論文要旨

クロスバスイッチを多段に結合した MIN (Multistage Interconnection Network) は，全てのメモリを同一コストでアクセス可能であり，全体として高い同時転送容量を持つため，中規模マルチプロセッサのプロセッサ-メモリ接続網として長い間検討が行われていた．一方で，MIN は，バスと異なり，各プロセッシングユニット (PU) が互いのメモリモジュール (MM) のアクセスを監視することができないため，スヌープ方式によるキャッシュ一致制御を行うことができず，MIN を用いたプロセッサ-メモリ接続法は最近あまり検討されていない．しかし，多数の PU や MM が単一チップ上に実装される時代が近い将来訪れると考えられ，MIN を用いたプロセッサ-メモリ接続法は，キャッシュ一致制御の問題が解決されれば，再び検討の対象になると考えられる．

MIN を用いたマルチプロセッサによるキャッシュ制御の基本は，MM 側にディレクトリを設ける方法である．しかし，完全な共有情報をディレクトリに保持するフルマップ方式では，アドレス空間のサイズに応じて必要なメモリ容量が増えてしまうため，ハードウェアコストの点で問題があり，また，ネットワーク越しにディレクトリが参照されるために，ディレクトリのアクセス時間が増加する．このため，共有情報を縮約して保持することでディレクトリのハードウェアコストを削減する方法，キャッシュやディレクトリの一部を MIN のスイッチ内に設ける方法などが提案されているが，いずれもメモリ容量やアクセス時間の増大を十分に解決できておらず，改善の余地が残されている．

そこで本論文では，第一の改善の試みとして提案された MINC(MIN with Cache control mechanism) をゲートアレイに実装し，実機 SNAIL−2 (SSS−MIN Network Architecture ImpLementation 2) 上で評価用アプリケーションを動作させて評価を行う．MINC は，共有情報を縮約して MM 側に保持し，また，スイッチ上に大容量のキャッシュやディレクトリを保持すること回避することで実装を容易にしつつ，メモリ容量やアクセス時間の問題の解決を図る方式である．

そして，MINC の評価の結果，PU 数が増えたときに MINC のキャッシュ制御方式が原因でシステムの処理能力が低下する問題が明らかとなったため，MIN を構成する各スイッチ内に設けた小容量のテンポラリディレクトリ (TD) のみで共有情報を保持するキャッシュ制御方式 MINDIC (MIN with DIrectory Cache switch) を提案する．MINDIC では，各スイッチ内の TD に比較的最近アクセスされたキャッシュラインの共有情報を保持し，共有されたキャッシュラインへの書き込み要求が発生した時に

無効化要求を PU の各キャッシュに対して発行する．これにより，キャッシュの一貫性を維持しつつ，MM 側にディレクトリを配置する必要性を無くしている．また，この方式は，TD のエントリ不足時にキャッシュ一致制御にエラーを生じる問題点があるため，この問題点を解決する３種類のプロトコルを提案する．そして，トレースドリブンシミュレータによる３種類のプロトコルの予備評価を行い，最も良い性能を示した Eviction プロトコルに対してクロックレベルシミュレータを構築し，フルマップ方式によるキャッシュ制御方式との比較検討を行う．その結果，MINDIC の TD を 2048 エントリ程度設けることで，フルマップ方式によるデレクトリ管理方式と同様の効率良いキャッシュ制御を実現できることを示す．また，ディレクトリに必要なメモリ容量を大幅に削減できることを示す．

Abstract

Multistage Interconnection Networks (MINs) have been well researched as an interconnection medium between processing units (PUs) and memory modules (MMs) for medium scale multiprocessors.Unlike shared buses that allow to access only one MM at a time, MINs enable multiple accesses simultaneously. Snoop caches commonly used in shared bus connected multiprocessors are not available because of the nature of MINs, thus, they usually maintain cache coherence by the directories provided on MMs. However, this method requires not only large amount of memory spared for the directories but also extra latency caused by the accesses to slow large memory.

Although several methods are proposed as solutions, for example, providing a part of cache or directories inside switching elements, they are still suffering from large memory expense, directory access latency, complicated structure of switching elements.

To solve these problems, we proposed two cache consistency mechanism in this paper. First, we propose the MINC(MIN with Cache control mechanism), in which the bit-map of the reduced hierarchical directory is equipped only on MM. The MINC are evaluated with a real machine SNAIL−2(SSS−MIN Network Architecture ImpLementation 2). Empirical implementation results show that the cache controlled by MINC improves the performance for a small system.

Second, we propose a novel MIN structure called MINDIC(MIN with Directory Cache switch). In this method, small directories called Temporary Directory (TD) are located only in switching elements, without providing directories on MMs. With the TDs in each switching element, we can maintain cache consistency with low latency nand low memory cost. Since the TDs must be implemented inside the switching elements, its size is limited. If the entry of the TD is full at the registration, a cache coherence problem for the cache line occurs. That is, invalidation messages for the cache line which could not be registered in the TD can't be generated.

To cope with such cases, we proposed three different protocols. In our preliminary examination using a trace driven simulator, the eviction protocol achieved the best performance in the three protocols. Therefore, we evaluate the eviction protocol in detail using instruction level simulator. Our results show that MINDIC is equivalent to the architecture with full-mapped directories in execution performance.

目 次

図目次

表目次

第1章　緒論

転送能力の高いクロスバスイッチを多段に結合した MIN (Multistage Interconnection Network) は，全てのメモリを同一コストでアクセス可能であり，全体として高い同時転送容量を持つため，中規模の並列計算機の PU-メモリ接続網として長い間検討が行われていた．一方で，MIN は，バスと異なり，各プロセッシングユニット (PU) が互いの共有メモリアクセスを監視することができないため，バスを PU-メモリ接続網とした並列計算機で用いられるスヌープ方式によるキャッシュの一致制御を行うことができない．このため，並列計算機の大規模化としては，バス結合で構成された小規模な並列計算機同士を接続する NUMA 方式が主となり，MIN を用いた PU-メモリ接続法はあまり検討されてこなかった．

一方，現在よりもより多くの PU やメモリモジュール，及びそれを接続するネットワークが単一チップ上に実装される時代が近い将来訪れると考えられる．MIN による接続は，近接したモジュール間の通信であっても，ある一定のレインテンシを必要としてしまう問題があるが，単一チップ上の実装では高速転送が可能であることから，このレイテンシを低く抑える事が可能である．さらに MIN では，より多くの PU が実装される事により発生するメモリアクセスの増大に対応可能な高いスループットを持つ．このため，MIN による PU-メモリ接続網は，キャッシュ一致制御の問題が解決されれば，再び検討の対象になると考えられる．

MIN を用いた並列計算機によるキャッシュ制御法の基本は，メモリモジュール (MM) 側にキャッシュラインの共有情報を保持するディレクトリを設ける方法である．しかし，完全な共有情報をディレクトリに保持するフルマップ方式では，アドレス空間のサイズに応じて必要なメモリ容量が増えてしまうため，ハードウェアコストの点で問題がある．また，ネットワーク越しにディレクトリが参照されるため，ディレクトリのアクセス時間が増加してしまう．このため，共有情報を縮約して保持することでディレクトリのハードウェアコストを削減する方法[GWM90][MH94][AGGD01][HKN+00]，キャッシュの一部を MIN のスイッチ中に設ける方法[MBLZ89][RL00]，ディレクトリを設ける方法[NB93][BNA94][IBN00]など様々な方法が提案されている．しかし，これらの方法はいずれもメモリ容量やアクセス時間の増大を十分に解決できておらず，改善の余地が残されている．

そこで本論文では，第一の改善の試みとして提案された MINC(MINwith Cache control mechanism) について説明し，その実装及び評価について記す．MINC は，共有情報を縮約して MM 側に保持し，また，スイッチ上に大容量のキャッシュやディレクトリを保持すること回避することで，キャッシュ制御機構の実装を容易にしつつ，メモリ容量やアクセス時間の問題の解決を図っている．MINC の評価は，MINC chip をゲートアレイに実装して，キャッシュ制御機構に MINC chip を用いた実機 SNAIL−2(SSS−MIN Network Architecture ImpLementation 2) 上で評価用アプリケーションを動作させることで行っている．

そして，MINC の評価の結果，PU 数が増えたときに MINC のキャッシュ制御方式が原

因でシステムの処理能力が低下する問題が明らかとなったため，第二の改善の試みとして，MIN を構成する各スイッチ内に設けられた小容量のテンポラリディレクトリ (TD) のみで共有情報を保持するキャッシュ制御方式である MINDIC(MIN with DIrectory Cache switch) を提案する．MINDIC では，各スイッチ内の TD にキャッシュラインの共有情報を保持し，共有されたキャッシュラインへの書き込み要求が発生した時に，スイッチから無効化要求を PU の各キャッシュに対して発行する．この TD のみを用いてキャッシュの一致制御を行うことで，MM 側にディレクトリを配置する必要性を無くしている．

しかし，単にスイッチ内に TD を設けただけであると，TD のエントリ不足時に共有情報が保持できず，キャッシュ一致操作を正確に行うことができなくなってしまう．そこで，常に問題なくキャッシュ一致操作を可能とするための 3 種類の転送プロトコルを提案し，トレースドリブンシミュレータによる予備評価により検討を行う．そして，予備評価の結果，最も良い性能を示したプロトコルに対してクロックレベルシミュレータを構築し，フルマップ方式によるキャッシュ制御方式との比較検討を行う．更に，フルマップ方式と比較したハードウェア量についても検討を行う．

筆者は，本研究を行うにあたり，先ず，安川氏らによって提案され[Yas96]，亀井氏ら[Kam97]によって設計が行われた，ディレクトリを縮約して保持するキャッシュ制御方式である MINC を Chip Express の 0.6μm LPGA(Laser Programmable Gate Array) へと実装し，MINC chip を作成した[TTTH98]．その後，その MINC chip を使用したスイッチ結合型マルチプロセッサの実機 SNAIL−2 を実装し，MINC chip の評価を行った．この実機による評価については，2005 年 3 月に ELSEVIER 社 Parallel Computing Vol31 へ下記の題名で掲載された．

> ” The performance of SNAIL-2(a SSS-MIN connected multiprocessor with cache coherent mechanism)“[TDM+05]

なお，MINC は田辺氏らにより，クロックレベルシミュレータによる評価も行われている[TMS+03]．MINC は，評価を行った結果，PU 数が増加すると無効化要求パケットの再送回数が大幅に増加してしまう問題があることが明らかとなった．

そこで，PU 数が増加した状況においても効率良くキャッシュ制御を実現することができる新たなキャッシュ制御方式の検討を行い，スイッチ内のテンポラリディレクトリを利用してキャッシュ一致制御を行う MINDIC の提案，評価を行った．MINDIC の提案，評価については，2005 年 10 月に電子情報通信学会論文誌へ下記の題名で掲載される．

> “テンポラリディレクトリを持つキャッシュ制御用多段結合網 MINDIC の設計と評価” [MSA05]

以降，第 2 章では，まず，ディレクトリ方式によるキャッシュ制御機構の背景について述べ，第 3 章で関連研究の説明や問題点の指摘を行う．第 4 章では，MINC の実装と評価を示し，第 5 章では，MINDIC の提案，3 種類の転送プロトコルの説明を示す．そして，第 6 章でトレースドリブンシミュレータによる転送プロトコルについての予備評価を行いう．続いて予備評価で最も評価の高かった転送プロトコルに基づいてクロックレベルシミュレータを構築し，詳細な評価を行う．また，ハードウェアコストについての検討も行う．最後に，第 7 章において結論を述べる．

第２章　研究の背景

本章では，MIN を用いた並列計算機で用いられるキャッシュ制御方式について説明し，ハードウェアによるキャッシュ制御方式であるディレクトリ方式について述べる．その後，MIN を用いた並列計算機でディレクトリ方式を採用した際の基本となるディレクトリ管理方式である階層ビットマップディレクトリ方式について紹介する．

2.1　オンチップマルチプロセッサ

半導体集積技術は安定的に向上しつつあり，１つのチップ上に集積することが可能なトランジスタ数が増加するにつれて，さまざまなプロセッサの高速化技術が登場している．

1980 年代後半から 1990 年代半ばには，１つの命令の処理を，読み込み，デコード，実行，書き込み等の複数の段階に分けて複数の命令の各段階を並列に実行するパイプライン処理技術や，命令レベル並列性を動的に抽出してスケジューリングし，out of order で命令発行や実行を並列に行うスーパースカラ技術がプロセッサに組み込まれるようになった．しかし，命令レベル並列性には上限があることから，スーパースカラ度を引き上げたとしても上限以上の並列処理を行うことができず，ハードウェアを複雑化することに見合った性能向上に限界があった．

また，1990 年代後半には，コンパイラにより静的に命令レベル並列性を抽出し，依存関係にない命令を一つの命令としてまとめて実行する VLIW(very long instruction word) 方式を採用したプロセッサが登場した．VLIW 方式は，スーパースカラより先に研究が行われていた方式であるが，バイナリプログラムを再コンパイルする必要性があるという問題を有している．動的に命令レベル並列性を抽出するスーパースカラとは異なり，複雑なハードウェアを必要としないことから動作周波数の向上でプロセッサを高速化することができる．しかし，やはり命令レベル並列性の上限により性能向上に限界があり，次世代アーキテクチャの決定打にはなりきれていない．

このように，SIMD(Single Instruction stream Multi Data stream) 型処理を前提としたプロセッサでは，ハードウェア量の有効利用が困難であることが明らかとなってきた．

そこで，1990 年代後半から命令レベルではなくスレッドレベルでの並列性を抽出してプロセッサを高速化する技術が注目されるようになる．マルチスレッディング技術では，複数のスレッドを切り替えながらプログラムを実行することで，ボトルネックの一つであるメモリアクセスのレイテンシを隠蔽することができる．しかし，マルチスレッディング技術では，一つのプロセッサコアで複数のスレッドを処理するシステムであるため，データ依存関係の少ない複数のスレッドが並列して処理されておらず，更なる並列処理による高速化が期待されるようになった．

そのため，複数の命令コードを並列に実行する MIMD(Multi Instruction stream Multi Data

stream) 型のオンチップマルチプロセッサが注目されることとなり，2000 年以降は，商用のオンチップマルチプロセッサが普及し始めている．

2.2　スイッチ接続型マルチプロセッサ

搭載するプロセッサ数が少ないオンチップマルチプロセッサでは，複数のプロセッサとメモリをバスで接続する構成を前提としている．しかし，バスは一度に 1 つのプロセッサしか使用することができず，同時に複数のデータ転送を行うことができないことから，接続するプロセッサ数が増えてくると，バスの転送能力がシステムのボトルネックとなってしまう．そこで，プロセッサとメモリ間の接続網をスループットの高いクロスバ等のスイッチで接続する構成が検討されている．メモリを複数のメモリモジュールに分割して構成し，各プロセッサが独立してそれぞれのメモリモジュールをアクセスすることが可能なスイッチ結合型マルチプロセッサとすることで，高速なメモリアクセスを実現することができる．近年，オンチップマルチプロセッサのプロセッサとメモリ間の接続網として様々なオンチップネットワークの研究が行われている．

一方、スイッチ結合型のネットワークとして，クロスバを多段に接続して数十〜数百のプロセッシングユニットおよびメモリモジュールを接続する MIN 接続型マルチプロセッサの研究が古くから行われてきた．クロスバは入出力数が増えるに従って必要なハードウェア量が大きく増加してしまうという問題を有する．そこで，MIN は，入出力数の少ない小規模なクロスバを複数接続して接続網を構成することで，全体としてのハードウェア量を抑える構成となっている．

図 2.1 に一般的な MIN 接続型マルチプロセッサの構成を示す．クロスバで構成されるスイッチングエレメントを多段に接続する構成であるが，その接続方式によって転送能力が異なってくる．図 2.1 に示したものは，ブロッキング網の一つである Omega 網と呼ばれるものである．MIN を大きく分けると次のように分けることができる．

- ブロッキング網
  異なった行き先に対するパスが衝突する可能性のある接続方式
  例：Omega 網，Generalized Cube 網, Augumented Data Manupilator(ADM) 網, Delta 網

- リアレンジブル網
  スケジューリングを行うことで，すべての入力からすべての異なった行き先に対して衝突しないパスを形成することができる接続方式．ブロッキング網より高い転送能力を持つ．
  例：Benes 網，Wacksman 網

- ノンブロッキング網
  スケジューリングの必要なく，すべての入力からすべての異なった行き先に対して衝突しないパスを形成することができる接続方式．
  例：Clos 網，Batcher-Banyan 網

ノンブロッキング網は確かに高い転送能力を持つが，同一の行き先のパケットが存在すれば出力で競合が発生するので，衝突自体がなくなるわけではない．このため，ノンブロ

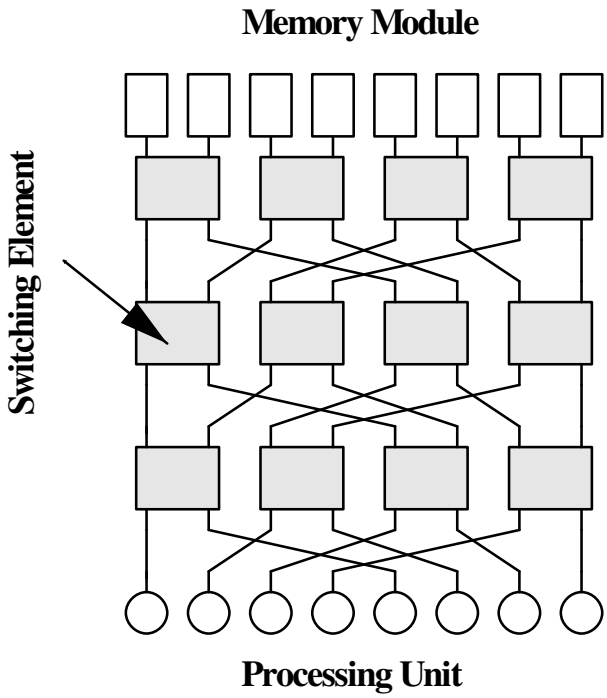

図 2.1: 一般的な MIN 接続型マルチプロセッサ (Omega 網)

キング網でも通過率が 65%程度で頭打ちとなってしまうため，多重出力可能な構成も検討されている．

- MBSF(Multi Banyan Switching Fabrics)
  図 2.2 で示すように，独立した Banyan 網を複数用意し，トラフィックを分散する方法.
- EBSF(Expanded Banyan Switching Fabrics)
  Banyan 網のサイズを整数倍に拡張し，MBSF と同等程度の通過率を持たる方法．図 2.3 は 入力数 8，Banyan 網のサイズ 3 倍の EBSF を示している．入力が 8 なので最初のステージの 12 のエレメントのうち 8 つだけが使われ，残りの 4 つのエレメントは使用されない.
- TBSF(Tandem Banyan Switching Fabrics)
  TBSF は、図 2.4 に示すように banyan 網 (omega 網) を直列に接続し、各網の出口にバイパス路を設けた構造を持つ。banyan 網を通過して目的の宛先に到着したパケットはバイパス路によりメモリモジュールに送られ、衝突により目的の宛先に到着できなかったパケットのみが次の段の banyan 網に入力される。3 以上の banyan 網を接続すればノンブロッキング網より通過率は高くなるが，結合網の最大通過時間が大きいという問題がある．
- PBSF(Piled Banyan Switching Fabrics)
  TBSF では接続された各網において，パケットが衝突するまでは正しくルーチング

されるにもかかわらず，それまでのルーチングの結果は次の網に対して全く貢献しない．この点を改良し，PBSF では図 2.5 に示すように banyan(omega) 網を階層的に接続した構造にしている．図のように最上層と最下層を除く層のスイッチングエレメントは水平方向の入出力を 2 つずつ，垂直方向を 2 つずつ，計 4 入力 4 出力を持つ．2 つ以上のパケットが衝突した場合，正しくルートされなかったパケットは下の層のネットワークに送られる．

パケットはまず最上層のネットワークに入力される．最上層において水平方向に進む 2 つのパケットが，あるエレメントで出力リンクの衝突が起こると，片方のパケットは希望の方向に送られ衝突に敗れたパケットは一つ下の層のエレメントに送られる．2 層目以下では，水平方向同士の衝突に加えて上層から送られて来たパケットとも競合する．この場合、最大で 3 つのパケットが一つの出力リンクを競合する (水平方向からのパケット 2 つと垂直方向からのパケット 1 つ) が，これらのパケットのうち一つは正しい出力へと送られ，もう一つは更に下層へ，残りの一つはスイッチングエレメント内で消滅する．最下層で 3 つのパケットが競合した場合，下層へ送る事ができないため 3 つのパケットのうち一つだけが正しく転送され，残りのパケットはエレメント内で消滅する．いずれの場合も，パケットが消滅した場合にはプロセッサ側に NAK が返されるようになっており，ネットワークインタフェースによって次のフレームで再送される。PBSF において下層の網は、従来型の MIN がエレメント内に持つパケットバッファと同様の効果があり、通過率、通過時間両方の改善が期待できる。

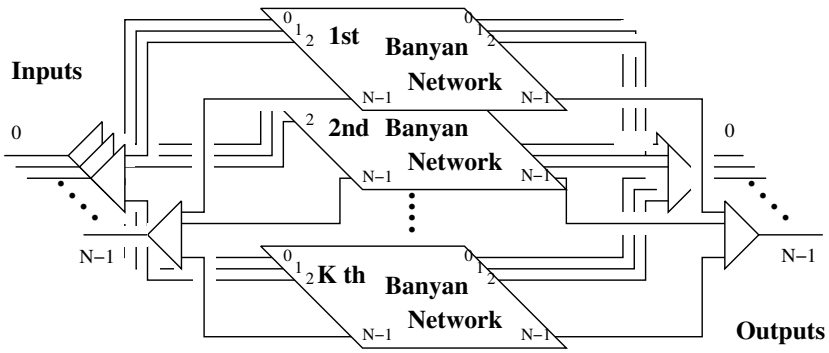

図 2.2: MBSF (Multi Banyan Switching Fabrics)

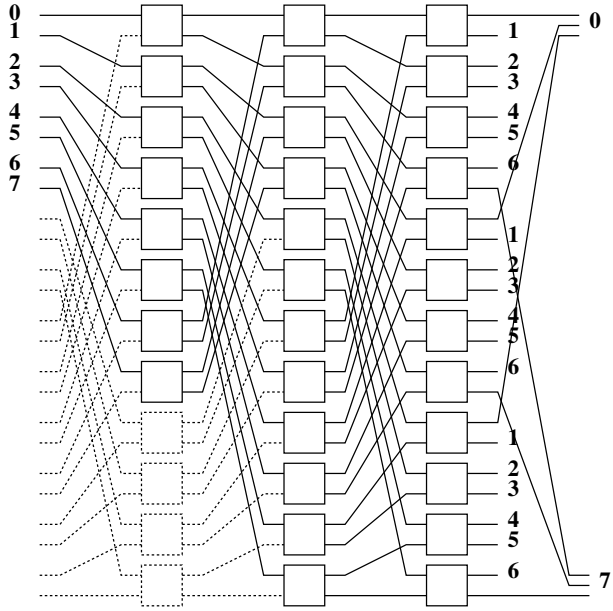

図 2.3: EBSF(Expanded Banyan Switching Fabrics)

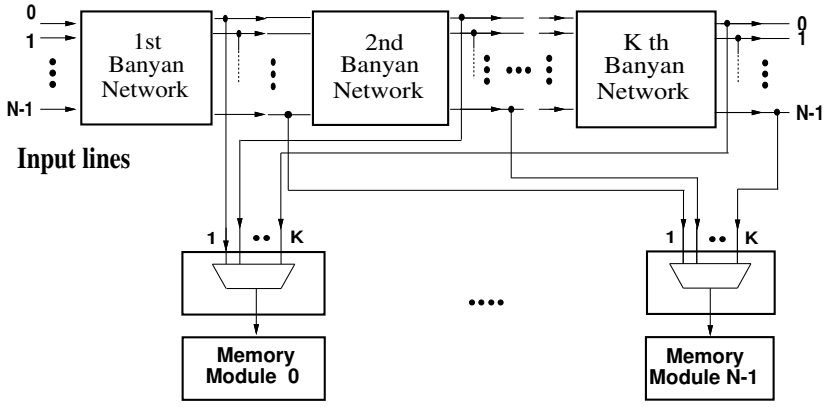

図 2.4: TBSF(Tandem Banyan Switching Fabrics)

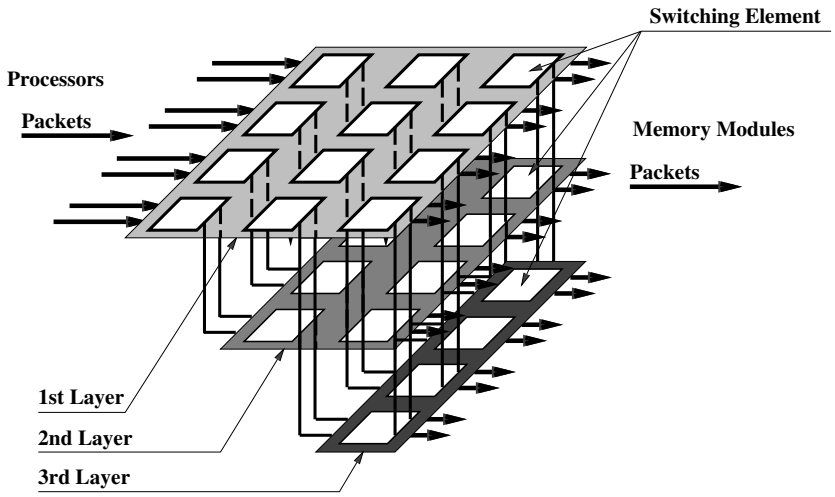

図 2.5: PBSF (Piled Banyan Switching Fabrics)

2.3　MIN 接続型マルチプロセッサの具体例

次に，MIN をプロセッサ-メモリ間の接続網に用いたマルチプロセッサについて，具体的な実装例を示す．

2.3.1　Cedar

Cedar は，1983 年に Illinois 大学で開発されたマルチプロセッサである．いままで演算レベルでデータ駆動的な処理を行っていたデータフローマシンに代わって，マクロデータフロー処理の概念が提案され，これに適した構成として実装された．図 2.7 に示した，Alliant 社の小規模マルチプロセッサ FX8 をクロスバで接続したクラスタを，図 2.6 のように，2 段の MIN(Omega 網) を介して Global Memory に接続する構成をとっている．1993 年に 32 プロセッサのシステムが稼働し，詳細な評価が発表されたが，それによると MIN の転送遅延が大きいことが，性能の低下の原因となってしまっていた．

2.3.2　Cenju-4

Cenju-4 は，1997 年に NEC が開発した，最大 1024 ノード構成の分散共有メモリをサポートしたマル千プロセッサである．図 2.8 のように，ノードは，プロセッサ R10000, 1Mbyte の２次キャッシュ，512MByte まで拡張可能なメモリ，PCI バス，メッセージ通信や DSM を実現する制御チップで構成される．また，各ノードは，4 入力 4 出力のクロスバスイッチを多段に構成した MIN で接続されている．

ディレクトリ方式の無効化型プロトコルでキャッシュ一致制御を行っており，ポインタ形式とビットパターン形式を組み合わせたディレクトリを採用することで，ディレクトリに必要なハードウェアコストを削減している．

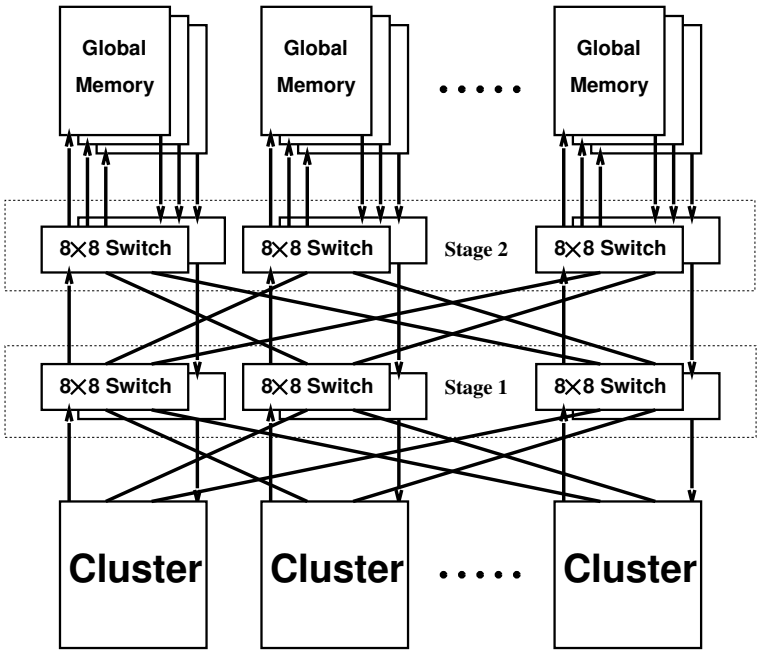

図 2.6: Cedar の構成

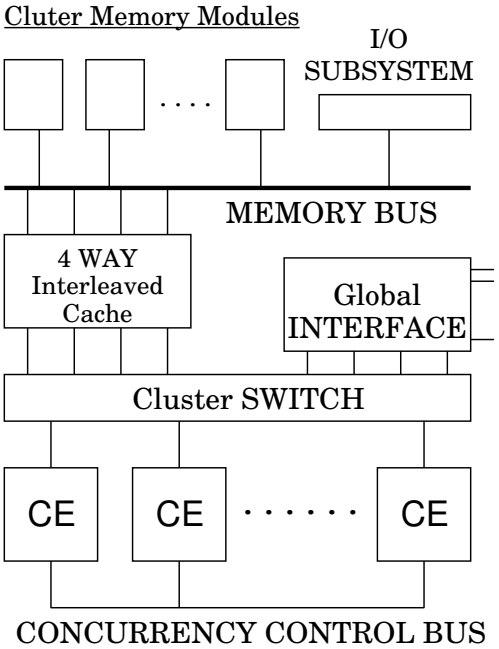

図 2.7: Cedar のクラスタ内構造 (FX-8)

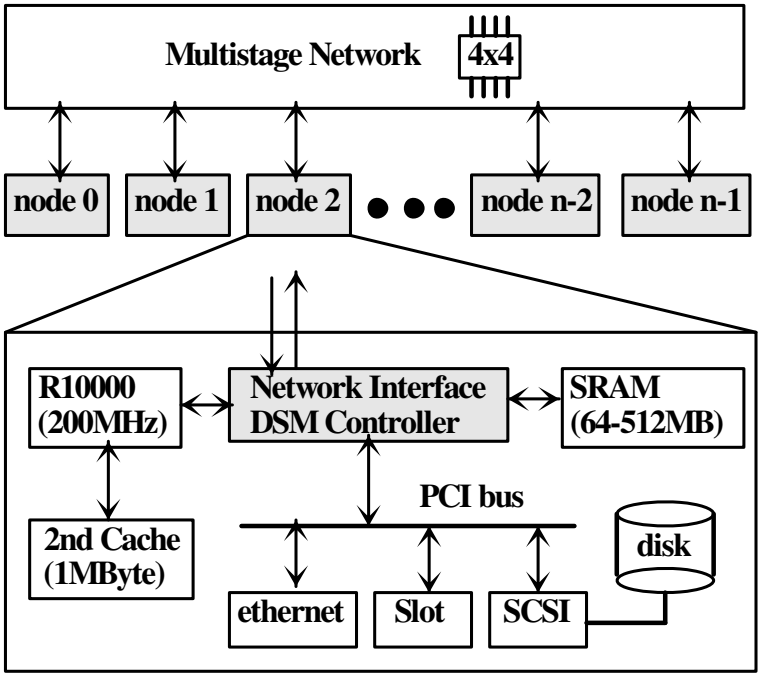

図 2.8: Cenju-4 の構成

これらの具体例を見ると，MIN 接続型マルチプロセッサは次のような問題点を有する．

- Cedar で問題となったように，MIN 接続型は多段のクロスバで構成されるためネットワークレイテンシが大きいという問題．
- Cenju-4 でディレクトリ方式のキャッシュ制御機構を採用しているように，バス接続型マルチプロセッサで用いられているスヌープ方式を利用したキャッシュ一致制御を採用することができず，キャッシュ一致制御が複雑になり，ハードウェアコストが高いという問題．

オンチップマルチプロセッサの接続網として MIN を採用した場合，接続網がプロセッサやメモリモジュールと同じチップ上に構成されるため，ネットワークレイテンシを大幅に削減することが可能である．

しかし，ハードウェアコストが高くなる問題が十分に解決されていないことから，MIN 接続型マルチプロセッサを用いたオンチップマルチプロセッサの登場には到っていない．そこで，このハードウェアコストの問題について検討を行っていることが重要である．

2.4　MIN におけるキャッシュ制御方式

MIN を用いたマルチプロセッサでは，その規模が大きくなるにつれ，共有メモリアクセスレイテンシが大きくなる．そこで，共有メモリに対するプライベートキャッシュをプロセッサ側に設けることができれば，システム全体の性能が格段に上がると考えられる．

しかし，MIN 結合型マルチプロセッサでは，バス結合型のように，スヌープ機構を用いたキャッシュ制御を行なうことができない．そこで，MIN 上でキャッシュ制御を行なう方法として，次に挙げるような，ソフトウェアを用いる方法とハードウェアで制御する方法が提案されている．

2.4.1 ソフトウェアによる方法

主にコンパイラ等による事前解析の結果を利用する方式．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，この方式ではキャッシュの効果が制限されてしまう．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[Vei86]，参照マーキング法[E+85]，部分的にハードウェアの助けを借りる方法[CV88]等様々な方法が提案されている．

2.4.2 ハードウェアによる方法

CC–NUMA で用いられるディレクトリ方式を適用する方式．

キャッシュの一貫性を維持するために，キャッシュラインの共有関係を保持するディレクトリを持つ．キャッシュに書き込みが起きた場合は，このディレクトリを参照してキャッシュラインを共有するプロセッサを調べ，これらに対し無効化メッセージあるいは更新データを送る．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ビットマップを保持するフルマップ方式をはじめ，リミテッドポインタ方式[ASHH88]，チェインドディレクトリ方式[JLGS90]，特定の場合にソフトウェアでエミュレーションする方式[CKA91]など様々な方式が提案されている．

ハードウェアによる方法は，上で挙げたようなコンパイラなどのソフトウェアによる方法に比べ，解析を必要としない分汎用性があり，キャッシュ制御方式として有利である一方，付加機能を設けなくてはならないため，スイッチングエレメントの構造が複雑になる問題がある．しかし，最近の LSI 実装技術の向上により，複雑な制御機構をスイッチ内部に組み込むことが可能となったため，ハードウェアによるキャッシュ制御機構が提案されるようになった．

2.4.3 階層ビットマップディレクトリ方式

CC–NUMA で用いられる最も基本的なディレクトリ管理方式は，全てのプロセッサのキャッシュ所有情報を管理し，それぞれのプロセッサに直接指示を出すことのできるフルマップ方式である．その中で，結合網が木構造か，それに類する階層構造を持つ場合に有効な方式として，階層ビットマップディレクトリ方式がある．

まず，木構造のネットワークの根の位置にメモリ，葉の位置にプロセッサが存在する場合を想定し，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．MIN はメモリをルートとする Fat-Tree 構造を内包しているため，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を自然に導入することができる．そのため，MIN により PU-メモリ間を接続するマルチプロセッサのキャッシュ制御機構として，広く採用されている．

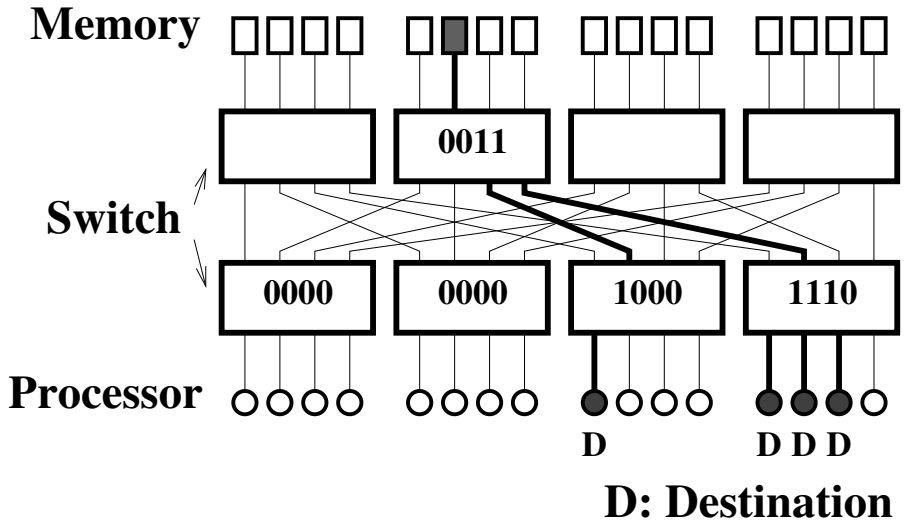

図 2.9: 階層ビットマップディレクトリ方式

図 2.9 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** を完全に指定することができる．しかし，単に全プロセッサを直接指示するビットマップを保持する方式では 1 ラインあたり n^m bit (m:ステージ数 n:スイッチサイズ) で宛先を指定できるのに対して，階層ビットマップディレクトリの場合は，1 ラインあたり $\sum_{k=1}^{m} n^k$bit 必要であり，より多くのビット数を必要としてしまう問題を有している．

なお，本論文中では，主に第 6 以降で比較対象としてフルマップ方式を示しているが，それらのフルマップ方式はこの階層ビットマップ方式を採用している．

2.5 ディレクトリ方式の問題点

スヌープ機構を用いずにキャッシュの一貫性を維持するためには，キャッシュラインの共有関係を示すディレクトリをいずれかに持つ必要がある．最も単純なフルマップ方式では，ディレクトリは対応する共有メモリのキャッシュライン単位に設けられ，そのラインのコピーを保持しているキャッシュの位置を記録する．あるキャッシュに書き込みを行う場合に，対応するディレクトリを参照し，そのラインを共有するキャッシュに無効化メッセージを転送することでキャッシュの一貫性を維持する．

この手法は，二つの問題点を持つ．

- 共有メモリのキャッシュライン数に対応するディレクトリが必要で，必要メモリ量

が大きい.

- MIN を介して共有メモリ側に設けられたディレクトリをアクセスするための遅延が大きい.

そこで，ディレクトリ方式に基づく並列計算機のキャッシュ制御機構として，キャッシュメモリやディレクトリを配置する位置や，ディレクトリに登録するビットマップを縮約して保持する方式がいくつか提案されている．次の章ではこれらの方式を紹介する.

第3章　関連研究

ディレクトリ方式の問題点として，ディレクトリに必要なメモリ量が大きい点や，ディレクトリのアクセス遅延が大きい点を前記したが，これらの問題を解決するために，さまざまなディレクトリ方式のキャッシュ制御機構が提案されている．

それらの方式をまとめると，キャッシュメモリやディレクトリを配置する位置を工夫することでメモリ量やアクセス遅延の解決をはかる方式，ディレクトリに登録するビットマップを縮約して保持することで必要なメモリ量を削減する方式に分けることができる．

この章では，これらの方式の関連研究を紹介する．そして，最後にこれらの関連研究の問題点をまとめ，メモリ量の問題とアクセス遅延の問題を十分に解決できていないことを示す．

3.1　キャッシュメモリやディレクトリの配置に特徴がある方式

3.1.1　MHN(Memory Hierarchy Network)

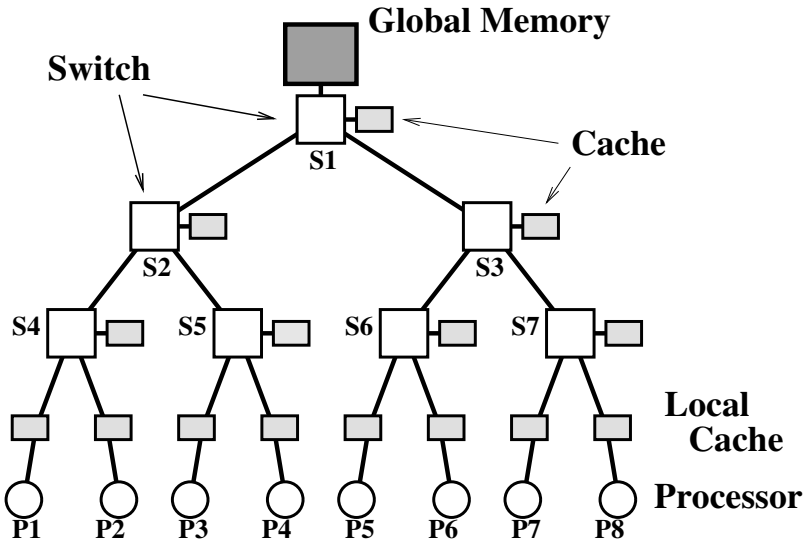

図 3.1: MHN の概観

MHN［MBLZ89］は，図 3.1 の様に，Tree 状のネットワークで構成されており，Tree の根の部分に共有メモリ，葉の部分に各プロセッサが配置されている．各プロセッサはローカルキャッシュを保持しており，さらに，各スイッチ中にも，そのエレメントを根とする

プロセッサのための共有データのキャッシュが搭載されている．

データブロックのコピーがシステム中に複数存在する状態を許さない Single Copy Protocol であり，共有データへのライトが起こった際に余計なコヒーレンス制御パケットの発生を抑え，トラフィックを少なくしている．また，プロセッサがデータブロックをアクセスした際，そのデータブロックはプロセッサの方向に 1 ステージだけマイグレーションされる．これにより，プロセッサ番号が近いもの同士だけがそのデータを利用する場合，そのデータアクセスのレイテンシは小さくなる．

しかしながら，ローカルキャッシュにデータが入るまでに少なくともステージ数と同じ回数のアクセスを行わなければならないため，ローカルキャッシュのヒット率が低くなってしまう．また，頻繁にマイグレーションパケットが発生することから，ネットワーク負荷が高くなってしまう．さらに，多くのプロセッサが共有するデータブロックは共有メモリ側のステージにマイグレートされていくため，アクセスレイテンシは大きくなってしまうなどの問題点がある．

3.1.2 CAESAR(CAche Embedded Switch ARchitecture)

MHN では無駄なコピーとデータ一貫性に関するロスを防ぐためにコピーは一つに制限されていたが，この考え方を発展させた CAESAR［RL00］は，スイッチ内にキャッシュを組み込み，複数のアクセスに対する応答が可能な現実的な構成を提案している．

CAESAR のスイッチングエレメントの構造は図 3.2 に示したように，ローカルキャッシュの他に，すべてのスイッチングエレメント内にキャッシュを持つ．

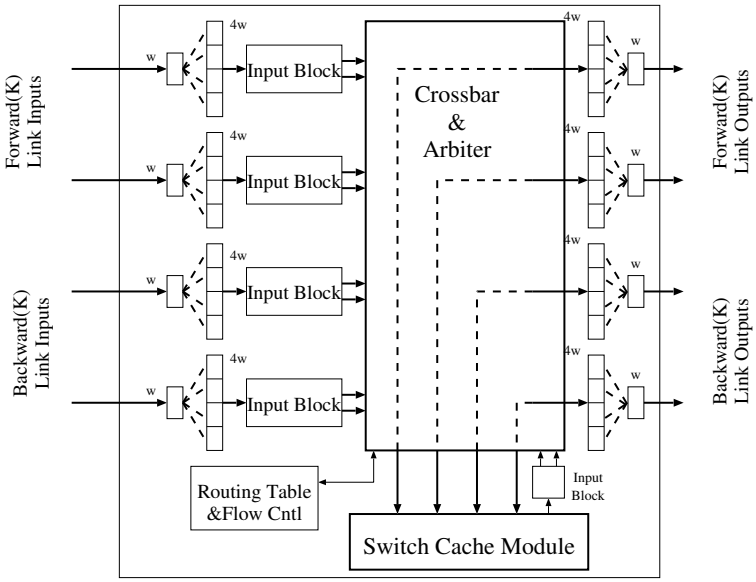

図 3.2: CAESAR のスイッチングエレメント

リードアクセスの際にローカルキャッシュにヒットした場合は，そのままプロセッサ内で処理が終わるが，ミスした場合は，共有メモリにリードリクエストのパケットが送られ

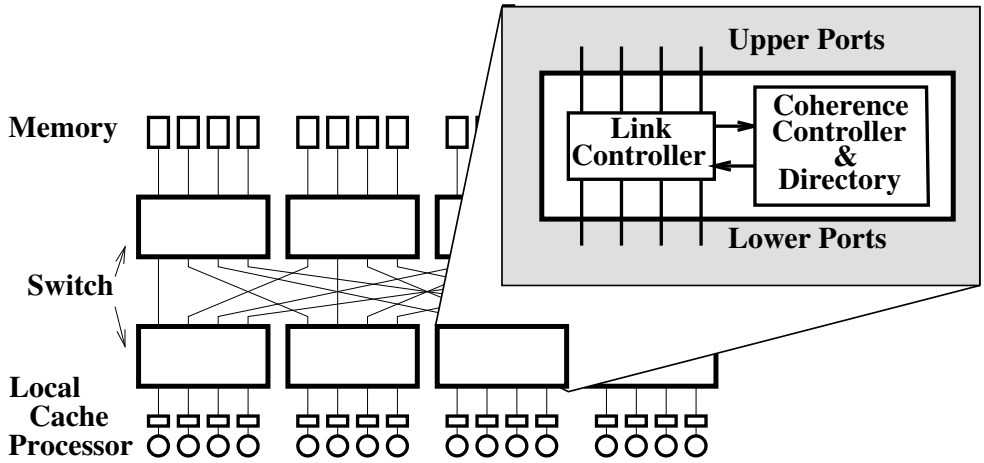

図 3.3: MIND の概観

る．そのとき，通過するスイッチングエレメントではパケットをチェックして自分がそのキャッシュラインを保持しているか調べ，保持している場合には，そのスイッチングエレメントがアクノリッジを生成し，要求元のプロセッサにキャッシュラインを送る．このように，スイッチングエレメントのキャッシュにヒットした場合はネットワーク途中でデータを得ることができるためにローカルキャッシュにミスした時のレイテンシを減らすことができる．スイッチングエレメントでキャッシュラインを保持していない場合は，共有メモリにリードリクエストのパケットが到達し，アクノリッジが生成される．いずれの場合も，アクノリッジが通過するスイッチングエレメントでは，自分がそのキャッシュラインを保持しているか調べ，キャッシュしていない場合はスイッチ内のキャッシュに登録する．

ライトアクセスの際はプロセッサが共有メモリにライトパケットを送るが，そのとき通過するスイッチングエレメントでそのラインを保持しているか調べ，キャッシュしている場合にはラインの無効化を行う．

しかし， MHN と CAESAR は各スイッチ内部のキャッシュに比較的大きなメモリ容量が必要となってしまう．

3.1.3 MIND(MIN with Directory)

MIND [NB93]は，MHN と CAESAR と異なり，各スイッチ内にキャッシュ自体は持たず，ディレクトリのみを置く方式である．

MIND では，各プロセッサにローカルキャッシュを持たせ，そのキャッシュ情報は各スイッチ内のディレクトリによって保持されている．各ディレクトリは階層ビットマップであり，自分の真下にあるスイッチングエレメントに関して，データブロックの所有情報をビットマップの形で保持している．

4×4 スイッチを用いた MIND を図 3.3 に示す．スイッチングエレメントはクロスバで構成され，エレメント内部には，ルーティング情報に従ってポート同士をリンクさせる Link Controller と，ディレクトリを保持しコヒーレンスアクションを制御する Coherence

	State	Presence bits			
		ch_1	ch_2	••••	ch_k
Block_1					
Block_2					
• • • •				••••	
Block_Nsb					

図 3.4: MIND で用いられているディレクトリ

Controller が設けられる．キャッシュの無効化メッセージは，図 3.4 に示したディレクトリに従って必要なプロセッサにのみにマルチキャストされる．

また，MIND ではコンパイラにより以下の 2 つのデータに分割され，別々のメモリモジュールに配置される．

- 単一のプロセッサのみが使用するプライベートデータ
- 複数プロセッサから使用される共有データ

これにより，プロセッサからの共有メモリアクセスパケットが各スイッチングエレメントに入った際，そのパケットの目的地メモリモジュールをチェックするだけで，共有データへのアクセスであるかどうかが判定できる．

仮に，そのパケットが共有データへのアクセスであった場合，エレメント内でディレクトリが参照されキャッシュ情報がチェックされる．そして，必要であればコヒーレンス制御パケットが発行される．それ以外のプライベートデータへのアクセスや，メモリからのリプライは単純に Link Controller を通過するのみで良い．

MIND は，スイッチングエレメントをクロスバで構成させていることから，通過率が高いという特徴を持つ．更に，MHN とは異なり，共有データのコピーがシステム中に複数存在することを許している (Multiple Copy Protocol) ので，プロセッサが持っているローカルキャッシュは高いヒット率を誇っている．

しかし，共有メモリへのリクエストパケットが必ずスイッチ内部のディレクトリを参照するためネットワークレイテンシが大きくなってしまう．また，スイッチ構造が複雑化することにより，スイッチサイズもせいぜい 4×4 程度以下となってしまう．さらに，ディレクトリが階層ビットマップディレクトリ方式であり，各スイッチ毎に，自身の直下にあるノードに対する全共有データのキャッシュ情報を保持しなければならないため，ディレクトリに必要なメモリ量が多く，スイッチのチップ外にディレクトリ管理用のメモリを設ける想定となっている．このことが，ディレクトリの参照に必要な時間を増大させ，ネットワークレイテンシを更に大きくしている．

3.1.4 MBN(Multistage Bus Network)

従来型の MIN では，スイッチングエレメントがクロスバで構成されているため，キャッシュ制御パケットのマルチキャストが困難であった．そこで，従来型 MIN でのリンク利用率が高々10%〜15%程度である事を踏まえて，バスが持つスヌーピング能力，ブロードキャスティング能力を利用しようとしたものが MBN［BNA94］である．MBN の概観を図 3.5 に示す．スイッチングエレメントの構成以外はキャッシュ制御プロトコルを含め MIND と全く同じである．

MBN のスイッチは，キャッシュコヒーレンス維持に関する制御を行なう TC(Traffic Controller)，スヌーピングバスに対するアクセスを制御する BAC(Bus Access Controller) からなる．TC は，ディレクトリと Coherence Contoller を保持しており，バスを通過する全てのリクエストを監視している．リクエストが共有データに関するアクセスであれば，ディレクトリを参照してキャッシュ情報をチェックし，必要であればコヒーレンス制御パケットが発行される．それ以外のプライベートデータに関するアクセスやメモリアクセスに対する応答パケットであれば，速やかに先に転送する．

MBN の利点として，MIND ほど Coherence Controller が混雑せず，レイテンシが小さくなることが挙げられる．これは，MIND では共有データへのリクエストパケットの全てが Coherence Controller を通過する必要があったのに対して，MBN では，スヌーピングバスがフィルタの役目を果たし，本当にコヒーレンスアクションが必要なリクエストのみを，Coherence Controller に送る事が可能であるからである．もう一点は，MIND と同様に，Multiple Copy Protocol を採用しているため，ローカルキャッシュで高いヒット率を示すということである．ただし，MBN は，スイッチ内部にバスを用いているため，MIND に比べて通過率さがってしまうという欠点もある．また，MIND と同様にディレクトリに必要なメモリ量が多く，スイッチのチップ外にディレクトリ管理用のメモリを設ける想定となっているため，ディレクトリの参照による遅延が大きくなってしまう．

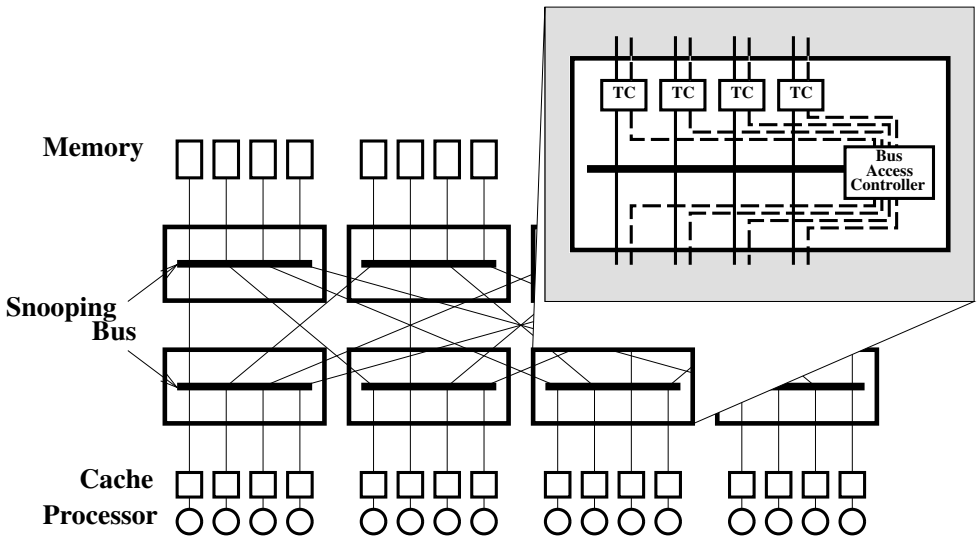

図 3.5: MBN の概観

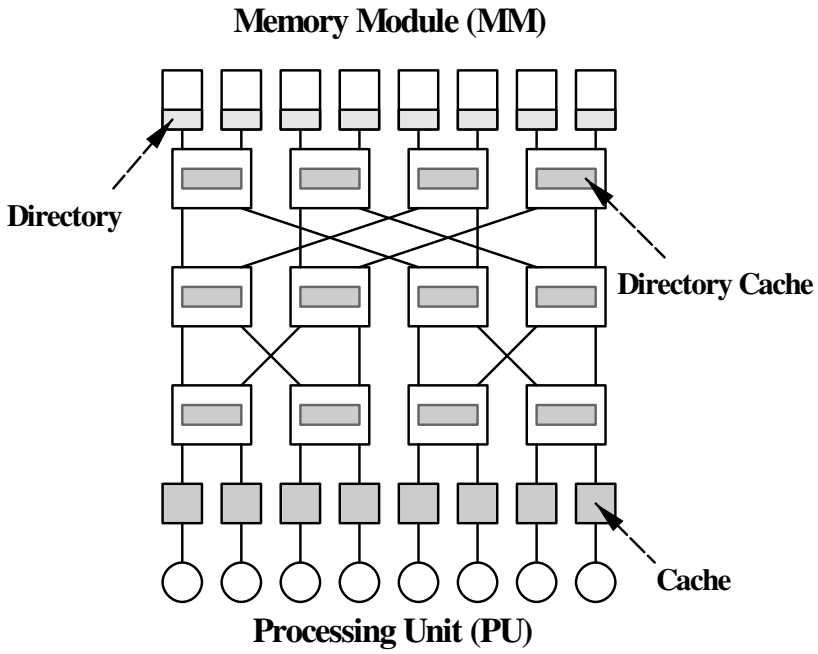

図 3.6: DRESAR の概観

3.1.5　DRESAR

MIND と MBN は，スイッチに配置するのがディレクトリのみであるが，依然としてスイッチに必要となるメモリ量が多く，スイッチのチップ外にディレクトリ管理用のメモリを設ける想定であるため，ディレクトリのアクセス時間が大きい．DRESAR［IBN00］は，この問題の解決し，スイッチ自体の構成の実現可能性を高めた方式である．

図 3.6 に示すように，DRESAR はディレクトリを共有メモリ側へ配置し，各スイッチ内に設けたディレクトリキャッシュにディレクトリのコピーのみを搭載してスイッチ内のメモリ量を削減すると共に，ディレクトリのアクセス時間を短くしている．

DRESAR は，ライトバック方式を採用し，ライトアクセスの後に発生する可能性がある，共有メモリとプロセッサのキャッシュの内容が異なる状況でのリードアクセスを高速化することを主な目的としている．

ライトバック方式では，ライトアクセスをしたプロセッサのキャッシュにしか有効なデータが存在しない状況で，他のプロセッサが共有メモリへリードアクセスした場合，共有メモリに設けられたディレクトリを参照して，共有メモリからキャッシュラインを保持しているプロセッサのキャッシュに対して，リードアクセスしたプロセッサへのキャッシュラインの転送命令を送信する．その後，ラインを保持しているキャッシュからリードアクセスを出したキャッシュへのデータ転送が行われる．

DRESAR では，各スイッチ内にディレクトリキャッシュを設け，キャッシュラインの転送命令を共有メモリではなくスイッチから送信することで，高速化を実現している．スイッチではライトアクセスへのアクノリッジを監視し，ライトアクセスしたプロセッサの位置をディレクトリ内に保持する．そのため，他のプロセッサがリードアクセスを送信し，ス

イッチ内のディレクトリにライトアクセスしたプロセッサの位置が保持されていた場合は，そのスイッチからキャッシュラインの転送命令を，キャッシュラインを保持しているプロセッサに送信することができる．

しかし，DRESAR は，共有メモリ側にフルマップ方式のディレクトリが必要なことから，共有メモリ側に膨大な量のメモリが要求される．

3.1.6 ANT

ANT(**A**ddress transfer **N**etwork with **T**emporary directory) [Tan03]は，テンポラリディレクトリ (TD) と呼ぶ小容量のディレクトリをクロスバスイッチ内に搭載したアドレス転送用ネットワークである．

アドレス転送用ネットワーク ANT の構成を図 3.7 に，また，ANT を用いたシステムの概観を図 3.8 を示す．図 3.7 で示す構成例は，4×4 のスイッチ構成であり，ネットワークの入力部には，入力されたパケットを格納する Input Buffer を設けられている．Input Buffer から出力されたパケットはキャッシュ制御を行なう Temporary Directory Unit(TD Unit) へと入力される．TD Unit に入力されたパケットは，要求の種類や，キャッシュのヒット，ミスによって適切な宛先にパケットが転送されるように経路情報を書き換えられる．宛先としては，メモリアクセス用に MM があるだけではなく，キャッシュ制御用の無効化パケットやキャッシュラインの転送を行なうために PU がある．TD Unit から出力されたパケットは，経路情報に応じクロスバを介して，MM や PU に送信される．

ANT はクロスバスイッチに内蔵された小規模な SRAM にディレクトリ情報を保持することで，高速なディレクトリ情報の参照を可能にしている．SRAM 上に保持できなくなったディレクトリ情報に対応するキャッシュラインを PU 側のキャッシュから無効化ことで，全体として低コストなキャッシュ管理を実現している．

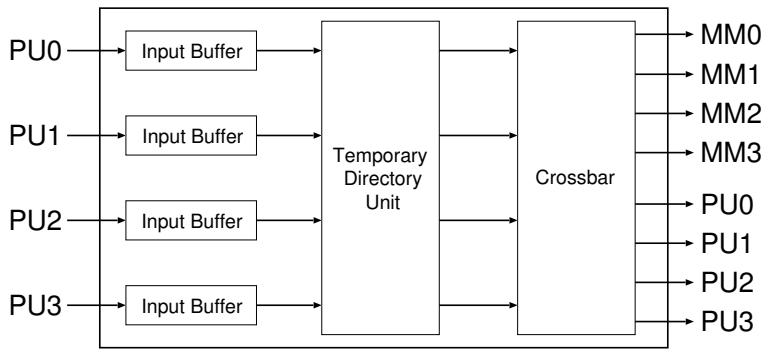

図 3.7: ANT の構成

3.1.7 Sparse-Directory

Sparse-Directory [GWM90]は，CC-NUMA のプロセッサノードに設けられたディレクトリを小容量のメモリで構成し，共有情報を保持できない場合は，不要な共有情報に該当す

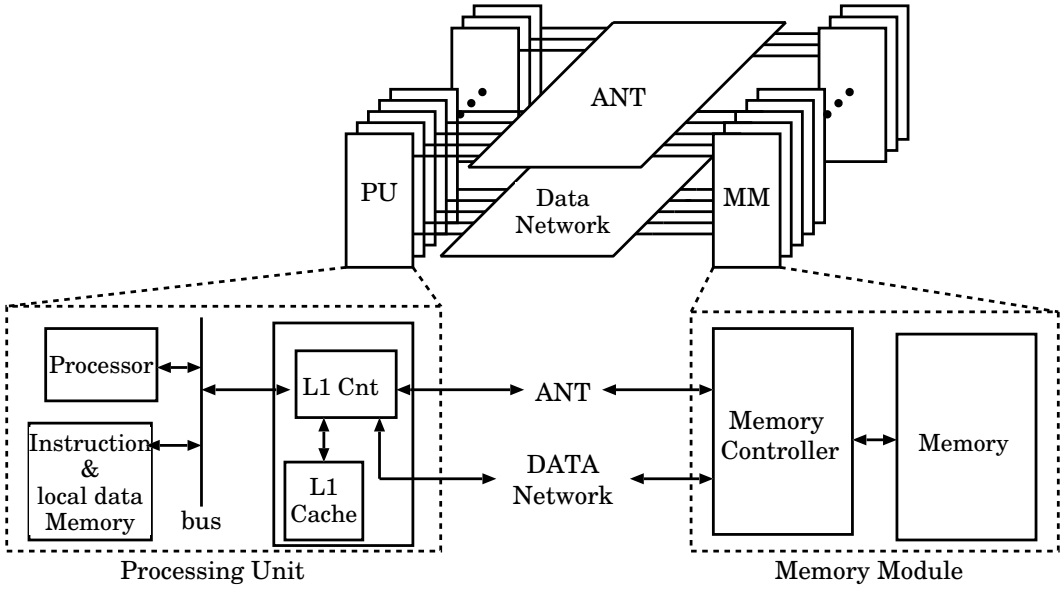

図 3.8: ANT を用いたシステムの概観

るキャッシュラインを無効化してキャッシュの一貫性を維持することで，必要なメモリ容量を大幅に削減した方式である．

ディレクトリにキャッシュの共有情報を登録しようとした際に，ディレクトリのエントリがすでに使用されており，新たな共有情報を保持できない場合，すで登録されている共有情報から一つのエントリを選び出して追い出し，該当するキャッシュラインに無効化パケットを送信する．このようにしてキャッシュを無効化して，エントリに空きを作成することで，新たな共有情報を登録することができる．

各プロセッサノードにキャッシュされているキャッシュラインを無効化することから，頻繁にエントリ不足による無効化が発生するとキャッシュのヒット率が低下してしまうが，エントリ数を十分に確保することでエントリ不足による無効化の影響をほとんど受けることなく，メモリ容量の大幅な削減が可能である．

しかし，MIN のスイッチ上などのネットワーク上にディレクトリを設ける構成でなく，ネットワークを介して接続された各ノード上にディレクトリを設ける構成であるため，PU からネットワーク越しにディレクトリをアクセスする必要があり，アクセス遅延が大きい．

3.2　ディレクトリを縮約して保持する方式

ディレクトリの容量を削減するために，保持するディレクトリを縮約し，従来より少ないビット数で共有情報を出来るだけ正確に記録する方法が提案されている．ここでは，それらの方式について紹介する．

3.2.1 Coarse Vector

Coarse Vector [GWM90]は，共有者数が i まではリミテッドポインタ，共有者数が i を超えたら図 3.9 に示すような，K プロセッサのグループ毎に 1 ビットを用いるビットベクタ方式にスイッチする方式である．

K はディレクトリとして使えるメモリの容量によって決めることができる．ノード数が N のとき，グループに含まれるノード数を K とすると，キャッシュライン当たりに必要なディレクトリは $\lceil \frac{N}{K} \rceil$ ビットとなる．

図 3.9 は，ノード数 16，$K = 4$ の例である．○はキャッシュを保持していないノード，●はキャッシュを共有するノードを表しており，● を含むグループのビットを 1，含まないグループのビットを 0 として，共有情報をビットベクタで表現している．

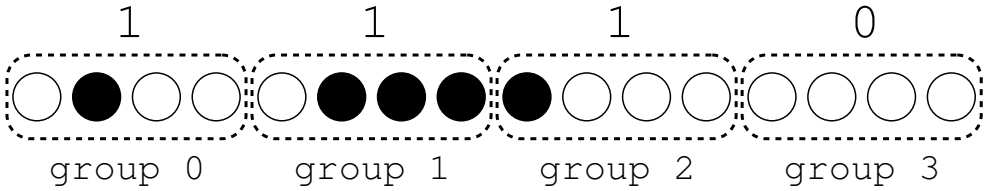

図 3.9: Coarse Vector

3.2.2 Gray-Tristate

Tristate

Tristate は，$logN$ 桁のコードを用い，1 つの桁に 2 ビットを割り当て，3 つの状態を表す方式である．共有コードの j 番目の桁は，もし全共有者 ID の j 番目のビットが 0 なら 0，1 なら 1，0 と 1 が両方存在すれば both とする．both の数が k の場合，無効化メッセージ数は 2^k となる．

Tristate では，図 3.10 に示すように，ノードに割り当てる番号は順番に 0，1，2，... としているため，隣接する 2 つのノード間のビットの違いが $k(k > 1)$ となり，隣接する 2 共有者への無効化メッセージは 2^k 発生してしまう．

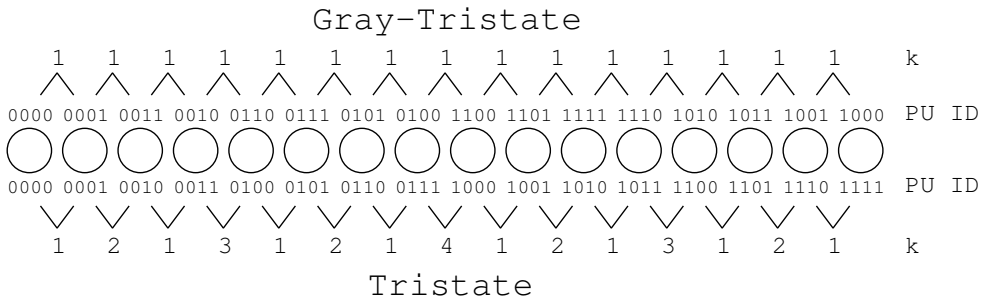

図 3.10: Tristate と Gray-Tristate

Gray-Tristate

binary reflected gray code を用いて Tristate の隣接ノード間で共有する場合を改善したのが Gray-Tristate［MH94］である．Gray-Tristate では，gray コードを用いることで，図 3.10 に示すように，隣接ノード間のビットの違いが全て 1 ビットになっており，2 つの隣接ノード間での共有の際に発生する不必要な無効化メッセージの発生を排除している．

gray コードのエンコード及びデコード方法の違い，Tristate の変種として以下の 3 つが提案されている．

- Gray-hardware: ハードウェアで gray コードのエンコード，デコードを行う．1 次元的な隣接共有のみに対応している．1 次元の gray コード変換は，ノード ID を右にビットシフトしたものと元のノード ID の XOR を取ることで実現できる．
- Gray-software: ソフトウェアで gray コードのエンコード，デコードを行う．サポートする隣接共有の次元を柔軟に変えることができ，この 3 つの中で最も性能が良い．さらに，Tristate と同じハードウェアで実現できる利点もある．
- Home: 全共有者の j ビット目がホームノードと同じ場合は 0, 一つでも異なる場合は 1 とすることにより，1 桁を表現するためのビット数が 1 で済む．これにより Tristate と比較してディレクトリ容量は半分に削減できる．しかし，ホームノードが常に共有に参加している状態でないと大量な無駄無効化メッセージが発生してしまい性能悪化を招くので，データの配置を考慮しなければならない．

3.2.3 Multilayer clustering

Multilayer Clustering［AGGD01］では，ノードは論理的に木構造を構成していると見なし，あるキャッシュラインの共有者全てがサブ木に含まれるように根を選び，その根が存在するレベルで共有情報を表す．この Multilayer Clustering を基に，3 つの共有コードが提案されている．

- Binary Tree (BT): ホームノードを含む木で全共有者をカバーし，その木の高さが最小になるようにする．図 3.11 の例は，● がホームノード，◍ が共有者を表している．網で示した，ホームノードを含み全ての共有者をカバーするサブ木の高さは level 3 であり，これを共有情報とする．

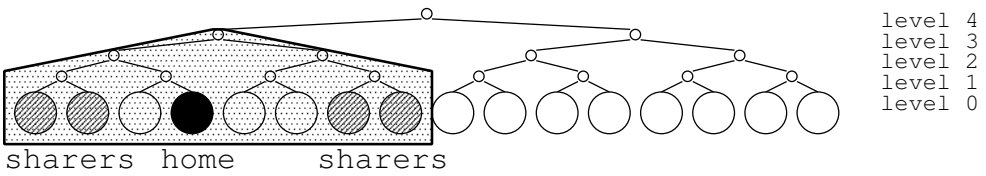

図 3.11: Binary Tree

- Binary Tree with Symmetric Nodes (BT-SN): ホームノードからみて対称な位置にあるノードであるシメトリックノードを導入し，ホームノードを含む木またはシメトリッ

クノードを含む木で全共有者をカバーする．図 3.12 の例では，● がホームノード，▨ がシメトリックノードである．

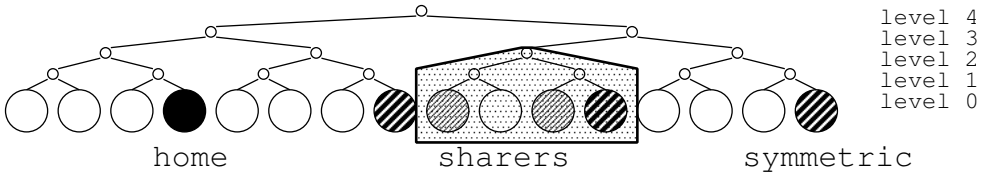

図 3.12: Binary Tree with Symmetric Nodes

- Binary Tree with Subtrees (BT-SuT): 共有者が 1 つの場合は直接それを特定する．共有者が複数の場合は，図 3.13 のようにホームノードを含む木とシメトリックノードの 1 つを含む木で全共有者をカバーすることにより，離れたノード間での共有にも対応する．

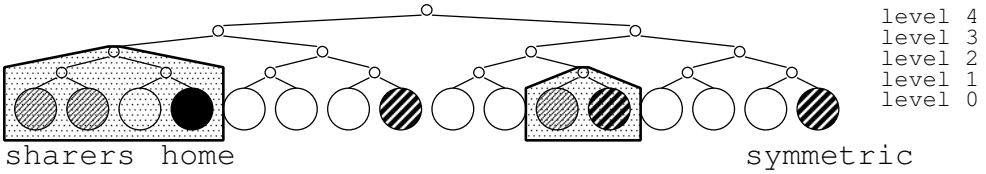

図 3.13: Binary Tree with Subtrees

3.3 従来のディレクトリ方式を用いたキャッシュ制御方式のまとめ

必要なメモリ容量の問題を解決するための手法として，縮約した共有情報をディレクトリに保持することで，ディレクトリに必要なメモリ量を削減する方法は，フルマップ方式で管理するよりはメモリ量を削減できる．しかし，ディレクトリを縮約したことにより無駄な無効化が生じたり，管理が複雑となる問題点がある．また，依然として MIN を経由してディレクトリをアクセスする必要がある点でアクセス遅延が問題となる．

一方で，MIN の持つ階層構造を利用し，MIN のスイッチ内にキャッシュ自体や，ディレクトリまたはディレクトリのコピーを内蔵する方法を紹介した．

MHN(Memory Hierarchy Network) では，無駄なコピーとデータ一貫性に関するロスを防ぐためにコピーは一つに制限させた．この考え方を発展させた CAESAR(CAche Embedded Switch ARchitecture) は，スイッチ内部にキャッシュを組み込み，複数のアクセスに対する応答が可能な現実的な構成を提案している．しかし，MHN と CAESAR は各スイッチ内部のキャッシュに比較的大きなメモリ容量が必要となる．

これに対し，MIND(MIN with Directory) [NB93]は，各スイッチ上にキャッシュ自体は持たず，ディレクトリのみを置く方式である．キャッシュの無効化メッセージは各スイッチ内でディレクトリが参照され，必要なプロセッサにのみマルチキャストされる．また，マルチキャストをより効率化するため，MIN の各スイッチをバスで構成する MBN(Multistage BusNetwork) [BNA94]も提案されている．しかし，スイッチに配置するのがディレクトリのみでも依然としてスイッチに必要となるメモリ量が多く，スイッチのチップ外にディレクトリ管理用のメモリを設ける想定であるため，ディレクトリのアクセス時間が大きい．

この問題の解決のため，スイッチ自体の構成の実現可能性を高めた DRESAR［IBN00］も提案されている．DRESAR はディレクトリを共有メモリ側へ配置し，スイッチ内にはディレクトリのコピーのみを搭載してスイッチ内のメモリ容量を削減すると共に，ディレクトリのアクセス時間を短くしている．しかし，共有メモリ側にフルマップ方式のディレクトリが必要なことから，共有メモリ側に膨大な量のメモリが要求される．

また，小容量のメモリのみでディレクトリを構成する方法として，ANT と Sparse-Directory が提案されているが，ANT は小規模のプロセッサ数での構成を想定したものであり，また，Sparse-Directory は，ネットワーク上にディレクトリを設ける構成でなく，ネットワークを介してディレクトリをアクセスするため，アクセス遅延が大きい．

従来のキャッシュ制御機構の問題点

そこで，これまでに述べた方式の問題点をまとめると次のようになる．

スイッチのハードウェアコストの増加とネットワークレイテンシの増大

スイッチ内部にキャッシュ自体やディレクトリを設ける方式では，キャッシュ制御用の複雑なハードウェアを各スイッチングエレメントに付加する必要がある．また，MHN ではキャッシュ自体，MIND・MBN ではディレクトリと，膨大なメモリをスイッチングエレメント内部に実装するため，ネットワーク全体では膨大なハードウェア量を必要とする．

また，各スイッチングエレメントがキャッシュ制御を自律的に行なうため，パケットがスイッチを通過する度に，ディレクトリをアクセスし，キャッシュ制御を行なわなくてはならない．さらに，現在の LSI 実装技術では，全ディレクトリを保持できるだけのメモリを，制御回路と共に 1 つのチップに載せることはできず，チップの外に設ける必要がある．そのため，チップ外部のメモリをアクセスに時間がかかってしまい，高速動作が困難となる．

スイッチ外にディレクトリを設けることによるディレクトリアクセス時間の増大

Sparce-Directory のように，スイッチ外にディレクトリを設ける方式では，ネットワークを介してディレクトリをアクセスに行く必要があるので，ディレクトリを参照するために時間がかかってしまう．

以上，様々な方式が提案されているが，いずれもメモリ容量やアクセス時間の問題を十分に解決できておらず，改善の余地がある．そこで，これらの問題点を改善するために，次の２つの要件を満たすキャッシュ制御機構が求められている．

- チップ内に実装できる小容量のメモリのみでディレクトリを構成
- PU を接続するネットワーク内にディレクトリを配置

そこで，これらの要件を満たす方式の検討を行うが，先ずは第 1 の試みである MINC について， 第 4 節で説明し，その評価結果を示す．その後，第 5 節では，別の方式である MINDIC を提案し，評価を行う．

第4章　MINC

4.1　MINC の概観

MIND は，スイッチ内にキャッシュを持つ必要はないが, ディレクトリを持つ必要があり, 同様にスイッチ内に大きなメモリと頻繁な同時アクセスを必要とする．MIND ではチップ外メモリを想定しているためアクセス時間が大きく，ステージ数が大きい場合に問題となる．また，DRESAR はスイッチ構成を工夫してこの問題の解決を図っているが，共有メモリ側にフルマップ方式のディレクトリが必要なことから，共有メモリ側に膨大な量のメモリが要求される．MINC(MIN with Cache control mechanism) はこのような問題を解決し，実装が容易な制御機構とすべく提案された，キャッシュ制御機構である．

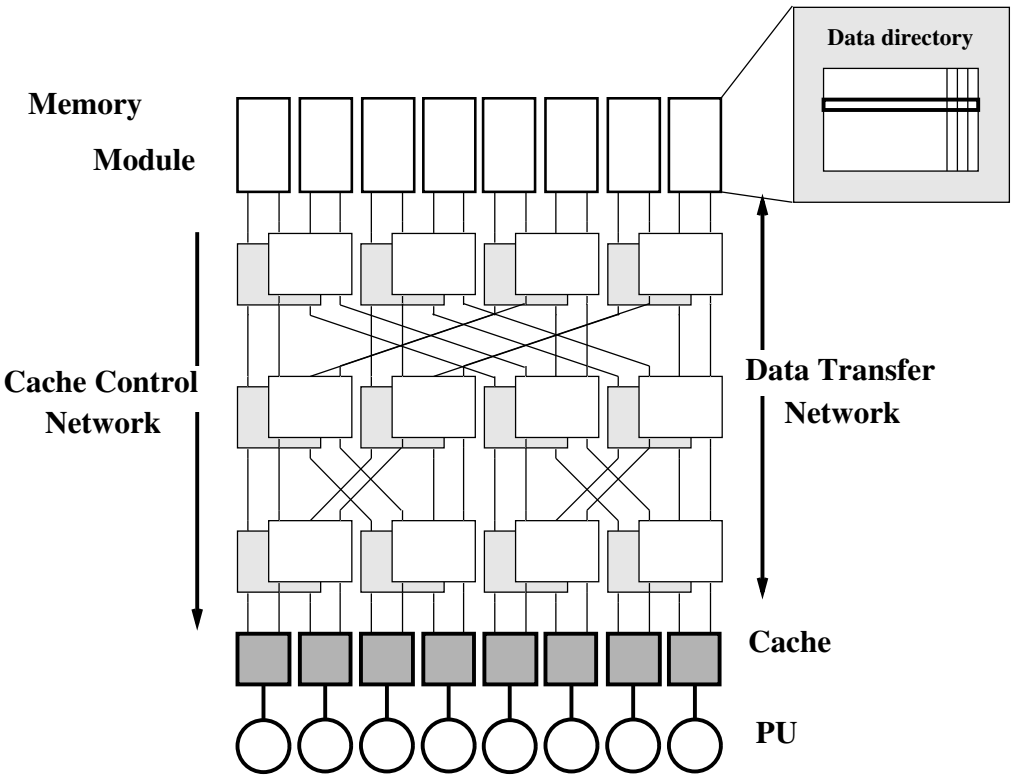

図 4.1: MINC の構成

MINC では，スイッチ上に大容量のキャッシュやディレクトリを保持すること自体が実装を困難にする要因と考え，ディレクトリは共有メモリ，キャッシュはプロセッサと直結

させ，スイッチには，キャッシュ制御を容易かつ効率的に実行するための機構のみを持たせた．このため，MINC におけるプロセッサ-メモリ間接続は，図 4.1 に示すように，プロセッサ・メモリ間のデータ転送を行う Data Transfer Network と，キャッシュ制御パケット転送を行う Cache Control Network に分かれている．Data Transfer Network は特に限定されておらず，転送能力の高いクロスバや多段結合網を用いることができる．

また，ライトスルー方式を採用しており，最新のデータとキャッシュ情報が常に共有メモリに存在している．共有メモリへのライトアクセスが発生した時には，ディレクトリが参照されて，キャッシュラインを有するプロセッサに対して無効化パケットがマルチキャストされる．

MINC ではディレクトリ管理のために，超並列マシン JUMP–1 のディレクトリ制御用に考案されたビットマップのビット数を縮約する縮約階層ビットマップディレクトリ（RHBD:Reduced Hierarchical Bit-map Directory）方式[NKA98]の中の SM(Single Map) 法を用いている. 書き込み要求の発生時に，ディレクトリの共有情報が参照され，キャッシュラインを保持している PU に対して，MM から無効化要求パケットが MIN 構造のネットワークを介して転送される．MINC では，共有情報が縮約されているため必要以上の無効化要求パケットが発生するという問題を有している．

そこで，MINC は無駄なマルチキャストパケットによりネットワークが混雑し，通過率が下がってしまうことを防ぐために，Backward MIN 上に枝苅りキャッシュを設け， 無駄パケットの数を減らしている．

4.2 RHBD(Reduced Hierarchical Bit-map Directory) 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るため，スイッチを構成するチップの外部にメモリを設ける必要がある．しかし，メモリをチップ外に出してしまうと，アクセス速度が低下する上，さらに階層毎にディレクトリを参照すると，大きな時間を要する．

そこでメモリ容量を節約するために，ディレクトリの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であり，

- 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になる．

これら 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[KYN+95]が，MINC では次の 2 つの方法について検討している．

- **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う．ある節で送信元を含む枝にパケットが送られると，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 4.2 に三進木を用いた摸式図を示す． この図で **s** が送信元のプロセッサ， **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プロセッサである．従って， **d** の無い ● は無駄なパケットを受けとるプロセッサを示す．

階層ビットマップを用いた場合，1 ラインあたり n^m bit のメモリが必要だが，縮約方式を用いた場合， わずか $n \times m$ bit のメモリでディレクトリを構成することができる．

また，複数の節で同一のビットマップを使うことから，一度の転送で使用するビットマップも少なくなるため，Backward MIN を通過するパケットのヘッダに各ステージで用いるビットマップを入れておき， そ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とができる．これにより，スイッチング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．

RHBD 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， スイッチ内で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， パケット転送の実行速度も向上する．

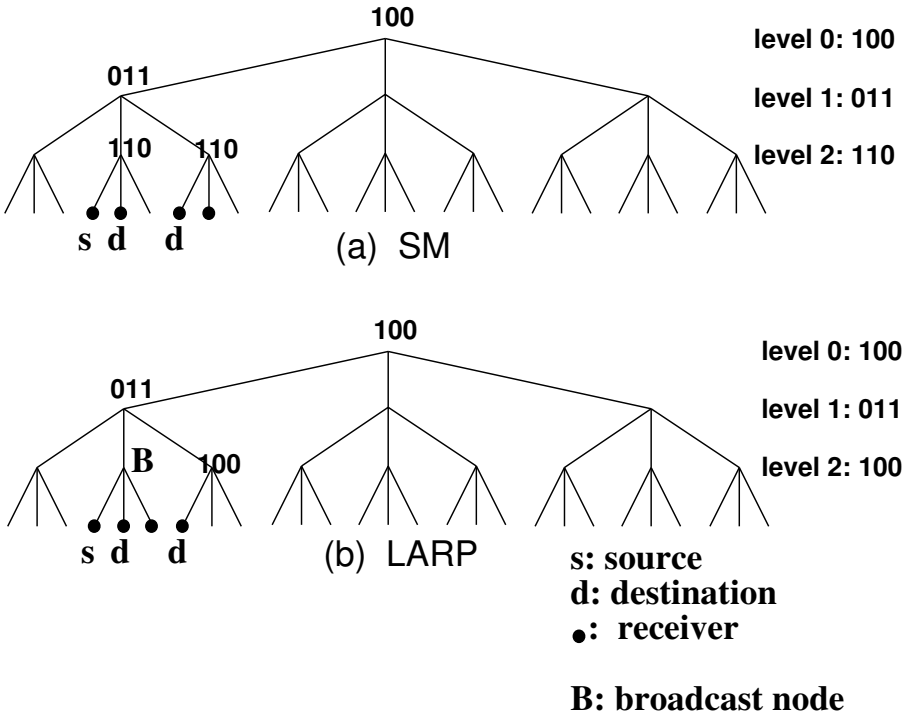

図 4.2: ディレクトリ縮約方式

4.3 枝刈りキャッシュ

縮約方式は大きな利点を持つ一方，送る必要のないプロセッサにもパケットが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，無駄なパケットによる Backward MIN の混雑が性能低下の原因となってくる[KYN+95]．

そこで，Backward MIN 上の特定の階層のスイッチに，枝刈りキャッシュと呼ばれるチップ内部に実装できる程度の小容量のビットマップキャッシュを設ける．図 4.3 は，LARP 型の縮約方式に枝刈りキャッシュを適応させたものである．枝刈りキャッシュは以下のように動作する．

- 読み出したラインを Backward MIN を通して要求元のプロセッサに転送する際，通過する枝刈りキャッシュ上にラインアドレス，対応するビットマップを登録する．既にエントリが存在した場合は，ビットマップの変更を行う．
- Backward MIN を通して更新データあるいは無効化パケットを送る時，枝刈りキャッシュが参照され，アドレスが一致したら枝刈りキャッシュのビットマップが利用され必要な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，枝刈りキャッシュ上のエントリは，無効化パケットを送った時点で削除する．

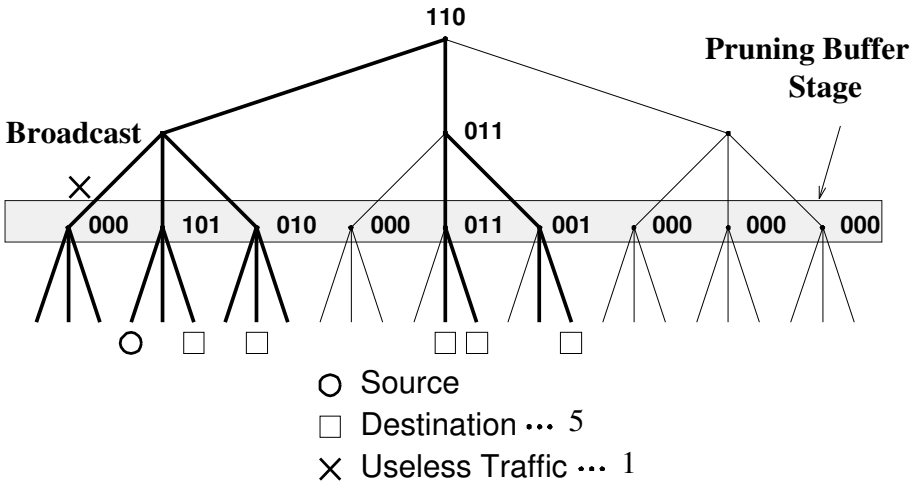

図 4.3: 枝刈りキャッシュ

枝刈りキャッシュは高速なアクセスが要求されるため，スイッチ内に実装することが必要であるため，大容量のメモリを利用することができない．したがって，枝刈りキャッシュの領域がなくなる場合が頻繁に起きると考えられる．そこで，枝刈キャッシュにエントリする時に，既に枝刈キャッシュが埋まっていてエントリできない場合，エントリを諦める．しかし，エントリが失われた後に，再びエントリが行なわれると，掃き出された枝刈キャッシュに対応するビットがセットされていないのに，その枝刈キャッシュの管理下にあるプロセッサがラインを持っていることがあり得る．この場合，枝刈キャッシュと共有メモリにあるディレクトリの間で情報の不整合が生じ，パケットを正しい宛先に転送できなくなる．

情報の不整合の発生について，枝刈キャッシュのエントリを諦めた場合の例を図 4.4 に示す．プロセッサ A が共有メモリのデータを読んだ時に，キャッシュラインはデータ転送用のネットワークを通ってプロセッサに送られるが，枝刈キャッシュが埋まっているとエントリできない (図 4.4 上)．その後，書き込みの発生などで枝刈キャッシュに空きができた後，先ほどとは別のプロセッサ B が，同じキャッシュラインのデータを読んだ時，枝刈キャッシュにはプロセッサ B のみが登録される．従って，実際にキャッシュラインのコピーを持つプロセッサは A と B だが，枝刈キャッシュの指し示す情報はプロセッサ B のみとなり，不整合が生じる (図 4.4 下)．このままマルチキャストを行なうと，枝刈キャッシュでパケットの宛先を指定した結果，プロセッサ A にはパケットが届かない．

そこで，マルチキャストパケットが正しく届くことを保証するために，枝刈キャッシュが埋まっていた時の動作として，枝刈キャッシュへの登録を諦め，共有メモリに対してエントリに失敗したことを知らせる方式を採用している．

4.3.1 アクノリッジ信号線のパケット化

キャッシュ一致制御のパケットを Backward-MIN 上でマルチキャストした時，転送の途中でパケットが衝突し，目的のプロセッサのうちの一部のプロセッサにのみパケットが届いて，残りの一部には届かないことがあり得る．

図 4.5 にマルチキャストパケットが衝突する例を示す．図では，メモリモジュール a から プロセッサ A, B, C, D にマルチキャストを行ない，同時に メモリモジュール b から プロセッサ B にパケットを転送する．ここで，メモリモジュール b から送られたパケットが，メモリモジュール a から送られるマルチキャストパケットよりも優先度が高く，ステージ 2 のスイッチで図のように衝突した場合，マルチキャストパケットは，プロセッサ B にのみ届かない．この場合，届かなかったプロセッサに対して再送する必要があるが，単純にアクノリッジ信号の AND をとると，全ての宛先に届くようになるまで，アクノリッジが返らなくなる．また，宛先を完全に特定するためには，アクノリッジを宛先毎に収集して，共有メモリ側でビットマップを再構成しなくてはならない．

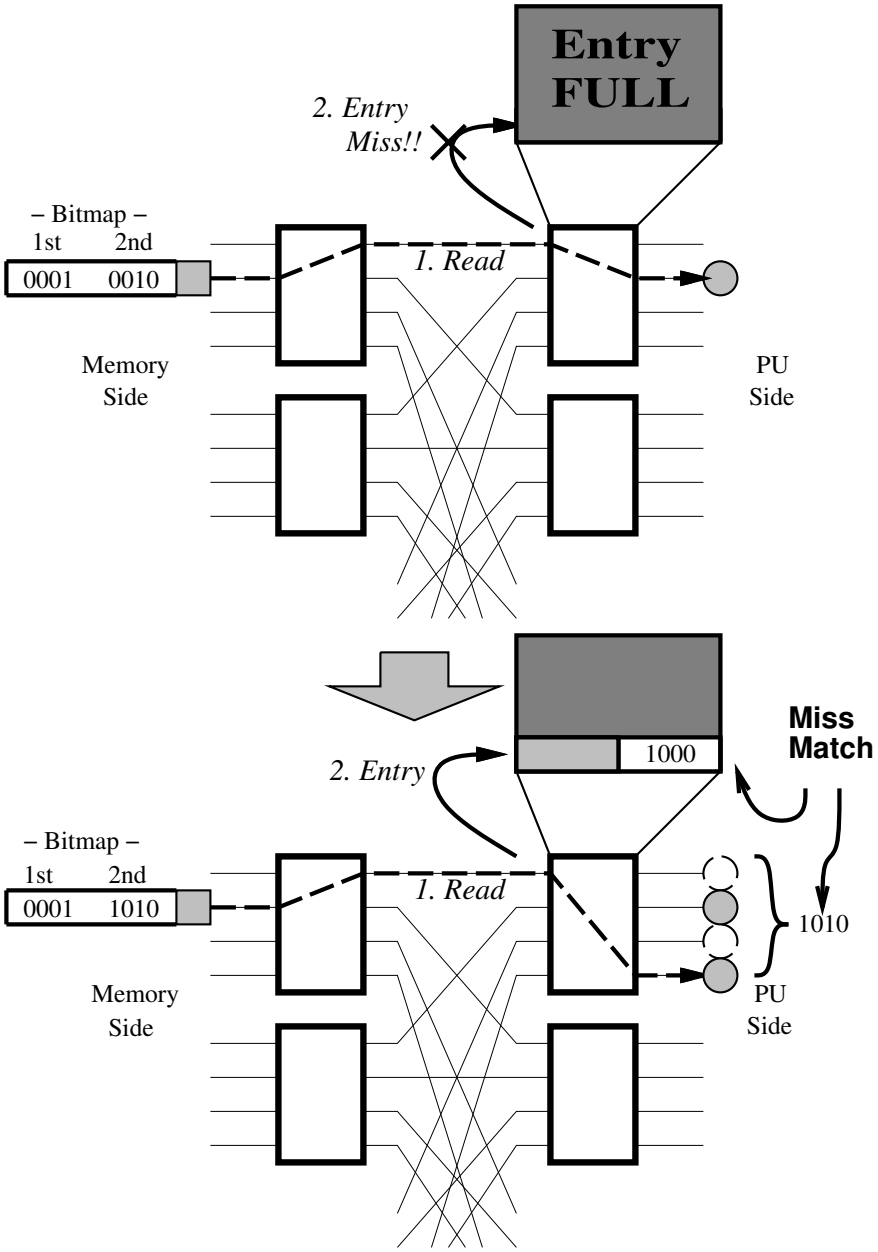

図 4.4: 枝刈キャッシュのエントリを諦めた場合の問題点

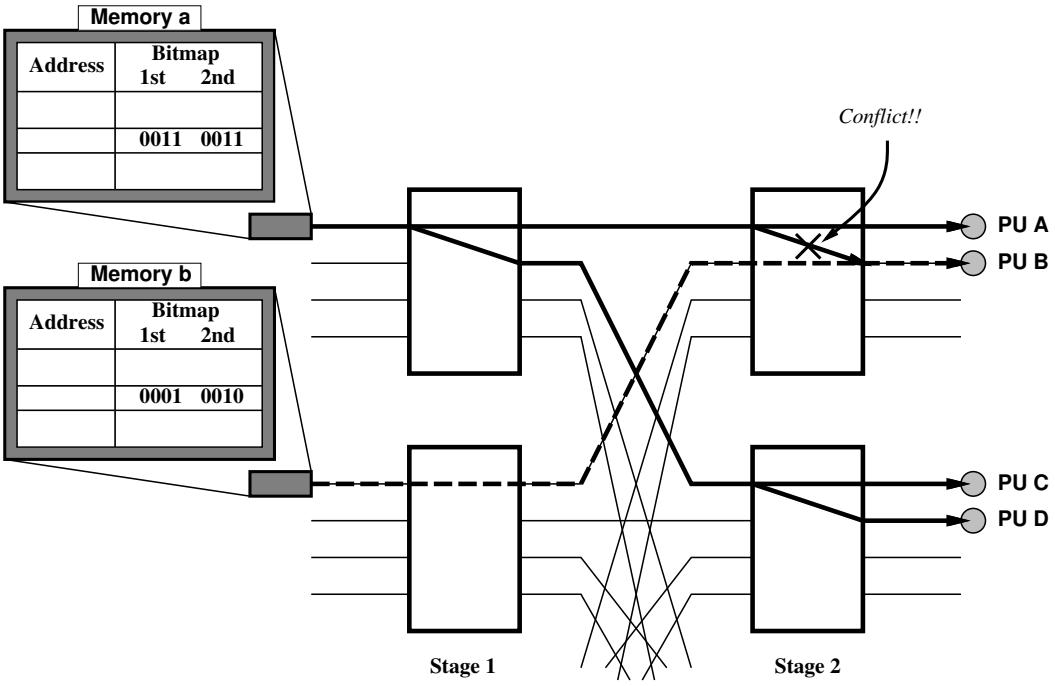

図 4.5: マルチキャストパケットの衝突

MINC では，頻繁に無効化が発生した場合，優先度の低いキャッシュ制御パケットがマルチキャストを妨げられ，いつまでも再送を繰り返す可能性がある．この問題に対して，パケットに優先度を設けて再送パケットの優先度を高くする方法や，通過率そのものを向上させて混雑を緩和する方法を取る必要があるが，単純に適用した場合，次にあげる問題が発生する．

- パケットの優先度を設け，再送のたびに増やしていく場合，キャッシュ制御パケットはマルチキャストするため，データ転送のような 1 対 1 で通信をする場合に比べて優先度の範囲を広くする必要がある．しかし，優先度の範囲を広くすると，スイッチの調停回路が複雑になり，性能の低下が大きい．
- 通過率を上げるために，MIN を多重出力にする方法が提案･調査されている[THY94]．しかし，1 対 1 の通信を前提としているため，マルチキャストパケットを挿入した場合には，高い通過率を得ることができない．また，いずれの方法でも一つの出力に対するパスが複数存在し，枝刈キャッシュを設けることができないため，無駄パケットの抑制を効率良く行なうことができない．

以上から，通過率や優先度ではなく，再送の際に宛先を特定することによって，パケットの数を減少させる方法が有効である．しかし，ディレクトリが縮約されているため，再送する宛先を特定するためにはネットワークの途中で発生したパケットの衝突に関する情報を，共有メモリ側に知らせなくてはならない．

そこで，アクノリッジを信号線ではなくパケットにして，各ステージにおける衝突状況を共有メモリ側に返送する．共有メモリ側では，返送された衝突状況に基づいて再送する

際の宛先情報を加工し，無駄なパケットの再送を減らすことができる．図 4.6 に，アクノリッジパケットによって衝突状況を共有メモリに知らせる例を示す．図 4.5 と同じ状況だが，各スイッチに衝突状況を記憶しておくレジスタを設け，入力毎に転送できたかどうかを記憶しておく (図 4.6 上)．そして，パケットが転送された後に，衝突状況をアクノリッジパケットとして，ステージ単位で共有メモリ側に返送する (図 4.6 下)．同じステージのスイッチの情報は，AND をとって返送する．

この方法では，縮約されたビットマップに従って再送を行なうため，完全に特定できるわけではない．例えば，図 4.6 の場合は，PU D にも再送されてしまう．しかし，返送されてきたアクノリッジパケットと，縮約されたビットマップを比較するだけで再送用のビットマップを再構成することが可能となり，パケットの数を 1/(スイッチサイズ) に減らすことができる．

また，先の枝苅キャッシュへのエントリミスを知らせるビットを追加することも可能である．

4.4 SNAIL−2 の実装

MINC のネットワークは，シミュレーション時間や必要なメモリ量の問題を解決した実機による評価が行われた． 実機 SNAIL-2 上で実アプリケーションを動作させることで，MINC によるキャッシュ制御機構の実現性を明確にしている．

4.4.1 SNAIL−2 の概観

第 4.7 図に SNAIL–2 の構成を示す．SNAIL–2 のプロセッシング (PU) と共有メモリ (MM) は，PBSF トポロジの SSS 型 MIN(以降 PBSF) によるデータ転送用ネットワークと，キャッシュ制御に用いる MINC ネットワークに接続される．

PU には，ローカルメモリ，共有メモリのキャッシュであるローカルキャッシュが置かれており，MM には，共有メモリ，キャッシュライン所有情報のディレクトリが配置される．データ転送用ネットワークは 8 個の PBSF チップで構成され，キャッシュ制御用ネットワークは 1 個の MINC チップで構成される．16 台の PU と MM は，データ転送用ネットワークとキャッシュ制御用ネットワークにそれぞれ接続されている．

SNAIL−2 と外部との通信は， PU 内にある RS-232C, Ether インタフェースを用いて行われ，ホストであるワークステーションに接続される．これらにより，SNAIL−2 への実行プログラムの転送や計算結果の確認を行う．

プロセッサの共有メモリへのアクセスは，PBSF チップで構成されるデータ転送用ネットワークを通して行われ，キャッシュ制御に関するパケットはすべて MINC チップのキャッシュ制御用ネットワークで転送が行われる．

PU は，キャッシュを利用することで共有メモリアクセスのレイテンシを小さくすることができる．共有メモリへの書き込みが起こった場合，PU 内のキャッシュにそのデータが存在しているのなら無効にし，PBSF ネットワークを通して MM へ書き込みパケットを転送する．MM は，ディレクトリを参照し，書き込みが行なわれたデータをキャッシュしている PU へ無効化パケットを MINC ネットワークを通して転送する．PU はこの無効化

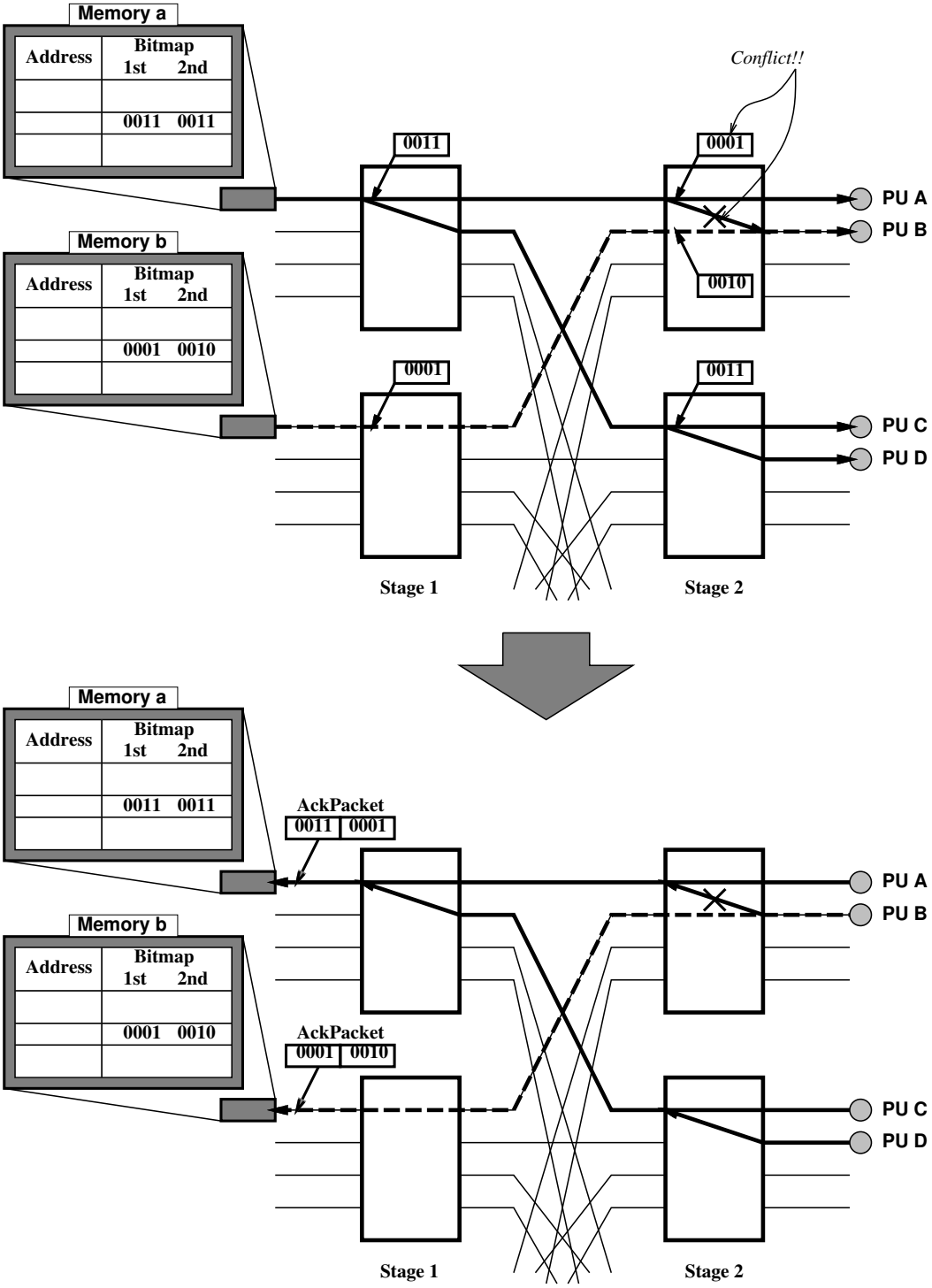
Memory a
Address
Bitmap
1st 2nd
0011 0011
Memory b
Address
Bitmap
1st 2nd
0001 0010
Conflict!!
0011
0001
0010
0001
0011
PU A
PU B
PU C
PU D
Stage 1
Stage 2
Memory a
Address
Bitmap
1st 2nd
0011 0011
AckPacket
0011 0001
Memory b
Address
Bitmap
1st 2nd
0001 0010
AckPacket
0001 0010
PU A
PU B
PU C
PU D
Stage 1
Stage 2

図 4.6: 衝突状況の返送

パケットを受け取り，キャッシュにそのデータが存在していたら無効化する．次に，プロセッサが共有メモリへのアクセスを行った時の詳細な流れをまとめるた．

- Read アクセス

  まず PU 内のローカルキャッシュが参照される．有効なデータがある場合はそのデータがプロセッサに渡される．存在しない場合，キャッシュラインのアクセス要求パケットが，データ転送用ネットワークを通して MM に転送される．MM は要求を受け取ると，ディレクトリにキャッシュラインの所有情報を登録し，アクセス要求を受けたラインを共有メモリから読みだす．そして，データ転送用ネットワークの帰りのネットワークを通し，キャッシュラインを PU に転送する．同時に，キャッシュ制御用ネットワークに枝刈りキャッシュ登録パケットを転送し，枝刈りキャッシュの登録を行う．PU 側に転送されたキャッシュラインは，ローカルキャッシュに保存され，プロセッサにデータが渡される．

- Write アクセス

  プロセッサが Write アクセスを発行すると，そのままデータ転送用ネットワークを通して MM に Write データが送られる．MM は，共有メモリにデータの書き込みを行い，それと同時に，ディレクトリを参照して，書き込まれたデータを含むキャッシュラインを所有しているプロセッサの特定を行う．そして，キャッシュ制御用ネットワークを通して，ラインを所有しているプロセッサに対し，無効化パケットのマルチキャストを行う．無効化パケットを受け取った PU は，実際に自分がそのラインを所有している場合はローカルキャッシュの無効化を行う．

4.4.2 SNAIL−2 のハードウェア構成

第 4.7 図に示したように，SNAIL−2 は，Processing Unit(PU)，Memory Module(MM)，データ転送用ネットワーク (PBSF)，キャッシュ制御用ネットワーク (MINC) で構成される．ここでは，それらのハードウェア構成の詳細を説明する．

Processing Unit(PU)

CPU は，FPU と MMU を備えた 32 bit MIPS RISC プロセッサ (IDT 79R3081E-50MJ) を用いている．動作周波数は最大 50MHz であり，プロセッサ外部とは 25MHz 通信を行う．各 PU は，各 PU はインストラクションやコヒーレンス維持の必要のないデータなどをおく 2MB のローカルメモリ，ワークステーションとの通信を行うための Ethernet と RS232C のインタフェース，ブート ROM を備えている．また，ライトスルーで，2way set associative 方式のローカルキャッシュは，データを記憶するための 1MB の SRAM と，タグメモリである 32KB の Dual port RAM で構成される．ネットワークインタフェースやキャッシュ制御を行う PU コントローラは CPLD(ALTERA EPF10K100GC503-4) で構成され，

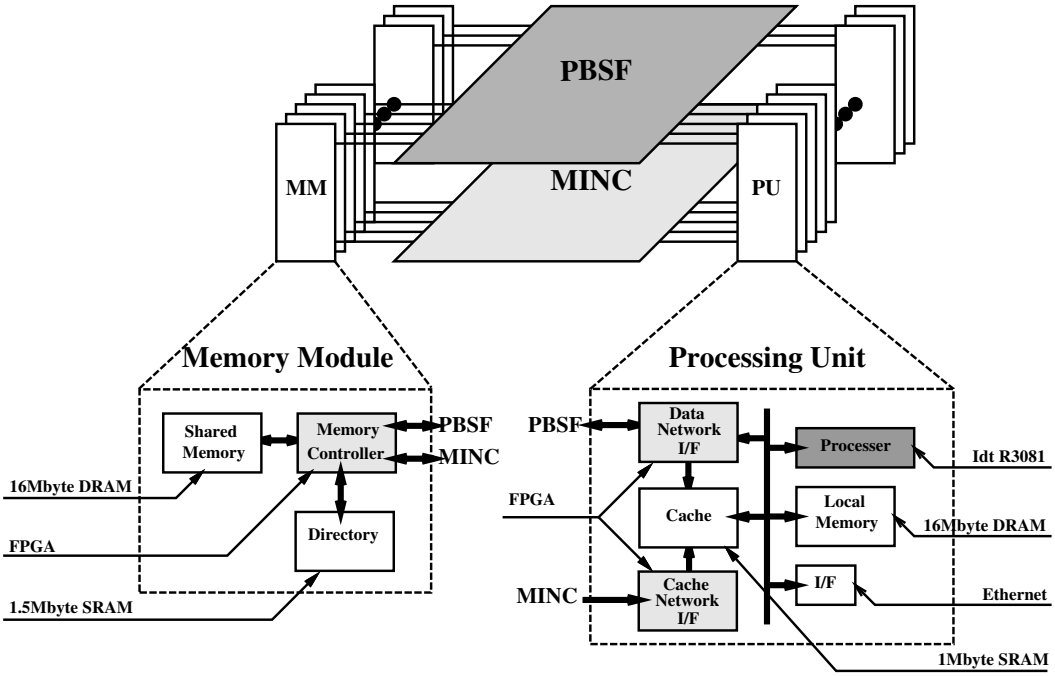

図 4.7: SNAIL−2 の構成

Memory Module(MM)

MM は 6MB の DRAM を持ち，16 個の MM で構成される SNAIL−2 は合計 256MB の共有メモリを持つ．データはキャッシュメモリのラインサイズ毎にインターリーブされて共有メモリに記憶される．ディレクトリは 1.5Mbyte の SRAM で構成され，RHBD 方式により縮約されたビットマップと枝刈りキャッシュへの登録状態を保持する. MM の制御を行う MM コントローラは，CPLD(ALTERA EPF10K50RC240-4) で構成され，ネットワークとのインタフェースやディレクトリの制御を行う．共有メモリへのアクセスに対しては，全共有メモリ領域に対して Test&Set と Fetch&Dec の同期機構をサポートしている．

データ転送用ネットワーク (PBSF)

16PU，16MM に対応する入出力を持った PBSF チップを用いた．PBSF チップは，SSS アーキテクチャの転送方式を用いており，フレームに同期して PU からのアクセス要求や書き込みデータが MM 側へ転送され，ルートが決定されると，次のフレームに，そのルートを逆にたどることで，MM から PU 側にキャッシュラインが転送される．笹原らによって実装された[Sas95]．

PBSF チップは，図 4.8 に示したような，2×2，2×4，4×2 のスイッチを用いて，2 レイヤで実装された．出力部には，ピン数を減らすためにマルチプレクサが用いられている．メッセージコンバイン機能は，最上層のネットワークでのみにが実装されている．また，静的にパケットの優先度が決定されることにより発生するスタベーションの問題を回避するために，入力するパケットに優先度を持たせことができる．再送パケットの優先度を高

くすることで，同じパケットが何度も再送されることを防ぐことができる．16 入出力規模のネットワークでは 4 レベルの優先度があればスタベーションは回避できるので，4 レベルの優先度が設けられている．

PBSF チップの仕様は表 4.1 の通りである．

最大動作周波数	100MHz
ビット幅	2bit(PU → MM) 1bit(MM → PU)
ネットワークサイズ	16 入力, 16 出力
最大バンド幅	300Mbits/sec × 16
セル使用数	17356
信号ピン数	116
テクノロジ	0.5μm CMOS sea-of-gates

表 4.1: PBSF チップの仕様

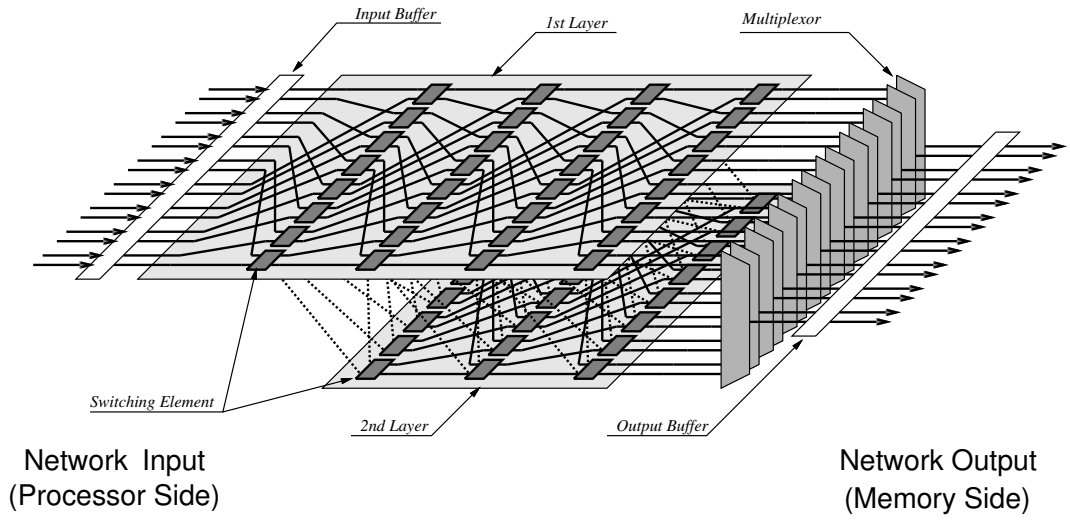

図 4.8: PBSF チップのハードウェア構成

キャッシュ制御用ネットワーク (MINC)

キャッシュ制御には，16 入出力を持った MINC チップを用いる．キャッシュ制御用パケットはディレクトリの情報より共有メモリ側で作られ，MINC チップを通して共有メモリ側から PU 側にマルチキャストされる．MINC チップも PBSF チップと同様に SSS アーキテクチャを用いているため，フレームに同期してパケットの転送を行い，PU 側からはそのパケットに対する Acknowledge パケットが戻ってくる．MM 側では，この Acknowledge パケットの衝突情報から，マルチキャストしたすべての PU にキャッシュ制御パケットが転送されたかを確認し，転送されていない場合は必要なプロセッサのみに再送を行う．

MINC チップのネットワークは，4×4 のスイッチングエレメントを利用したオメガ網で構成される．また，シミュレーションによる評価[HYNA96]より，枝刈りキャッシュは連想度を 2，エントリ数を 256 とし，また，4 段階の優先度制御を行うことと決定された．[Kam97]．

MINC チップは，大規模集積システム設計教育研究センターの試験研究 (B)「VLSI 設計教育カリキュラムの開発」の一環として，Chip Express 社の 0.6μ LPGA (Laser Programmable Gate Array) へ実装された[TTTH98]．LPGA は，レーザーで配線を切断することで回路を構成するゲートアレイであり，論理合成後までをユーザーが行い，配置配線以降は Chip Express 社が行う．論理合成後のネットリストを渡してからチップが実装されるまでが 1,2 週間という短期間であり，小数での発注が可能である，また，FPGA よりも高速に動作するという長所がある．しかし，FPGA と比較して遥かにコストが高く，回路構成の変更も行えないため，開発途中のチップの試作や製品の大量生産には向かないデバイスである．

実装に用いた LPGA の仕様を次に示す．

- パッケージ
  391 ピン PGA(信号 264 ピン)
- 電気的特性
  電源電圧 5V，入出力とも TTL インタフェース
- ゲート数
  最大 100K ゲート (50K ゲート推奨 + 64Kbit メモリセル)

LPGA デバイスの制約を考慮し，実装された MINC チップの仕様は表 4.2 の通りである．

最大動作周波数	50MHz
ビット幅	4bit(packet transfer), 4bit(acknowledge)
ネットワークサイズ	16 入力, 16 出力
Max bandwidth	400Mbits/sec × 16
基本セル使用数	26477
メモリセル使用数	60Kbit
信号ピン数	262
テクノロジ	0.6μm LPGA
最小フレーム長	30 clock

表 4.2: MINC チップの仕様

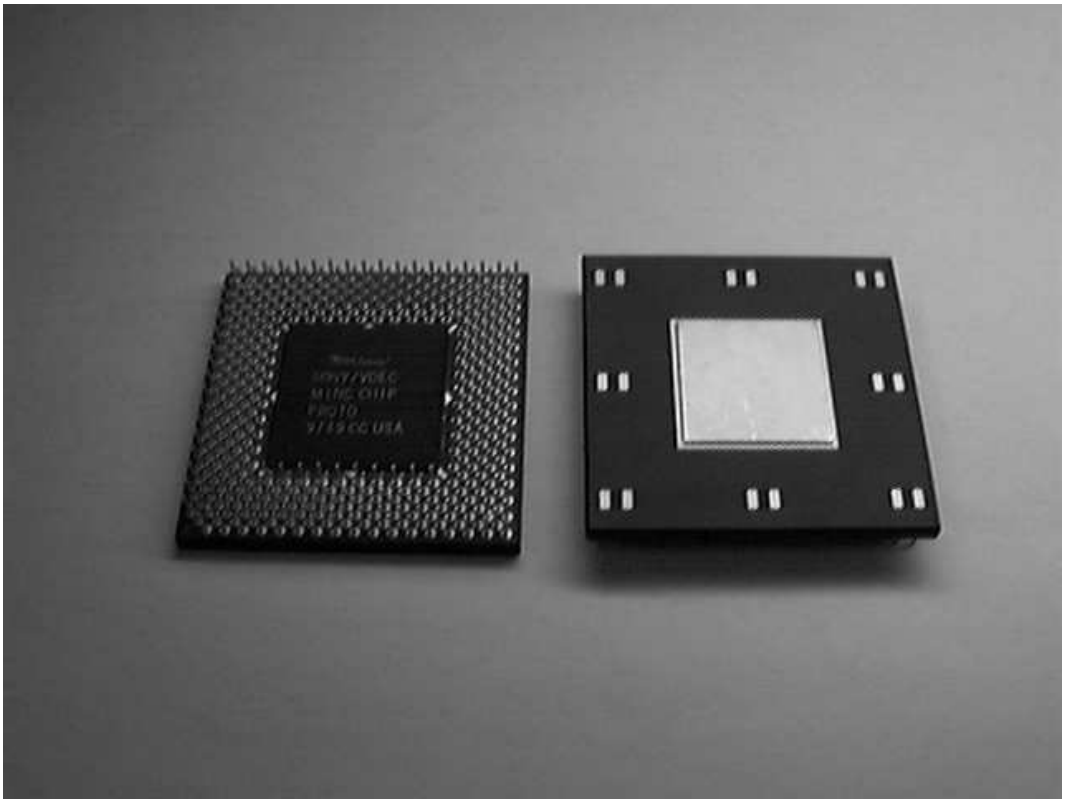

図 4.9: MINC チップ

4.4.3 SNAIL−2 の基板構成

SNAIL−2 を実装するためにプリント基板の作成を行った．基板は，フィルムに焼いて基板にする直前のガーバーデータまで作成し，その後の工程を業者に依託した．ガーバーデータの作成は，Mentor Graphics 社の CAD である，Board Station を用いて行った．基板は実装規模と標準的な規格を考慮して，以下のようにした．

- サイズ: 366.7mm × 400mm，版圧 1.6mm
- 6 層基板 (信号: 4 層, 電源: 1 層, グランド: 1 層)
- 両面実装

また，基板は次の 2 種類を作成した．

SNAIL−2 プロセッサ・メモリボード

プロセッサ・メモリボードは，2 個の PU と 2 個の MM を実装した．一部で共通のクロック信号を使っている以外は，それぞれ完全に独立して動作する．PU，MM ともコネクタが用意されており，ケーブルを用いてネットワークボードと接続される．コネクタの出

力部には信号線をケーブルを通すことによる劣化を防ぐためにバスバッファの IC(74244) を実装し，入力部には，信号の反射を防ぐためのテブナン終端を行った．また，デバッグ用に主要なバスをピンヘッダに接続し，ロジックアナライザやオシロスコープなどで信号を容易に監視できるようにした．

プロセッサ・メモリボードを写真を図 4.10 に示した．基板の左半分が 2 つの PU であり，右半分が 2 個の MM である．2 個の PU は独立しており，キャッシュメモリ，タグメモリ，PU コントローラ，プロセッサ，ROM，ローカルメモリをそれぞれ備えている．I/O 部は，基板の上側にまとめて実装した．PU の外部とのインタフェースは，RS-232C, Ethernet を用いるが，Ethernet は，実装面積やコストの問題から一方の PU のみに実装した．表示用の LED は 1PU につき 2 個ずつ実装した．2 つの MM も独立しており，それぞれディレクトリ用 SRAM,MM コントローラ，共有メモリ用 SIMM を備えている．

SNAIL−2 ネットワークボード

ネットワークボードは，PBSF ネットワークを構成する 8 個の PBSF チップ，MINC ネットワークを構成する 1 個の MINC チップ，フレームクロック作成やリセットの制御をする FPGA を実装した．基板左側には，PU との接続のためのコネクタ，右側には，MM との接続のためのコネクタが設けられている．これらは，ケーブルを用いてプロセッサ・メモリボードと接続される．

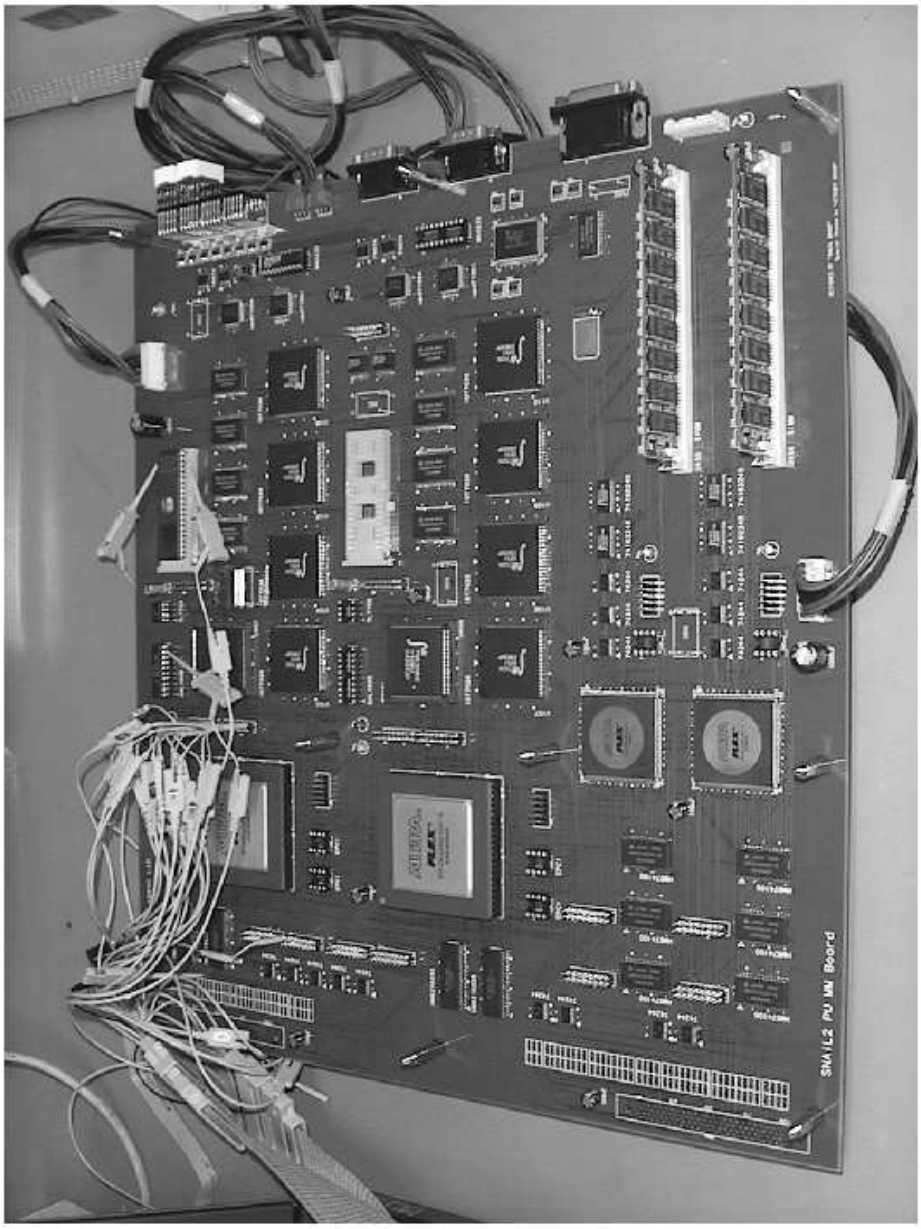

図 4.10: プロセッサボードの外観

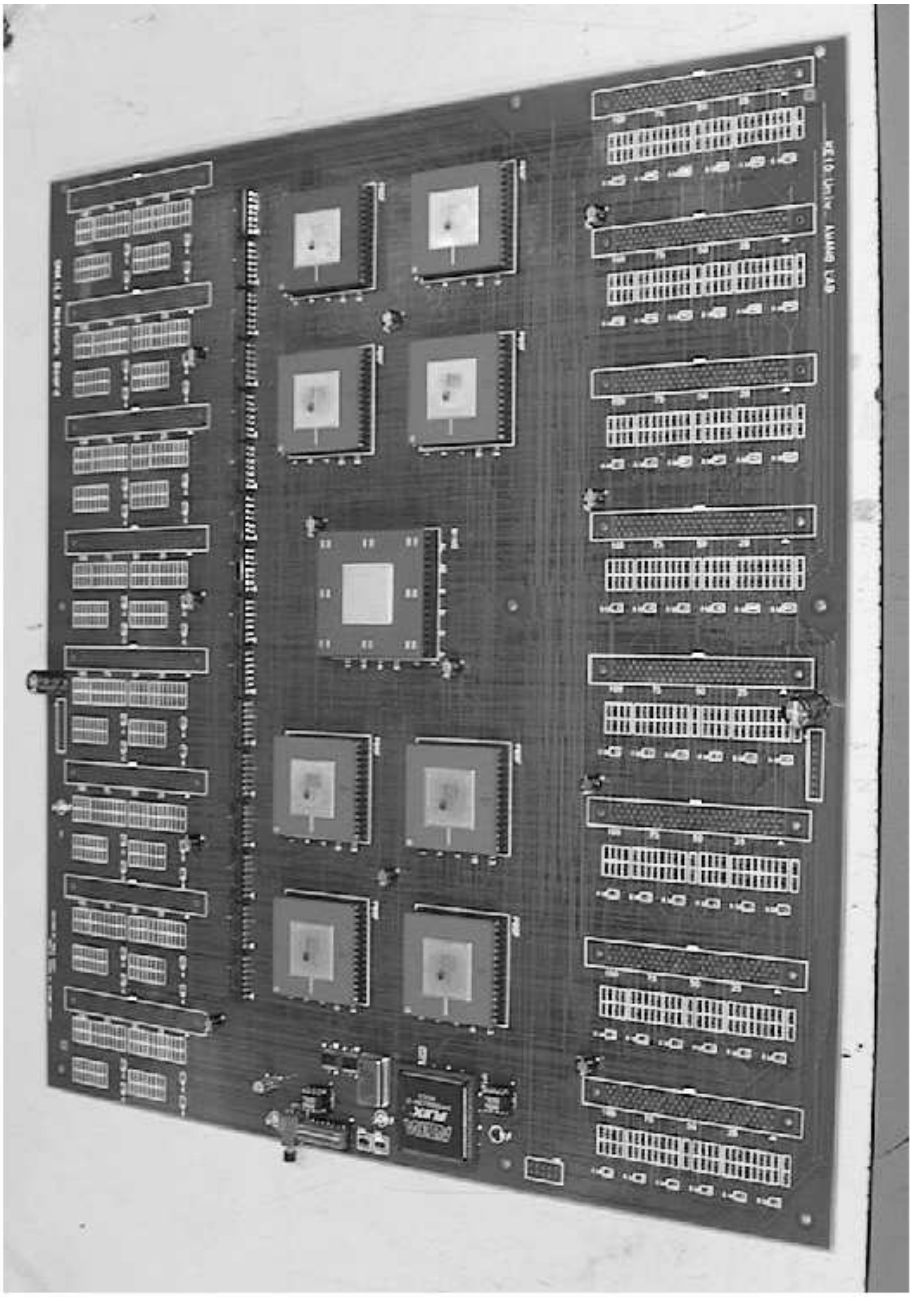

図 4.11: ネットワークボードの外観

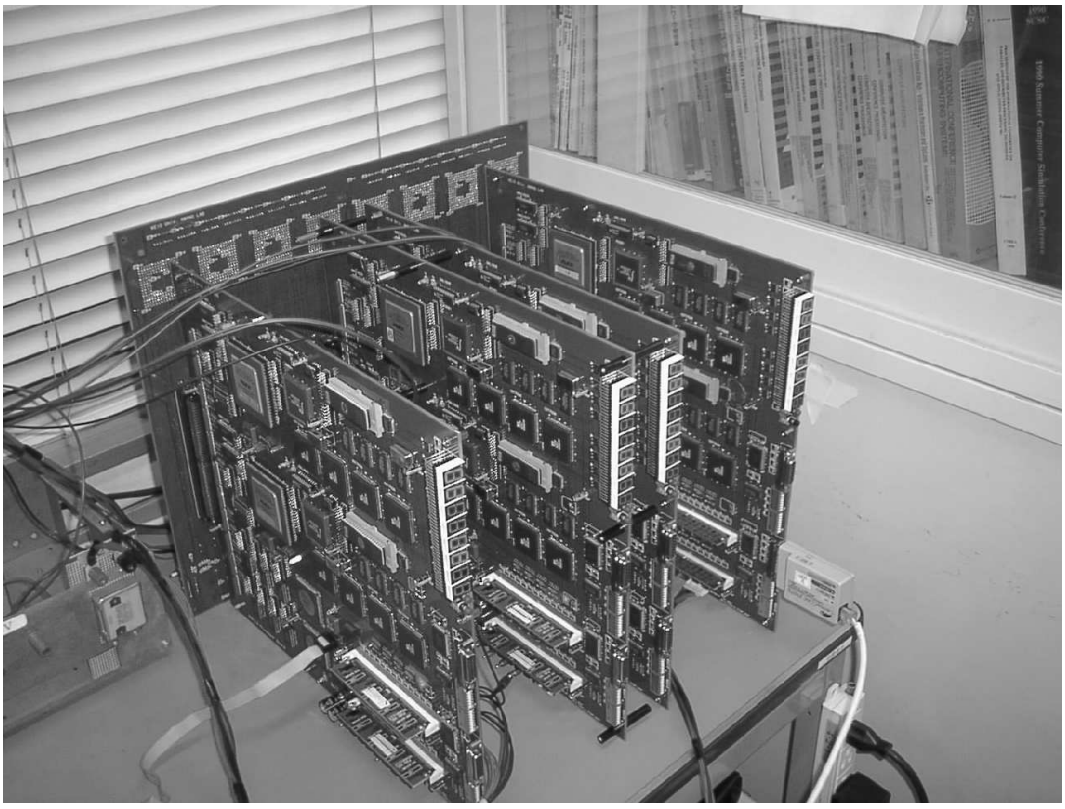

図 4.12: SNAIL−2

4.5 SNAIL−2 の評価

4.5.1 評価環境

SNAIL−2 は，16PU，16MM で構成し，プロセッサの内部動作周波数が最大 50MHz, 外部動周波数が最大 25MHz で設計されている．

しかしながら，プロセッサボードとネットワークボード間を接続するフラットケーブルの電気的特性の問題から予定の台数，動作周波数では安定動作が困難であったため，実際には 8PU，16MM で，表 4.3 に示すようにシステム全体の動作周波数を低くし，安定動作させて評価を行った．

また，メモリアクセス時間は表 4.4 に示すように，フレーム長は 40clock〔4μsec〕とした．実行命令は，コヒーレンスを維持する必要がないため本来はプロセッサ内のインストラクションキャッシュを用いることができるが，今回は実装上の問題で使用しないようにしている．キャッシュコントローラにはライトバッファが実装されているので，１回のライトアクセスでプロセッサが長く待たされることはないが，ライトアクセスが連続した場合はキャッシュ制御がライトスルー方式なので表 4.4 の時間待たされることとなる．ネットワーク上で衝突が発生した場合はメモリアクセス時間はさらに継ぎのフレームまで引き延ばされることとなる．

並列アプリケーション集 SPLASH–2〔Stanford ParalleL Applications for SHared memory–

2〕を用いて評価を行った．今回は，以下の 3 つのアプリケーションを実装し，プロセッサ数 1，2，4 のそれぞれで評価を行った．

なお，アプリケーションのバリア同期には，SNAIL–2 に実装されている Fetch & Dec 機構を利用した．また，ローカル変数はローカルメモリへ配置され，インストラクション及び共有データは共有メモリへ配置される．

- LU: 密行列を下三角行列と上三角行列に分解する．

  LU では行列をブロック毎にわけて各プロセッサが並列実行していく．ブロック間でデータを共有するためにプロセッサ間通信を行うが，その数は比較的少ない．そのため，同期の回数もあまり増えず，比較的並列制を抽出し易い．行列のサイズは，256×256 とした．

- FFT: $\sqrt{n}$ を基数とする 6 ステップのアルゴリズムを用いた 1 次元複素数の高速フーリエ変換で，プロセッサ間の通信が最小限になるように最適化されている．

  プロセッサ間で共有するデータの数が少なく，プロセッサ間通信も少ない．そのため，同期の回数も少なくなる．データサイズを大きくしても，プロセッサ間通信の数は変わらないため，どのデータサイズでも同じように並列性を抽出できる．データサイズは 2^{18} とした．

- Radix: 基数ソートである．キーの桁ごとに処理する整列アルゴリズム．各ステップ毎にプロセッサ間でデータをやり取りする必要があるので，Ocean ほどではないが，LU や FFT と比較すると通信量は多くなり，同期の回数も増加する．要素数は 524288 個.

- Ocean: 渦や海流の境界をもとに大規模な海洋の動きをシミュレーションする．プロセッサ間で共有するデータの数も大きく，演算中のプロセッサ間の共有データのやりとりも多いので，プロセッサ間通信はかなり大きなものとなる．そのため，同期の回数も増加し，並列性を十分に引き出すことが難しい．海洋のサイズは，130 x 130 で評価した．

Processor	〔Internal〕	20MHz
	〔External〕	10MHz
PU Controller	〔Controller〕	10MHz
	〔Network Interface〕	10MHz
MM Controller	〔Controller〕	10MHz
	〔Network Interface〕	10MHz
PBSF Network		10MHz
MINC Network		10MHz

表 4.3: 各回路の動作周波数

read/write	アクセスの種類	メモリアクセス時間〔clock〕
リード	キャッシュヒット	9
	キャッシュミス	120〜159+〔80×n〕
	キャッシュ無し	88〜127+〔80×n〕
	ローカルメモリ	6
ライト	キャッシュヒット	83〜122+〔80×n〕
	キャッシュミス	83〜122+〔80×n〕
	キャッシュ無し	55〜94+〔80×n〕
	ローカルメモリ	6

n: 再送回数

表 4.4: メモリアクセス時間

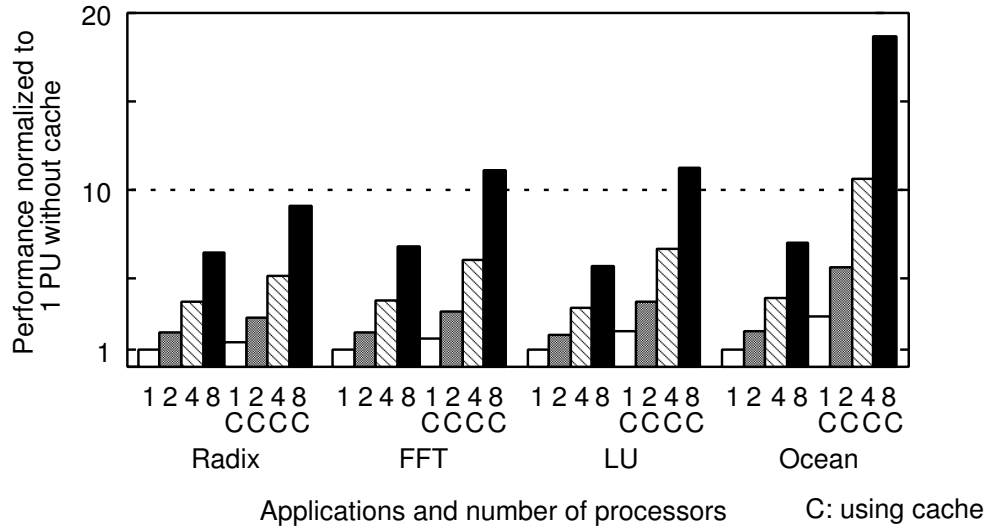

図 4.13: スピードアップ率

4.5.2 評価結果

図 4.13 は，キャッシュを使用しないときの 1PU を基準とした性能評価を示している．LU 以外のアプリケーションでは台数に比例して性能が向上している．LU ではアプリケーションの最終段階で不規則なロードが発生するため性能の向上がやや低くなっている．

図 4.14 はキャッシュを使用したときの 1PU を基準とした性能評価を示している．すべてのアプリケーションでキャッシュを使用することによる性能が向上しており，40-190% の性能向上を達成している．また，図 4.15 はキャッシュのヒット率を示している．共有しないデータはローカルメモリ上に配置しているため，キャッシュのヒット率は高くないが，MM に配置された共有データに対しては十分にアクセスレイテンシを小さくすることができる．

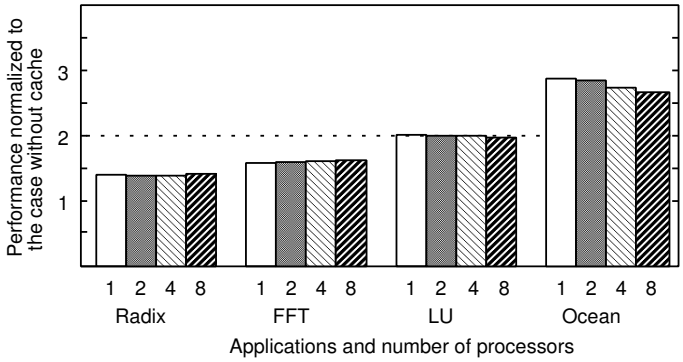

図 4.14: キャッシュの効果

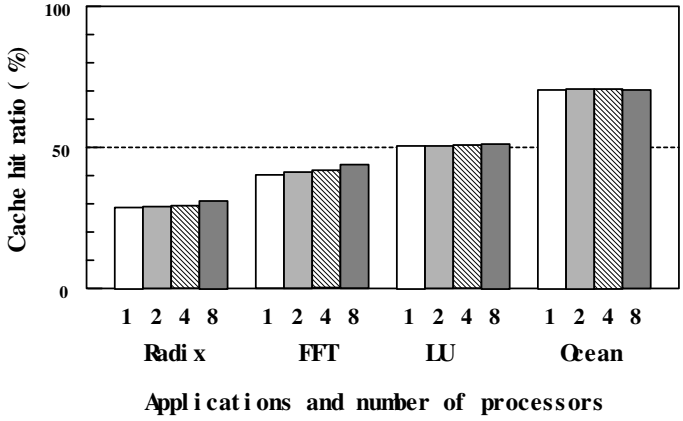

図 4.15: キャッシュヒット率

図 4.5 は，枝苅キャッシュの評価を示す，枝苅りキャッシュの使用時／未使用時の無効化パケットの衝突率を示している．衝突率の高い FFT と，データの共有率が高い Ocean では，8PU のときに枝苅りキャッシュにより，不必要な無効化パケットが削除されて衝突率が低下している．しかし，枝苅りキャッシュへの登録パケットの影響から，RADIX，LU では逆に衝突率が上がってしまうという問題があることがわかった (表 4.5). また，図 14 は枝苅りキャッシュを使用したときの性能向上を示しているが，どのアプリケーションにおいても登録パケットの処理が影響し，性能が低下する結果となってしまった．登録パケットは MM から MINC chip へ転送されるが，MINC chip 内において衝突が発生すると，その都度，再生が繰り返される．その間に，MINC chip に投入されるパケットは入力バッファに貯められることになるが，入力バッファが満たされてしまうと，MM は PBSF を介して転送された読み出し／書き込みパケットを受け取ることができなくなり，大幅に性能の低下につながってしまう．

PU	枝刈りキャッシュ未使用時			枝刈りキャッシュ使用時		
	無効化パケット	登録パケット	合計	無効化パケット	登録パケット	合計
2	5591737	0	5591737	5592360	5597389	11189749
4	5592215	0	5592215	5591867	5597499	11189366
8	5591786	0	5591786	5592026	5593624	11185650

表 4.5: Memory Module から MINC chip へ転送されるパケット数 (LU)

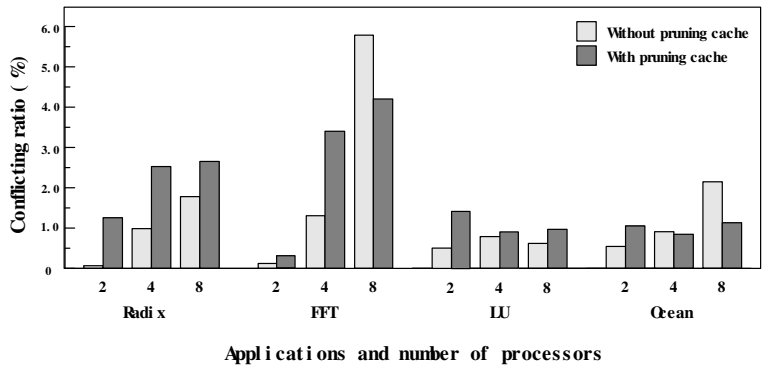

図 4.16: 無効化パケットの衝突率

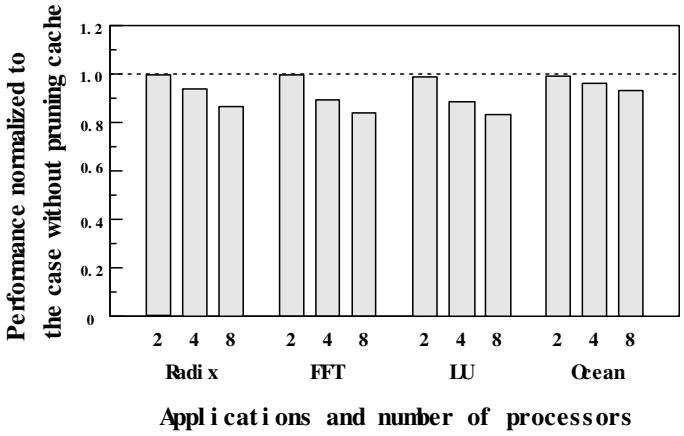

図 4.17: 枝刈りキャッシュの効果

表 4.6: 無効化パケット平均再送回数

PU 数	8	16	32	64
枝刈りバッファ無	0.13	0.37	13.09	36.98
枝刈りバッファ有	0.23	0.30	15.68	37.52

4.6 MINC の問題点

MINC はプロトタイプ SNAIL-2 上で実機動作し，その結果以下のことが明らかになった[TDM+02]．MINC を用いて一貫性制御を可能としたキャッシュを付加した効果は大きいが，MINC の Tree を用いた無効化のマルチキャストは，マルチキャスト自体が衝突すると大きな遅延を発生し，ネットワークを遅延させる．

また，SNAIL–2 はクロックレベルシミュレータで 64PU までの評価が行われ，枝刈りバッファは，登録するためのパケットの損失が，枝刈りの効果を相殺してしまうことがわかった．表 4.6 に，LU 分解のクロックレベルシミュレーションにおける PU 数と再送回数を示す．PU 数が増えると衝突に基づく再送回数が急激に増加してしまい，枝刈りバッファの効果が十分でないことがわかる．

原因の一つは，MINC で用いた縮約階層ビットマップ方式が，共有関係が長い時間保持される CC-NUMA のページ管理用に提案されて方式であることに由来する．中規模の UMA 型のキャッシュは，頻繁にデータ交換が行なわれる場合には，頻繁に無効化と転送が行なわれる．SNAIL-2 の場合，共有を行なわないデータはローカルメモリに格納するため，対象となるキャッシュラインは全て共有データであり，頻繁な無効化パケットのマルチキャストの要因となる．

第5章　MINDIC

5.1　MINDIC の概観

本論文では，MIN 接続でキャッシュの一致制御を取る新しい方法として，ディレクトリを共有メモリ (MM) 側に設けず，MIN を構成する各スイッチ内に小容量のテンポラリディレクトリ (TD) を設け，その上にディレクトリ構造を動的に構成する MINDIC(MIN with DIrectory Cache switch) を提案する．

図 5.1 に，MINDIC を用いたシステムの構成の一例を示す．各 PU と MM は，メモリアクセス要求およびキャッシュ制御パケットの転送を行う MINDIC と，キャッシュラインのブロック転送を行なうデータ転送用結合網の，分離された 2 つの結合網を介して接続される．書き込みは Write Through 方式の Direct Write を用い，MINDIC 内では読み出し要求，書き込み要求，無効化要求の短いパケットのみを転送する．

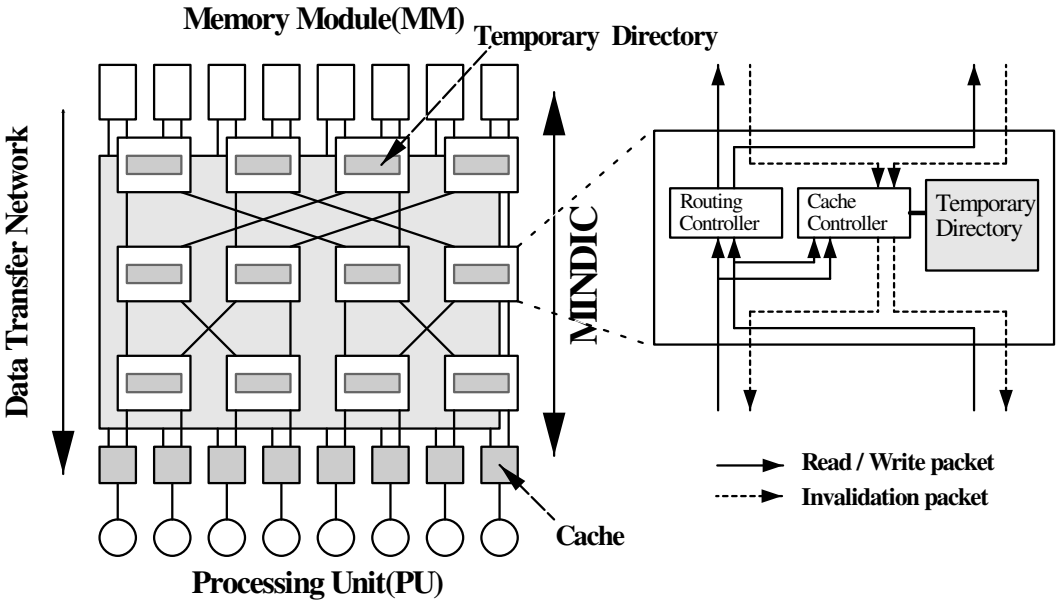

図 5.1: MINDIC の構造

MINDIC の最大の特徴は，共有メモリを構成する MM にディレクトリを設けず，スイッチ内に設けられた数 K エントリの TD でキャッシュラインの共有情報を保持する点である．TD に登録できる共有情報の数は制限されるため，共有情報は短期的にのみ保持される．TD は小容量のメモリで実現でき，スイッチのチップ内部に実装できるため，高速アクセスが可能である．ディレクトリを用いた従来のキャッシュ制御方式と MINDIC のそれ

表 5.1: キャッシュ及びディレクトリの配置

	PU	Network		MM
	cache	cache	directory	directory
Full-map	○	-	-	○
MHN，CAESAR	○	○	-	○
MIND，MBN	○	-	○	-
DRESAR	○	-	(DC)	○
Sparse-Directory	○	-	-	(TD)
MINDIC	○	-	(TD)	-

ぞれについて，キャッシュやディレクトリが，PU，Network，MM のどの部分に配置されるかを表 5.1 に示し，構成の違いをまとめた．

MINDIC は，MIND と同様にネットワーク内に設けたディレクトリでキャッシュ一致制御を行うが，ディレクトリが TD として構成されるので，小容量のメモリしか必要とせず，ネットワークのチップ内部に実装することができる．また，TD をネットワーク越しにアクセスされる各ノード上に配置する Sparse-Directory と異なり，高速にディレクトリをアクセスすることが可能となっている．従来のキャッシュ制御方式では，MIND に Sparse-Directory の方式を適用し，ネットワーク内に設けた TD のみでキャッシュ一致制御を行うことについては，検討が行われていない．

なお，MINDIC の Temporary Directory(TD) は，DRESAR の Directory Cache(DC) と似た構成であったため，MINDIC が提案された初期の段階では，Directory Cache と呼んでいた．しかし，TD は DC のようにディレクトリのコピーを記憶するものではないため，明確に区別するために別の表現を用いることとなった．その際，MINDIC の名称については変更を行わなかった．

5.2 基本動作

基本的に，MINDIC のスイッチでは，読み出し要求，書き込み要求，無効化要求の三種類のパケットを取り扱う．また，MM はスイッチから受け取った要求に応じてデータの読み出しや書き込み制御を行うのみで，ディレクトリの管理などキャッシュ制御に関する動作は一切行わない．PU 側のキャッシュは，無効化要求パケットを受け取った際に該当するキャッシュラインを保持していれば無効化を行う以外，特別な操作は必要としない．

ここからは，読み出し要求時と，書き込み要求時の動作をそれぞれ詳細に説明する．

5.2.1 読み出し要求時の動作

図 5.2(a)〜(d) は，PU から MM へ 2 回の読み出し要求が発生したときの動作を順に示している．次にそれぞれの動作について説明する．なお，図中の Read_A はキャッシュライン A のデータに対する読み出し要求を表し，TD 内の数字 (例えば，1010_A) は，キャッシュラ

イン A に対する共有情報であるビットマップを表している．

(a) PU から読み出し要求が発生すると，先ず PU に接続されたキャッシュが参照され，データが存在するときは，キャッシュからデータが PU へ送信される．キャッシュにデータが存在しない場合は，読み出し要求パケットが MINDIC を介して MM へ転送される．

(b) 読み出し要求パケットを転送する際，パケットが通過する各スイッチ内の TD には，読み出すキャッシュラインの共有情報が登録される．MM にパケットが到達すると，データ転送用結合網を介して MM から PU 側へキャッシュラインが転送され，PU にキャッシュされるとともに，必要なデータが PU へ渡される．

(c) 次に，別のプロセッサから同じキャッシュラインに対して読み出し要求が発生したときの動作を示す．PU に接続されたキャッシュにデータが存在しないときは (a) と同じように，読み出し要求パケットが MM へ転送される．

(d) パケットが通過するスイッチ内の TD では，読み出すキャッシュラインの共有情報を登録されるが，既に同じキャッシュラインの共有情報が登録されている場合は，そのビットマップを更新することで，共有情報を登録する．

5.2.2 書き込み要求時の動作

図 5.2(a)〜(d) の読み出し要求の動作が行われた後に，PU から MM へ同じキャッシュラインへのの書き込み要求が発生したときの動作を図 5.3(a)〜(d) に示す．図中の Write_A はキャッシュライン A のデータに対する書き込み要求を表している．

(a) PU で MM への書き込み要求が発生すると，書き込み要求パケットが MINDIC を介して共有メモリへ転送される．この際，パケットが通過する各スイッチにおいて TD が参照される．TD のエントリに，書き込み要求に対応するキャッシュラインの共有情報が登録されているときは，登録されている共有情報に従ってキャッシュラインの無効化要求パケットが生成される．

(b) 生成された無効化要求パケットは，対応する下位のスイッチへマルチキャストされる．

(c) 下位のスイッチでも，上位のスイッチから無効化要求パケットを受け取ると TD が参照される．そして，登録されている共有情報に従って，対応する出力に無効化要求パケットがキャストされる．

(d) 無効化要求パケットを受け取った PU 側のキャッシュでは，該当するキャッシュラインを無効化することで，キャッシュの一貫性を保持する．

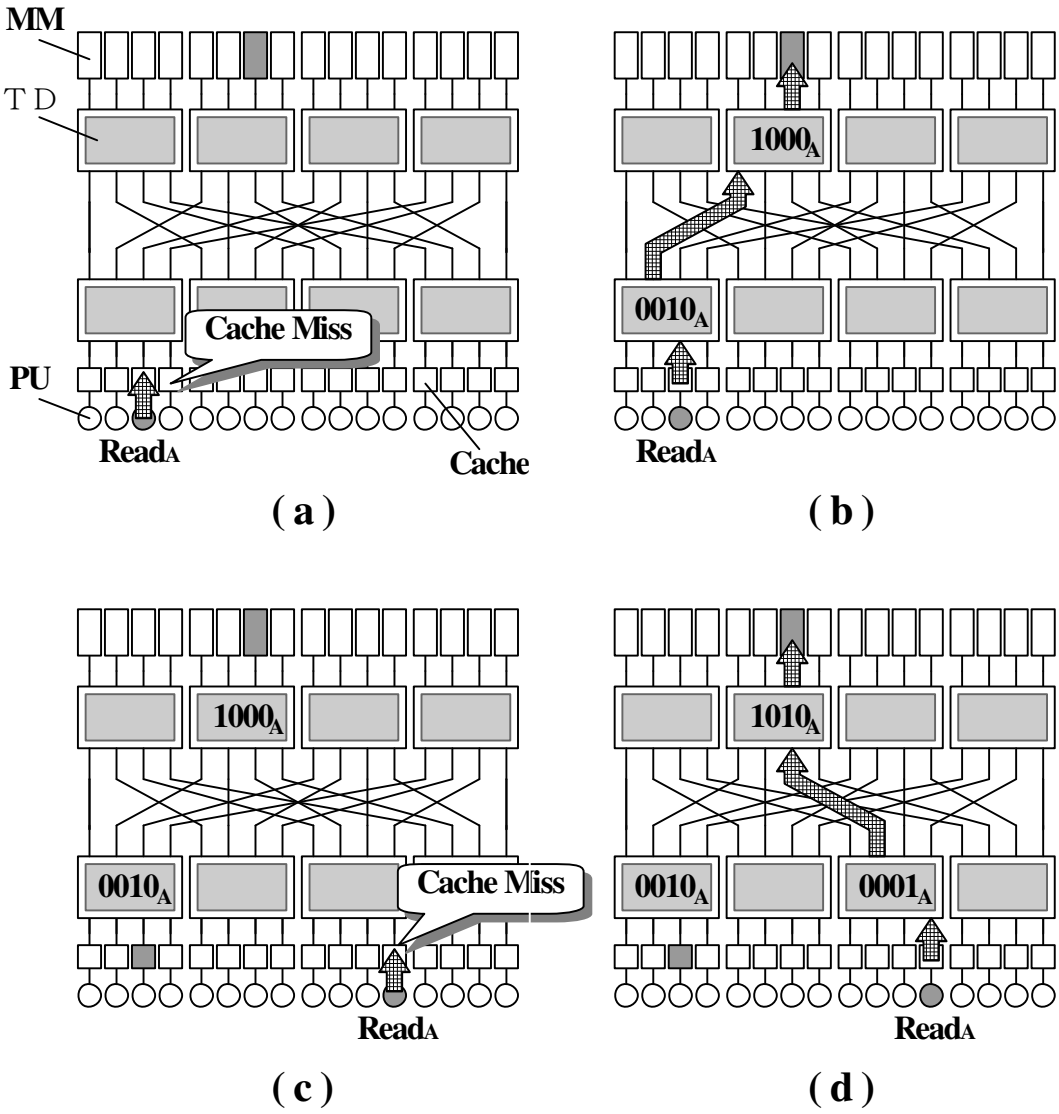

図 5.2: MINDIC の基本動作 (読み出し要求)

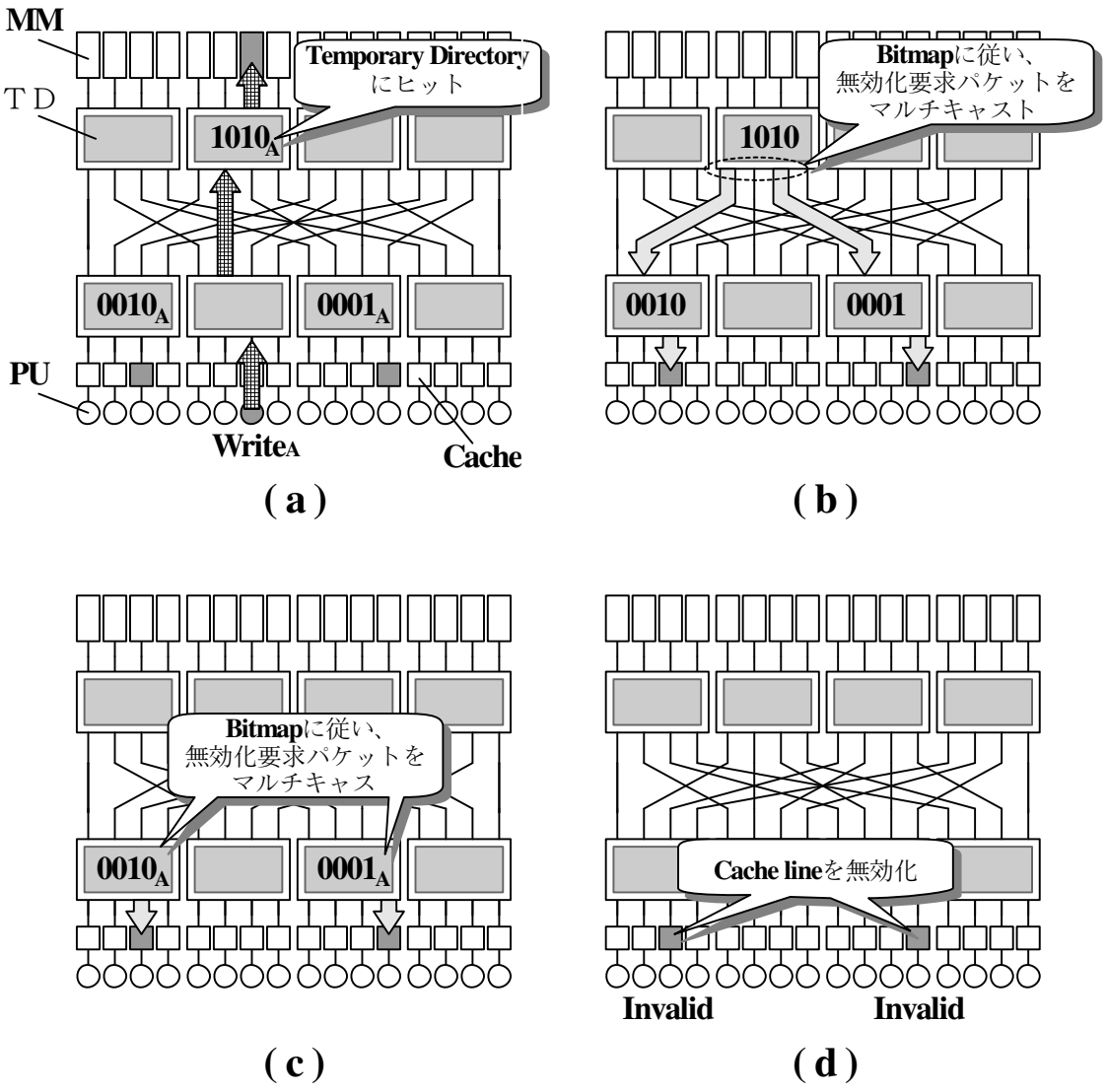

図 5.3: MINDIC の基本動作 (書き込み要求)

5.3 テンポラリディレクトリの構成

図 5.4 に，MINDIC の各スイッチに設けられているテンポラリディレクトリ (TD) の構成を示した．TD は，少容量のメモリで構成されており，入力リンクに対応する PU のキャッシュラインの共有情報をビットマップの形で保持している．そして，一般のキャッシュと同様に各要求パケットに転送アドレスの一部をインデックスとして参照される．図 5.4 の TD は 2way であるが，共有情報は各インデックスに対し，way 数分しか保持することはできない．

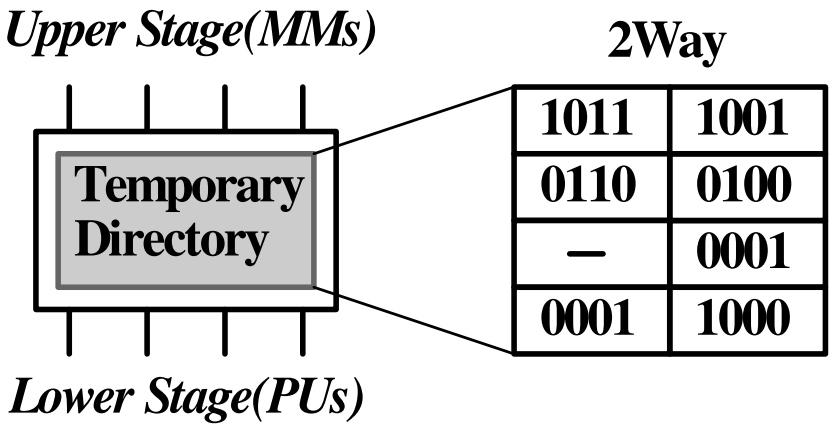

図 5.4: テンポラリディレクトリの構成

スイッチが各要求パケットを受信した時は以下のようにの動作する．

- 読み出し要求
  読み出し要求パケットが PU 側のステージから入力されたら，読み出しアドレスのインデックス部を用いて TD を参照する．TD 内の共有情報の参照でミスした場合，TD に共有情報を登録可能であれば，パケットが入力されたリンクに対応するビットを 1，他を全部 0 としてこのビット列を登録する．一方，TD にヒットした場合，パケットが入力されたリンクに対応するビットを 1 として既存のビットマップを OR-write して共有情報を更新する．

- 書き込み要求
  書き込み要求パケットが PU 側のステージから入力されたら，読み出し要求時と同様に，MM への書き込みアドレスのインデックス部を用いて TD を参照する．TD にヒットした場合，TD から共有情報を表すビットマップを読み出し，対応するビットが 1 の入力リンク全てに無効化パケットを逆送する．そして，TD のエントリを破棄する．これにより，書き込みが発生したキャッシュラインを保持する PU 側のキャッシュのエントリが無効化される．なお，PU から MM へ向かう書き込みデータは，書き込み要求パケットに続いて MINDIC 上で転送される．

- 無効化要求
  無効化要求パケットが MM 側のステージからスイッチに入力された場合，無効化パケットが有するキャッシュラインのアドレスの一部をインデックスとして TD を参照する．TD にヒットした場合，TD から得られた共有情報に従って，キャッシュラインを保持する PU へ向けて無効化パケットをマルチキャストする．そして，このスイッチ内の TD のエントリを破棄する．

5.4 転送プロトコル

MINDIC は前記した基本動作を行うが，TD は小容量のメモリであるため，新たにキャッシュラインの共有情報を登録する際に，先に他のラインに対する古い共有情報が登録されており，共有情報を登録できないことがある．具体的には，次の場合に登録できない．

- TD に，インデックス A に対応するエントリの空きがない状態で
- 共有情報が登録されていない，同じインデックス A のキャッシュラインに対する，後発の読み出し要求があった場合

この場合，先発か後発の読み出し要求に対する共有情報のうち，いずれかの共有情報を破棄しなければならず，単に基本動作を行うのみでは正常にキャッシュの一致制御を行うことができない．

例えば，キャッシュライン A に対する読み出し要求が発生したときに，スイッチ上の TD が一杯のため登録できず，そのライン A に対する書き込み要求が発生したときに問題となる．キャッシュライン A に対する読み出し要求の共有情報がどこにも保持されていないため，書き込み要求による無効化要求が発生せず，適切な無効化が行なわれない．

そこで，共有情報が破棄された場合にもキャッシュの一致制御を正確に行えるようにするために，次の 3 つの転送プロトコルを提案する．

- Dangerous bit プロトコル

  後発の読み出し要求の共有情報を登録せず，登録しなかった旨の情報をスイッチ内で管理する方式

- Invalidation broadcast プロトコル

  後発の読み出し要求の共有情報を登録せず，登録しなかった旨の情報をメモリモジュールで管理する方式

- Eviction プロトコル

  TD に登録されている情報を追い出して，後発の読み出し要求の共有情報を登録する方式

次に，それぞれのプロトコルについて詳細な動作を説明する．

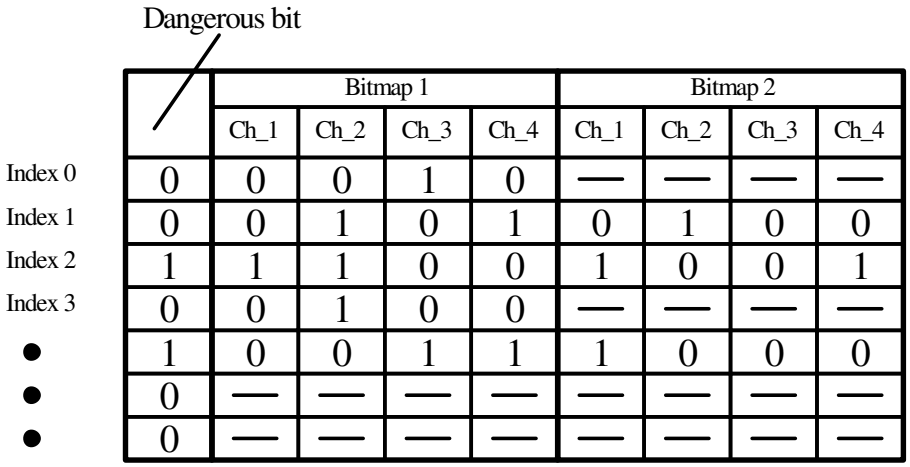

図 5.5: Dangerous bit プロトコルの TD の構成

5.4.1 Dangerous bit プロトコル

先に TD に登録されている共有情報を保持し続け，新たな読み出し要求に対応する共有情報を登録しない方式．図 5.6(a) に示すように，スイッチ内の TD のエントリ毎に Dangerous bit を設け，新たな読み出し要求に対する共有情報が未登録であるをことを示す未登録情報が，Dangerous bit に記録される．2way の TD は図 5.5 のように，エントリ毎のビットマップと Dangerous bit の組で構成される．

図 5.6(b) のように，読み出し要求パケットがスイッチに入力されると，TD の対応するエントリに共有情報を登録しなければならないが，エントリに空きがない場合，TD の対応するインデックスに Dangerous bit がセットされる．この時，共有情報はどこにも登録されずに，MM へ読み出し要求パケットが転送されて，データ転送用ネットワークを介してキャッシュラインの転送が行われる．

そして，図 5.6(c)〜(d) のように，書き込み要求パケットがスイッチに入力された際には，TD の Dangerous bit が参照され，TD の対応するインデックスに Dangerous bit がセットされていると，無効化要求パケットを生成し，入力リンク以外の下位ステージの全リンクへパケットをブロードキャストする．下位ステージのスイッチでは，上位ステージのスイッチから無効化要求パケットを受け取ると，TD の共有情報に従って無効化要求パケットの転送が行われ，無効化パケットが PU 側へ到達してキャッシュラインの無効化が行われる．

Dangerous bit がセットされたインデックスでは，先に登録されたエントリ以外は利用不能になってしまうが，このようなインデックスの共有情報と Dangerous bit は，バリア同期が発生した時に一斉にクリアされる．

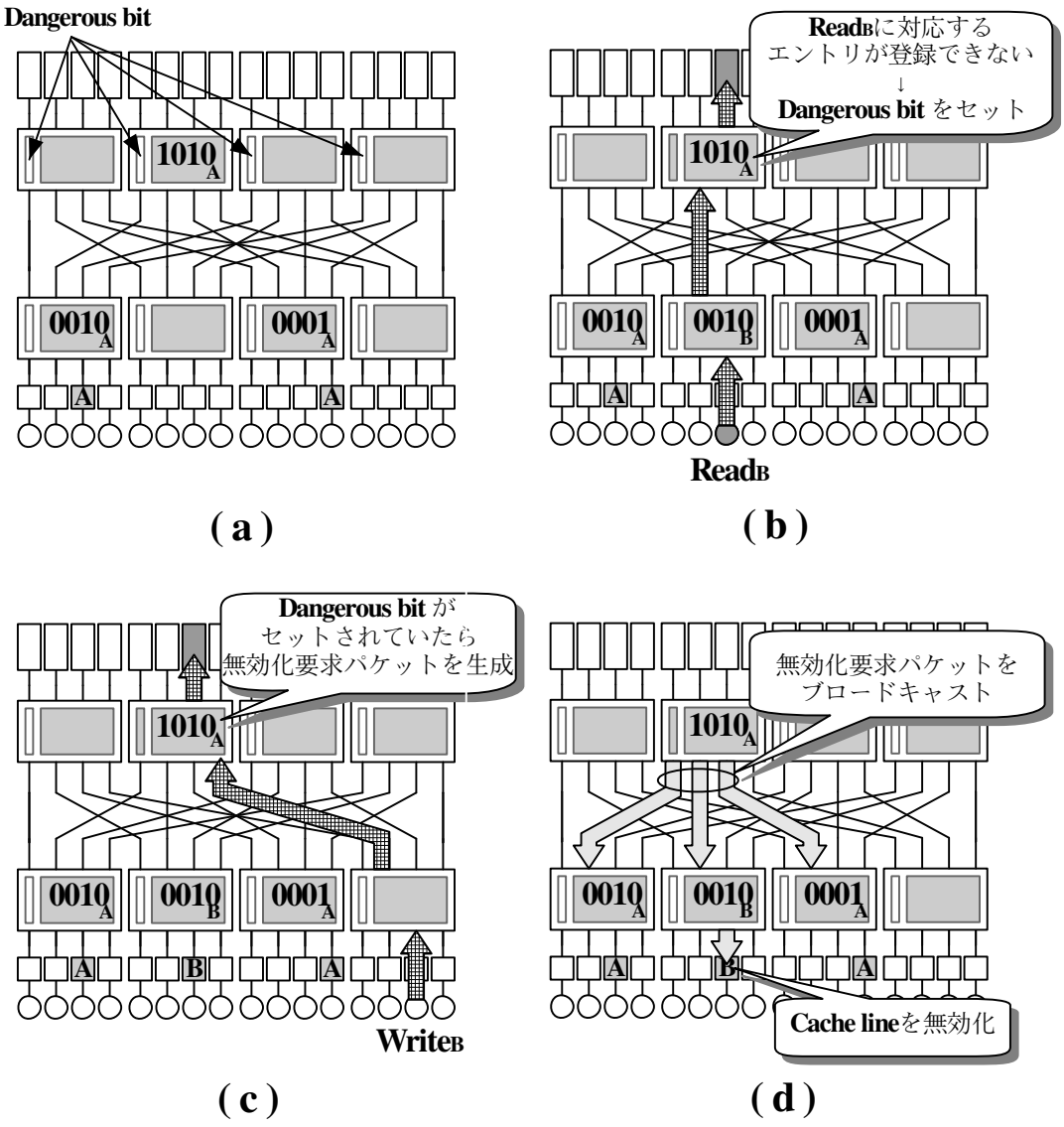

図 5.6: Dangerous bit プロトコルの動作

	Bitmap 1				Bitmap 2			
	Ch_1	Ch_2	Ch_3	Ch_4	Ch_1	Ch_2	Ch_3	Ch_4
Index 0	0	0	1	0	—	—	—	—
Index 1	0	1	0	1	0	1	0	0
Index 2	1	1	0	0	1	0	0	1
Index 3	0	1	0	0	—	—	—	—
●	0	0	1	1	1	0	0	0
●	—	—	—	—	—	—	—	—
●	—	—	—	—	—	—	—	—

図 5.7: Invalidation Broadcast プロトコルの TD の構成

5.4.2 Invalidation Broadcast プロトコル

Dangerous bit プロトコルと同様に，先に TD に登録されている共有情報を保持し続け，新たな読み出し要求に対応する共有情報を登録しない方式．未登録情報を保持する場所が Dangerous bit プロトコルとは異なり，図 5.8(a) に示すように，MM のキャッシュライン毎の 1 ビットのディレクトリである Broadcast bit に未登録情報が記録される．これにより，Dangerous bit プロトコルのように，スイッチ内に Dangerous bit を持つ必要がなくなるので，スイッチ構造を簡略化することができる．なお，Broadcast bit を記憶するためのメモリが MM に必要となるため，このプロトコルでは MINDIC の基本構造に反して MM に小容量のディレクトリを設けていることになる．

TD は図 5.7 のような構成であり，これは次に説明する Eviction プロトコルと同様である．

図 5.8(b) のように，読み出し要求パケットがスイッチに入力されたると，TD の対応するエントリに空きがない場合，読み出し要求パケットにフラグがセットされて MM へ転送される．MM ではフラグがセットされていると，Broadcast bit に未登録情報をセットされる．

そして，図 5.8(c)〜(d) のように，書き込み要求が MM に到着した際に未登録情報をセットされていると，MM から特別無効化要求パケットを下位ステージに発行する．特別無効化要求パケットを受け取ったスイッチは，下位ステージへの全リンクに特別無効化要求パケットをブロードキャストする．この特別無効化要求パケットは不要な PU へも大量に転送されることになるが，無効化が必要なキャッシュラインを必ず無効化することができる．

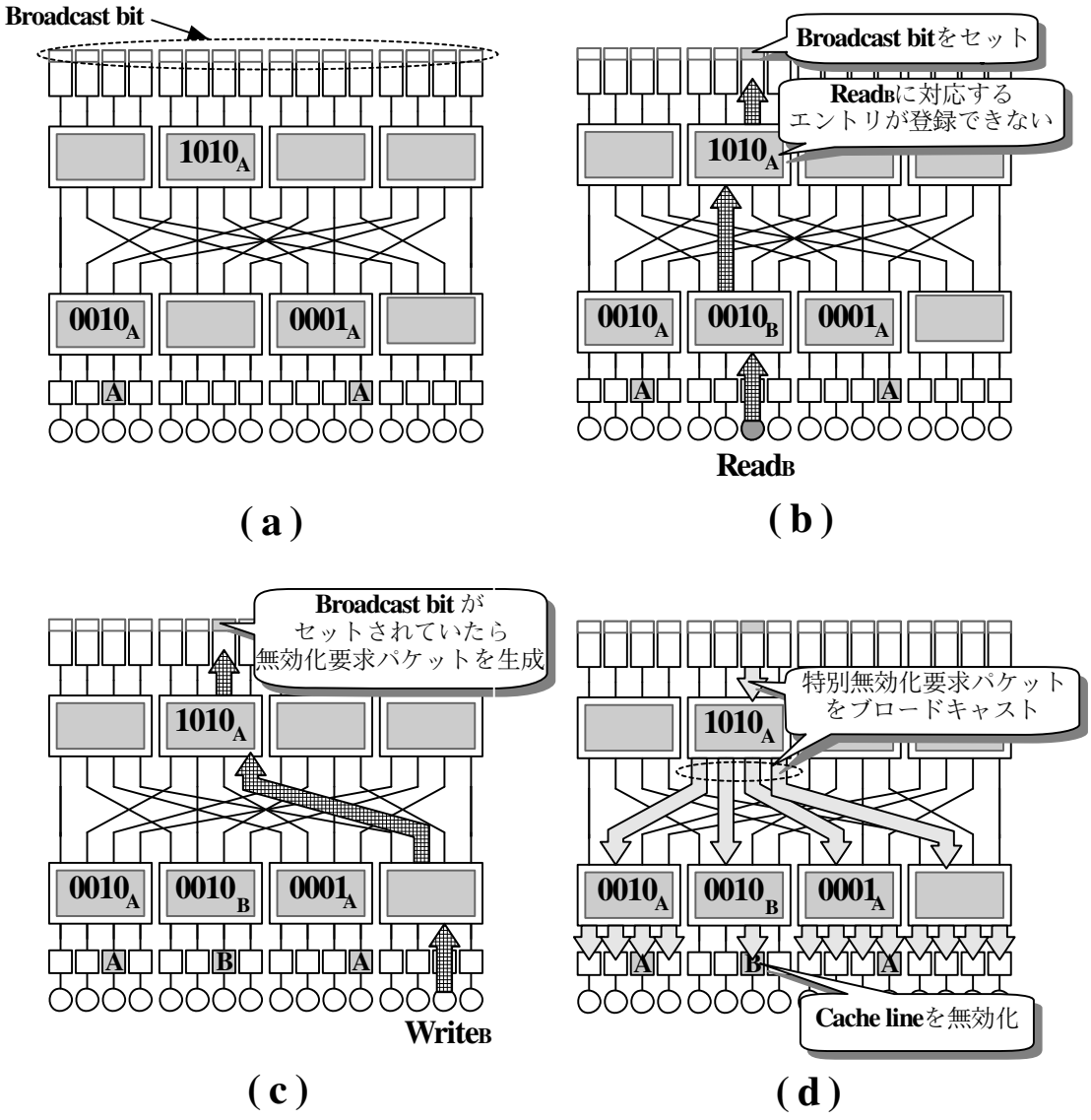

図 5.8: Invalidation Broadcast プロトコルの動作

5.4.3 Eviction プロトコル

先に TD が登録されており，新たな読み出し要求の共有情報を登録するエントリが存在しない場合，先に TD に登録されている共有情報を破棄し，新たな読み出し要求に対応する共有情報を登録する方式．共有情報を破棄する際に，対応する PU 側のキャッシュを無効化することで，PU 側でキャッシュされているラインの共有情報は，必ず TD に登録された状態に保つことができる．

図 5.10(a) に示したように，キャッシュライン A が 2 つの PU にキャッシュされた状態で，他の PU からキャッシュライン B に対する読み出し要求 Read_B が MM へ発生した場合の動作を説明する．この時，MM 側のスイッチ内の TD にはキャッシュライン A の共有情報が登録されており，キャッシュライン B に対する読み出し要求の共有情報をすぐに登録することができない．

このように，スイッチ内の TD に共有情報が登録することができないとき，スイッチは次のように動作する．

(1) インデックスの競合するラインから，LRU により追い出すエントリを決定

(2) 追い出すエントリのキャシュラインの無効化要求を PU 側に発行

(3) 追い出しによって空いたエントリに共有情報を登録

MM 側のスイッチでは，キャッシュライン A の共有情報に基づいて図 5.10(b) に示すように無効化要求パケットをマルチキャストし，PU 側のキャッシュライン A が無効化を行う．その後，キャッシュライン B に対する読み出し要求の共有情報を TD に登録することができる．

また，図 5.9 にスイッチの入出力チャンネルが 4×4 で連想度が 2 の TD の構成を示した．インデックスが対応するエントリに空きがあればビットマップは登録され，空きがない場合は前記したように LRU によって登録されていたエントリが追い出され，新たに登録される．

	Bitmap 1				Bitmap 2			
	Ch_1	Ch_2	Ch_3	Ch_4	Ch_1	Ch_2	Ch_3	Ch_4
Index 0	0	0	1	0	—	—	—	—
Index 1	0	1	0	1	0	1	0	0
Index 2	0	0	1	0	1	0	0	1
Index 3	0	1	0	0	—	—	—	—
●	0	0	1	1	0	1	0	0
●	—	—	—	—	—	—	—	—
●	—	—	—	—	—	—	—	—

図 5.9: Eviction プロトコルの TD の構成

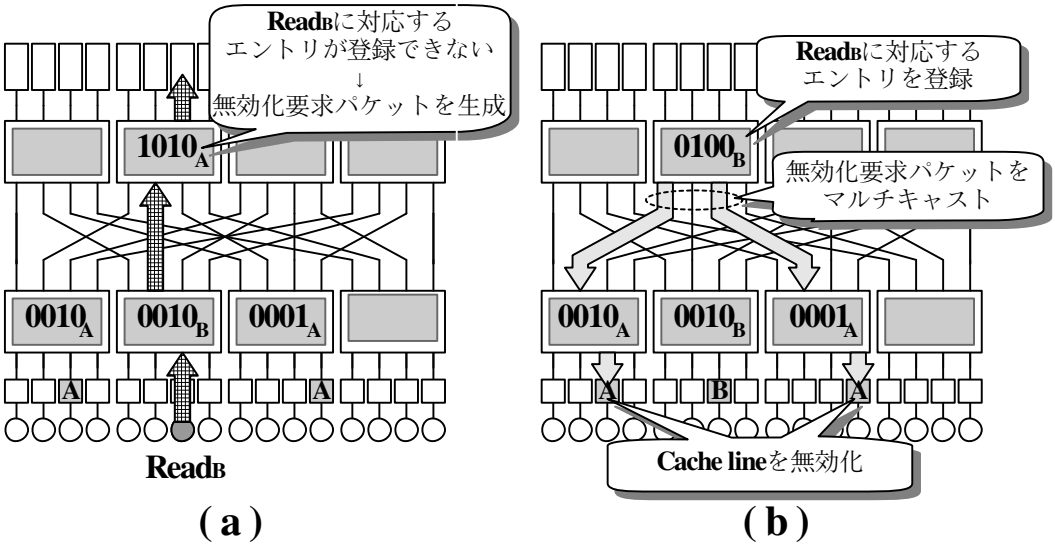

図 5.10: Eviction プロトコルの動作

第6章　MINDICの評価

本章では，クロックレベル汎用シミュレータ ISIS を用いた MINDIC についての評価を行う．まず，トレースデータを利用したトレースドリブンシミュレーションにより MINDIC の 3 種類の転送プロトコルの比較検討を行う．その後，3 種類の転送プロトコルのうち，最も効率良いキャッシュ制御を実現できる Eviction プロトコルを採用した MINDIC のクロックレベルシミュレータの実装を行い，従来技術と比較した詳細な評価を行う．

最後に，ハードウェア量について，ディレクトリに必要なメモリ容量やスイッチ回路規模についての評価も行う．

6.1　ISIS

MINDIC の評価を行うために，本研究室で開発されたクロックレベル汎用並列計算機シミュレータライブラリ ISIS［WA01］を利用してシミュレータを構築し，シミュレーションを行っている．まずは，ISIS についての紹介する．

ISIS は，並列システムの性能評価，プログラム開発用シミュレータを構築するための C++言語用のライブラリツールである．プロセッサ，メモリ，バス等の並列計算機を構成する代表的な部品の挙動をクロック単位で記述した機能ブロック，機能ブロック間の接続のためのインターフェースとしてポート，送受信される情報を表わすパケットが，それぞれ基本要素としてクラスで提供されている．

また，必要ならばそれぞれのクラス階層から必要なクラスを取り出し，その派生クラスで意図した機能を実装し新たな機能ブロックを構成する．そして，それぞれの機能ブロック間の接続を記述してゆくことにより，比較的容易にシミュレータを構成することができる．

第 6.1 図にライブラリ構成を示す．ユーザは ISIS ライブラリのクラス群を用いてトップモジュールを記述する．用意されていないクラスは基底クラスの派生クラスとして実装を行う．記述したシミュレータはコンパイルし，リンクすることで実行ファイルとすることができる．

6.2　トレースドリブンシミュレータによる予備評価

3 つの転送プロトコルを比較するために，各転送プロトコルで動作する MINDIC のトレースドリブンシミュレータを実装して評価を行った．トレースデータには，キャッシュ制御機構として MINC を採用したスイッチ結合型並列計算機 SNAIL–2［TDM+02］のクロックレベルシミュレータによるメモリアクセス命令のトレースを用いた．

MINC では，PU にキャッシュを，MM にディレクトリを配置し，共有情報を RHBD(Reduced Hierarchical Bit-map Directory) 方式を用いて縮約してディレクトリに保持することで，必

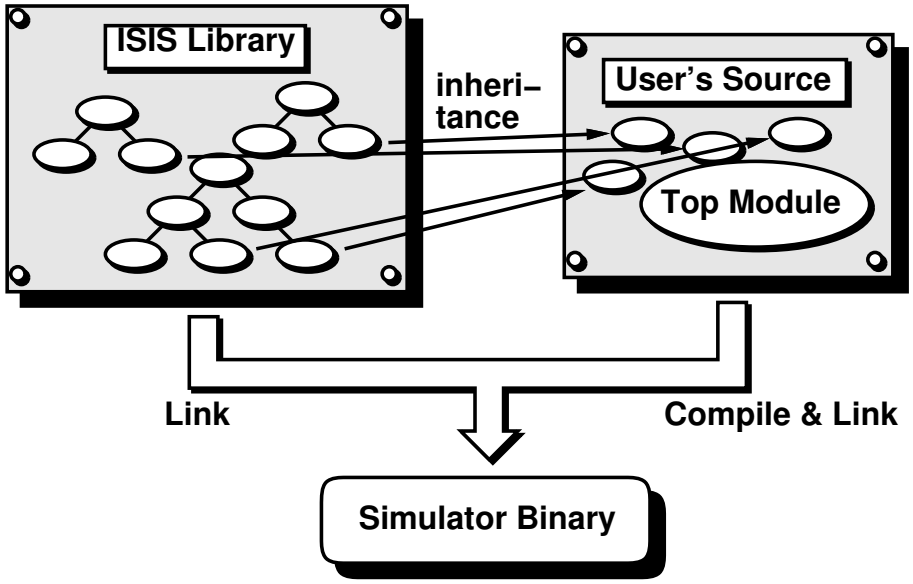

図 6.1: ISIS のライブラリ構成

要なメモリ量を削減したキャッシュ制御方式である．書き込み要求の発生時に，ディレクトリの共有情報が参照され，キャッシュラインを保持している PU に対して，MM から無効化要求パケットが MIN 構造のネットワークを介して転送される．MINC では，共有情報が縮約されているため必要以上の無効化要求パケットが発生するという問題を有している．

トレースデータは，PU から PBSF ネットワークへ出力されるアクセス要求をクロックレベルシミュレータを動作させて抽出した．クロックレベルシミュレータでは，PU 数は 4，各 PU のキャッシュは 256KB, 2-way，キャッシュラインサイズは 32 byte とし，共有アドレス空間を持つ並列計算機で広く用いられている並列ベンチマークプログラム集 SPLASH-2(Stanford ParalleL Applications for SHard memory-2) [SME+95]の 4 つのアプリケーション実行し，抽出を行った．抽出したトレースデータの発行命令数を表 6.1 に示す．

表 6.1: トレースデータの発行命令数

	Read	Write	Test & Set	Fetch & Dec
RADIX	589411	832595	76	69
FFT	1621926	7025179	0	53
LU	2368951	4718639	0	213
OCEAN	908616	3024499	946	6168

6.2.1 トレースドリブンシミュレータの実装

MINDIC の転送プロトコルの評価を行うために，第 6.2 図に示すトレースドリブンシミュレータの実装を行った．トレースプロセッサはトレースデータから入力パケットを生

成し，スイッチに投入可能であれば投入する．トレースプロセッサ，スイッチは共に ISIS を用いて実装されており，クロックレベルシミュレータとして動作可能である．スイッチは，書き込み/読み出しパケットを転送する順方向ポートが 4×4，無効化パ ケットを転送する逆方向ポートが 4×4，ディレクトリキャッシュは，2way, 1024 エントリの構成とした．

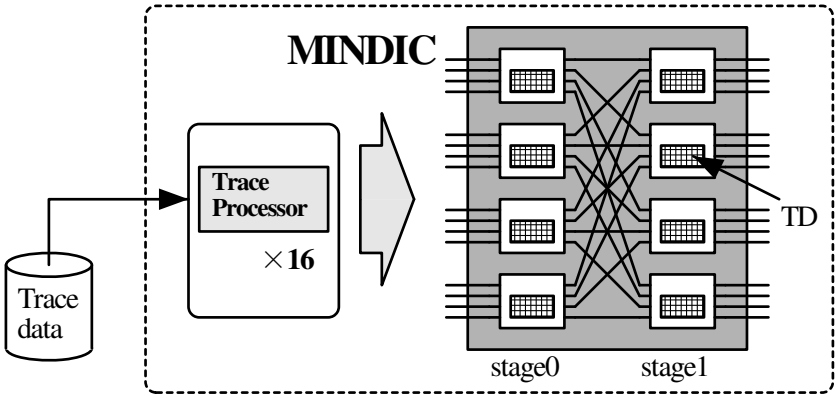

図 6.2: トレースドリブンシミュレータの構成

6.2.2 トレースドリブンシミュレータの評価結果

テンポラリディレクトリのヒット率

PU 側からスイッチに入力された読み出し要求と書き込み要求に対するテンポラリディレクトリのヒット率について検討する．

読み出し要求アクセスのヒット率は，各アプリケーション毎のプロセッサのデータ共有率に影響される．データ共有率の低い，Radix，LU，FFT では，読み出し要求アクセスのヒット率は低くなり，Ocean はデータ共有率が高いのでヒット率が他のアプリケーションよりも高くなると考えられる．また，Dangerous bit プロトコルと Invalidation Broadcast プロトコルでは，TD のエントリ数が少ないと，読み出し要求アクセスの共有情報が TD に登録されなくなるので，読み出しアクセスのヒット率は低くなると考えられる．同様に，書き込み要求アクセスのヒット率もエントリ数が少ない状況では，低くなると考えられる．

そこで，PU 側からスイッチに入力された読み出し要求と書き込み要求に対する TD のキャッシュヒット率を計測し，図 6.3〜図 6.6 に示した．図 6.3〜図 6.6 の書き込み要求入力時の TD のヒット率を見ると，全てのアプリケーションにおいて Eviction プロトコルを用いた場合のライトアクセスで，常に高いヒット率を示すことが確認できる．Dangerous bit プロトコルと Invalidation Broadcast プロトコルでのライトアクセスは，TD のエントリ数が少なくなるに従って TD に登録されない共有情報が増加してしまうため，TD のヒット率が低下している．なお，Ocean 以外のアプリケーションで，読出し要求に対する TD のヒット率が低いのは，キャッシュラインが共有される PU 数が少ないアプリケーションであるからと考えられる．

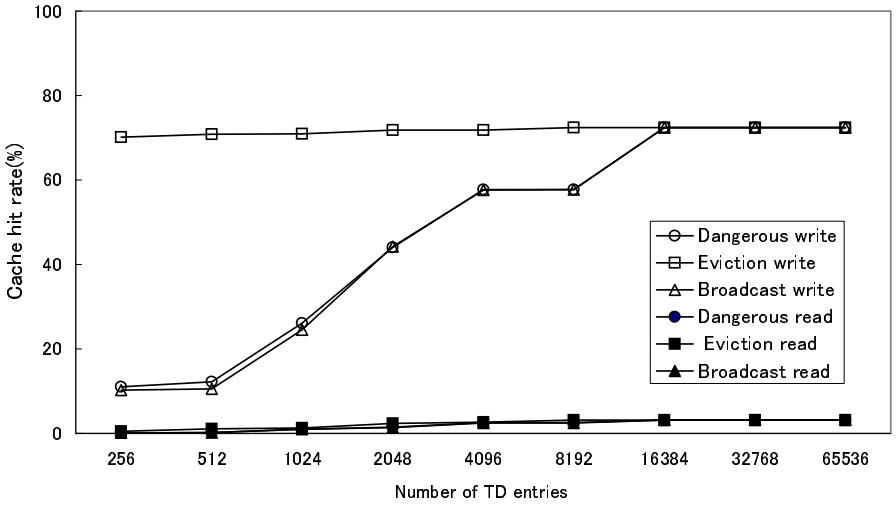

図 6.3: テンポラリディレクトリのヒット率 (16PU，Radix，TD:1way)

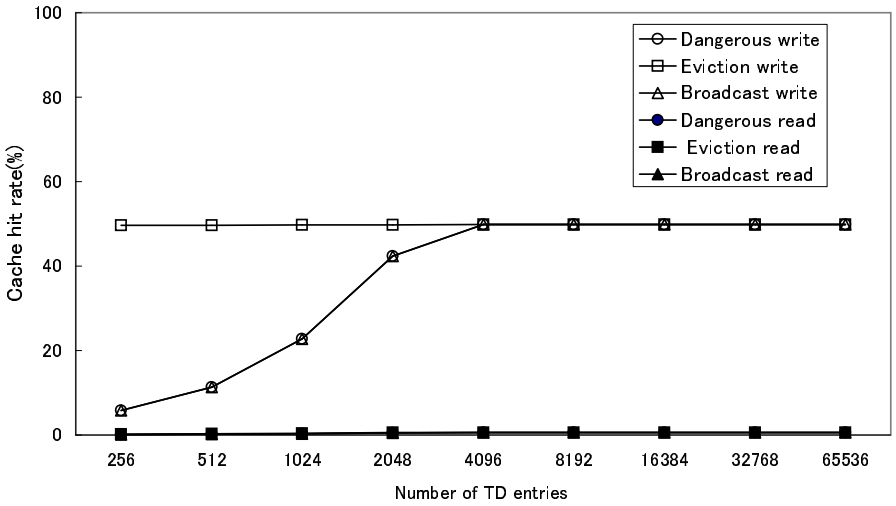

図 6.4: テンポラリディレクトリのヒット率 (16PU，LU，TD:1way)

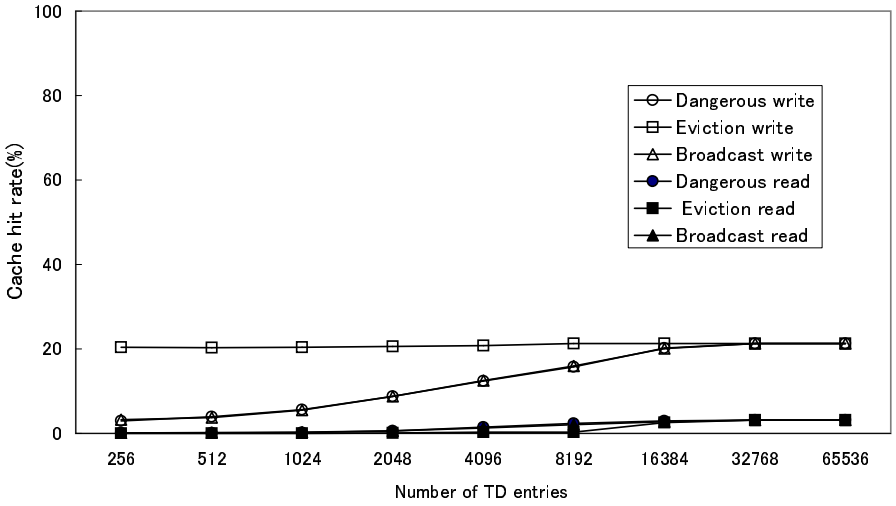

図 6.5: テンポラリディレクトリのヒット率 (16PU，FFT，TD:1way)

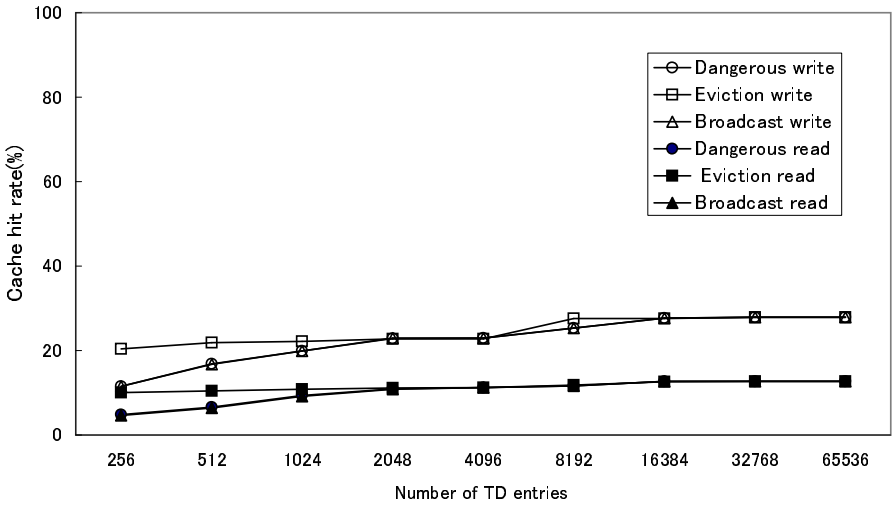

図 6.6: テンポラリディレクトリのヒット率 (16PU，Ocean，TD:1way)

無効化パケットの発生数

Dangerous bit プロトコルと Invalidation Broadcast プロトコルで共有情報が登録されない状況が発生すると，大量の無効化パケットのマルチキャストが発生してしまうと考えられる．そこで，無効化パケットの発生状況についても計測した．

PU 側へ送出された無効化パケットの総数を図 6.7〜図 6.10 に示した．なお，MINC をキャッシュ制御機構として用いた際の無効化パケットの総数を 1 としている．図 6.7〜図 6.10 の無効化パケットの発生数を比較すると，Eviction プロトコルを用いた場合に，8192 エントリ以下のエントリが少ない状態においても無効化パケットによるネットワークの混雑が少なくなることが確認された．

Dangerous bit プロトコルと Invalidation Broadcast プロトコルでは，エントリ数が少なくなるにしたがって，Dangerous bit やディレクトリの未登録情報が頻繁にセットされることになり，急激に無効化パケット発生数が増加してしまっている．

TD の連想度を上げて TD を効率良く使用するこで，Eviction 以外のプロトコルにおいて，TD のヒット率が急激に低下するエントリ数や，無効化パケットが大量に発生するエントリ数をより少ないエントリ数とすることはできる．しかし，TD の連想度，PU 数，キャッシュサイズを異なる条件としたとしても，図 6.3〜図 6.6 の TD のヒット率や，図 6.7〜図 6.10 の無効化パケットの発生数の 3 つの転送プロトコルの傾向は同様であると考えられる．

これらの結果から常に高い TD のヒット率を示し，無効化パケットの発生数が少ない Eviction プロトコルが 3 つのプロトコルにおいて最も有効な転送プロトコルであることが確認された．

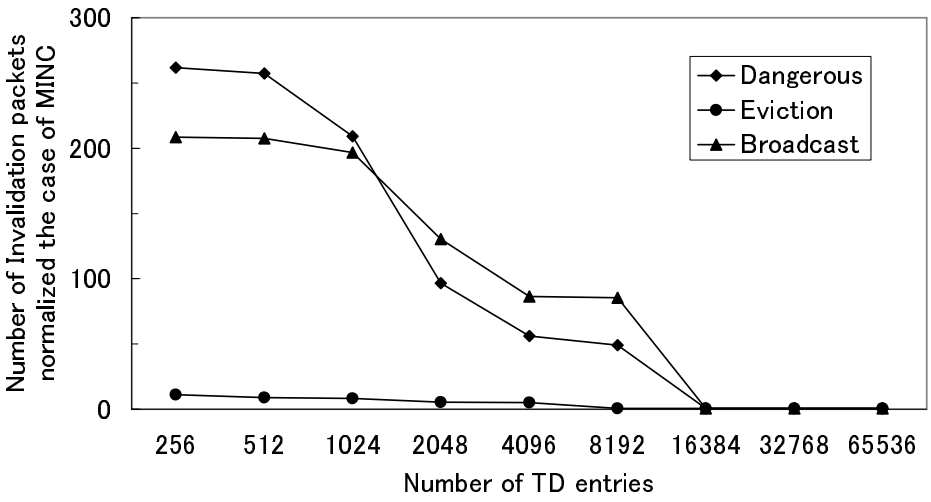

図 6.7: 無効化パケットの発生数 (16PU，Radix，TD:1way)

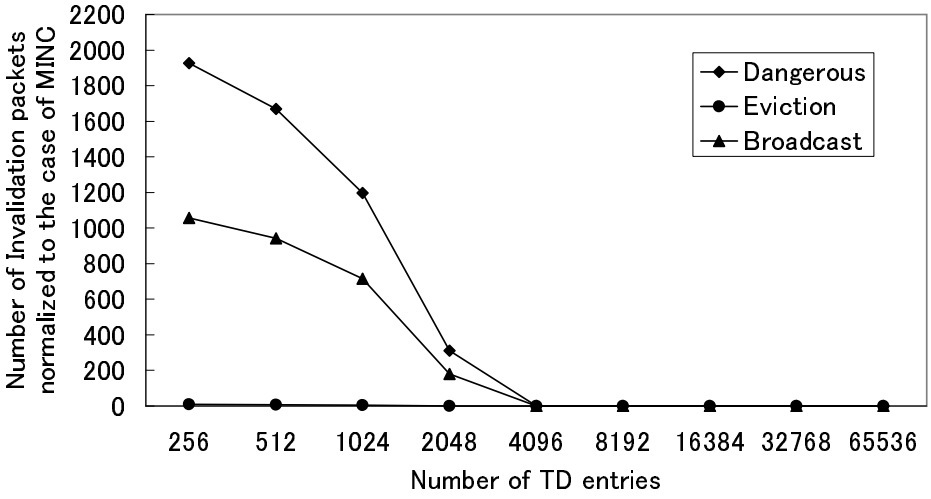

図 6.8: 無効化パケットの発生数 (16PU，LU，TD:1way)

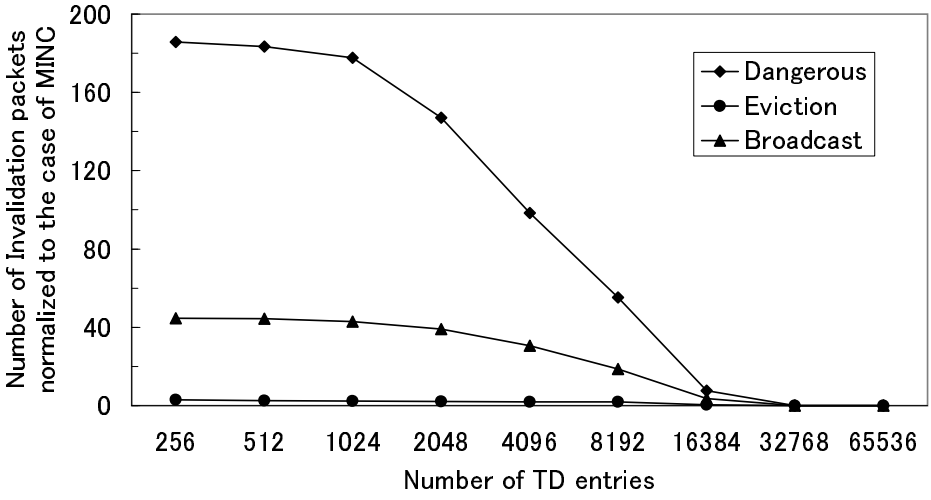

図 6.9: 無効化パケットの発生数 (16PU，FFT，TD:1way)

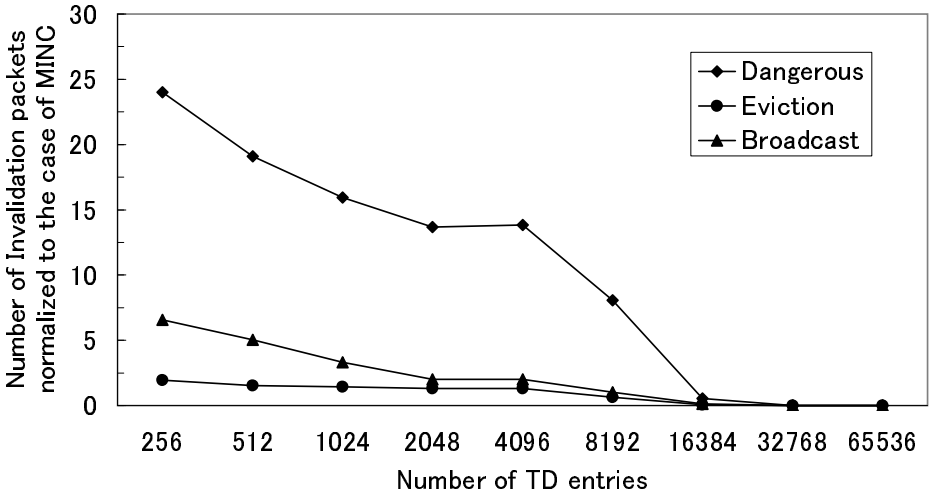

図 6.10: 無効化パケットの発生数 (16PU，Ocean，TD:1way)

ステージ毎の無効化パケットの発生数

ここでは，無効化パケットの発生する原因について検討を行う．無効化パケットは，全てのスイッチで発生する可能性があるが，Dangerous bit プロトコルと Invalidation broadcast プロトコルでは，共有情報を登録できず，Dangerous bit 及び Broadcast bit が設定されていた場合に発生する無効化パケットのブロードキャストが無効化パケットの大量発生の原因になると考えられる．また，Eviction プロトコルの無効化パケットの発生は読み出しパケット転送の際に，TD からエントリを追い出したことによるものが多く発生すると考えられる．

図 6.11〜6.12 に，無効化パケットの発生数の変化が大きい 4096〜32768 エントリにおける，無効化パケットの発生数の詳細を示した．各グラフは，各ステージにおいて下位ステージへ出力される無効化パケットの総数を表しており，無効化パケットの発生原因別に色分けされている．Upper switch とは上位ステージから受け取った無効化パケットにより発生した無効化パケットのことである．

図 6.11 で，Dangerous bit プロトコルは TD が 8192 エントリ以下で無効化パケットが大幅に増加してしまっている．stage1 の内訳を参照すると，上位ステージである stage1 から下位ステージである stage0 へ転送された無効化パケットの 99%程度が，書き込み要求時の TD miss 時に Dangerous bit がセットされていたことにより発生した無効化パケットのブロードキャストが原因となっている．stage0 の From upper switch が示すように，この無効化パケットは stage0 に転送されて更に増幅されるため，無効化パケットの大幅な増加につながることがわかる．

また，図 6.12 の Invalidation Broadcast プロトコルでも，TD が 8192 エントリ以下になると無効化パケットが多くなっている．書き込み要求時に MM のブロードキャストビットがセットされていたことにより MM から発生する無効化ブロードキャストパケットは

stage1 に入力され stage0 へブロードキャストされる．この無効化ブロードキャストパケットを受け取った stage0 では更に PU 側へ無効化パケットをブロードキャストするので，無効化パケットの数が大幅に増幅してしまう．

図 6.13 を見ると，Eviction プロトコルでは読み出し要求時に TD にミスし，先に TD に登録されていたエントリを追い出すことにより発生する無効化パケットの影響が大きいことが確認できる．

しかし，無効化パケットの発生数自体が少なく抑えられており，無駄な無効化パケットの発生を十分に抑制できており，効率的なキャッシュ制御が可能であることがわかる．

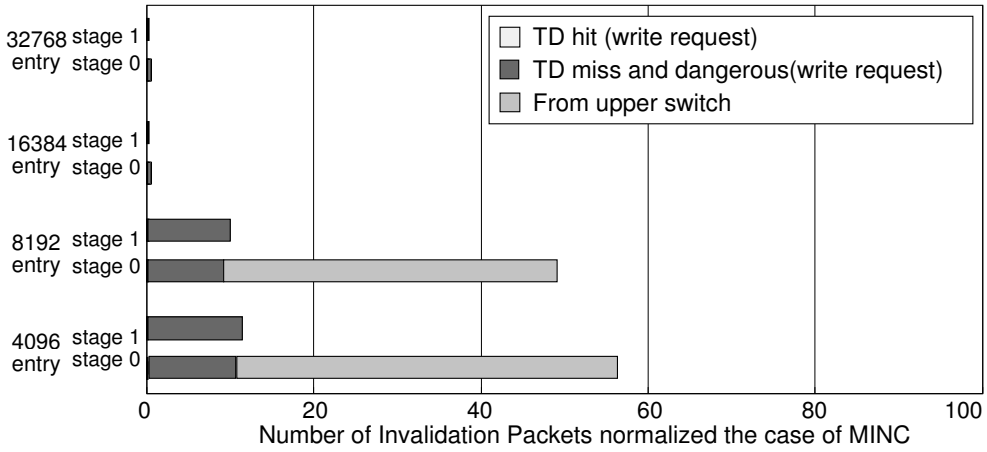

図 6.11: 無効化パケットの発生数 (16PU, Radix, TD:1way, Dangerous bit プロトコル)

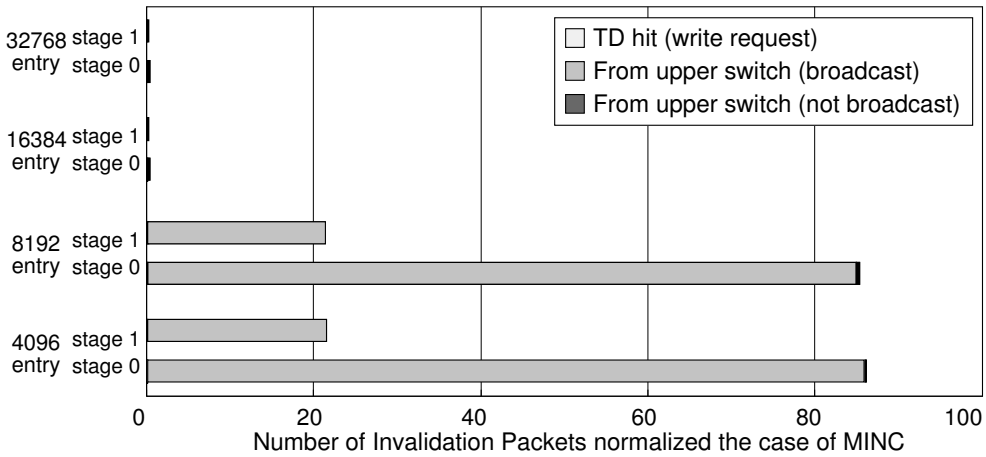

図 6.12: 無効化パケットの発生数 (16PU, Radix, TD 1way, Invalidation Broadcast プロトコル)

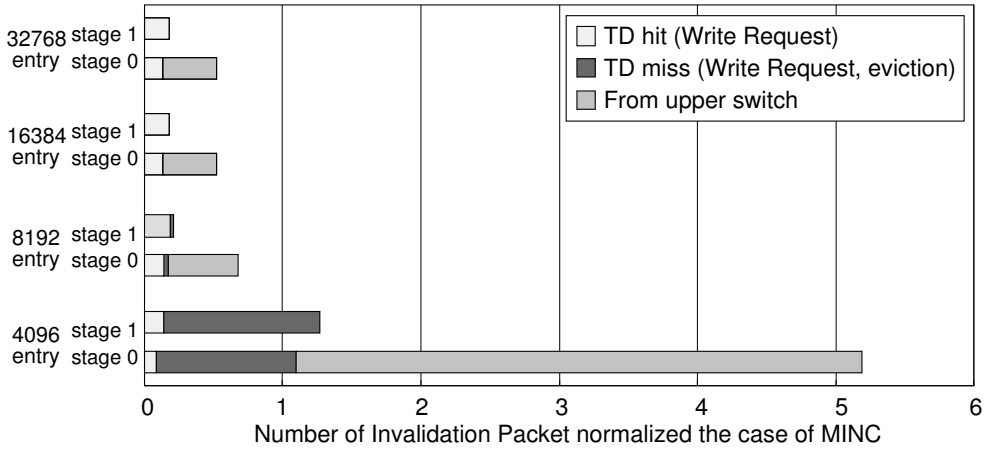

図 6.13: 無効化パケットの発生数 (16PU, Radix, TD:1way, Eviction プロトコル)

Eviction プロトコルにおける連想度の影響

次に，無効化パケットの抑制に最も有効な Eviction プロトコルにおける，連想度の違いを検討する．TD は容量が限られているので，連想度を増やすことで TD を効率良く使用することで，無駄な追い出しを少なくし，無効化パケットの発生数を抑えることができると考えられる．

TD の連想度と無効化パケットの発生数の関係を図 6.14〜6.17 に示した．Radix では，1024〜8192 エントリの間では，連想度が高いほど無効化パケットの発生数を抑えることができる．エントリ数を 16384 以上にするとエントリが十分に大きくなるため，連想度の影響は少くなると考えられる．また，FFT でも Radix と同様の傾向にあり，4096〜16384 エントリの間では連想度が高いほど無効化パケットの発生数が少く，それ以上のエントリ数では連想度の影響は現れなくなっている．LU では，アクセスするアドレスのパターンの影響から 4096 エントリ以下でも連想度の影響は少くなっているものと考えられる．

アプリケーション毎の無効化パケットの発生数

最後に， Eviction プロトコルで連想度 4 としたときの無効化パケット発生数を図 6.18 に示す．4 つのアプリケーション全てにおいて，Eviction プロトコルを用いた連想度 4 の TD が 4096 エントリ以上あれば無効化パケットの発生数を MINC より少なくすることができることが確認できる．

以降，MINDIC の性能とコストについて従来手法と比較検討を行うために，後述するクロックレベルシミュレータによる評価およびハードウェアコストの評価を行うが，トレースドリブンシミュレータにより，最も効率的な転送プロトコルであることが確認された Eviction プロトコルのみを検討の対象とした．

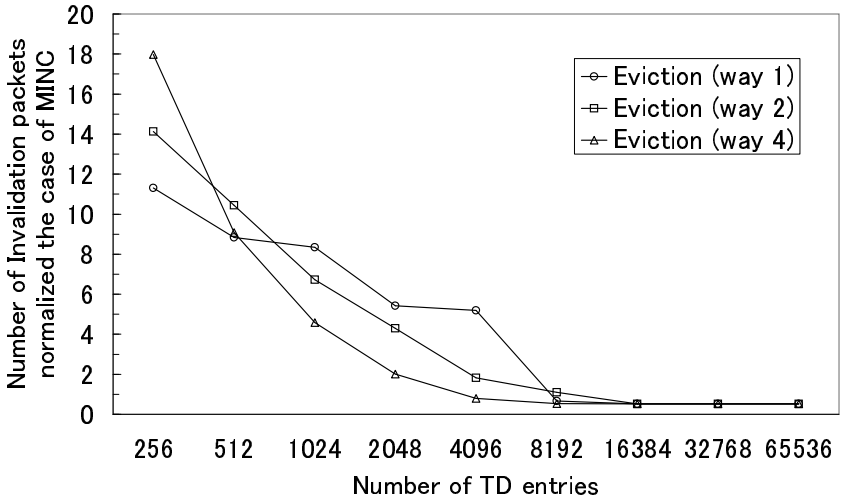

図 6.14: 連想度毎の無効化パケットの発生数 (16PU, Radix，Eviction プロトコル)

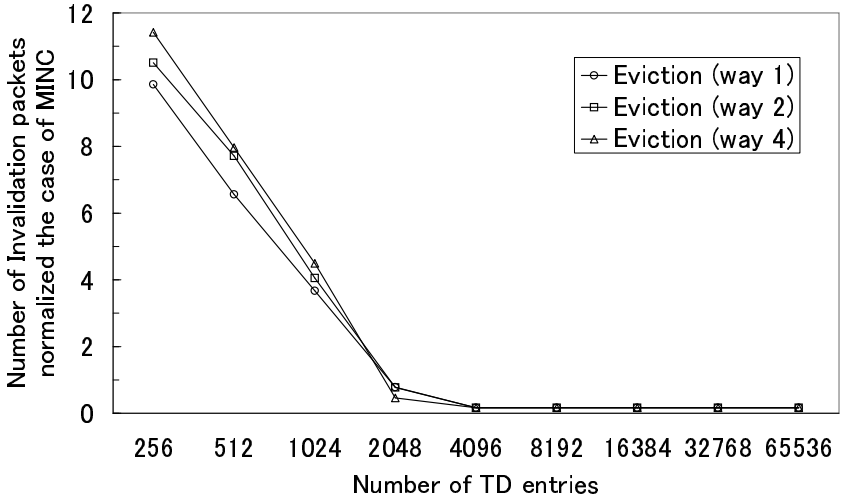

図 6.15: 連想度毎の無効化パケットの発生数 (16PU, LU，Eviction プロトコル)

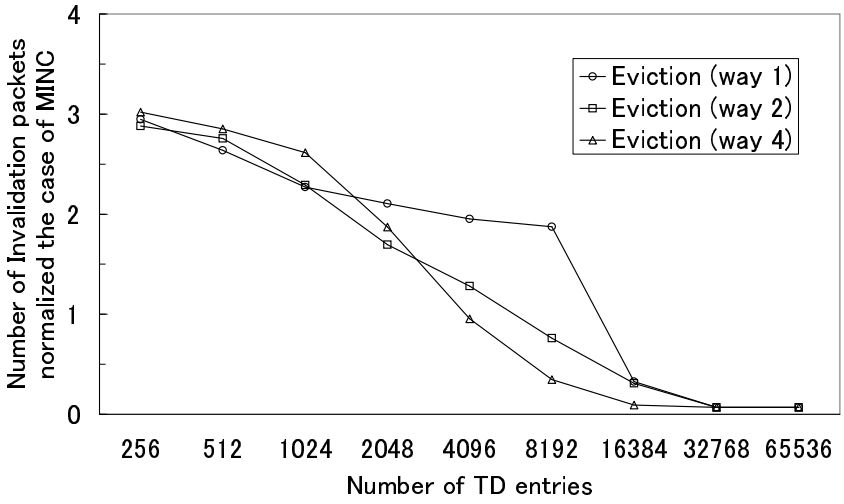

図 6.16: 連想度毎の無効化パケットの発生数 (16PU, FFT，Eviction プロトコル)

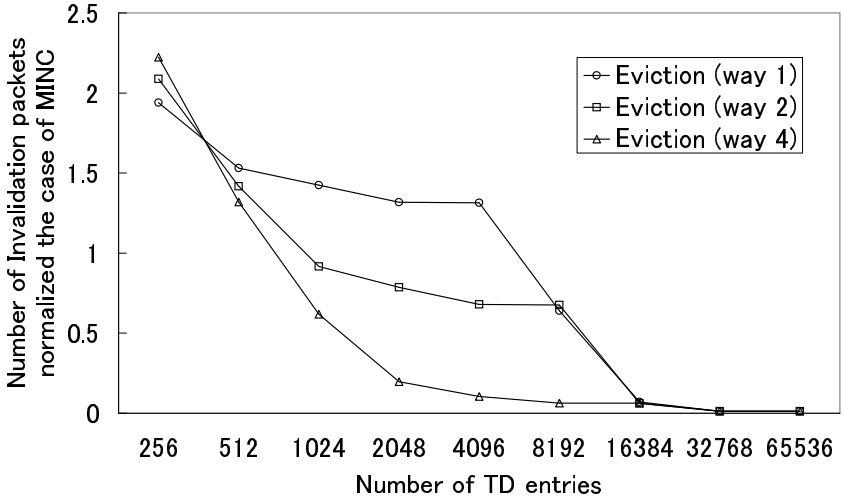

図 6.17: 連想度毎の無効化パケットの発生数 (16PU, Ocean，Eviction プロトコル)

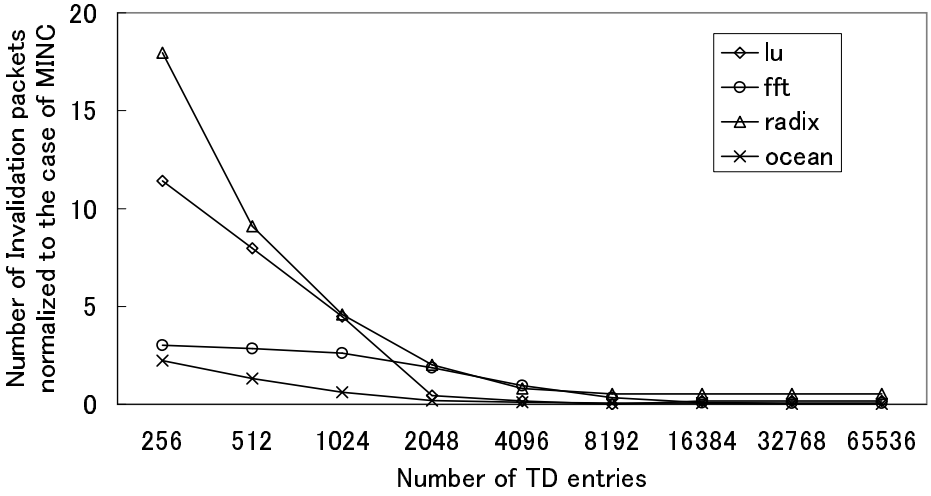

図 6.18: アプリケーション毎の無効化パケットの発生数 (16PU, Eviction プロトコル，4-way)

6.3 クロックレベルシミュレータによる評価

6.3.1 クロックレベルシミュレータの実装

MINDIC の評価のために，C++言語用のクロックレベル汎用並列計算機シミュレータライブラリ ISIS [WA01]を用いてクロックレベルシミュレータを実装した．このシミュレータを実装した時点において，ISIS がサポートしているプロセッサは，MIPS 社の R3081 のみであるので，R3081 を用いている．しかし，R3081 は処理能力が低く，ネットワークに与える負荷が小さくなってしまうことから，プロセッサとメモリの動作クロックをネットワークの動作クロックの 4 倍に設定し，ネットワークの負荷を上げて評価を行った．

シミュレータは，最大 64PU 構成のシステムで動作可能であり，PU と MM は，3 段のステージ，各ステージ 16 スイッチで合計 48 個のスイッチを有する MINDIC で接続されている．MINDIC を構成する各スイッチは，PU 側からの書き込み要求パケットと読み出し要求パケットを MM 側に転送する 4×4 の順方向ポート，MM 側からの無効化パケットを PU 側に転送する 4×4 の逆方向ポートを持つ．データ転送用ネットワークは，MINDIC と同様のスイッチサイズからなる MIN を用いている．命令及びローカルデータは各 PU のメモリに配置し，共有データはキャッシュラインごとにインタリーブして各 MM に配置した．

また，シミュレータは全ての構成が 1 つの LSI に実装されるオンチップマルチプロセッサを想定して，キャッシュやテンポラリディレクトリ (TD) に関するパラメータを表 6.2 に示すように設定している．シミュレータは，性能評価の比較検討を行うために，TD でキャッシュ制御を行う MINDIC だけでなく，MM に設けたフルマップ方式のディレクトリでキャッシュ制御を行うアーキテクチャ(FULL_MAP) でも MINDIC と同様の PU 数，ネッ

表 6.2: キャッシュとテンポラリディレクトリのパラメータ

Cache	Cache Size	1 ~ 1024 KByte/PU
	Associative	2 way
	Line size	8 ~ 1024 Byte
	Protocol	Write Through
TD	Entry	128 ~ 4096 /SW
	Associative	1，2，4 way

表 6.3: ネットワーク速度の比較

Processor	PU 数	PU の動作周波数	ネットワークの転送レート
CELL(IBM, ソニー, 東芝)	9	4GHz	192GBPS(21.3GBPS/PU)
MINDIC	2-64	1GHz	128GBPS(2GBPS/PU)

トワーク構成，キャッシュパラメータで動作させることができる．

次に，クロックレベルシミュレータで想定している MINDIC のネットワークの転送レート，キャッシュ容量が妥当な値であることを示す．MINDIC のクロックレベルシミュレータは，全ての構成が 1 つの LSI に実装されるオンチップマルチプロセッサを想定しているため，商用プロセッサとの比較を表 6.3，表 6.4 に示す．

ネットワークの転送レート

MINDIC のネットワークは 250MHz で動作する想定としており，各 PU とは 32 ビット幅のポートで接続されている．CELL のネットワークはデータ及び命令等の全ての情報を転送するネットワークであるので，MINDIC の転送レートも同様に，MM から PU へデータを転送するデータ転送用ネットワークの転送レートも加えている．表 6.3 では，CELL と比較して，MINDIC のプロセッサあたりの転送レートは低めとなっている．しかし，PU の動作周波数が低く，また，PU に使用する R3081 は 32 ビットプロセッサであり，多重命令発行，out-of-order 実行，分岐予測をサポートしていないなど，処理能力が低いことを考慮すれば妥当な想定であると考えられる．

キャッシュ容量

MINDIC は共有メモリである MM を含めて全て 1 つの LSI に実装される想定であるため， 表 6.4 では，MM を L2 キャッシュに相当するものとしてキャッシュ容量を比較している．

MINDIC の L1 data cache は他のプロセッサと同様の記憶容量とし，L2 cache は大きな記憶容量を設定している．これは，64PU が 1 つの LSI に実装できる程度に半導体の集積技術が向上した状況を想定しているためである．L2 cache のみ記憶容量を増加させている

表 6.4: L1,L2 キャッシュ容量の比較

Processor	PU 数	L1 data cache	L2 cach
Power 5(IBM)	2	32KByte/PU	1.92MB/chip
MPCORE (ARM)	1〜4	1K〜16KByte/PU	128K〜2MB/chip
Pentium EE 840(Intel)	2	16KByte/PU	1MByte × 2/chip
Athlon 64 X2 4800+(AMD)	2	64KByte/PU	1MByte × 2/chip
MINDIC	2-64	32KByte/PU	16MByte × 16/chip

表 6.5: メモリアクセスレイテンシ

MINDIC	読み出し要求 (Cache hit)	2 clock
	読み出し要求 (Cache miss)	80 clock
	書き込み要求	64 clock
FULL_MAP	読み出し要求 (Cache hit)	2 clock
	読み出し要求 (Cache miss)	80 clock
	書き込み要求	64 clock

のは，L1 data cache の記憶容量を増加させるとキャッシュヒット率を向上すること以上に，アクセスレイテンシが大きくなることが影響し，処理速度低下につながると考えられるからである．

アクセスレイテンシ

MINDIC と FULL_MAP を用いた場合に，PU が MM をアクセスする際のアクセスレイテンシを表 6.5 に示す．読み出し要求の場合，キャッシュヒット時はデータをキャッシュから受け取ることができるので，MINDIC と FULL_MAP は共に 2clock のアクセスレイテンシとなる．ただし，ここでのレインテンシは，他の要求と競合しない場合の最短のレイテンシを挙げているため，Cache Hit 時のレイテンシ以外は，他のアクセスとのネットワーク上での競合により増加する．キャッシュミス時はスイッチを介して MM へ読み出し要求パケットを転送し，MM からデータを受け取る．MINDIC では，読み出し要求パケットが転送される各スイッチ毎に TD へのアクセスが行われるが，TD へのアクセスはパケットの転送と並行して行われるため，レイテンシには影響しない．そのため，MINDIC と FULL_MAP は共に各スイッチ毎に 4clock の遅延で読み出し要求パケットを転送し，MM でのデータ読み出し時間と PU へのデータ転送時間を含めて，80clock のアクセスレイテンシとなる．書き込み要求のレイテンシは，MM が更新されるまでの時間を示しているが，MINDIC と FULL_MAP とも，PU がネットワークに書き込み要求パケットを出力した時点で，次の処理に進むことができる．

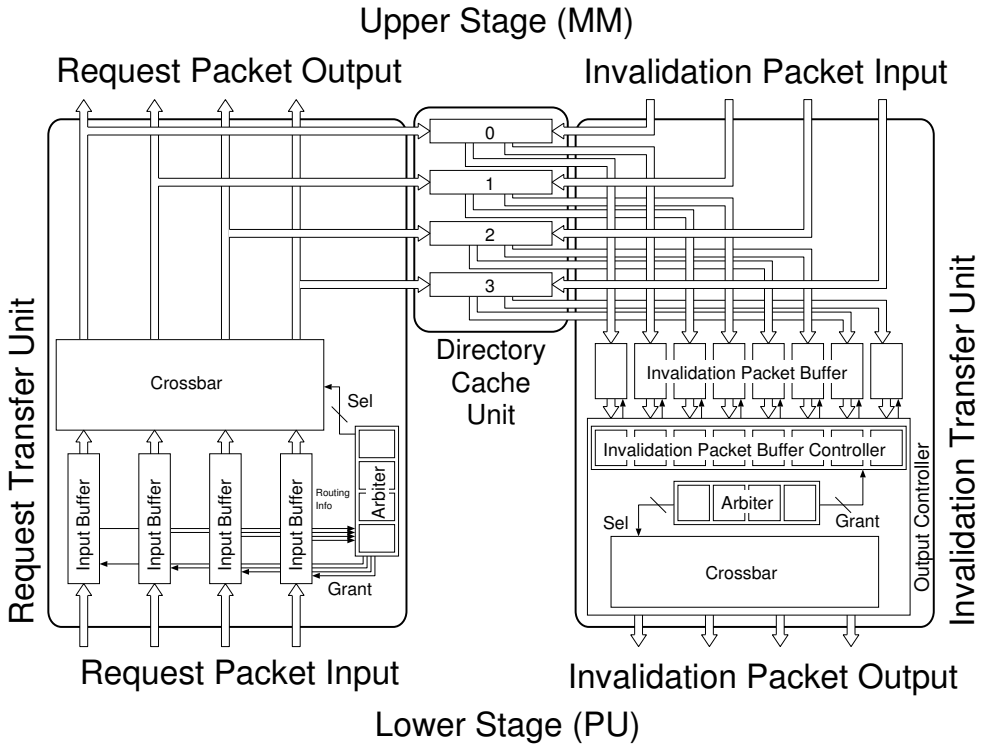

図 6.19: MINDIC スイッチの構成

6.3.2 クロックレベルシミュレータのスイッチ構成

図 6.19 に，構築した MINDIC のスイッチの内部構成を示す．MINDIC の各スイッチは，PU 側の下位ステージから MM 側の上位ステージへのメモリアクセス要求転送用に 4 入力 4 出力，上位ステージから入力された無効化要求パケットを下位ステージにマルチキャストするために 4 入力 4 出力，合計 8 入力 8 出力のポートをもつ．

下位ステージからのメモリ読み出し/書き込み要求は，スイッチのリクエスト転送部 (図 6.19 の左側ユニット) にある Input Buffer へ入力される．これらの要求は Crossbar を介して上位ステージへのリンクへ出力されるが，この際に出力リンク毎に設けられた TD が参照され，要求の種類に応じてキャッシュの共有情報の登録や更新，無効化パケットの生成が行なわれる．生成された無効化パケットは Invalidation Packet Buffer に取り込まれ，Output Controller により下位ステージへマルチキャストされる．上位ステージから逆送される無効化要求パケットは，無効化パケット転送部から入力され，下位ステージへマルチキャストされる．

TD は，下位ステージからのメモリアクセス要求と，上位ステージからの無効化要求の両方に同時に対応するために，Dual port RAM で実装されている．シミュレータでは，Eviction プロトコルを採用していることから，TD ユニットは次のように動作する．

読み出し要求

読み出すキャッシュラインの共有情報を TD へ登録する．TD のエントリが一杯で，これ以上共有情報を登録できない場合，LRU にしたがって既存の共有情報を破棄し，新規の共有情報を登録する．この際，キャッシュの一貫性を保つために，共有情報が破棄されるキャッシュラインを保持する PU のキャッシュを無効化するために無効化パケットを生成する．無効化パケットには，該当ラインを保持する PU へマルチキャストするためのビットマップが付加されており，無効化パケット転送部の Invalidation Packet Buffer に取り込まれる．

書き込み要求

書き込みが発生したラインを共有している PU のキャッシュを無効化するために，TD の共有情報をもとに無効化パケットを生成する．無効化パケットは，下位ステージへマルチキャストするためのビットマップが付加されており，無効化パケット転送部にある Invalidation Packet Buffer に蓄えられる．

無効化要求

MM から PU 方向へ逆送されてきた無効化パケットは，上位ステージとのリンク毎に設けられた TD を参照する．TD にヒットすれば TD から読み出したビットマップを付加して Invalidation Packet Buffer に送られる．TD にミスした場合は，無効化要求は消滅する．

6.3.3 アプリケーション

評価に用いるアプリケーションは，並列ベンチマークプログラム集 SPLASH-2［SME+95］から LU，FFT，RADIX の 3 つを選択し，PU 数 1〜64 で実行した．各アプリケーションの評価条件を表 6.6 に示す．

表 6.6: 各アプリケーションの評価条件

Application	Input
Radix	radix 1024, keys 16384
FFT	4096 complex doubles
LU	64×64 matrix, 16×16 element blocks

6.3.4 実行時間

TD のエントリ数による影響

MINDIC を用いた場合とフルマップ方式をキャッシュ制御に用いた場合のそれぞれでアプリケーションを実行し性能を比較した．各アプリケーションの実行に要したクロック数を図 6.20〜6.22 に示す．キャッシュサイズは 32KByte，キャッシュラインサイズは 128Byte，TD の連想度は 2 とした．

TD のエントリ数が実行時間に与える影響を見ると，どのアプリケーションにおいても TD のエントリ数が少ない場合は実行時間が延びてしまっている．しかし，アプリケーションや PU 数による差はあるものの，512〜2048 程度のエントリ数のみでも，FULL_MAP と比較して劣らない性能を示すことがわかる．

エントリ数が極端に少ない場合において，実行時間が伸びているのは，Eviction プロトコルを採用していることが原因となっている．Eviction プロトコルでは，TD に新しい共有情報を登録する際，TD が一杯であると先に登録されている，古い共有情報に対応する PU 側のキャッシュラインを無効化して TD に新しい共有情報を登録するため，PU でまだ必要であるキャッシュラインが無効化されてしまうことがあるため性能の低下を招く．この無効化が，TD のエントリ数の少ない場合に多く発生し，エントリ数を増やすに従って減少するため，図 6.20〜 6.22 では，適当なサイズのエントリ数を TD に持たせた場合には FULL_MAP と同等の実行速度を達成している．

TD に適当なサイズのエントリ数を用いた場合でも，Eviction プロトコルを用いたことによる無効化は発生し得る．しかし，ある程度のエントリ数があり Eviction による無効化の発生の間隔が長くなれば，無効化されるキャッシュラインが PU にとって必要でないキャッシュラインとなっている可能性が高くなる．このため，ある程度のエントリ数を用いることによって，無効化が発生してもこれが性能に与える影響が少なくなり，FULL_MAP に劣らない性能を実現している．

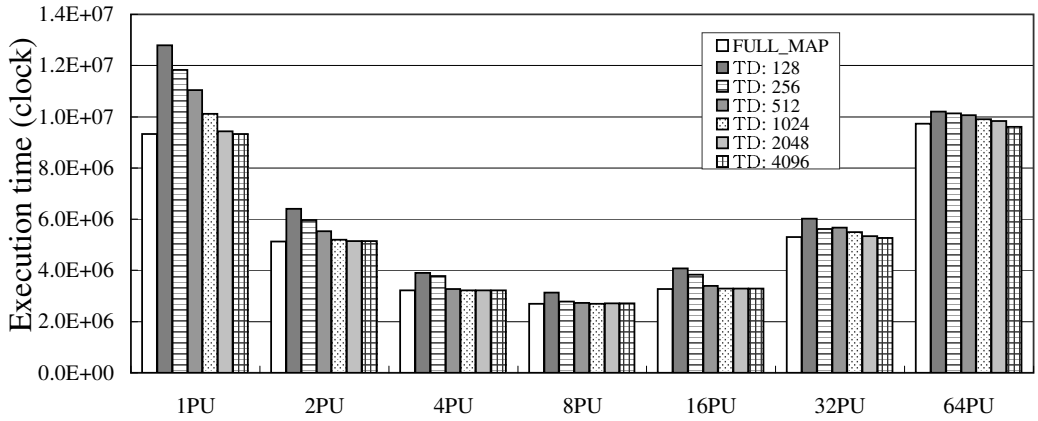

図 6.20: 実行時間 (RADIX，TD:2way)

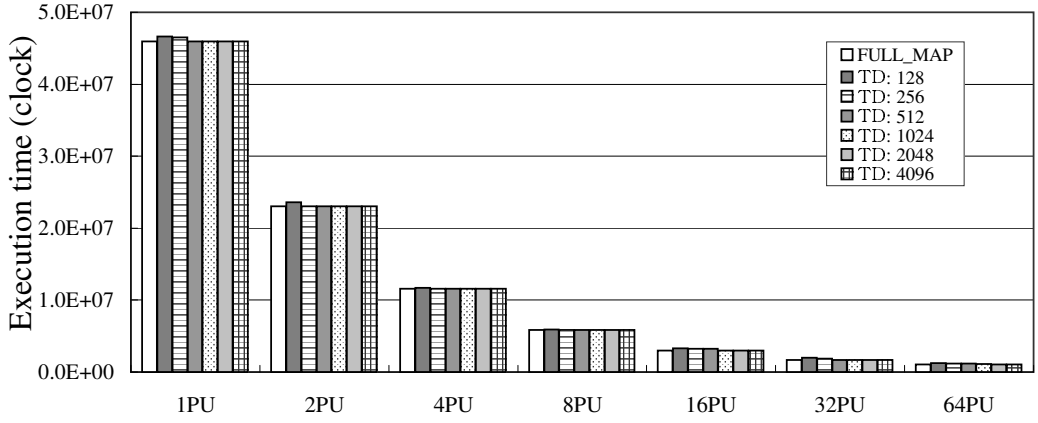

図 6.21: 実行時間 (FFT，TD:2way)

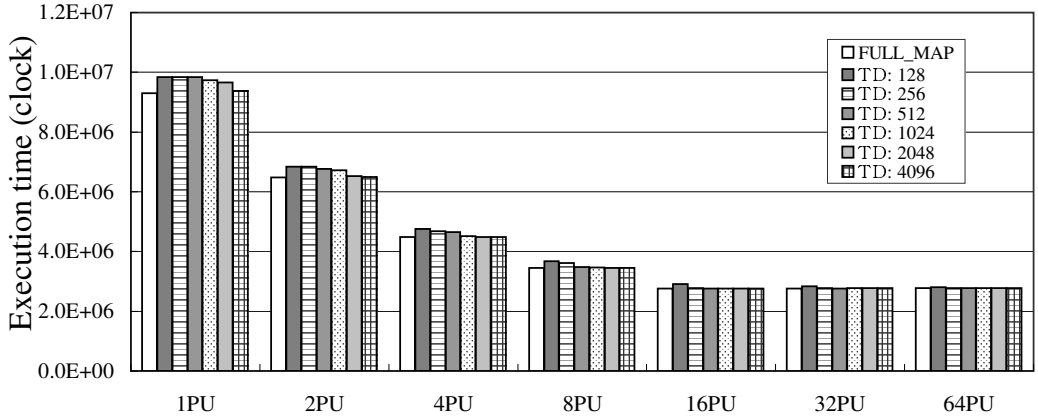

図 6.22: 実行時間 (LU，TD:2way)

TD の連想度による影響

64PU で TD の連想度を 1， 2， 4 として RADIX を実行した際にかかったクロック数を図 6.23 に示す．キャッシュサイズは 32KByte，キャッシュラインサイズは 128Byte とした．点線は FULL_MAP でかかったクロック数を示している．

2048 エントリ以上では，連想度にかかわらず，FULL_MAP とほぼ同等の実行時間で終了している．しかし，1024 エントリ以下のエントリ数が少ない場合には，連想度が 1 で実行時間が長く，連想度を高くすることで FULL_MAP と同様の実行時間を達成している．これより，エントリ数が少ない場合には，Eviction による無効化が実行時間に影響を与えやすいが，連想度を高くすることにより，この影響を緩和させることが可能であることがわかる．

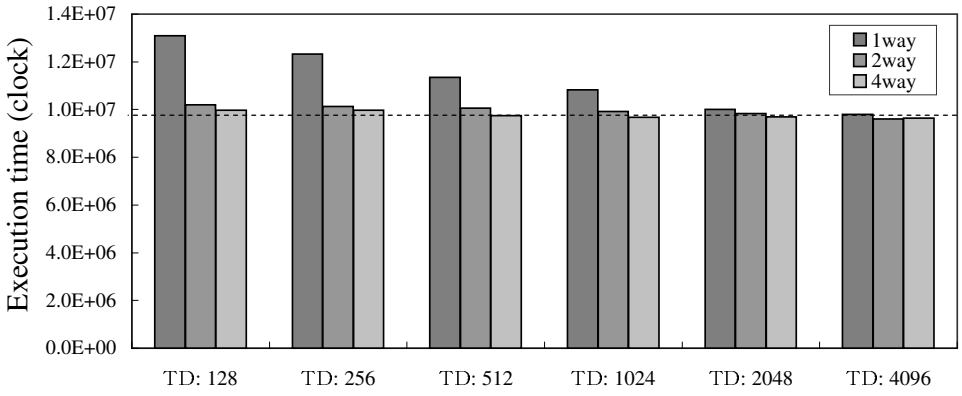

図 6.23: テンポラリディレクトリの連想度と実行時間 (64PU， RADIX)

キャッシュサイズとラインサイズによる影響

16PU でキャッシュサイズとキャッシュラインサイズを変化させ，RADIX を実行した際にかかったクロック数を図 6.24，図 6.25 に示す．TD のエントリ数は 2048，連想度は 1 とした．

図 6.24 では，キャッシュサイズが増えるに従って実行時間が短くなるものの，8KByte 以上では実行時間がそれ以上改善せず，ほぼ一定となった．また，図 6.25 では，キャッシュラインサイズを長くするに従って実行時間が短くなっているが，ラインサイズ 128〜256Byte で最も短くなり，512Byte 以上では実行時間が長くなっている．

これらの結果から，表 6.6 で示した評価条件では，キャッシュサイズが 32KByte，キャッシュラインサイズが 128Byte 程度にすることで，キャッシュの効果を十分に発揮できることがわかる．

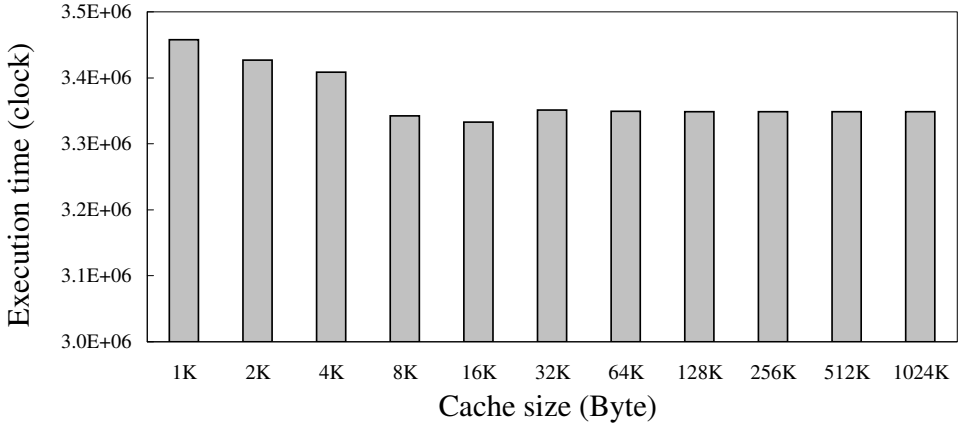

図 6.24: キャッシュサイズと実行時間 (16PU, RADIX，Cache line size:128Byte)

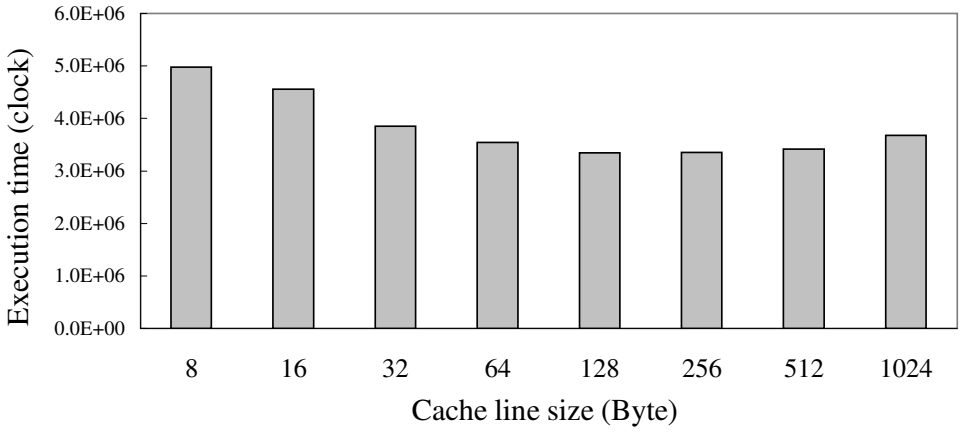

図 6.25: キャッシュラインサイズと実行時間 (16PU, RADIX，Cache size:32KByte)

6.3.5 キャッシュヒット率

図 6.26 は 64PU で RADIX を実行した時，PU の読み出し要求に対するキャッシュヒット率を表す．図中の Full は FULL_MAP のキャッシュヒット率を示す．RADIX と FFT では，TD のエントリが少ない 128〜1024 エントリではキャッシュヒット率が FULL_MAP に劣っているが，2048 エントリ以上ではほぼ同等のヒット率となっている．LU では 128 エントリ以上でほぼ同等のヒット率に達する．

このキャッシュヒット率は，MINDIC を用いた場合の実行時間に大きく影響する．図 6.20 〜 6.22 の 64PU での実行時間と比較すると， キャッシュヒット率の低い，RADIX と FFT の 128〜1024 エントリでは実行時間も長くなってしまっている．しかし，FULL_MAP と同等のキャッシュヒット率の RADIX と FFT の 2048 エントリ以上と，LU の 128 エントリ以上では，実行時間も FULL_MAP と同等であることがわかる．

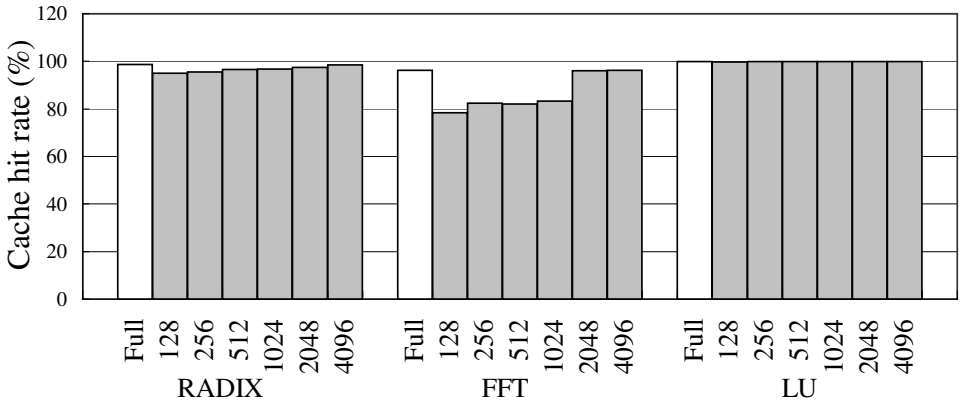

図 6.26: キャッシュヒット率 (64PU，TD:2way)

6.3.6 無効化パケットの発生数

TD が 128〜1024 エントリにおいてキャッシュヒット率が低下する原因を調べるため，無効化パケットの発生数および，発生原因を調べた．キャッシュサイズは 32KByte，キャッシュラインサイズは 128Byte とした．

図 6.27 は 64PU において TD エントリ数を変化させて各アプリケーションを実行した際に，PU が受けとった無効化パケット数を，FULL_MAP で発生した無効化パケット数の総数を 1 として表している．また，MINDIC を用いた時，PU が受け取る無効化パケットの発生原因は，次の 2 種類に分類することができる．

Write Hit

複数の PU が共有しているキャッシュラインに対する書き込み要求が TD にヒットした際に発生する．MM に近い上位のステージで発生した場合は，PU 側に逆送される途中のスイッチにおいても TD が参照され，ヒットした全ての PU へマルチキャストされる．

Eviction (Read Full)

PU による新規キャッシュラインの読み出し要求の際に，TD のエントリ不足により空きエントリを作る必要がある場合に，既存のエントリを追い出すことにより発生する．

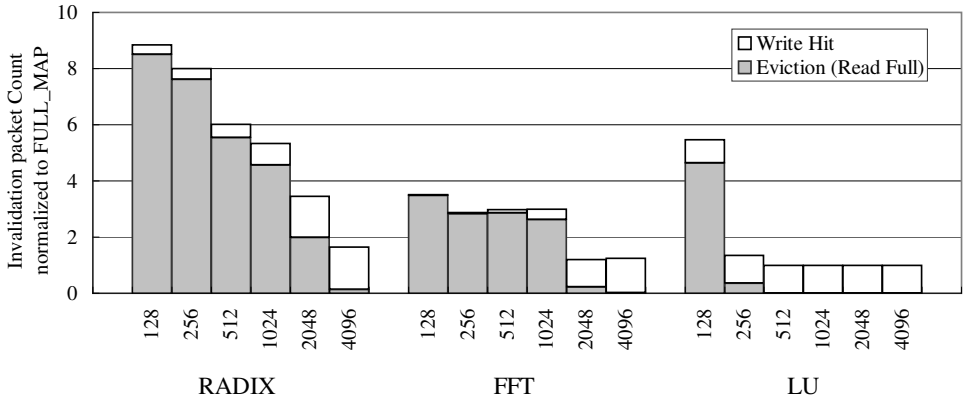

図 6.27: 無効化パケット数 (64PU，TD:2way)

図 6.27 で無効化パケットの総数を見ると，RADIX では 2048 エントリ以下で，FFT では 1024 エントリ以下で，LU では 128 で大量に発生しているが，それ以上のエントリ数になると急激に減り FULL_MAP とほぼ同等数に抑えられている．

また，無効化パケットの発生原因を見てみると，RADIX と FFT では，TD が 1024 エントリ以下では Eviction により発生する無効化パケットが大半を占めている．Eviction によるキャッシュラインの無効化は，MINDIC を用いない場合のキャッシュ制御においては発生しない余計なキャッシュラインの無効化であり，この無効化が多発するとアプリケーションの実行時間に悪影響をあたえてしまう．TD の エントリ数を十分に増やすことで，Eviction による無効化パケットを減らすことができ，RADIX では 4096 エントリで，FFT では 2048 エントリ以上で，LU では 256 エントリ以上で，Eviction によるキャッシュの不必要な無効化がほとんど発生しなくなる．

一方，Write Hit による無効化は，FULL_MAP で PU の書き込み要求発生時にキャッシュ制御のために発生する無効化と同様の働きをするものであり，LU の 512〜4096 エントリを見るとわかるように，エントリ数を十分に増やしても一定数以上減らすことは困難である．なお，MINDIC を用いた場合は，Eviction により書き込み要求前に無効化されることがあるため， Eviction による無効化パケットが多発しているエントリ数では，Write Hit に

よる無効化も少なくなる．

6.3.7 TD の Eviction による無効化の影響

頻繁に読み出されるキャッシュラインが，Eviction によりどの程度，無効化されてしまうか調べるため，Eviction により無効化されたキャッシュラインが，再度，同 PU に読み出された回数を図 6.28 に示した．キャッシュサイズは 32KByte，キャッシュラインサイズは 128Byte である．

再読み出し回数は，どのアプリケーションでもエントリ数が十分与えられると急激に減少しており，TD が 2048 エントリ程度あれば，Eviction によるキャッシュラインの無効化を原因とする性能低下が抑えられることがわかる．

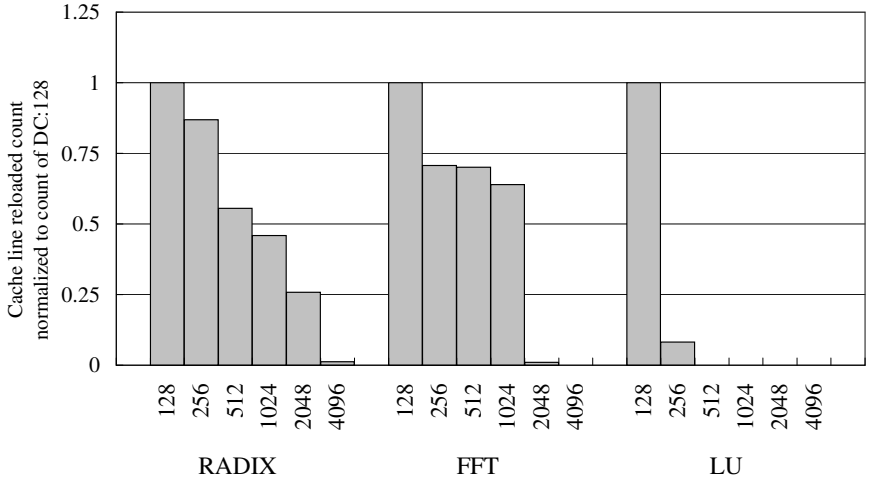

図 6.28: 再読み出し回数 (64PU，TD:2way)

6.4 ハードウェアコストの評価

6.4.1 ディレクトリ管理に必要なメモリ量の評価

ここでは，それぞれのエントリ数の場合に Eviction プロトコルで TD に必要とするメモリ量を検討する．フルマップ方式を用いた場合と MINDIC の Eviction プロトコルを用いた場合に，ディレクトリ管理に必要なメモリ量はそれぞれ次のようになる．

フルマップ方式を用いた場合

フルマップ方式では，m をスイッチのステージ数，n をスイッチサイズ (スイッチの MM から PU への経路の出力数) とすると， 1 ラインあたり $\sum_{k=1}^{m}(\frac{sharedmem}{linesize} \times n^k)$ のメモリ量が

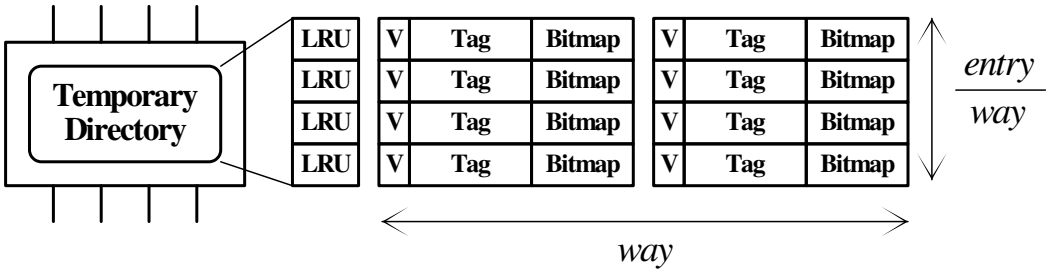

図 6.29: テンポラリディレクトリのハードウェア構造

必要となる．そのため，全体として必要なメモリ量 $DIR_{FULLMAP}$ は，次式で求められる．

$$DIR_{FULLMAP} = \sum_{k=1}^{m}(\frac{sharedmem}{linesize} \times n^k) \times \frac{sharedmem}{linesize} \text{ [bit]}$$

MINDIC の Eviction プロトコルを用いた場合

MINDIC の Eviction プロトコルを用いた場合，TD の構造は図 6.29 に示したようにビットマップ以外に，タグや LRU の制御ビットを保持する必要がある．また，各スイッチが同じメモリ量の TD を備えているので，1 スイッチあたりに必要なメモリ量にスイッチの数を掛けることで，全体として必要なメモリ量 DIR_{MINDIC} が，次式のように求められる．

$$DIR_{MINDIC} = (lru + (v + tag + td) \times way) \times \frac{entry}{way} \times switch \text{ [bit]}$$

$$tag = log_2 sharedmem - log_2 linesize - log_2 \frac{entry}{way} \text{ [bit]}$$

sharedmem	Memory Module の総メモリ量
linesize	キャッシュラインサイズ
switch	スイッチの総数
m	スイッチのステージ数
n	スイッチサイズ
tag	TD のタグのビット幅
td	TD のビット幅
v	TD の有効/無効を表すビット (1bit)
way	TD の連想度
entry	TD のエントリ数

16PU，16MM，共有メモリサイズ 256MByte，キャッシュラインサイズ 128Byte とし，8 つのスイッチから構成される MINDIC の Eviction プロトコルにおける連想度毎のディレクトリ管理に必要な全メモリ量を算出した値を表 6.7 に示した．また，比較のためにフルマップ方式で必要な全メモリ量を FULL_MAP として示した．

MINDIC で必要なメモリ量はスイッチ内に設けられる TD のメモリ量のみであり，5120KByte ものメモリ量が必要なフルマップ方式と比較して大幅に削減されていることがわかる．

表 6.7: ディレクトリ管理に必要なメモリ量

	1*way*	2*way*	4*way*
MINDIC (512*entry*)	8.5KByte	9.25KByte	10.5KByte
MINDIC (1024*entry*)	16KByte	17.5KByte	20KByte
MINDIC (2048*entry*)	30KByte	33KByte	38KByte
FULL_MAP	5120KByte		

6.4.2 スイッチのハードウェア規模の評価

MINDIC のスイッチを Verilog-HDL で記述し，SYNOPSYS 社の論理合成ツール Design compiler を用いて CMOS エンベデッドアレイ ASIC 0.18μm ライブラリを利用して論理合成を行い，動作速度とハードウェア量の評価を行った．

共有メモリサイズ 256MByte，キャッシュラインサイズ 32Byte，各スイッチ内の TD のエントリ数 2048，TD の連想度 2，4 で，TD に使用するメモリを除いたスイッチの論理合成結果は表 6.8 となった. MINDIC は，FULL_MAP 方式と比較して，6.32〜8.13 倍のゲート数を必要とするものの，CMOS エンベデッドアレイ ASIC 0.18μm においては，最大 7.3M ゲートの集積が可能であり，全体の 0.60〜0.77%程度であるから，十分実装可能なハードウェア量であるといえる．なお，TD の連想度 4 にすると連想度 2 に比べて 28.62%ゲート数が増加することがわかる．

表 6.8: 論理合成結果

スイッチ構成	Gate 数	最大動作周波数
MINDIC(TD:2way, 2048 エントリ)	43546	250MHz
MINDIC(TD:4way, 2048 エントリ)	56013	250MHz
FULL_MAP	6893	400MHz

また，MINDIC のスイッチは FULL_MAP のスイッチと比較すると複雑な構造であるため，最大動作周波数は低くなり，FULL_MAP の方が 1.6 倍高速に要求パケットを転送することができる．しかし，表 6.5 で示した FULL_MAP の読み出し要求 (Cache miss) のアクセスレイテンシ 80clock のうち，要求パケットに関するレイテンシは 12clock にすぎず，メモリからのデータ読み出しや PU へのデータ転送に関するレイテンシの影響が大きいため，動作周波数を考慮した，FULL_MAP の読み出し要求 (Cache miss) のアクセスレイテ

ンシは，MINDIC の 0.94 倍にすぎない．更に，図 6.26 に示したキャッシュヒット率が非常に高い状況での動作を考えれば，複雑な構造による MINDIC の動作周波数の低下は，実行時間への影響は非常に小さいものといえる．このことより，MINDIC のスイッチは，十分高速な動作周波数を実現しているといえる．

第7章　結論

本論文では，まず，RHBD 方式を用いて縮約した共有情報を MM 側に保持しすることで，ディレクトリ管理に必要なメモリ容量を減らし，共有情報の高速なアクセスを図る MINC について説明し，0.6μm ゲートアレイへ実装を行った．そして，実装された MINC chip の評価を行うために，MINC chip をキャッシュ制御機構に用いたマルチプロセッサ SNAIL−2 を実装し，評価用アプリケーションを動作させて評価を行った．その結果，キャッシュを搭載することでアプリケーションの処理速度を向上させることはできたものの，PU 数が増えたシステムにおいては，無効化要求パケットの再送回数が大幅に増加してしまう問題があることが明らかとなった．

そこで，別のキャッシュ制御方式である MINDIC を提案した．MINDIC は，MM にディレクトリを設けず，スイッチ内部に設けた小容量のテンポラリディレクトリ (TD) により共有情報を保持することで，キャッシュ制御を行う方式である．TD は少容量であるため，TD のエントリに既に共有情報が登録されている場合は，読み出し要求時に新たな共有情報を登録することができない状況が発生してしまう．そのような状況でもキャッシュ一致制御を正確に実行する必要があるので，それを実現する 3 種類の転送プロトコルを示した．

そして，トレースドリブンシミュレータにより，3 種類の転送プロトコルのシミュレーションを行い，Eviction プロトコルが最も効率の良い転送方法であることを示した．その後，Eviction プロトコルで動作する MINDIC の詳細な評価を行うために，最大 64 プロセッサで動作可能なクロックレベルシミュレータを実装し，評価用アプリケーションをシミュレータ上で動作させて評価を行った．

クロックレベルシミュレータによる評価では，フルマップ方式によるキャッシュ制御を用いたシステムと MINDIC を用いたシステムの性能を比較した結果，MINDIC は，連想度 4 であれば各スイッチに 512 エントリ，連想度 2 でも 2048 エントリ程度の TD を設けるだけで，フルマップ方式に劣らない性能を達成できることが確認された．

また，実行時間，キャッシュヒット率，無効化パケットの発生数をフルマップ方式と比較した結果，MINDIC では TD を連想度 4 であれば各スイッチに TD を 512 エントリ，連想度 2 でも TD を 2048 エントリ設けることで，フルマップ方式と同様の性能を達成できることがわかった．

更に，ハードウェア量の検討として，MINDIC とフルマップ方式でのキャッシュ制御に必要なメモリ容量を算出し，MINDIC はフルマップ方式と比較して大幅に少ないメモリで実現可能であることを示した．また，16 プロセッサ規模の MINDIC を，Verilog-HDL を用いて実装し，TD の連想度 2，2048 エントリの条件で，スイッチ 1 つ当たり 43546 ゲートと比較的低コストで実現できることを確認した．

このように，スイッチ接続型マルチプロセッサにおけるキャッシュ制御機構として，MINDIC は必要はハードウェアを削減しつつ，高速なディレクトリアクセスを実現できることにつ

いて述べたが，次のような今後の課題を有している．

一つは，Eviction プロトコルの MINDIC において，TD をより効率良く利用する追い出し方を検討することである．本論文中では読み出し要求パケットの転送時に共有情報を追い出す際，LRU を採用した方式を説明した．この方式では，TD 上の共有情報を登録する場所は，読み出し要求先のアドレスによって固定であるため，TD に空きがあるにも関わらず，共有情報の追い出しが発生することがあると考えられる．この問題を解決し，TD を高速にアクセス可能な構成で，更に効率良く TD を利用する制御方式を検討することが期待される．

また，クロックレベルシミュレーションにおいて，全ての構成が 1 つの LSI 内に実装される想定とし，評価プログラムが全て LSI 内の MM に存在する状況で評価を行ったが，MM を L2 cache として構成し，LSI 外部へのアクセスが発生する状態での評価についても検討する必要がある．

また，書き込み要求を処理する際，ライトスルー方式ではなく，ライトバック方式に対応することも課題となっている．ライトバック方式では，書き込み要求の発生時にメモリモジュール側へすぐに書き込み要求パケットが転送されないため，MINDIC の TD にキャッシュ制御に関する情報を登録することができない．そして，最新のデータが必ずメモリモジュールに存在する保証もないことから，キャッシュ一致制御を正確に行うために，新たな制御機構を導入する必要がある．

更に，オンチップマルチプロセッサとして LSI に実装したときの PU，MM，接続網等の LSI 上での配置方法，オンチップマルチプロセッサを複数接続するようなシステムへの対応なども検討の余地が残されている．

2000 年以降，商用のオンチップマルチプロセッサの普及が始まり，今後，オンチップマルチプロセッサに対応するアプリケーションが増加するもとの考えられ，1 チップに要求されるプロセッサ数も大幅に増加していくものと予想される．それに伴い，PU と MM の接続網がバス接続型からスイッチ接続型へと変化しつつある流れの中で，必要なメモリ容量やハードウェア規模を考慮したスイッチ接続型マルチプロセッサのキャッシュ制御方式は，益々重要なものとなっていくであろう．

謝 辞

本研究の機会を与えて下さり，終始御指導下さった慶應義塾大学理工学部情報工学科 天野 英晴 教授に深く感謝いたします．

また，本研究をまとめるにあたり，貴重な御助言を頂きました，慶應義塾大学理工学部情報工学科 笹瀬 巌教授，寺岡 文男教授，山崎 信行助教授に深く感謝致します．

本研究を始めるにあたり，亀井 貴之 氏 (現在 株式会社東芝) に様々な御助言を頂きました．特に，MINC chip の実装に関しては多大なる御協力を頂きました．深く感謝いたします．

SNAIL グループの星野 智則 氏 (現在 日本電気株式会社)，白石 大介 氏 (現在 キヤノン株式会社)，茂野 真義氏には，実機 SNAIL–2 を実装するにあたり，共に取り組んでいただき，デバッグ等に尽力して頂きました．深く感謝いたします．

東京工科大学コンピュータサイエンス学部講師 塙 敏博 氏，SNAIL グループの田辺 靖貴 氏，薬袋 俊也 氏，住吉 正人 氏には，MINDIC の提案および評価に関して様々な御協力および御助言を頂きました．深く感謝いたします。

また，本研究に御協力頂いた，天野研究室の皆様には大変お世話になりました．感謝いたします．

最後になりましたが，これまで暖かく見守ってくれた両親，さまざまな面から支えてくれた緑川 弥生，元気づけてくれた緑川 結に心より感謝いたします．

2005 年夏

参考文献

［AGGD01］ M.E. Acacio, J. Gonzalez, J.M. Garcia, and Jose Duato. A new scalable directory architecture for large-scale multiprocessors. In *HPCA '01: Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (HPCA'01)*, p. 97. IEEE Computer Society, 2001.

［ASHH88］ A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz. An evaluation of directory schemes for cache coherence. In *Proc. of 15th ISCA*, pp. 280–289, 1988.

［BNA94］ L. N. Bhuyan, A. K. Nanda, and T. Askar. Performance and reliability of the multistage bus network. In *Proc. ICPP*, pp. 26–33, 1994.

［CKA91］ D. Chaiken, J. Kubiatowicz, and A. Agarwal. Limitless directories: A scalable cache coherence scheme. In *In ASPLOS IV*, pp. 224–234, Apr. 1991.

［CV88］ H. Cheong and A.V. Veidenbaum. A cache coherence scheme with fast-selective invalidation. In *Proc. of 15th ISCA*, pp. 299–307, 1988.

［E+85］ J. Edler, et al. Issues related to mimd shared-memory computers the nyu ultracomputer approach. In *Proc. of 12th ISCA*, pp. 126–135, 1985.

［GWM90］ A. Gupta, W. Weber, and T. Mowry. Reducing memory and traffic requirements for scalable directory-based cache coherence schemes. In *1990 International Conference on Parallel Processing*, Vol. I, pp. 312–321, St. Charles, Ill., 1990.

［HKN+00］ 細見岳生, 加納健, 中村真章, 広瀬哲也, 中田登志之. 並列計算機 Cenju-4 の分散共有メモリ機構. 情報処理学会論文誌, Vol. 41, No. 5, pp. 1400–1409, 2000.

［HYNA96］ T. Hanawa, H. Yasukawa, K. Nishimura, and H. Amano. Minc: Multistage interconnection network with cache control mechanism. In *Proc. PDCS*, pp. 310–317, 1996.

［IBN00］ R. Iyer, L. N. Bhuyan, and A. Nanda. Using switch directories to speed up cache-to-cache transfers in ccnuma multiprocessors. *Proc. of the 14th Int'l Parallel and Distributed Processing Symposium (IPDPS'00)*, pp. 721–728, 2000.

［JLGS90］ 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.

［Kam97］ 亀井貴之. 多段結合網に基づくキャッシュコヒーレントネットワークの設計と実装. 慶應義塾大学大学院 理工学研究科 修士論文, 1997.

［KYN+95］ 工藤知宏, 好村公一, 福嶋泰仁, 西村克信, 楊愚魯, 天野英晴. 超並列計算機 JUMP-1 のクラスタ間結合網 RDT における階層マルチキャストによるメモリコヒーレンシ維持手法. 並列処理シンポジウム JSPP 論文集, pp. 257–264, May. 1995.

［MBLZ89］ H.E. Mizrahi, J.L. Baer, E.D. Lazowska, and J. Zahorjan. Introducing memory into the switch elements of multiprocesssor interconnection networks. In *Proc. of 16th ISCA*, pp. 158–166, 1989.

［MH94］ S.S. Mukherjee and M.D. Hill. An evaluation of directory protocols for medium-scale shared-memory multiprocessors. In *ICS '94: Proceedings of the 8th international conference on Supercomputing*, pp. 64–74. ACM Press, 1994.

［MSA05］ 緑川隆, 住吉正人, 田辺靖貴, 天野英晴. テンポラリディレクトリを持つキャッシュ制御用多段結合網 MINDIC の設計と評価. 電子情報通信学会論文誌, Oct. 2005.

［NB93］ 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.

［NKA98］ 西村克信, 工藤知宏, 天野英晴. Pruning Cache を用いた分散共有メモリのディレクトリ構成法. 情報処理学会論文誌, Vol. 39, No. 6, pp. 1644–1654, 1998.

［RL00］ R.Iyer and L.Bhuyan. Design and evaluation of a switch cache architecture for cc-numa multiprocessors. *IEEE Trans. on Comput*, Vol. 49, No. 8, pp. 779–797, 2000.

［Sas95］ 笹原正司. SSS アーキテクチャに基づくマルチステージネットワーク PBSF の実装. 慶應義塾大学大学院 理工学研究科 修士論文, 1995.

［SME+95］ S.C.Woo, M.Ohara, E.Torrie, J.P.Singh, and A.Gupta. The splash-2 programs: Characterization and methodological considerations. In *Proceedinfs of the 22nd International Symposium on Computer Architecture*, pp. 24–36, Jun. 1995.

［Tan03］ 田辺靖貴. Temporary Directory を持つアドレス転送用ネットワーク ANT の性能評価環境の設計と実装. 慶應義塾大学大学院 理工学研究科 修士論文, 2003.

［TDM+02］ T.Midorikawa, D.Shiraishi, M.Shigeno, Y.Tanabe, T.Hanawa, and H.Amano. Snail-2: a sss-min connected multiprocessor with cache coherent mechanism. In *Proc. of Parallel and Distributed Computing, Applications and Technologies*, pp. 17–24, 2002.

［TDM+05］ T.Midorikawa, D.Shiraishi, M.Shigeno, Y.Tanabe, T.Hanawa, and H.Amano. The performance of snail-2(a sss-min connected multiprocessor with cache coherent mechanism). *Parallel Computing*, Vol. 31, pp. 352–370, Mar. 2005.

［THY94］ T.Hanawa, H.Amano, and Y.fujikawa. Multistage interconnection networks with multiple outlets. In *Proc. ICPP*, Vol. 1, pp. 1–8, Aug. 1994.

［TMS+03］ 田辺靖貴, 緑川隆, 白石大介, 茂野真義, 塙敏博, 天野英晴. 命令レベルシミュレーションによる sss 型 min の評価. 情報処理学会論文誌 コンピューティングシステム第 3 号 (ACS), No. 44, pp. 169–179, 2003.

［TTTH98］ T.Midorikawa, T.Kamei, T.Hanawa, and H.Amano. The minc chip: Multistage interconnection network with cache control mechanism chip. In *Proc. on ASICON*, pp. 249–252, 1998.

［Vei86］ A.V. Veidenbaum. A compiler-assisted cache coherence solution for multiprocessors. In *Proc. ICPP*, pp. 1026–1036, 1986.

［WA01］ 若林正樹, 天野英晴. 並列計算機シミュレータの構築支援環境. 電子情報通信学会論文誌, 2001.

［Yas96］ 安川英樹. キャッシュ制御機構内蔵型多段結合網:MINC. 慶應義塾大学大学院理工学研究科 修士論文, 1996.

論文目録

本研究に関する論文

【公刊論文】

1. 緑川 隆, 住吉 正人, 田辺 靖貴, 天野 英晴, “テンポラリディレクトリを持つキャッシュ制御用多段結合網 MINDIC の設計と評価”, 電子情報通信学会論文誌, 2005 年 10 月 (採録決定).

2. Takashi Midorikawa, Daisuke Shiraishi, Masayoshi Shigeno, Yasuki Tanabe, Toshihiro Hanawa, Hideharu Amano, ” The performance of SNAIL-2 (a SSS-MIN connected multiprocessor with cache coherent mechanism)”, Parallel Computing, Vol.31, pp.352-370, Mar.2005.

3. 田辺 靖貴, 緑川 隆, 白石 大介, 茂野 真義, 塙 敏博, 天野 英晴,”命令レベルシミュレーションによる SSS 型 MIN の評価”, 情報処理学会論文誌 コンピューティングシステム第 3 号 (ACS), No.44, pp.169-179, 2003.

【国際会議】

4. Masato Sumiyoshi, Yasuki Tanabe, Takashi Midorikawa, Hideharu Amano, “Design and Evaluation of a Switch Architecture for Multistage Interconnection Network with Temporary Directory”, International Conference on Parallel and Distributed Computing Systems(PDCS 2004), Sep. 2004.

5. Masato Sumiyoshi, Takashi Midorikawa, Hideharu Amano, ”Design and Implementation of Switching Fabrics for Multistage Interconnection Network with Directory Cache” , COOL Chips VII, pp.75, Apr.2004.

6. Yasuki Tanabe, Takashi Midorikawa, Daisuke Shiraishi, Masayoshi Shigeno, Toshihiro Hanawa, and Hideharu Amano, ”Performance Evaluation of 3-Dimensional MIN with Cache Consistency Mainenance Mechanism”, Proc. of The 2003 International Conference on Parallel and Distributed Processing Techniques and Applications(PDPTA’03), pp.1155-1161, Jun. 2003.

7. Takashi Midorikawa, Daisuke Shiraishi, Masayoshi Shigeno, Yasuki Tanabe, Toshihiro Hanawa, Hideharu Amano, ”SNAIL-2: a SSS-MIN connected multiprocessor with cache

coherent mechanism", Proc. of the 3rd International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT2002), pp.17-24, Sept.2002.

8. Takashi Midorikawa, Takayuki Kamei, Toshihiro Hanawa, Hideharu Amano,"The MINC Chip: Multistage Interconnection Network with Cache control mechanism chip", Proc. of the 3rd International Conference on ASIC (ASICON '98), pp.249-252, Oct.1998.

9. Takashi Midorikawa, Takayuki Kamei, Toshihiro Hanawa, Hideharu Amano,"The MINC (Multistage Interconnection Network with Cache control mechanism)", Proc. of The Asia and South Pacific Design Automation(ASP-DAC'97) , pp.337-338, Feb.1998.

【研究会ほか】

10. 住吉 正人, 緑川 隆, 茂野 真義, 田辺 靖貴, 薬袋 俊也, 天野 英晴, "一時的にディレクトリを保持する MINDIC スイッチの設計と評価", 情報処理学会研究報告 2004-EVA-8, pp.19-24, 2004 年 3 月.

11. 住吉 正人, 緑川 隆, 田辺 靖貴, 天野 英晴," キャッシュ制御用多段結合網 MINDIC の設計と評価", 情報処理学会研究報告 ARC-9 (SWOPP2004), pp.1-6, 2003 年 3 月.

12. 緑川 隆, 田辺 靖貴, 天野 英晴," ディレクトリキャッシュスイッチを持つキャッシュ制御用多段結合網の検討, 電子情報通信学会技術研究報告 [コンピュータシステム] CPSY2003-14(SWOPP2003), pp.49-54, 2003 年 8 月.

13. 茂野 真義, 緑川 隆, 白石 大介, 田辺 靖貴, 天野 英晴," キャッシュ制御機構を持つスイッチ結合型並列計算機 SNAIL-2 の評価", 情報処理学会研究報告 2003-ARC03-152(HOKKE2003), pp.103-108, 2003 年 3 月.

14. 白石 大介, 緑川 隆, 茂野 真義, 田辺靖貴, 金森 勇壮, 天野 英晴,"キャッシュ制御用マルチキャストネットワーク MINC チップを用いたスイッチ結合型並列計算機 SNAIL-2 の評価", VLSI 設計技術研究会, 2002 年.

15. 田辺 靖貴, 緑川 隆, 白石 大介, 茂野 真義, 金森 勇壮, 塙 敏博, 天野 英晴," 多重出力可能な MIN の命令レベルシミュレータによる評価", 電子情報通信学会技術研究報告 [システム評価] (SWoPP'2003), pp.19-24, 2002 年 8 月

16. 白石 大介, 星野 智則, 緑川 隆, 金森勇壮, 天野 英晴," スイッチ結合型マルチプロセッサ SNAIL-2 のデータ転送用ネットワーク PBSF の評価",VLSI 設計技術研究会, 2001 年.

17. 星野 智則, 緑川 隆, 金森 勇壮, 白石 大介, 天野 英晴," キャッシュ制御機構を持つスイッチ結合型マルチプロセッサ SNAIL–2 の評価", 電子情報通信学会技術研究報告 [コンピュータシステム] CPSY2000-41, pp.9-16, 2000 年.

18. 星野 智則, 緑川 隆, 天野 英晴,"キャッシュ制御機構を持つスイッチ結合型マルチプロセッサ SNAIL–2 の実装", 電子情報通信学会技術研究報告 [コンピュータシステム] CPSY99-70, pp.63-70, 1999 年.

その他の論文

【研究会ほか】

19. 薬袋 俊也, 緑川 隆, 田辺 靖貴, 茂野 真義, 天野 英晴, ”オンチップマルチプロセッサ向け内部接続網の検討”, 情報処理学会研究報告 2003-ARC-153, pp.1-6, 2003 年 5 月.

20. 米田 卓司, 若林 正樹, 緑川 隆, 西村 克信, 天野 英晴, “並列計算機のための相互結合網シミュレータ SPIDER”, 電子情報通信学会技術研究報告 [コンピュータシステム] CPSY97-109, pp.59-66, 1998 年 1 月