Pruning Cache を用いた分散共有メモリのディレクトリ構成法

西村克信[†]　　工藤知宏[‡]　　天野英晴[†]

[†]慶應義塾大学 理工学部

[‡]東京工科大学 工学部　(現在 新情報処理開発機構)

多くのプロセッサでデータ共有を行なう，大規模な CC-NUMA 型の並列計算機を対象とした，ディレクトリを動的に構成する手法について提案し，評価を行なう．Pruning Cache と RHBD (*Reduced Herarchical Bitmap Directory*) を組み合わせて用いることにより，少ないコストでより高い性能を得ることができる．評価の結果，従来の 1 対 1 転送の方式に比べて，転送容量の点でほぼ等しく，レイテンシの点で有利あることがわかった．

Pruning Cache: A Dynamic Directory Generation Scheme For Distributed Shared Memory

The pruning cache directory method is proposed for dynamic hierarchical directory management on large scale CC-NUMA systems in which a node aggressively shares data with other nodes. By a combination with RHBD (*Reduced Herarchical Bitmap Directory*) method, dynamic directory is quickly formed and managed with a small additional hardware. From the probabilistic simulation of a large system, it appears that the combination of the pruning cache and the RHBD achieves better latency and bandwidth than those of traditional directory management methods based on 1-to-1 message passing.

1　はじめに

現在，キャッシュを持つ分散共有メモリ型並列計算機 (*Cache Coherent Non Uniform Memory Access model* — CC-NUMA) の開発が各地で行われている [2, 4, 7]．

これらのシステムでは分散共有メモリを実現するために，キャッシュディレクトリを数十バイト程度の大きさのライン単位で管理し，無効化型のプロトコルによりキャッシュの一致制御を行っている．この方法は様々な評価や実システム上での経験 [4] から，数十から数百プロセッサの規模ならば高い効率を得られることが明らかになっている．

ディレクトリは，データを共有しているプロセッサ，即ち無効化/更新のためのメッセージの宛先プロセッサを示すものである．任意数のプロセッサを指定するためにはシステム全体のプロセッサ数と等しいビット数のビットマップが必要でありフルマップと呼ばれる．しかし，システム全体のプロセッサ数が多い場合には必要とするメモリ量が多くなり過ぎてフルマップを用いることは困難である．そこで，ライン単位にディレクトリを管理し，無効化型プロトコルを用いる場合，無効化メッセージの宛先は，1 ないし 2 がほとんどであるといわれることから [1]，従来，リミテッドポインタ方式，チェインドディレクトリ方式 [5, 10] などのディレクトリ構成方式が用いられてきた．

前者の方法では，データを共有するプロセッサ数が一定の数までは，それらのプロセッサを指し示すポインタをキャッシュのページ/ラインに持たせる．最初に提案された方法では，共有するプロセッサ数がポインタ数を超える場合は，無効化を行なって数を制限するか，ブロードキャストに切替える．しかし，これではプロセッサ数がポインタ数を越えた場合の性能低下が激しいため，ソフトウェアのエミュレーションに切替える方法も提案され [3]，MIT の Alewife[2] に利用されている．

後者の方法は，SCI(Scalable Cache Interface) で標準化されているもので，データを共有するプロセッサを差すポインタでリスト構造を作る．プロセッサ間でリストをたどるのを避けるため，メモリを管理

するプロセッサのメモリ中に動的にリスト構造を作るダイナミックポインタ法も提案され，Stanford の FLASH[7] で用いられている．

しかし，システムが大規模化し，共有するメモリ量が増大するにつれ，ライン単位でディレクトリを持つとディレクトリに必要なメモリ量が非常に大きくなってしまう．そこで，ディレクトリに必要なメモリ量を削減し，ディレクトリをキャッシングして処理を高速に行うために，数キロバイト程度の大きさのページ単位でディレクトリを管理し，ラインには共有状態を示す数ビットのフラグのみを付加して，データ転送はライン単位で行なう方式が考案され，超並列マシン JUMP-1[17] や Tempest[9] で利用されている．この方式では，同一ページのいずれかの共有状態のラインのコピーを持つプロセッサはそのページ全体を共有することになるため，共有するプロセッサ数が多くなる．また，頻繁にデータをやりとりする場合，無効化型よりも更新型のプロトコルの方が有利であるが，更新型では一般に同一のラインまたはページを共有するプロセッサ数が多くなる [16]．このため，リミテッドポインタ法では頻繁にソフトウェアによるエミュレーションが発生するし，チェインドディレクトリ法では連鎖が非常に長くなりいずれも性能の低下を招くと考えられる．

本論文では，ページ単位に管理を行なう場合や更新型のプロトコルを利用する等，データ共有が多い場合にフルマップを実行時に動的に生成する手法である Pruning Cache を提案し，縮約階層ビットマップディレクトリ [14] と組み合わせて両者の弱点を補う方法を述べる．さらにトレースおよび確率モデルにより，その性能の評価を行う．

2 Pruning Cache

ページ単位の管理を行なう場合あるいは更新型プロトコルを利用する場合，キャッシュの容量の不足による追い出しを除けば，新たにそのデータを共有するプロセッサが現れなければ，共有しているプロセッサの集合は変化しない．

システム規模が大きくなると共有データすべてについてフルマップのディレクトリを持つのは難しい．一方，それぞれのプロセッサが，自身があるアドレスのデータを持っているかどうかを判断することは簡単である．そこで，一旦広い範囲に無効化/更新メッセージを送り，その結果そのアドレスのデータを持っていないプロセッサは自ら送り手に知らせることによってフルマップディレクトリを動的に生成する手法が，本論文で提案する Pruning Cache である．

この手法では，はじめ更新メッセージは全ての宛先プロセッサを含むプロセッサの集合に対して送られる．この集合が全プロセッサであればブロードキャストとなる．メッセージを受け取ったプロセッサは，そのデータをキャッシュしている場合には acknowledge メッセージを，そのデータをキャッシュしていない場合には not-acknowledge メッセージを返す．送信元がこれらのメッセージを収集する際に，acknowledge を返したプロセッサに対応するビットには 1 を，not-acknowledge を返したプロセッサには 0 を立てたビットマップを作れば，フルマップを動的に生成することができる．引き続く更新ではこのフルマップを用いて宛先プロセッサのみにメッセージを送ることができる (図 1)．

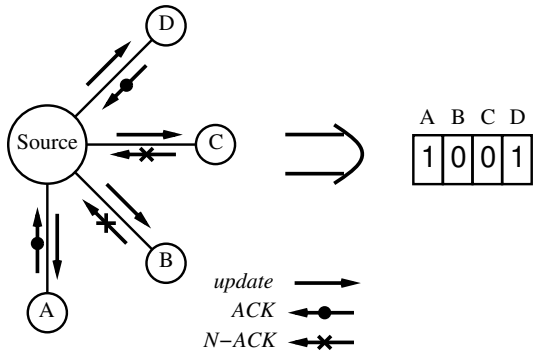

図 1: Pruning Cache

メモリアクセスに局所性があれば，動的に生成したフルマップを有限個持つことにより十分な効率を得ることができると考えられる．そこで，生成したフルマップを格納するキャッシュを用意する．これが Pruning Cache である．Pruning Cache の置き換えは LRU などで管理すれば良く，追い出されたアドレスへのアクセスが起きた時には再びより広い範囲のプロセッサの集合に対してメッセージを送ってフルマップを再度生成する．

Pruning Cache は，一旦メッセージを送ったプロセッサのうちで実際には必要としないプロセッサへは，それ以降更新メッセージを送らないように通信路の枝刈り (pruning) をしていることになる．

一旦 Pruning Cache 上にエントリが作られた後でプロセッサのキャッシュメモリから該当するデータが追い出された場合には，次の更新時に not-acknowledge が返され Pruning Cache の状態が更新される．また，新たにデータを共有するプロセッサが加わった場合には，その時点で共有するプロセッサを全て含む範囲に再度更新メッセージを送り，Pruning Cache のエントリが存在すれば一旦無効にして設定しなおす．

2.1 階層ビットマップディレクトリへの適用

更新メッセージの宛先数が多い場合，それらのメッセージを 1 対 1 通信で逐次的に送ると，全てのメッセージを送信するのには非常に時間がかかる．この問題に対処するには，同一のメッセージが通信路の途中で複製されて複数の宛先に伝達される木構造状のマルチキャストを用いることが考えられる．

木構造状にマルチキャストするためには，木の各節においてどの枝にマルチキャストするかを示す必要がある．各節におけるマルチキャスト先をそれぞれビットマップで表す方法が階層ビットマップディレクトリ方式である [16]．

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葉 (プロセッサ) にマルチキャストすることを考える．木構造のネットワークであるから，各節において，送信先の葉を含む枝にのみパケットを送れば，結果として必要な葉にパケットが届くことになる．図 2に，3 進木の根から “● ” でマークされた葉にパケットを送る場合について示す．このとき，各節に 3bit のビットマップを与えることにより送信先を完全に指定できる．階層ビットマップディレクトリは，同一の経路を同一のパケットが複数回通ることがなく，マルチキャストを効率良く行えるという特徴がある．

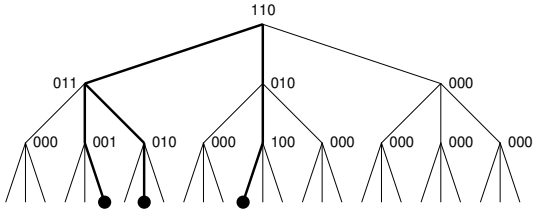

図 2: 階層ビットマップディレクトリ方式

階層化されていない場合には，送信元のプロセッサにおいて，全プロセッサ数と等しいビット数のフルマップを格納できる幅を持つ Pruning Cache を用意しなくてはならない．しかし，階層ビットマップディレクトリ方式に Pruning Cache を適用すると，各節でその節の持つ枝の数分のビット幅の Pruning Cache を持つだけで，動的に階層ビットマップディレクトリを生成することができる．

2.2 縮約階層ビットマップディレクトリとの組合せ

Pruning Cache では，はじめに全ての宛先プロセッサを含むプロセッサの集合に対して更新メッセージを送る．この集合が全てのプロセッサであればブロードキャストとなる．この操作は Pruning Cache のミス時にのみ行われるので，回数は比較的少ないと考えられるが，大規模なシステムでは少ない回数のブロードキャストでもオーバヘッドが大きく性能が低下する可能性がある．

そこで，ページ毎に容量の少ない簡略なディレクトリを用意し，Pruning Cache ミス時にメッセージを伝達するプロセッサ数を少なくすることが考えられる．超並列計算機 JUMP-1[15] で用いられている縮約階層ビットマップディレクトリ (*Reduced Hierarchical Bit-map Directory* — RHBD) 方式 [16] はその一つの手法である．

2.2.1 縮約階層ビットマップディレクトリ方式

階層ビットマップディレクトリの各節のビットマップについて

- 同一階層の節ではすべて同じビットマップを用いる．
- 特定の節ではブロードキャストを用いる．

の 2 つの方針をもとに，縮約を行うディレクトリの構成法が RHBD 方式である．

木の階層ごとに一つだけビットマップを用いるので，m 階層の n 進木の結合網においてディレクトリを管理するのに必要なビット数は $m \times n$ となる．

縮約階層ビットマップディレクトリの最も大きなメリットは，ビットマップの縮約により，ビットマップそのものをパケット中に持たせることができるため，ディレクトリの引き直し等のコストがかからないことである．このため，縮約前の階層ビットマップディレクトリ方式に比べてマルチキャストに要する時間を大幅に短縮することができる．

この縮約階層ビットマップディレクトリ方式には，縮約手法の違いにより LPRA 法，LARP 法，SM 法の 3 つの手法が提案されている [16]．ここでは，SM 法についてのみその縮約手法を示す．

SM 法

パケットは，根から各階層に与えられたビットマップにしたがって順にマルチキャストされる．各階層でのビットマップは，縮約前のビットマップの階層ごとの論理和となる．この例を図 3に示す．

この例では，各階層で単一のビットマップ (110，011，111) を用いマルチキャストを行っている．各階層のビットマップは，縮約前の階層ビットマップディレクトリ方式 (図 2) の各階層ごとの論理和になっているのがわかる．たとえば，最下位層では 001，010，100 の論理和である 111 を用いる．

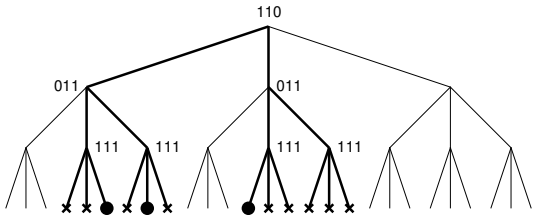

図 3: SM 法

最終的に，“● ” で示した葉に対してパケットが送られる．しかし同時に，“× ” で示した葉に対して無駄なパケットが送られてしまう．

この手法は，他の 2 つの手法に比べて，宛先数が少ない場合や，規則性のある宛先に対して効率良く指定できる点に特徴がある [16]．

2.2.2 Pruning Cache の適用

LPRA に対して Pruning Cache を適用する例を考える．はじめ，無効化 / 更新メッセージは図 3 のように送られる．この際に acknowledge/not-acknowledge メッセージの収集が行われ，図 4のように各節にビットマップがキャッシュされる．

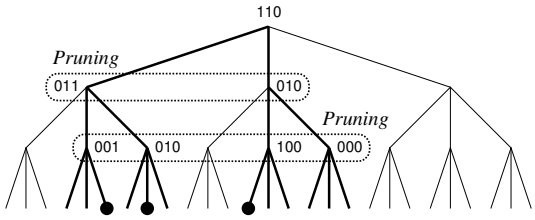

図 4: RHBD への Pruning Cache の適用

これを逆の立場から見ると，RHBD を単独で用いた場合は，常に無効化/更新メッセージが図 3のように送られることになり，多くの無駄パケットが発生する．これが RHBD の致命的な欠点であった．Pruning Cache の利用により，ヒット時にはこれらの無駄パケットを大幅に削減することができる．

3 評価

3.1 トレースを用いた枝刈り効率の評価

Pruning Cache の効果は，どの程度の確率でエントリがマッチして実際にメッセージを削減できるか，すなわちヒット率に依存する．そこで，実際の並列プログラムのトレースデータを用いてシミュレーションを行ないヒット率を調査した．

評価は，SPLASH 並列プログラム集 [8] の中から，Choleskey 問題のアドレストレースを用いて行なった．分散共有メモリは 4Kbyte のページ単位で管理され，分散共有メモリのラインサイズは 32byte である．この場合，ひとつのメッセージの宛先は平均 6 程度になる．

ノード数は 64 とし，単純な 4 進木を用いた．Pruning Cache は，RHBD 方式と組み合わせてルータ内部に装備した場合，外部メモリの参照が必要なくなり，高速なメッセージ転送が可能となる．この場合，チップ面積の制限からエントリ数はさほど多くとることはできない．そこで，ここでは，エントリ数 16，32，64 についてそれぞれ Direct，2 way set associative，4 way set associative の構成に基づき，ヒット率を測定した．Pruning Cache エントリの追い出しアルゴリズムは LRU を用いている．

SM 法を用いた際の Pruning Cache のヒット率の測定結果を図 5に示す．エントリ数が 16 の場合，ヒット率は 70%代であるが，32 を越えると 2 way 以上では 90%を越し，64 の場合は 90%を上回るヒット率が得られる．同一エントリ数では当然 way 数が多いものがヒット率が高いが，エントリが 32 を越えると，direct と 2 way の差ほど 2 way と 4 way の差は大きくない．4 way は 4 セットの比較器を必要とするため，実装を考慮すると 32 エントリ，2 way 程度の構成がコストに比較して高い性能が得られることがわかる．

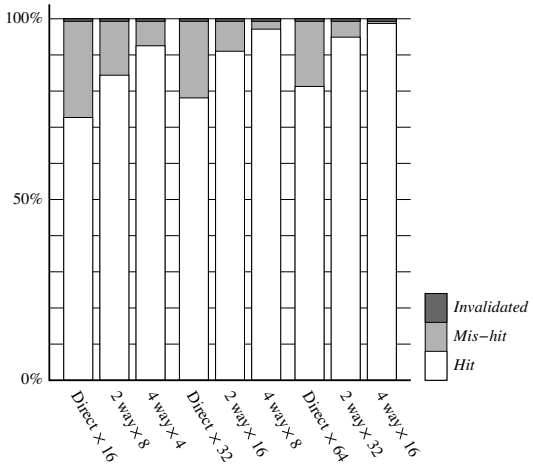

図 5: キャッシュのヒット率

3.2 RDT を用いたシステムでの評価

次に実際に大規模な並列計算機上での実装を念頭において行なった評価結果を示す．大規模システムはトレースドリブンによるシミュレーションが困難であることから，今回の評価は確率モデルを用いて行なった．Pruning Cache を装備した RHBD は階層構造を持つ任意の結合網に適用可能であるが，ここでは超並列計

算機 JUMP-1 で用いられた結合網 RDT(*Recursive Diagonal Torus*)[19] 上に実現し，評価する．

3.2.1 RDT 上での 8 進木の実現

RDT は図 6に示すように 2 次元トーラスに目の粗いトーラス (上位トーラス) を 45 度傾けて次々に重ねた階層構造を持つ．上位トーラスのひとつのノードから図 7に示すようにマルチキャストを 2 回行うことにより，8 進木を構成することができる．メッセージがマルチキャストされる範囲は送信元のノードを中心とし亀甲状のパタンになる．この範囲をテリトリと呼ぶ．

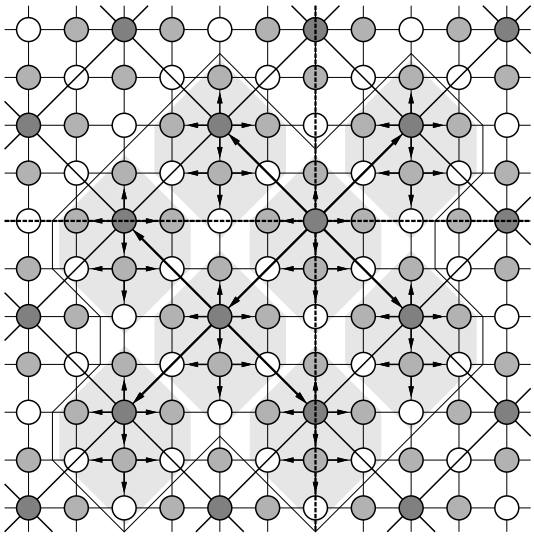

図 6: 相互結合網 RDT

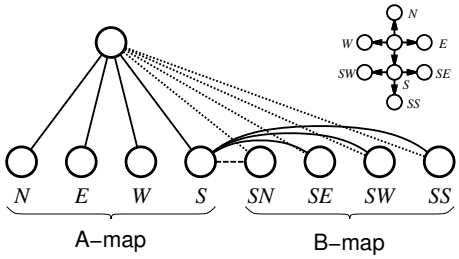

図 7: RDT 上での 8 進木

テリトリは，マルチキャストを始める木の高さに従い，送信元を中心に同心円状に広がる．図 6は，ランク 1 トーラスからのマルチキャストのテリトリ，すなわち 8 進木の 2 階層目からのマルチキャストの様子を示す．この図ではメッセージは周辺 6 4 ノードに対してマルチキャストされる．RDT は，マルチキャストを開始する上位トーラスを多数持つため，マルチキャストを行う木構造は Fat Tree となり，根付近での混雑が起こらない利点がある．この 8 進木を利用して RHBD により分散共有メモリを実現するための RDT ルータチップは JUMP-1 用に既に実装され，稼働している．Pruning Cache は，このルータチップ内に簡単に組み込むことができる．

3.2.2 RDT シミュレータ

RDT 上で RHBD を実現した場合の Pruning Cache の効果を調べるため，パケットレベルのシミュレータを作成した．このシミュレータは実装された RDT ルータチップを基に，パケット長，パケットがノードを通過する時間等のパラメータを与えることにより，クロック単位でレイテンシィの測定を行うことができる．パケットの転送に関しては実装されたチップに忠実に，バーチャルチャネルを用いたデッドロックフリールーティングをサポートし，RHBD の 3 つの手法 (SM 法，LPRA 法，LARP 法) 全てを備えている．

RHBD および Pruning Cache の性能は，アプリケーションプログラムのプロセス間のデータ共有の状況のみならず，プロセスのマッピングにも大きく影響を受ける．そこで，キャッシュの一致のために送るパケットの宛先は乱数を用いて決定した．まず，乱数を用いて任意の送信ノードを選び，さらに乱数を用いて一定の数の宛先ノードを選ぶ．この際宛先ノードは，マッピングの局所性を再現するため，東西方向および南北方向が互いに独立に，送信元から標準偏差 5 で散らばっているものとした．

乱数を用いて宛先ノードを決定する方式では Pruning Cache の詳細をシミュレーションしても意味がないため，Pruning Cache を装備する階層および Pruning Cache のヒット率 (枝刈り率) をパラメータとして与えてシミュレーションを行った．なお，RDT のノード数は 256(16×16) とし，パケットの通過時間は 3 クロック/ホップ，パケットの長さは 8 フリットとして評価した．

3.2.3 Pruning Cache の効果

RHBD の木構造のランク 1(すなわち木の高さ 2 の節) および最下位層に Pruning Cache を付加した場合について，マルチキャストしたパケットが全ての宛先に届くまでの平均レイテンシィを図 8および図 9に示す．

前節の評価では Pruning Cache のヒット率はエントリ数 32，way 数 2 で 90%を越えているが，サイズと木構造の構成が異なることを考えて，80%に設定した．また，宛先数は 4 および 6 の結果をそれぞれ示した．この評価では横軸は平均パケット発生間隔であり，これが小さくなる程頻繁にパケットが発生するため結合網が混雑し，レイテンシィが大きくな

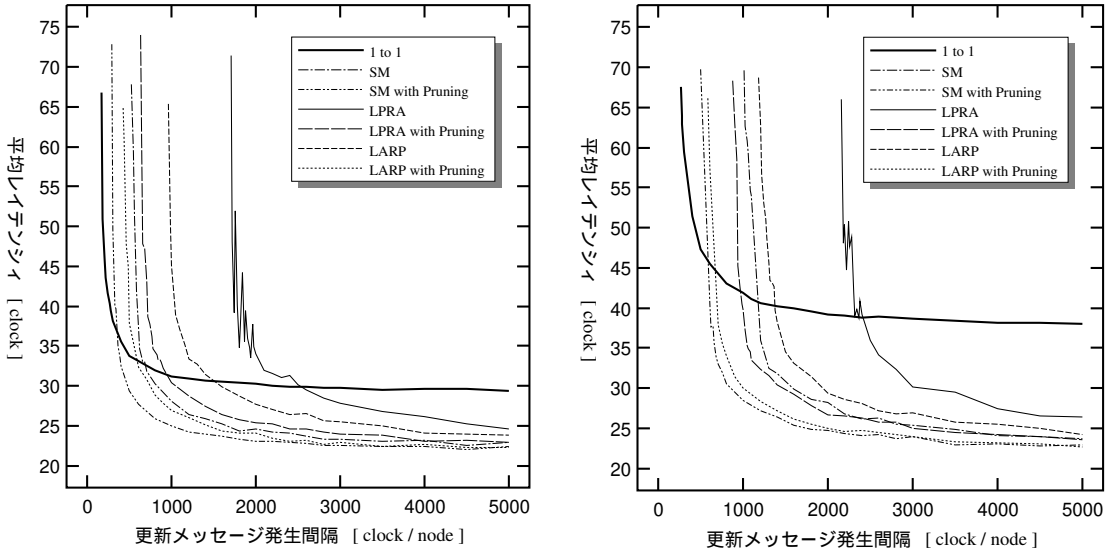

図 8: ランク 1 に Pruning Cache を装備した場合

る．レイテンシィが急激に大きくなる発生間隔が結合網の利用可能な転送容量を示すと考えられる．比較のため，マルチキャストするかわりに，1 対 1 転送を宛先数分繰り返した場合についても併せて示す．

RHBD は，Pruning Cache を装備しない場合でも，平均パケット発生間隔が長ければ平均レイテンシィが 1 対 1 転送に対して 30%-80%程度優れている．これはマルチキャストを効率良く行えるためである．ところが平均パケット発生間隔が短くなると，レイテンシィが急激に悪化してしまう．これは，RHBD を用いたために発生する無駄パケットのために，結合網の転送容量が飽和してしまうためである．Pruning Cache を用いると，平均パケット発生間隔にかかわらず結合網の混雑時のレイテンシィが改善されている．このため結合網の利用可能な転送容量 (すなわち飽和を引き起こすパケットの生成間隔) は 4~8 割ほど大きくなっている．

また，Pruning Cache は，最下位層に装備した場合により効果的である．最下位層に装備した場合，利用可能な転送容量は，ほとんど 1 対 1 転送と等しくなっている．

3.2.4 Pruning Cache へのヒット率

次に Pruning Cache のヒット率の影響を調べるため，前述の LPRA 法に Pruning Cache を付け加えた場合に関してヒット率を変化させた結果を図 10に示す．宛先数は 4 とし，ヒット率は 25%，50%，75%，100%とした．

この結果によると，Pruning Cache のヒット率に従い，利用可能な転送容量はほぼ線形に変化することがわかる．前節の評価によると 32 エントリ，2 way の構成でヒット率は 90%を越えており，かなり高い性能が期待される．ただし，前節の評価結果は比較的小規模なシステムであるため，より大規模なシステムに対する評価が必要である．

4 関連研究との比較

Pruning Cache に関しては同様なアイディアによる研究が Scott らにより行なわれている [11, 12, 13]．しかしこの研究では，ページ単位の管理や更新型プロトコルを考えられておらず，概念の提示に留まっている．また，他の方式との組み合わせによる，ブロードキャスト時のオーバヘッド削減等について十分検討されていない．

塙らは同様なアイディアを MIN に適用した MINC[18] を提案しているが，この研究でも無効化型のキャッシュを念頭におくため，Cache の動的形成は読み出し動作時に行なわれる．このため，無効化メッセージを一回送るだけで，Cache の役割は終わってしまい，コストに見合う性能を得られるかどうか疑問である．さらに，これらの研究では実際的なトレースや，具体的なルータに基づく評価がなされていない．

階層型ディレクトリを効率良く管理する方法としてはチェーンドディレクトリを Tree 構造に拡張した Scalable Tree Protocol[6] が提案されている．この方法は，動的に木構造を構成する点で Pruning Cache と類似点があるが，ノード内で大規模なメモリ上のポインタをたどる必要があり，ルータ内で全ての処

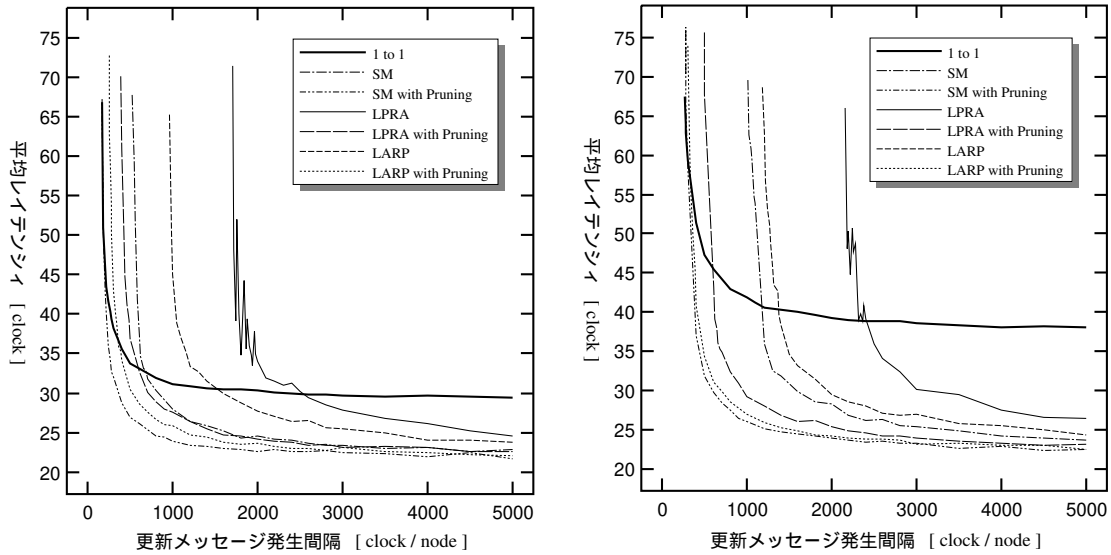

図 9: 最下位層に装備した場合

理が終了する Pruning Cache+RHBD に比べてノード上での処理時間の点で不利である．

5 おわりに

大規模な CC-NUMA 型の並列計算機で，多くのプロセッサによりデータが共有される場合に有効なディレクトリを動的に構成する手法である Pruning Cache を提案した．さらに，階層ビットマップディレクトリ方式のひとつである RHBD と組み合わせて用いることにより，低コストで効率的な分散共有メモリのディレクトリ管理を行なう方法を示した．トレースデータによる評価の結果，64 プロセッサのシステムでは，32 エントリ，2 way set associative の構成で 90%を越えるヒット率が得られることがわかった．さらに，確率モデルを用いた大規模なシステムのシミュレーションの結果によりヒット率がある程度以上大きければ，Pruning Cache と RHBD 方式の組合せにより，従来の 1 対 1 転送の方式に比べて，転送容量の点でほぼ等しく，レイテンシの点で有利あることがわかった．

今回の評価は，ノード数が小さいシステムはトレースに基づく評価を行なったが，大きいシステムは確率モデルを用いざるを得なかった．これは大規模なシステムをトレースドリブンで評価することが時間的にきわめて難しいためであるが，今後，シミュレータの効率化により，さらに大規模なシステムについて実プログラムに基づく評価を行なっていく方針である．

参考文献

[1] A.Agarwal, et al. An Evaluation of Directory Schemes for Cache Coherence. In *Proc. 15th ISCA*, pp. 280 – 289, 1988.

[2] D.Chaiken and A.Agarwal. Software-Extended Coherent Shared Memory: Performance and Cost. In *Proc. The 21st ISCA*, pp. 314 – 324, 1994.

[3] D.Chaiken, J.Kubiatowwicz, and A.Agarwal. LimitLESS Directories: A Scalable Cache Coherence Scheme. In *Proc. ASPLOS IV*, pp. 224 – 234, 1991.

[4] D.Lenoski, et al. The Stanford DASH Multiprocessor. *IEEE Computer*, Vol. 25, No. 3, pp. 63 – 79, 1992.

[5] D.V.James, et al. Distributed-Directory Scheme: Scalable Coherent Interface. *IEEE Computer*, Vol. 23, No. 6, pp. 74 – 77, 1990.

[6] H.Nilsson and P.Stenstrom. The Scalable Tree Protocol – A Cache Coherence Approach for Large Scale Multiprocessors. In *Proc. 1992 SPDP*, pp. 498 – 506, 1992.

[7] J.Kuskin, et al. The Stanford FLASH Multiprocessor. In *Proceedings. of The 21st International Symposium on Computer Architecture*, pp. 302 – 313, 1994.

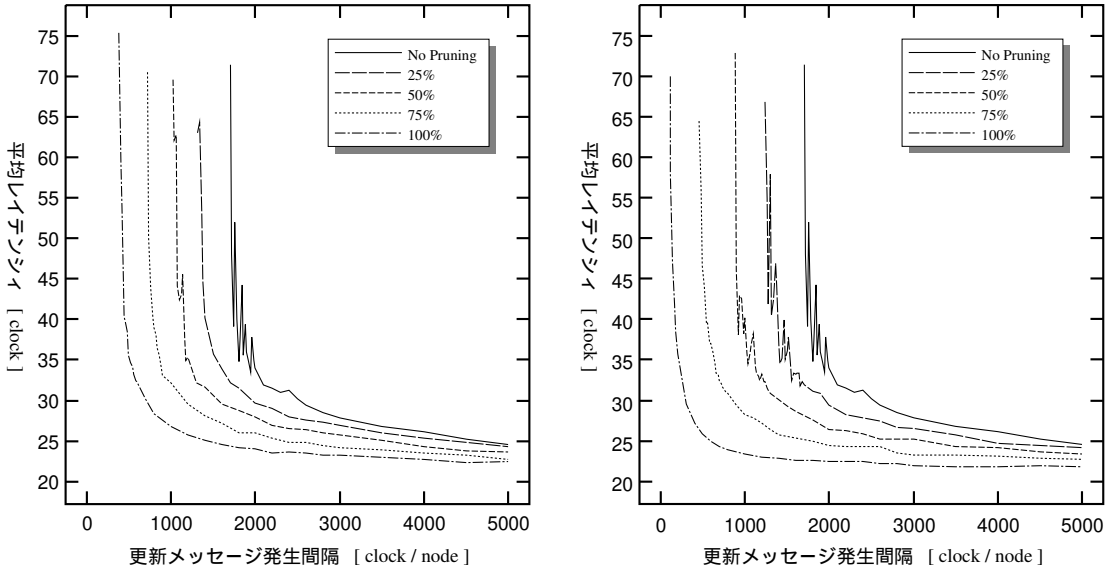

図 10: ヒット率の性能に及ぼす影響

[8] J.P.Singh, W.Weber, and A.Gupta. SPLASH: Stanford Parallel Applications for Shared-Memory. Tech. report, computer system laboratory, Stanford University, 1992.

[9] M.D.Hill, J.R.Larus, and D.A.Wood. Tempest: A Substrrate for Portable Parallel Programs. In *Proc. COMPCON '95*, pp. 327 – 332, 1995.

[10] M.Thapar and B.Delagi. Distributed-Directory Scheme: Stanford Distributed-Directory Protocol. *IEEE Computer*, Vol. 23, No. 6, pp. 78 – 80, 1990.

[11] S.L.Scott. A Cache Coherence Mechanism For Scalable, Shared-Memory Multiprocessors. In *Proc. 1991 ISSMM*, pp. 49 – 59, 1991.

[12] S.L.Scott. *Toward The Design Of Large-Scale, Shared-Memory Multiprocessors*. PhD thesis, University of Wisconsin, 1992.

[13] S.L.Scott and J.R.Goodman. Performance of Pruning-Cache Directories for Large-Scale Multiprocessors. *IEEE Trans. Parallel and Distributed Processing*, Vol. 4, No. 5, pp. 520 – 534, 1993.

[14] 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'95 論文集, pp. 257 – 264, 1995.

[15] H. Tanaka, Y. Muraoka, M. Amamiya, N. Saito, and S. Tomita, editors. *The Massively Parallel Processing System JUMP-1*. オーム社, 1996. ISBN4-274-90083-5.

[16] 西村克信, 工藤知宏, 西宏章, 楊愚魯, 天野英晴. 相互結合網 RDT 上での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. 情報処理学会論文誌, Vol. 37, No. 7, pp. 1367 – 1377, 1996.

[17] 松本尚, 平木敬. Memory-Based Processor による分散共有メモリ. 並列処理シンポジウム JSPP'93 論文集, pp. 245 – 252, 1993.

[18] 安川英樹, 舟橋啓, 西村克信, 塙敏博, 天野英晴. キャッシュ制御機構内蔵型多段結合網:minc. 並列処理シンポジウム JSPP'96 論文集, pp. 129 – 136, 1996.

[19] Y.L.Yang and H.Amano. Message transfer algorithms on the recursive diagonal torus. *Proceeding of the 1994 International Symposium on Parallel Architectures*, Vol. Algorithms and Networks, pp. 310 – 317, 1994.