2004－ARC－159　(24)
2004／7／31

キャッシュ制御用多段結合網 MINDIC の設計と評価

住吉正人[†]　田辺靖貴[†]
緑川隆[†]　天野英晴[†]

MIN (Multistage Interconnection Network) を用いたマルチプロセッサにおける効率の良いキャッシュ制御手法 MINDIC を提案する．MINDIC は，MIN を構成する各スイッチ内部にのみ小容量のディレクトリを保持することにより低レイテンシなキャッシュ制御が可能であり，ハードウェアコスト面でも有利である．詳細なシミュレーション環境を構築しアプリケーションを動作させ性能評価を行なったところ，MINDIC では，小容量のメモリを用いることで完全なディレクトリを用いた場合と同等な性能を達成できることがわかった．

Design and Evaluation of Multistage Interconnection Network with Cache Coherence Mechanism

Masato Sumiyoshi,[†] Yasuki Tanabe,[†] Takashi Midorikawa[†]
and Hideharu Amano[†]

We proposed MINDIC (MIN with DIrectory Cache switch), a novel MIN structure that consists of switches with small temporary directory. Building temporary directory dynamically in each switching element, we can maintain cache consistency with low latency and low memory cost. We built precise clock level simulation environment and evaluated MINDIC by executing parallel benchmark programs. As a result, MINDIC achieved as high performance as a system with large full directory in shared memory. Synthesis report using 0.18μm CMOS process shows that the hardware cost is small enough for implementation

1. はじめに

中規模並列計算機向けの接続網として，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) が検討されている．MIN では各プロセッシングユニット (PU) がお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に，バス結合型の接続網で用いられているスヌープ方式によるキャッシュ制御が行えない．このため，MIN におけるキャッシュ制御はメモリモジュール (MM) 側にディレクトリを設ける方法が一般的である．しかし，アドレス空間のサイズに応じてディレクトリ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情報が参照されるために，キャッシュ制御の際のレイテンシが増加する．このため，キャッシュの一部を MIN のスイッチングエレメント中に設ける方法[1)2)]，ディレクトリを設ける方法[3)4)] 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利用量やアクセス時間の増大，スイッチの複雑化を招き，実機で利用されるに至ってはいない．

これに対し，本研究室では，MIN を構成する各スイッチングエレメント内に小容量のディレクトリキャッシュ(DC) を設ける MINDIC[5)] (MIN with DIrectory Cache switch) を提案した．各スイッチングエレメント内の DC は，比較的最近アクセスされたキャッシュラインのディレクトリ情報をキャッシュし，これを利用し，キャッシュラインの無効化要求を各ノードに発行する．MINDIC は，スイッチ上でディレクトリ情報が管理できなくなったキャッシュラインを各ノードから無効化する点で従来手法と大きな違いがある．これによってメモリモジュール側でのキャッシュ管理の必要性を無くしている．

これまで，本研究室ではこの MINDIC について，ハードウェアコストの見積り，トレースドリブンのシミュレータにより評価を行なってきた．しかし，これらの評価では，実アプリケーション実行時の MINDIC の性能評価を行うことができなかった．そこで MINDIC のクロックレベルのシミュレータを構築した．本稿では，このシミュレータによる MINDIC の評価結果を報告する．

以降，第 2 節で MINDIC の概観と動作，第 3 節でスイッチの構成，第 4 節で構築したシミュレータについて述べ，第 5 節において，シミュレータを用いた評価結果について述べる．第 6 節では，シミュレータによる評価結果を踏まえハードウェアコストの検討を行う．

2. MINDIC

2.1 MINDIC の概観

図 1 に，MINDIC を用いたシステムの構成の一例を示

† 慶應義塾大学 理工学部
Amano Laboratory, Faculty of Information and Computer Science, Department for Science and Engineering, Keio University

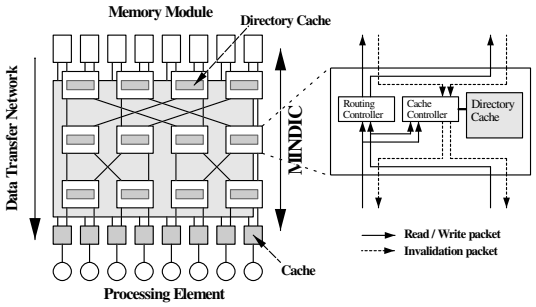

図 1 MINDIC の構造

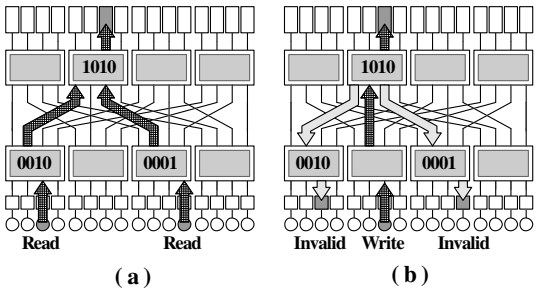

図 2 MINDIC の基本動作

す．各ノードは，メモリアクセス要求および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の結合網の，分離された 2 つの結合網を介して接続される．MINDIC においてキャッシュ制御を行うことにより，各 PU は，コンシステンシが維持された共有データをキャッシュする事ができる．

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．MINDIC を用いたキャッシュコンシステンシ管理の対象とするのは共有データ領域に限定し，複数 PU で共有することのない命令データおよびローカルデータはキャッシュコンシステンシをとらずに管理する．

MINDIC の最大の特徴は，共有メモリを構成する MM にはディレクトリを設けずに，MINDIC のスイッチ内に数 K エントリのディレクトリキャッシュ(DC) を設け，キャッシュラインの共有情報を短期的に管理する点である．DC は小容量のメモリを用いるため，共有情報の登録ができるキャッシュラインの数が制限される．このため，DC はキャッシュラインの共有情報を一時的に保持するものとなっている．

2.2 基 本 動 作

MINDIC において，各スイッチ内の DC は図 2 に示すように，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示すビットマップを保持する．一般のキャッシュ同様 DC は転送アドレスの一部をインデックスとして参照される．

PU による MM の読み出し要求が発生した際，PU に接続されたキャッシュにデータが存在しない場合，読み出し要求パケットが MINDIC を介して MM へ転送される．この際，要求パケットが通過する各スイッチ内の DC に，読み出しデータに関する共有情報が登録される (図 2(a))．読み出し要求により MM から読み出されたデータは，データ転送用ネットワークを介して PU へ転送される．

PU による 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 DC が参照される．DC のエントリにヒットすれば，図 2(b) に示すように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パケットが生成される．生成された無効化パケットはキャッシュラインを保持する全ての PU へマルチキャストされ，該当するラインを無効化する．このようにしてキャッシュの一致制御を行う．

スイッチの基本動作の詳細は以下の通りである．MINDIC のスイッチが取り扱うのは読み出し要求，書き込み要求，無効化要求の三種類である．

- 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 DC を参照する．DC 内の共有情報の参照でミスした場合，共有情報を登録可能であれば，スイッチの各入力リンクに対して，読み出し要求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共有情報として DC に登録する．一方，DC にヒットした場合，パケットの入力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 し共有情報を更新する．
- 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スイッチ上の DC を参照する．DC にヒットした場合，DC 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し，MM に対する書き込みが発生した共有データをキャッシュに保持する PU のエントリを無効化する．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，無効化パケットの対象のキャッシュラインのアドレスのインデックスにより DC を参照する．DC にヒットした場合，DC から得られたキャッシュラインの共有情報に従って，キャッシュラインを共有する PU へ向けて，無効化パケットを PU 側のステージにマルチキャストする．この際，このスイッチ内の DC のエントリは無効化する．

PU のキャッシュは，無効化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，特殊な操作の必要はない．同様に，MM はディレクトリを持たず，キャッシュ制御に関する特別な操作は何もしない．

2.3 転送プロトコル

MINDIC は前記した基本動作を行うが，DC には容量不足やコンフリクトにより共有情報が登録できない場合が

ある．このような場合キャッシュの一致制御が正しく行なえない．この問題を解決するために 3 つの転送プロトコルが提案された．

トレースドリブンシミュレーションによる評価[6] から，3 つのプロトコルのうち，Eviction プロトコルを用いた場合にネットワークに発生する無効化パケットによる混雑が最も少なくなり有効なキャッシュ制御プロトコルであることがわかった．本稿では Eviction プロトコルを用いてアプリケーションを実行した際の性能評価を行う．

Eviction プロトコル　PU による MM への読み出し要求がスイッチに入力された際，スイッチ内の DC において共有情報を格納するエントリに空きがない場合，Eviction プロトコルでは以下のように動作する．

(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ッ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

これにより，PU 側でキャッシュされているラインのディレクトリ情報は，必ず DC に登録された状態に保つことができ，キャッシュ一致制御を正しく行うことができる．

3. MINDIC スイッチの構成

MINDIC の各スイッチは下位ステージ (PU 側) から上位ステージ (MM 側) 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．図 3 に，設計したスイッチの内部構成を示す．

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 3 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 DC 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．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．

それぞれの要求に対する Eviction プロトコルにおける DC ユニットの動作を以下に示す．

- 読み出し要求
  読み出すキャッシュラインの共有情報を DC へ登録する．DC 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(Least Recently Used) 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，キャッシュの一貫を保つために，破棄する共有情報に従って該当す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の無効化パケットを生成する．こうして発生した無効化パケットには，該当ラインを保持する複数の PU へのマルチキャストのためのビットマップが付加されている．そして，無効化パケット転送部に設けられた Invalidation Packet Buffer に取り込まれる．
- 書き込み要求
  DC からの共有情報をもとに，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の無効化パケットを生成する．生成された無効化パケットには，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，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 の方向へ逆送されてきた無効化要求パケットは，上位ステージとのリンク毎に設けられた DC を参照する．DC にヒットすれば DC 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．ミスした場合，無効化要求は消滅する．ここで参照する DC は，リクエスト転送部の対応する上位ステージへの出力リンクと共有されている．

このようにして Invalidation Packet Buffer に取り込まれた無効化要求パケットは，Output Controller により下位ステージへマルチキャストされる．

DC 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．同アドレスに対する参照が DC メモリの 2 つのポートで同時に発生する場合はアービタが働くことにより，排他的に DC へのアクセスが行なわれる．

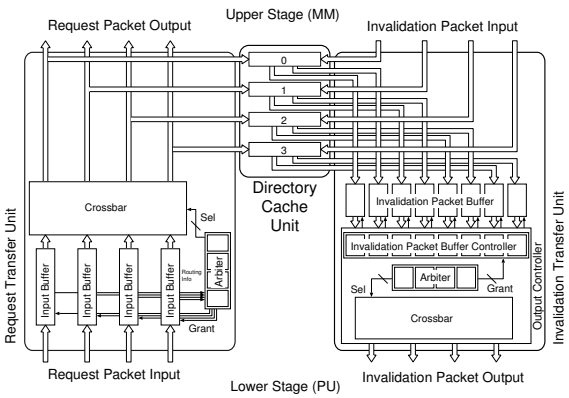

図 3　MINDIC スイッチの構成

表 1　キャッシュと DC のパラメータ

Cache	
Cache Size	32768 Byte/PU
Associative	2 way
Line size	128 Byte
Protocol	Write Through

DC	
Entry	128 ~ 8192 /SW
Associative	1, 2, 4 way

4. シミュレーション環境

MINDIC の評価のために開発したシミュレータは，本研究室で開発された 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[7] 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．

シミュレーションでは MIPS 社の R3081 をプロセッサモデルとして想定しているが，ネットワークに負荷を与えるために，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 倍に設定してシミュレーションを行った．

評価には，2 段のステージ, 各ステージ 4 スイッチで合計 8 つのスイッチを有した MINDIC を用いた，最大 16PU 構成のシステムを想定した．MIN 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．

命令及びローカルデータは各ノードのメモリに配置し，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．キャッシュや DC に関するパラメータは表 1 のようになっている．

4.1 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 (Stanford ParalleL Applications for SHard memory-2)[8] から LU, FFT, RADIX の 3 つを選択し，PU 数 1, 2, 4, 8, 16 で実行した．

- RADIX – キーの桁ごとに処理する整列アルゴリズム
  データ数: 65536
- 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ェ変換で，PU 間の通信が最小になるように最適化されている．
  データ数: 2^{12}
- LU – 密行列を下三角行列と上三角行列に分解する．
  行列のサイズ: 128×128

以降では，シミュレーションの各結果は，正確な評価のため，並列化されていない初期化処理を含めていない．

5. シミュレーションによる評価

MINDIC を用いずに，MM 上に full-mapped なディレクトリを保持しキャッシュ管理を行うアーキテクチャを用いた場合 (without_MINDIC) と，MINDIC を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する．

5.1 実 行 時 間

RADIX, FFT, LU の実行にかかったクロック数をそれぞれ図 4, 5, 6 に示す．DC の連想度は 2 としている．

5.1.1 DC のエントリ数による影響

図 4〜6 において，DC のエントリ数が性能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，MINDIC を用いた場合，DC のエントリ数が少ない場合は without_MINDIC と比較すると実行時間が延びてしまっているが，各スイッチ毎に DC のエントリ数を十分持たせることにより without_MINDIC と同等の実行速度を達成できる．

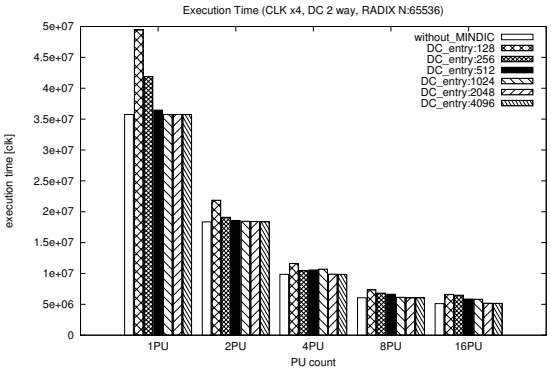

図 4 実行時間 (RADIX N:65536)

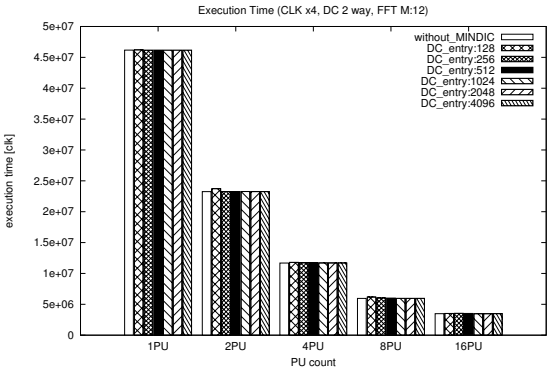

図 5 実行時間 (FFT M:12)

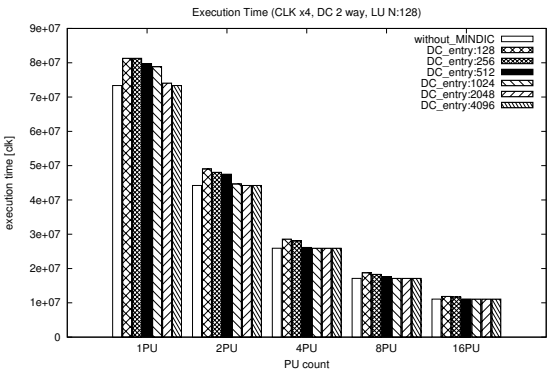

図 6 実行時間 (LU N:128)

16PU では，DC を 512 エントリ程度設ければ各アプリケーションの実行時間は without_MINDIC と比べて大きな差はないことがわかる．

5.1.2 DC の連想度による影響

16PU で DC 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を図 7 に示す．DC のエントリが少ない 1024 までは連想度が高い程実行速度が速くなっている．

これにより，エントリ数が少ない状況においては，連想度が高い方がより効率的に DC を利用できていることがわかる．しかし，2048 エントリ以上ではどの連想度でもほぼ同等の性能となる．512 エントリの場合，最も性能の悪かった RADIX では without_MINDIC と比較して連想度 2 で 87.46%，連想度 4 で 97.85%の実行速度となったが，連想度 2 でも FFT と LU では without_MINDIC と比較してそれぞれ 98.72%, 99.80%の実行速度を達成した．

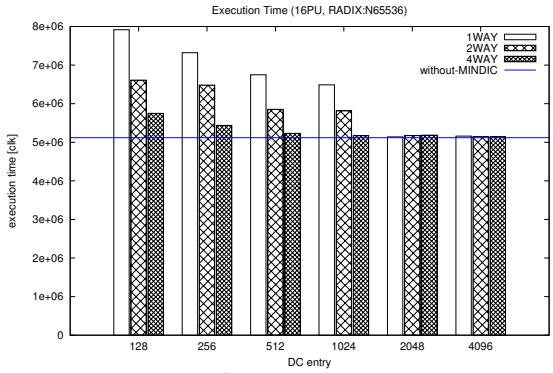

図 7 DC の連想度と実行時間 (16PU, RADIX)

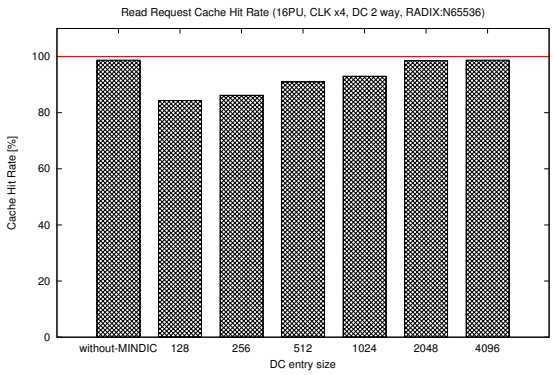

図 8 キャッシュヒット率 (16PU, DC:2way, RADIX)

5.2 キャッシュヒット率

図 8 は 16PU で RADIX を実行した際の Read 要求に対する共有データのキャッシュヒット率を表す．RADIX では，DC 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 without_MINDIC 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．このキャッシュヒット率の向上が MINDIC を用いた場合の実行時間の改善の大きな要因となっていると考えられる．いずれのアプリケーションにおいても，DC を 2048 エントリ設けることにより without_MINDIC と同等のキャッシュヒット率が得られた．

5.3 無効化パケットの発生数

128〜1024 エントリにおけるキャッシュヒット率低下の原因を調べるため，無効化パケットの発生数および，発生原因を調べた．

図 9 は 16PU において DC エントリ数を変化させて RADIX を実行した際 PU が受けとった無効化パケット数を表している．無効化パケットは，DC が 1024 エントリ以下の場合は大量に発生しているが，2048 エントリになると急激に減り，4096 エントリ以上では without_MINDIC とほぼ同等数に抑えられている．

さらに，図 9 は無効化パケットの総数を発生原因別に分類している．PU が受け取る無効化パケットは発生原因別に以下の 3 種類がある．

- Write Hit
  複数 PU が共有しているラインに対する書き込み要求が DC にヒットした際に発生するもの
- Invalidation Request

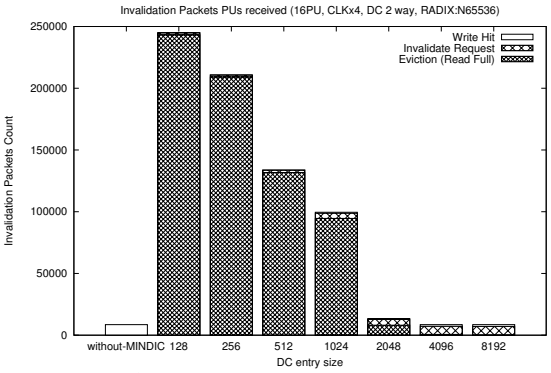

図 9 無効化パケット数 (16PU, DC:2way, RADIX)

  Write Hit により発生した無効化要求パケットがメモリモジュールに近い上位のステージから逆送され，これらの無効化要求パケットが PU に最も近いステージのスイッチにおいて DC にヒットして PU へマルチキャストされたもの
- Eviction (Read Full)
  PU による新規ラインの読み出し要求の際に，DC のエントリ不足により空きエントリを作る必要があり，既存の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

図 9 を見ると，DC が 1024 エントリ以下では，Eviction による無効化がその大半を占めている．Eviction による無効化が頻発すると，本来のキャッシュ一致制御においては必要でない余計なキャッシュラインの無効化が多発することとなり，アプリケーションの実行性能を落としてしまう．2048 エントリ以上では，書き込み要求が原因の無効化 (Write Hit + Invalidation Request) の割合が増加しており，8192 エントリ設けると，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．

5.4 DC の Eviction による影響

頻繁に Read 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っているか調べるため，図 10 に，RADIX を実行した際に，DC の Eviction により発生した無効化パケット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の合計を示した．DC の連想度は 2 とした．

DC エントリを 128 から増加させていった場合，再度読み出しされる回数は 2048 エントリになると急激に減少しており，Eviction によるキャッシュラインの無効化が招く性能低下を抑えられている．

以上の結果より，無効化パケットの発生数から考慮すると，今回のシステムにおいて，DC に必要なエントリ数は 2048 であると言える．しかし，図 4，5，6 を見ると，エントリ数を 512 エントリから，2048 エントリに変更しても性能向上は，0.19〜13.11%程度である．特に，DC のエントリ数が 512 でも連想度を 4 とした場合の実行時間は，without_MINDIC と同等であった．このため，無効化パケットの発生数は多いが，実行時間で考慮すると，512 エントリのみでも without_MINDIC と同等の性能が発揮

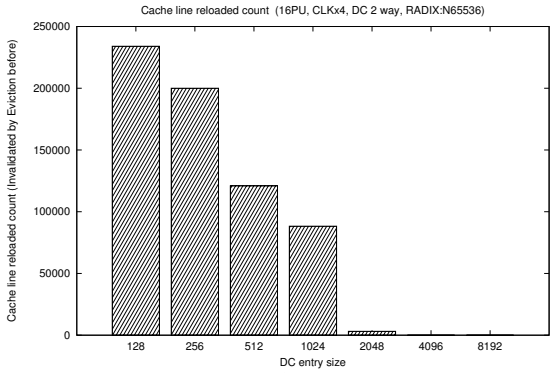

図 10 Eviction されたラインが再度 Read された回数 (16PU，RADIX)

表 2 ディレクトリ管理に必要なメモリ容量

512 entry	2048 entry	without_MINDIC
11 KByte	40 KByte	200 KByte/Memory(MByte)

できると言える．

5.5 トレースシミュレータによる評価との比較

トレースデータを用いた評価[6] では，無効化パケットの発生数を抑えることに注目し，DC の連想度 4 の場合でスイッチ毎に 4096 エントリ必要であるという結論となった．しかし，今回構築した詳細なシミュレーション環境でアプリケーションの実行を行った結果，実際には 512 エントリで十分であることが判明した．

6. ハードウェアコストの評価

第 5 節で行った評価から，各スイッチにおいて最低限 512 エントリは必要であり，また，2048 エントリの DC を設ければ各アプリケーションにおいて十分な性能が得られることがわかった．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 DC に必要とするメモリ容量を検討する．

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ディレクトリ管理に必要な全メモリ容量を算出した結果は表 2 のようになる．MINDIC ではディレクトリ管理に用いられるメモリ容量を大幅に削減可能であることを確認するため，MM へディレクトリを持たせるフルマップ方式で必要とするディレクトリ容量を比較対象として表示している．MINDIC でディレクトリ管理に必要なメモリ容量は 512 エントリの場合 8 スイッチ合計で 11KB，2048 エントリで 40 KByte 程度であり，スイッチ内に実装することが十分に可能であることが確認できる．

また，MINDIC のスイッチを Verilog-HDL で記述し，ハードウェア量の評価が行われている[6]．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 DC のエントリ数 4096，DC の連想度 2，4 で，DC 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 3 となっており，実装可能なハードウェア量であることが確認されているが，連想度 4 にすると連想度 2 に比べて 29.44%ゲート数が増加する．

表 3 論理合成結果 (タイミング制約 4*nsec*)

DC associativity	DC entry	Gate
2 way	4096	43328
4 way	4096	56084

7. 結　　論

一時的な共有情報であるテンポラリディレクトリを内部に持つスイッチで構成される多段結合網 MINDIC を提案し，キャッシュ一致制御のためのプロトコルを示した．

シミュレーション環境を構築し，3 つプログラムを最大 16PU で動作させるシミュレーションを行った．完全なディレクトリをもつアーキテクチャを比較対象とし，実行時間を比べた結果，MINDIC では DC を連想度 4 とすれば各スイッチに 512 エントリ，連想度 2 の場合でも 2048 エントリ設けることで十分な性能を達成できることがわかった．また，連想度を 2 から 4 にした場合のスイッチのゲート数の増加は，29.44%のみであった．

更に，Verilog-HDL を用いた実装の結果，スイッチ 1 つ当たり約 43328 ゲートと比較的低コストで実現できることを確認した．DC エントリ数を 512 として MINDIC でディレクトリに必要なメモリ容量を計算した結果，スイッチ 1 つあたり 1.42 KByte，全スイッチの合計でも 11 KByte と小容量のメモリで実現可能であることも確認できた．

参 考 文 献

1) 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.
2) 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.
3) 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.
4) 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.
5) 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. 電子情報通信学会コンピュータシステム研究会, CPSY2003-14, pp. 49–54, 2003.
6) 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 2002-EVA-8, pp. 19–24, Aug.2004.
7) 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, Vol. J84D-I, No. 3, pp. 1–10, 2001.
8) 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, 1995.