Vol. 39 No. 6 情報処理学会論文誌 June 1998

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[†]

Pruning Cacheは，大規模なCC-NUMA型並列計算機においてディレクトリを動的に構成する手法である．この方法は，ページ単位で共有関係を管理したり，更新型のプロトコルを用いるなど，データ共有を行うプロセッサ数が多い場合に特に有効である．さらに，システムが階層型結合網を持つ場合，縮約階層ビットマップディレクトリ法(RHBD: *Reduced Hierarchical Bitmap Directory*)を組み合わせて用いることにより，互いの弱点を補うことができ，より高い性能を得ることができる．トレースドリブンシミュレーションによる評価の結果，多くのアプリケーションプログラムにおいて，32エントリ2 wayの構成で75%以上のヒット率を実現することが分かった．さらに大規模な階層型結合網を持つシステムに関して確率モデルにより評価した結果，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であることが分かった．

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

Katsunobu Nishimura,[†] Tomohiro Kudoh[††] and Hideharu Amano[†]

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 Hierarchical Bitmap Directory*) method, dynamic directory is quickly formed and managed with a small additional hardware. Trace driven simulation shows that the hit ratio of the 2-set associative cache with 32 entries is more than 75% in most of applications. 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

キャッシュコヒーレントな分散共有メモリを持つマルチプロセッサシステム(CC-NUMA: *Cache Coherent Non Uniform Memory Access model*)は，システムサイズに応じスケーラブルな性能が得られる可能性を持つ一方，現在の小規模バス結合型マルチプロセッサからのプログラム移植が容易であることから，各地で開発が行われている[1)~5)]．現在のCC-NUMAの多くは，メモ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分散共有メモリを実現している．様々な評価や実システム上での経験[1),2)]から，数十から数百プロセッサの規模ならばこの方法によって高い効率が得られることが明らかになっている．

また CC-NUMA では，他のプロセッサのメモリをキャッシュするため，データを共有しているプロセッサを示すキャッシュディレクトリが必要であり，その構成方式と管理方式がコストおよび性能に大きく影響する．最も簡単なディレクトリ構成法は，システム全体のプロセッサ数に等しいビット数のビットマップで表すフルマップ法である．しかし，この方法はシステムを構成するプロセッサ数が大きくなると，必要とするメモリ量が多くなる点に問題がある．そこで従来のCC-NUMA では，無効化型プロトコルを用いてディレクトリをライン単位に持つならば，ラインを共有するプロセッサ数はそのほとんどが1ないし2である[6)]ことを利用して，リミテッドポインタ方式やチェインドディレクトリ方式[7),8)]などのディレクトリ構成方式が用いられてきた．

リミテッドポインタ方式では，一定数のポインタをキャッシュライン単位に用意しておき，そこにデータ

† 慶應義塾大学理工学部
Faculty Science and Technology, Keio University
†† 新情報処理開発機構
Real World Computing Partnership

を共有するプロセッサを示すポインタを格納する．この方法は，共有プロセッサ数がポインタ数を超える場合に問題が生ずる．最初に提案された方法では，無効化を行って共有数を制限するか，ブロードキャストに切り替えていた．しかし，これでは共有プロセッサ数がポインタ数を超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り替える方法が提案され[9]，MIT の Alewife[1] で利用されている．

一方，チェインドディレクトリ法は，データを共有するプロセッサを示すポインタでプロセッサのキャッシュ間にまたがるリスト構造を作る方式で，SCI(*Scalable Cache Interface*) で用いられている．また，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[3] で用いられている．

これらのディレクトリ構成方法は，中規模のシステムでは有効であるが，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ことによるメモリ量の増大が無視できなくなる．そこで，ディレクトリに必要なメモリ量を削減するために，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う方式が考案され，超並列計算機 JUMP-1[10] や Tempest[11] で利用されている．この方式では，あるページ内のいずれかのラインを共有する場合，そのページ全体を共有することになるため，必然的に共有プロセッサ数が増加する．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．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さらに多くなることが知られている[12]．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，このようにページ単位に管理を行ったり更新型のプロトコルを利用する場合において，共有プロセッサを表すフルマップを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[13] 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，その性能評価を行う．

2. Pruning Cache

システムの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，自分があるアドレスのデータを共有しているかどうかを判断することは簡単である．そこで，いったん広い範囲に無効化/更新メッセージを送り，そのアドレスのデータを共有しているプロセッサが自ら送り手に知らせ，その結果からフルマップディレクトリを動的に生成する手法を提案する．

2.1 Pruning Cache の原理

ページ単位の管理を行う場合や，更新型プロトコルを用いる場合は，ページまたはラインを共有しているプロセッサの集合は，一定の状態に達した後は変化することが少ないと考えられる．動的に生成されたフルマップディレクトリは，この集合が変化しない間は，正しい状態であるといえる．

ここで提案する手法では，1 回目の無効化/更新メッセージはブロードキャストされる．このメッセージを受け取ったプロセッサは，そのデータを共有している場合には acknowledge メッセージを，そのデータを共有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生成すれば，フルマップディレクトリを動的に生成することができる．引き続く 2 回目以降のメッセージは，先程生成されたビットマップを用いることにより，目的のプロセッサのみに送ることができる（図 **1**）．

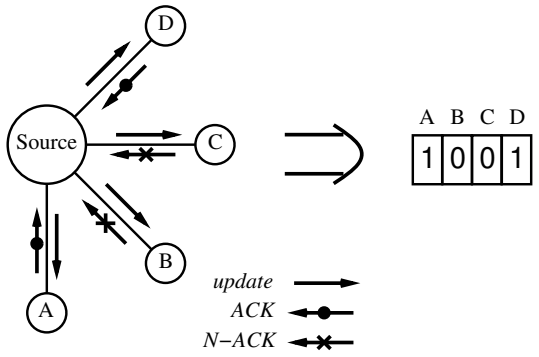


図 **1** Pruning Cache
Fig. 1 Pruning Cache.

メモリアクセスに局所性があれば，動的に生成したビットマップを有限個保持する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，生成したビットマップを格納するためのキャッシュメモリを用意

する．このビットマップを保存しておくキャッシュのことを Pruning Cache と呼ぶ．このキャッシュメモリにより，いったん，メッセージを送信したプロセッサのうち，実際には必要としないプロセッサへはそれ以降のメッセージは送られない．つまり，Pruning Cache により通信路の枝刈り (pruning) をしていることになる．

この Pruning Cache のエントリが足りなくなった場合，LRU などの管理手法により追い出されることになる．また再び，Pruning Cache から追い出されたデータに対して読み書きが行われた場合，1 回目の時と同じように，メッセージをブロードキャストすることによりビットマップを生成しなおす．

以下，Pruning Cache の基本的な操作を示す．

2.1.1 Pruning Cache への追加

あるプロセッサが今まで共有していなかったライン (またはページ) に対してアクセス要求を発生する場合，そのプロセッサはそのデータを管理するプロセッサ(ホーム)に対して，共有開始のためのメッセージを送信する．このメッセージを受け取ったホームは，共有プロセッサが増えたことを示すメッセージをブロードキャストする．このとき Pruning Cache 中にビットマップが存在すればこれを無効にし，このブロードキャストに対する応答メッセージによりビットマップを再生成する．

この方法は，共有プロセッサが増えるたびにブロードキャストが行われるため，性能の低下を招く可能性がある．そこで，後に述べるように，縮約型階層ビットマップ方式と組み合わせて用い，ブロードキャストされる範囲を限定することにより性能の低下を防ぐ．

2.1.2 Pruning Cache からの削除

なんらかの理由により，プロセッサのメモリから共有データが追い出された場合，次のメッセージを受け取った際に not-acknowledge メッセージを返す．このことにより，Pruning Cache 中のビットマップが更新され，それ以降のメッセージは，そのプロセッサに対し送信されない．

また，Pruning Cache 中のビットマップのすべてのビットが 0 になった場合，つまり共有しているプロセッサが 1 つもなくなった場合，そのエントリは Pruning Cache から削除される．

2.2 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，すべ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 [12]．

今，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うことができる．

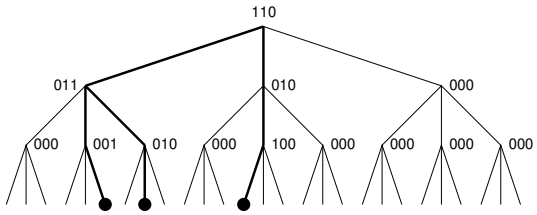


図 2 階層ビットマップディレクトリ方式
Fig. 2 Hierarchical bitmap directory.

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．

2.3 縮約階層ビットマップディレクトリとの組合せ

先に述べたように，Pruning Cache ではビットマップの生成のために，メッセージのブロードキャストが必要となる．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．

実際にはこのメッセージはブロードキャストされる必要はなく，そのアドレスのデータを共有するすべて

の宛先プロセッサに届きさえすればどのような範囲に送られてもよい．そこで，ページごとに容量の少ない簡略なディレクトリを用意し，この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[14] で用いられてい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[12] は簡略なディレクトリ管理方式の 1 つである．

2.3.1 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
- ある節以下はブロードキャスト

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 1 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[12] が提案されている．ここでは SM 法についてのみその縮約手法を示すが，本論文で提案する Pruning Cache の組合せは，どの縮約手法に対しても有効である．

SM 法では，パケットは根からそれぞれの階層に対して 1 つずつ与えられたビットマップに従って順にマルチキャストされる．このとき各階層で利用されるビットマップは，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．

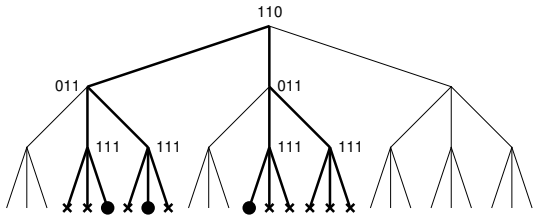


図 **3** RHBD/SM 法
Fig. 3 RHBD/SM scheme.

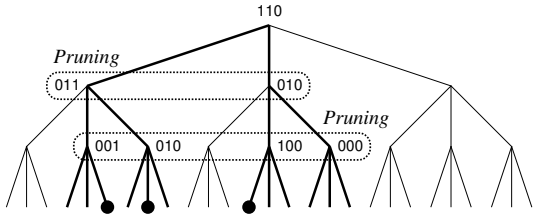


図 **4** RHBD への Pruning Cache の適用
Fig. 4 RHBD and Pruning Cache.

この例では，各階層で単一のビットマップ（110，011，111）を用いてマルチキャストを行っている．図中で用いられるビットマップは，縮約前の階層ビットマップディレクトリ方式（図 2）で用いられたビットマップを階層ごとに論理和した結果であることが分かる．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．

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[12]．

2.3.2 Pruning Cache と RHBD の組合せ

Pruning Cache と RHBD 方式を組み合わせて用いた場合，最初のメッセージは RHBD 方式を用いて図 3 のようにマルチキャストされる．これに対する応答メッセージの収集が各節ごとに行われ，図 4 のようにビットマップが生成/キャッシュされる．

次にメッセージを送る場合，Pruning Cache にヒットするればそのビットマップが用いられ，ミスした場合に限り RHBD 方式を用いてマルチキャストされる．Pruning Cache の立場からは，ミスした場合にブロードキャストではなく RHBD 方式でマルチキャストが行われるため，ミス時におけるオーバヘッドが低減される．

一方，RHBD 側から考えると，RHBD を単独で用いた場合に，つねに無効化/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が，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．以上のように RHBD と Pruning Cache の組合せは，容易に実装が可能で，かつ両方の方法の欠点を互いに補うことができる．

2.4 Pruning Cache のプロトコル

Pruning Cache は，キャッシュ制御のメッセージ数

を削減することが目的であるため，無効化型プロトコル，更新型プロトコルの両方について，プロトコル制御の細部に依存せず用いることができる．しかし，ライン単位の管理で無効化型プロトコルを用いる場合は，そのラインにいずれかのプロセッサが書き込みを行うごとに共有関係がクリアされてしまう．したがって，無効化のたびに Pruning Cache のエントリを再生成しなければならず，事実上その効果を失ってしまう．このため，Pruning Cache は

(1) ページ単位の管理で無効化プロトコル/更新型プロトコルを用いる場合
(2) ライン単位の管理で更新型プロトコルを用いる場合

に有効である．

ここでは，ページ単位で管理を行う場合の無効化プロトコルについて，階層構造を持つ結合網における Pruning Cache の操作を具体的に述べる．この方法は，Typhoon[15] や JUMP-1[14] で行われている方法で，共有関係の管理はページごとに設けられたディレクトリを用いて行う．各プロセッサノードでは，ディレクトリはページごとにのみ持つが，そのメモリを管理しているホームではラインごとに状態を持ち，データ転送もライン単位で行われる．ここでは最も簡単なプロトコルを想定する[16]．すなわち，それぞれのキャッシュラインの状態は，以下の 3 つの状態をとる．

- **I** — 無効
- **S** — 共有
- **O** — 他のキャッシュにはコピーが存在しない

またホームメモリ上でのラインの状態は，以下の 3 つの状態をとる．

- **U** — キャッシュされていない
- **C** — キャッシュされており，キャッシュの内容は主記憶と一致する
- **D** — キャッシュの内容は主記憶と一致しない

プロセッサ間のメッセージは 1 対 1 通信および階層構造を用いたマルチキャストで送られる．以下にそれらのメッセージを示す．

読み出し要求（**1 対 1**） プロセッサからホームへラインの転送を要求する．そのページに対する初めてのアクセスかどうかを示すビット（*share request*）を含む．
読み出しライン転送（**1 対 1**） ホームから要求元にラインを転送する．
書き戻し要求（マルチキャスト） ホームからプロセッサにラインの書き戻しを要求する．
書き戻しライン転送（**1 対 1**） プロセッサからホームにラインを転送する．
書き込み要求（**1 対 1**） プロセッサからホームへラインの書き込みを行う．そのページに対する初めてのアクセスかどうかを示すビット（*share request*）を含む．
再構成要求（マルチキャスト） ホームからプロセッサへ，共有状態の再構成を要求する．
無効化要求（マルチキャスト） ホームからプロセッサへ，ラインの無効化を要求する．
応答メッセージ（階層構造の下から上へ） プロセッサからホームへ，マルチキャストメッセージが届いたことを示す．階層構造の途中で収集される．その下の階層で共有されているかどうかを示す情報を含む．

ここで，マルチキャストメッセージは，階層構造を利用してそれぞれの階層ごとにマルチキャストされる．マルチキャストは，Pruning Cache 中にそのページに対するエントリが存在していれば，エントリ中のビットマップに従って行われる．エントリが存在しなければメッセージ中の RHBD のビットマップに従って行われる．再構成要求の場合は，いったんそのエントリを無効にした後，マルチキャストを行う．

マルチキャストメッセージを受信したプロセッサは，そのページを共有していれば応答メッセージの *acknowledge* ビットをセットし，そうでなければリセットして返す．応答メッセージを受けた Pruning Cache はその内容によってエントリを再構成し，自分より下の階層の応答メッセージを収集して *acknowledge* ビットの論理和をとり上の階層に転送する．

以下に，最も簡単な無効化プロトコルの操作を示す．更新プロトコルは，無効化要求の代わりに書き込みデータをマルチキャストするように変更することにより容易に実現できる．

2.4.1 読み出し操作

プロセッサのキャッシュの状態が **S**，**O** の場合，直接キャッシュからデータを読み出す．状態が **I** の場合，ホームに対して読み出し要求メッセージを出す．そのページに対して初めて要求を出す場合，*share request* ビットをセットしておく．

この要求を受け取ったホームは，ホームメモリ上のディレクトリを検索し，ラインの状態を調べる．状態が **D** ならば書き戻し要求をマルチキャストし，これに対して転送されてきたラインを受け取りホームメモリに書き戻す．次に，要求メッセージ中の *share request* ビットを調べ，セットされていな場合，読み出しラインを発生元に 1 対 1 通信で転送する．セットされいる

場合，読み出しラインを発生元に 1 対 1 通信により送るとともに再構成要求をマルチキャストする．再構成要求の応答メッセージを収集し終わったら，ホームのディレクトリの状態を **C** とする．応答メッセージを収集する前に同一ラインに対して次の要求が到着した場合，その要求の処理は待たされる．

読み出し要求を出したプロセッサは，ホームから転送されたラインをキャッシュに格納しその状態を **S** とする．

2.4.2 書き込み操作

プロセッサのキャッシュの状態が **O** の場合，直接キャッシュにデータを書き込む．**I**，**S** の場合，ホームに対して書き込み要求を送る．そのページに対して初めて要求を出す場合，*share request* ビットをセットしておく．

要求を受け取ったホームは，ホームメモリ上のディレクトリを検索し，ラインの状態を調べる．このとき読み出し要求同様，必要があれば書き戻しを行う．次に，*share request* ビットを調べ，セットされていない場合，必要に応じて読み出しラインを発生元に 1 対 1 通信で転送し，無効化要求をマルチキャストする．セットされている場合，必要に応じて読み出しラインを発生元に 1 対 1 通信により送るとともに無効化および再構成要求をマルチキャストする．いずれの場合も応答メッセージを収集し終わったら，ホームのディレクトリの状態を **D** とする．応答メッセージを収集する前に同一ラインに対して次の要求を受け取った場合，その要求の処理は待たされる．

書き込み要求を出したプロセッサは，ホームから転送されたラインにデータを上書きし，キャッシュに格納しその状態を **O** とする．その他の無効化要求を受けたプロセッサは，そのラインの状態を **I** とする．

2.4.3 Pruning Cache の再構成

Pruning Cache のエントリは，追い出される可能性がある．また，それぞれのプロセッサのキャッシュラインは追い出される可能性があり．その状態は現在の共有関係を正確に反映するものではない．しかし，エントリが存在しない場合は，RHBD のビットマップによって，共有される可能性のあるプロセッサすべてにメッセージが送られる．また，余分なメッセージを受け取ったプロセッサは，*acknowledge* ビットをリセットして応答メッセージを返すことにより，それ以降のメッセージを受け取らずに済む．したがって，無駄なメッセージが増えることがあっても，キャッシュの一貫性に問題を生じるわけではない．

また新たに，共有するプロセッサが増え，Pruning Cache を再構成する必要が生じた場合，本来はそのプロセッサが属する枝だけを再構成すればよい．しかし実装によっては，そのプロセッサがどの枝に属するのかを特定するのに莫大なコストがかかる．一方，RHBD 方式を用いた場合，無駄なメッセージは生ずるものの，全体にブロードキャストすることに比べればはるかに少ないメッセージ数で共有する可能性のある枝すべてに再構成メッセージをマルチキャストすることができる．そこで本方式では，再構成が必要な場合，いったん，現在のエントリを無効にして，RHBD 方式を用いて共有する可能性のある枝すべてに再構成要求をマルチキャストする方法を選択した．

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，どの程度の確率でパケットがエントリに一致して無駄な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い，ヒット率を評価した．評価には，並列ベンチマークプログラム集 SPLASH2[17] に含まれる 4 つのプログラム (FFT，LU，Ocean，Radix) を並列計算機シミュレータ ISIS[18] で実行し，このとき出力されるアドレストレースデータを用いた．なお，アドレストレースの制限からノード数は 64 とし，ディレクトリ管理には 3 階層の 4 進木を用いた．

3.1.1 メッセージ発生回数

分散共有メモリの管理は，超並列計算機 JUMP-1 と同様に，コヒーレンス維持は 32byte ごとのライン単位で行われるが，共有プロセッサを示すディレクトリは 4Kbyte ごとのページ単位で作成される．また，分散共有メモリのコヒーレンス維持には更新型プロトコルを用いる．このとき，分散共有メモリ管理のために発生するメッセージ数，その本来の宛先数，および RHBD 方式を用いたときに実際に送られる宛先数を表 **1** に示す．

表 **1** メッセージの宛先数
Table 1 Number of destination.

	発生回数	本来の宛先数	実際の宛先数
FFT	4028482	110154548	144791521
LU	4304719	66863862	134106632
Ocean	53399253	957170367	2696172050
Radix	296035	11500186	14738705

この結果から，共有分散メモリ管理に RHBD 方式を用いた場合，本来の宛先数の約 1.5 ~ 3 倍のメッセー

表 2 Pruning Cache のヒット率
Table 2 Hit ratio of Pruning Cache.

	direct map			2 way set associative			4 way set associative		
	16	32	64	8×2	16×2	32×2	4×4	8×4	16×4
FFT	53.86%	84.12%	93.97%	57.74%	86.15%	96.38%	56.52%	88.30%	96.68%
LU	95.84%	99.75%	99.82%	96.64%	99.96%	99.99%	97.32%	99.97%	99.99%
Ocean	60.59%	94.96%	99.39%	66.35%	95.44%	99.72%	69.78%	99.39%	99.78%
Radix	60.24%	68.73%	74.47%	68.73%	75.61%	78.73%	73.76%	77.96%	79.56%

ジが送られてしまうことが分かる．したがってこの無駄パケットの削減，つまり Pruning Cache のヒット率が重要な意味を持つ．

3.1.2 Pruning Cache のヒット率

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が 16，32，64 について，それぞれ Direct map，2 way set associative，4 way set associative の各構成において，Pruning Cache のヒット率を測定した．なお，Pruning Cache のエントリの追出しアルゴリズムは LRU を用いた．前述のアドレストレースデータを用いた際の Pruning Cache のヒット率の測定結果を表 **2** に示す．

エントリ数が 16 の場合，一部を除いてヒット率は 60%前後であるが，エントリ数が 32 を超えるとヒット率が急激に向上し，64 の場合では，ほぼ 90%を上回るヒット率が得られた．同一エントリ数では当然 way 数が多いものの方がヒット率が高いが，エントリ数が 32 を超えると，direct と 2 way の差に比べて，2 way と 4 way の差は大きくない．当然ながら 4 way では比較器を 4 セット必要とするため，実装を考慮すると 2 way で 32 エントリ (16×2) 程度の構成が有利である．

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 **5** 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 1 つのノードから図 **6** 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．

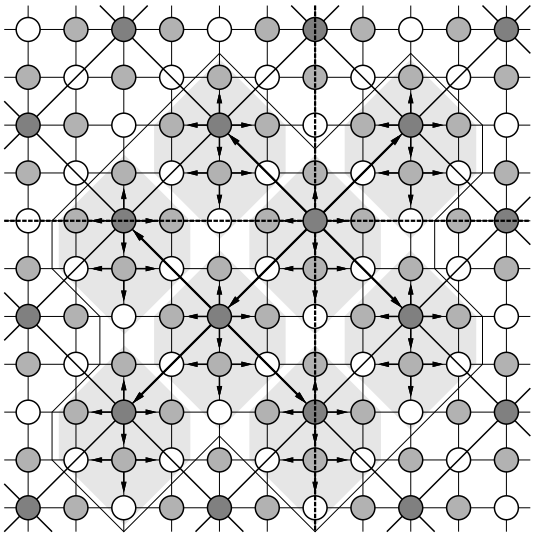


図 **5** 相互結合網 RDT
Fig. 5 Recursive Diagonal Torus.

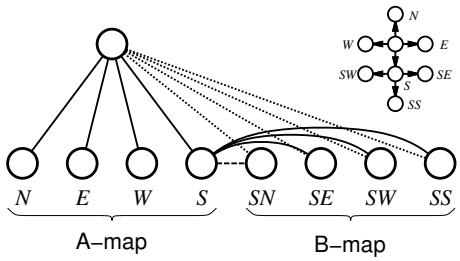


図 **6** RDT 上での 8 進木
Fig. 6 8-ary tree on the RDT.

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 5 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

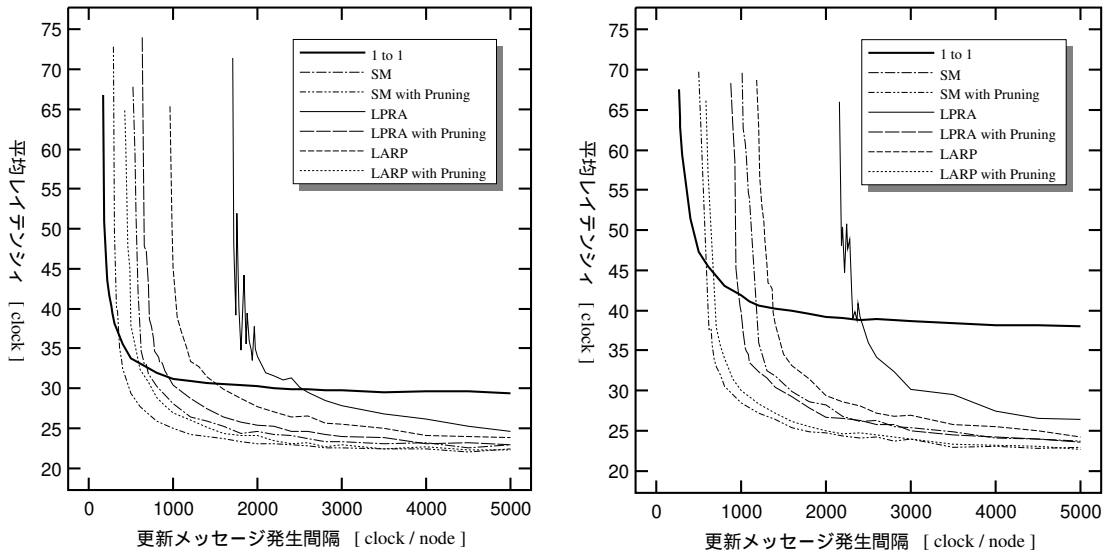


図 7 ランク 1 に Pruning Cache を装備した場合
Fig. 7 Latency of the combination (RANK-1).

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 64 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 用にすでに実装され，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) すべてを備えている．

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 (枝苅り率) 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) およびランク 0 (すなわち木の最下位層) 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すべての宛先に届くまでの平均レイテンシを図 7 および図 8 に示す．

Pruning Cache のヒット率は，前節の評価ではエントリ数が 64 の場合ほぼ 90%を超えていたが，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，過去の評価 [12),13)] に基づき，4 および 6 とした．この評価では横軸は平均パケット発生間隔であり，これが小さくなるほど頻繁にパケットが発生するため結合網が混雑し，レイテンシが大きくなる．レイテンシ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

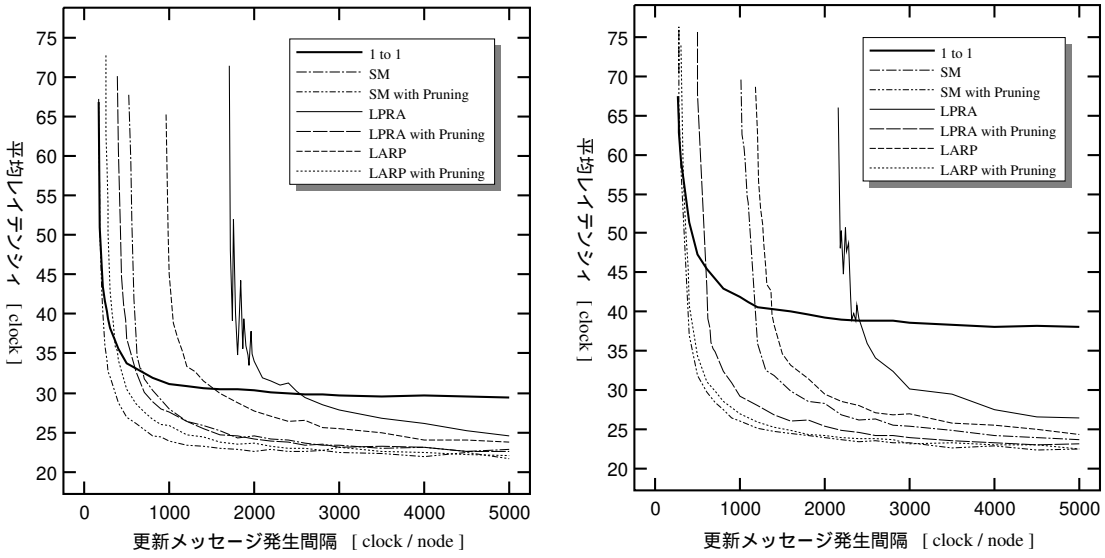


図 8 最下位層に装備した場合
Fig. 8 Latency of the combination (RANK-0).

め，従来のシステムが用いている方式として，マルチキャストを用いず，1 対 1 転送を宛先数分繰り返した場合についてもあわせて示す．

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 1 対 1 転送を行う場合に対してレイテンシが 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うことができるためである．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ことが分かる．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 **9** 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分かる．前節の評価から，ヒット率はかなり高いことが期待されることから，利用可能な転送容量も大きいことが期待される．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 らにより行われている [20)~22)]．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，概念の提示にとどまっている．また，他の方式との組合せによる，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[23)] 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われる．このため，無効化メッセージを 1 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[24)] 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

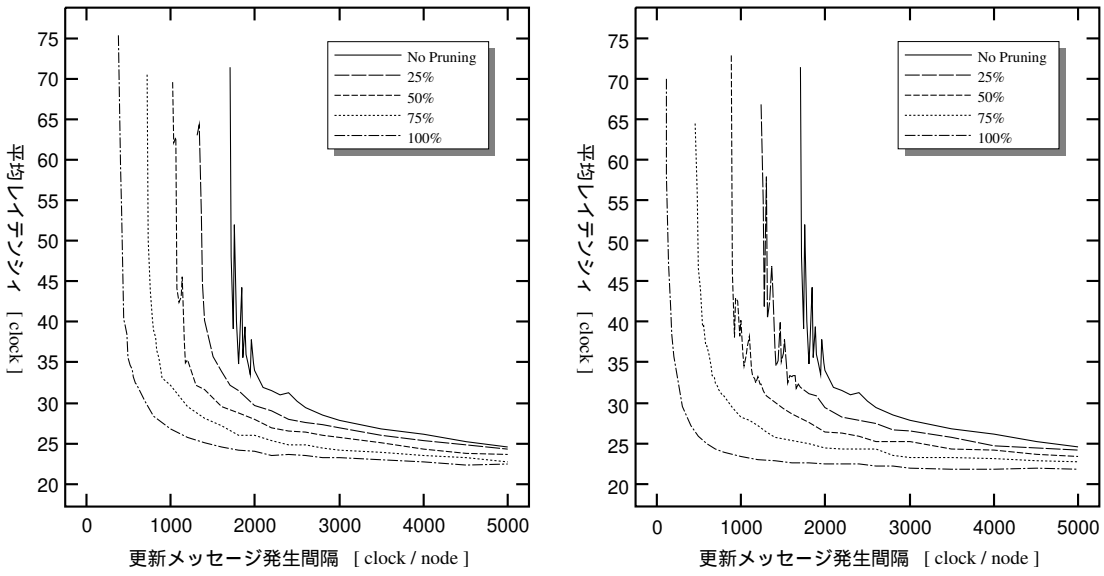


図 9 ヒット率の性能に及ぼす影響
Fig. 9 Various hit ratio.

インタをたどる必要があり，ルータ内ですべての処理が終了する 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 1 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%を超えるヒット率が得られることが分かった．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であることが分かった．

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ったが，大きいシステムは確率モデルを用いざるを得えなかった．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っていく方針である．

謝辞

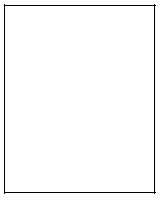
並列計算機シミュレータ ISIS を公開していただいた慶應義塾大学天野研究室の若林正樹氏，ならびにトレースデータの収集に協力してくれた同研究室の安生健一朗氏に感謝いたします。

参考文献

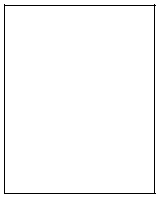
1) D.Chaiken and A.Agarwal: Software-Extended Coherent Shared Memory: Performance and Cost, *Proc. The 21st ISCA*, pp. 314 – 324 (1994).
2) D.Lenoski, et al: The Stanford DASH Multiprocessor, *IEEE Computer*, Vol. 25, No. 3, pp. 63 – 79 (1992).
3) J.Kuskin, et al: The Stanford FLASH Multiprocessor, *Proceedings. of The 21st International Symposium on Computer Architecture*, pp. 302 – 313 (1994).
4) Lovett, T. and Clapp, R.: STiNG: A CC-NUMA Computer System for the Commercial Marketplace, *Proc. The 23rd ISCA*, pp. 308 – 317 (1996).
5) Laudon, J. and Lenoski, D.: The SGI Origin 2000: A CC-NUMA Highly Scaleble Server, *Proc. of 24th ISCA* (1997).
6) A.Agarwal, et al: An Evaluation of Directory Schemes for Cache Coherence, *Proc. 15th ISCA*, pp. 280 – 289 (1988).

7) D.V.James, et al: Distributed-Directory Scheme: Scalable Coherent Interface, *IEEE Computer*, Vol. 23, No. 6, pp. 74 – 77 (1990).
8) M.Thapar and B.Delagi: Distributed-Directory Scheme: Stanford Distributed-Directory Protocol, *IEEE Computer*, Vol. 23, No. 6, pp. 78 – 80 (1990).
9) D.Chaiken, J.Kubiatowwicz and A.Agarwal: LimitLESS Directories: A Scalable Cache Coherence Scheme, *Proc. ASPLOS IV*, pp. 224 – 234 (1991).
10) 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).
11) M.D.Hill, J.R.Larus and D.A.Wood: Tempest: A Substrrate for Portable Parallel Programs, *Proc. COMPCON '95*, pp. 327 – 332 (1995).
12) 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).
13) 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).
14) Tanaka, H., Muraoka, Y., Amamiya, M., Saito, N. and Tomita, S.(eds.): *The Massively Parallel Processing System JUMP-1*, オーム社 (1996). ISBN4-274-90083-5.
15) Choi, L. and Yew, P. C.: Compiler and Hardware Support for Cache Coherence in Large-Scale Multiprocessors Design Considerations and Performance Study, *Proc. 23th ISCA* (1996).
16) 天野英晴: 並列コンピュータ, 昭晃堂 (1996). ISBN4-7856-2045-5.
17) J.P.Singh, W.Weber and A.Gupta: SPLASH: Stanford Parallel Applications for Shared-Memory, Tech. report, computer system laboratory, Stanford University (1992).
18) 若林正樹, 寺澤卓也, 山本淳二, 天野英晴: 並列計算機シミュレータ ISIS の実装, 電子情報通信学会総合大会講演論文集 情報・システム, p. 80 (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).
20) S.L.Scott: A Cache Coherence Mechanism For Scalable, Shared-Memory Multiprocessors, *Proc. 1991 ISSMM*, pp. 49 – 59 (1991).
21) S.L.Scott: *Toward The Design Of Large-Scale, Shared-Memory Multiprocessors*, PhD Thesis, University of Wisconsin (1992).
22) 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).
23) 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 (1996).
24) H.Nilsson and P.Stenstrom: The Scalable Tree Protocol – A Cache Coherence Approach for Large Scale Multiprocessors, *Proc. 1992 SPDP*, pp. 498 – 506 (1992).

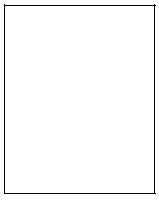
(平成 9 年 11 月 7 日受付)
(平成 10 年 4 月 ? 日採録)



西村 克信 (学生会員)
平成 6 年東京工科大学工学部情報工学科卒業．平成 8 年慶應義塾大学大学院理工学研究科計算機科学専攻修士課程修了．現在，同大学院後期博士課程に在学中．ほかに，FPGA を用いた教育用マイクロプロセッサシステムに興味を持つ．



工藤 知宏 (正会員)
昭和 61 年慶應義塾大学工学部電気工学科卒業．平成 3 年同大学大学院理工学研究科博士課程単位取得退学．東京工科大学工学部情報工学科助教授を経て，現在は新情報処理開発機構に所属．工学博士．軽量通信・メモリアーキテクチャの開発に従事．



天野 英晴 (正会員)
昭和 56 年慶應義塾大学工学部電気工学科卒業．昭和 61 年同大学大学院工学研究科電気工学専攻博士課程修了．現在，慶應義塾大学理工学部情報工学科助教授．工学博士．計算機アーキテクチャの研究に従事．