pr	各プロセッサがあるフレームでパケットを発行する確率
r	read パケット率
システムパラメータ	
N	システムのノード数 (256)
sw	スイッチサイズ (4x4)
PBst	枝刈りバッファを搭載するステージの番号
PBsz	枝刈りバッファのサイズ (4)
メモリパラメータ	
GM	共有メモリサイズ [bytes](64M)
BS	キャッシュラインサイズ [bytes](16)

ジからなる MIN を想定している．枝刈りバッファのサイズ (LTsz) は実装を考慮して小さめの 4 に設定した．他のパラメータは, Pr=0.3, GM=64Mbyte, BS=16byte として固定した．この条件で, 枝刈りバッファを入れるステージ (PBst) を変えて, パケットの平均再送回数を調べた結果を図 7に示す．なお, ステージ数はプロセッサ (キャッシュ) に近い側から 1,2,3 と順番に番号を付けている．評価に用いたデータは, それぞれの条件でシミュレータを 10,000 フレームだけ動作させたものである．図中の **prob**, **local** はそれぞれ,

- **prob** – Nonlocal モード 90%, Local モード 10% の割合でパケットを生成した場合
- **local** – Local モード 90%, Nonlocal モード 10% の割合でパケットを生成した場合

を, invalid/valid はそれぞれ, キャッシュプロトコルの無効化型/更新型を表している．先の評価でディレクトリ縮約法については, LARP 法より SM 法の方が良い性能を示す事が分かったので, 今回は SM 法に関して更に詳細な評価を行なっている．

図 7によると, 局所性のない場合でも再送回数は平均 4 回程度で済んでおり, マルチキャストを行なうのに要する時間が現実的であることがわかる．また, 無効化型・更新型いずれの場合もパケットに局所性のある方がより良い性能を示している．一方, update.local について, LTst を 3 にすると LTst=2,1 の時に比べ, 急激に転送回数が増加している．これは, MINC のトポロジが Baseline 網であるので, アクセスに局所性がある場合は下位ステージの方で初めて複数方向にマルチキャストパケットが枝別れしていく場合が多く, 上位ステージの方では枝刈りできないためである．その他, どの条件でも, LTst=1 の場合が最も低い値を示しており, 枝刈りバッファはステージ 1 に搭載するのが最も効果的であることがわかる．

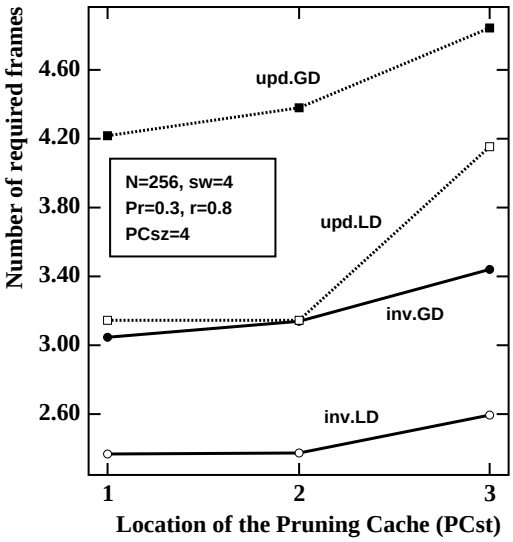

図 7: 枝刈りバッファの搭載段数 対 パケット平均再送回数

次に枝刈りバッファを 1 ステージ目に入れる場合に関して, 枝刈りバッファの効果について調べた．図 8 は, invalid.local と update.local については PBst=1 固定とし, 枝刈りバッファ非搭載型 (invalid) との比較を示している．プロセッサのアクセス発生確率が Pr=0.3 の時, バッファ非搭載型が平均約 3.5 回の再転送を必要とするのに対して, 枝刈りバッファ付きの方は約 1.5 回で済んでいる．このことから, 枝刈りバッファの効果は明らかである．また, Pr=0.35 になるとどの場合も再送回数が上昇するが, これは MIN に対するアクセスが増加し, 全体の飽和が始まっていることを示している．

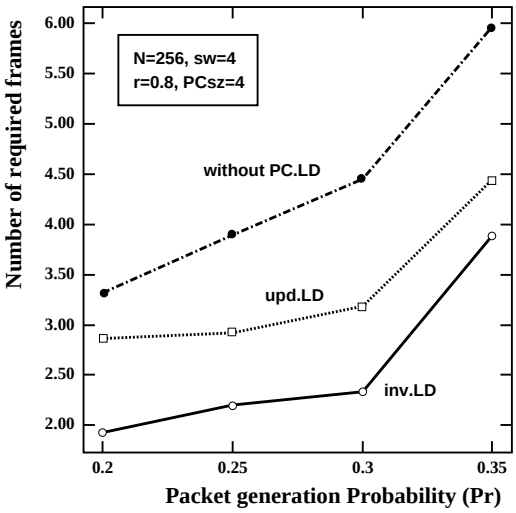

図 8: アクセス確率 対 パケット平均再送回数

最後に, 枝刈りバッファを 1 ステージ目に搭載した場合の枝刈りバッファのヒット率を調べた (表 2). invalid.prob の方が 48.52%であるのに対して, update.local ヒット率の方が 54.20%と高いのにも関わらず, invalid.prob の方が, マルチキャストをわずかであるがスムーズに行なっている. これは, 無効化型はいずれかのプロセッサが write を起こすとディレクトリを作り直すのに対して, 更新型はディレクトリ上のビットマップが保存される点に起因する. すなわち, 更新型の場合は既に不要になったキャッシュラインの更新のためのトラフィックが,MIN の混雑を引き起こす可能性がある.

表 2: 枝刈りバッファヒット率

invalid local	invalid prob	update local	update prob
77.87%	48.52%	54.20%	34.22%

5 結論

RHBD および枝刈りバッファを導入する事で高速なキャッシュコヒーレンス制御可能な多段結合網 MINC を構築した. 更にソフトウェアシミュレータでの評価を行い, 以下の結論が得られた.

1. 共有メモリに対してのアクセスに局所性がある場合, より効果的にコヒーレンス制御を行なうことができる.
2. ステージ 1 に枝刈りバッファを搭載すると最も効率良くマルチキャストを行なうことができる.
3. ディレクトリ縮約に関しては SM 法が有効である. キャッシュプロトコルに関しては, 更新型はマルチキャストの有効性を利用することができる一方, 不必要なキャッシュラインが存在すると交信量が増大するため, 不要となったラインを判別し, 無効化する機構が望まれる.

今回のシミュレーションは確率モデルに基づいたため, その結果は目安に過ぎない. 今後は MINC のスイッチングエレメントの設計を進める一方, トレースドリブンシミュレータを用いてさらに現実的な評価を行なう予定である.

参考文献

[1] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz. An Evaluation of Directory Schemes for Cache Coherence. In *Proc. 15th ISCA*, pp. 280–289, 1988.

[2] L.N. Bhuyan, A.K. Nanda, and T. Askar. Performance and Reliability of the Multistage Bus Network. In *Proc. of ICPP*, pp. I–26–I–33, 1994.

[3] H. Cheong and A.V. Veidenbaum. A Cache Coherence Scheme with Fast-Selective Invalidation. In *Proc. of 15th ISCA*, pp. 299–307, 1988.

[4] J. Edler and et al. Issues Related to MIMD Shared-memory Computers: the NYU Ultracomputer Approach. In *Proc. of 12th ISCA*, pp. 126–135, 1985.

[5] Amano H., Zhou L., and Gaye K. SSS(Simple Serial Synchronized)-MIN: a novel multistage interconnection architecture for multiprocessors. In *Proc. of the IFIP 12th World Computer Congress, Vol.I*, pp. 571–577, 1992.

[6] K. Hiraki, H. Amano, M. Kuga, T. Sueyoshi, T. Kudoh, H. Nakashima, H. Nakajo, H. Matsuda, T. Matsumoto, and S. Mori. Overview of the jump-1, an mpp prototype for general-purpose parallel computations. In *Proc. IEEE International Symposium on Parallel Architectures, Algorithms and Networks*, pp. 427–434, 1994.

[7] D.V. James, A. T. Laundrie, S. Gjessing, and G. S. Sohi. Distributed-Directory Scheme: Scalable Coherent Interface. *IEEE Computer*, Vol. 23, No. 6, pp. 74–77, 1990.

[8] T. Kudoh, H. Amano, T. Matsumoto, K. Hiraki, Y. Yang, K. Nishimura, K. Yoshimura, and Y. Fukushima. Hierarchical bit-map directory schemes on the RDT interconnection network for a massively parallel processor JUMP-1. In *Proc. International Conference on Parallel Processing*, 1995.

[9] H.E. Mizrahi, J.L. Baer, E.D. Lazowska, and Zahorjan J. Introducing Memory into the Switch Elements of Multiprocessor Interconnection Networks. In *Proc. of 16th ISCA*, pp. 158–166, 1989.

[10] A.K. Nanda and L.N. Bhuyan. Design and Analysis of Cache Coherent Multistage Interconnection Networks. *IEEE Trans. on Computers*, Vol. 42, No. 4, pp. 458–470, 1993.

[11] A.V. Veidenbaum. A Compiler-Assisted Cache Coherence Solution for Multiprocessors. In *Proc. ICPP*, pp. 1026–1036, 1986.

[12] 安川英樹, 天野英晴, 舟橋啓, 塙敏博. MINC: キャッシュ制御機構を持つ MIN. 信学技報, CPSY 95-95, pp. 7–12, 1995.

[13] 松本尚. 局所処理と非局所処理を分離並列処理するアーキテクチャ. 第 43 回情処全大 (6), pp. 115–116, 1991.