軽いハードウェアによる分散共有メモリ機構

田中清史[†,††] 松本尚[†] 平木敬[†]

並列/分散システムにおいて，汎用かつ容易なプログラミング環境を提供するために共有メモリが有効である．本論文では階層コヒーレンス管理および一般化されたコンバイニングのサポートにより効率の良い分散共有メモリを軽いハードウェアで実装する方式を提案する．提案する方式ではディレクトリに必要なメモリ量は要素プロセッサ数の対数オーダーである．このことから，超並列システムを構築する場合に各メモリブロックについて1ワード程度用意すれば十分であり，ディレクトリのアクセスコストが小さい．並列計算機プロトタイプお茶の水5号上に軽いハードウェアDSMおよび一般化されたコンバイニングを実装し評価を行った結果，我々の方式が並列化の効果を得ることが示された．

Lightweight Hardware Distributed Shared Memory

KIYOFUMI TANAKA,[†,††] TAKASHI MATSUMOTO[†] and KEI HIRAKI[†]

On a parallel/distributed system, it is necessary to provide efficient shared memory mechanisms for a general and convenient system. In this paper, we describe a lightweight method for constructing an efficient distributed shared memory system supported by hierarchical coherence management and generalized combining. In our method, the amount of memory required for directory is proportional to the logarithm of the number of clusters. This implies that only one word for each memory block is sufficient for covering a massively parallel system, and access costs of the directory are small. We have developed a prototype parallel computer, OCHANOMIZ-5, that implements this lightweight distributed shared memory and generalized combining with simple hardware. The results of evaluating the prototype show that our methodology provides the advantages of parallelization.

1. はじめに

並列/分散システムにおいて汎用性を維持し，容易なプログラミング環境を提供するためには共有メモリが有効である．その際，遠隔データ参照が実行性能を落とすのを避けるために共有データをキャッシュすることが求められる．キャッシュ方式を採用することはコンシステンシ維持管理が必要であることを意味する．分散共有メモリ（DSM）は，集中共有メモリ（SMP）が実現可能な規模[5]を超えた共有メモリシステムを構築する最も有力な方式である．DSMの実現方式は，ハードウェア，ソフトウェアによるアプローチともすでに多数提案されてきた[16),18)]．DSMを実現するためには，要素プロセッサの持つキャッシュメモリへのデータコピーの生成，キャッシュミス/ヒットおよび共有の有無等キャッシュされたブロックの状態の判定および判定結果等に基づくコヒーレンス管理，特に書き込み時のコヒーレンス維持操作の実行が必要である．要素プロセッサにおけるソフトウェア実行によりデータコピーの生成，キャッシュミス/ヒットの判定およびコヒーレンス管理を実現するアプローチでは，要素プロセッサを時分割でキャッシュ管理に使用するためソフトウェアオーバヘッドが大きい．

IVY[15]を源とする仮想共有メモリは，プロセッサが持つページ管理機構，すなわちプロセッサ内のTLBを含むMMUを使用し，ページフォルトトラップの発生時にソフトウェアにより遠隔メモリアクセスとコヒーレンス維持操作を起動する．共有されているデータへの書き込み時にはコンシステンシをとるためのソフトウェアオーバヘッドが存在する．また，データ転送/管理単位が基本的にページ単位の大きさであるためfalse sharingによりネットワーク性能が無駄に使用され性能が低下する可能性が大きい．

一方，プロセッシングノードにDSM管理を行う専用ハードウェアを用意することにより，ソフトウェア

† 東京大学大学院理学系研究科情報科学専攻
Department of Information Science, Faculty of Science, University of Tokyo
†† 日本学術振興会特別研究員
JSPS Research Fellow

オーバヘッドを削減することが可能である．しかし，ソフトウェアによる方法が特別なハードウェアを必要とせず，市販のPCやワークステーションクラスタ上で実現可能であるのに対し，DSM実現のために付加するハードウェアのコストを考慮しなければならない．ソフトウェア分散共有メモリにおいて，緩和されたコンシステンシモデル[8),12)]や最適化コンパイラ[10),20)]の利用により実行性能が向上してきているが，それに対してハードウェアによるシステムがコストに見合う性能向上を得られなければその存在意義がなくなる．

DSM実現のためのハードウェアコストを引き上げる主な要因は，共有状態を保持するディレクトリメモリの構成およびメモリ量，ネットワークにおける共有に必要な通信バンド幅の確保，またシステムによってはキャッシュ管理に特化したプロトコルプロセッサの使用があげられる．

ディレクトリに関して，共有情報を完全に持つものとして，Fullmapディレクトリ[3)]，階層bit-mapディレクトリ[7)]，LimitLESSディレクトリ[4)]，chainedディレクトリ[11),22)]がある．Fullmapディレクトリはハードウェア量が$O(n)$であり，大規模システムには適用が困難である．また，メモリブロックごとのディレクトリのサイズが1回のアクセスで参照できる幅を超えると，多段アクセスが必要となるため効率が悪い．LimitLESSディレクトリでは，共有数が多い場合にプロトコルプロセッサあるいは要素プロセッサによるソフトウェア実行が必要となることが大きなオーバヘッドとなる．同様にchainedディレクトリはリンクをたどるために，共有数が多い場合にレイテンシが大きく，無効化型プロトコルの使用により共有数を低く抑える必要がある．階層bit-mapディレクトリでは，同一メモリブロックの共有数が多くなるにつれ，メモリ使用量が大きくなる．したがって，無効化型プロトコルの使用によってメモリ使用量を抑えることが重要となる．また，各階層でメモリアクセスが必要となるため，この方法はレイテンシを増大させる傾向があり，これを低く抑えるために非常に高速なメモリが必要となる．

したがって，共有に関する完全情報を持つディレクトリ方式は大規模システムにおいて，ハードウェア量が大きいか，レイテンシが大きいか，あるいはプロトコルプロセッサ/プロトコル処理のオーバヘッドが大きいということが問題となる．

一方，共有に関する情報を不完全に持つことによって，プロセッサ台数に比例しないメモリ量でディレクトリを実現する方式として，Limitedディレクトリ[2)]，疑似fullmapディレクトリ[25)]がある．Limitedディレクトリは，ディレクトリのアクセス幅が大きいという問題があり，また，共有数がlimitより大きい場合に全体へのbroadcastが必要となり，通信コストが非常に大きくなる．したがって，この方法は無効化型プロトコルと組み合わせることが必須となる．疑似fullmapディレクトリ方式では，ディレクトリが階層ごとの情報を持つため，読み出し幅が小さくないこと（32ビット程度），またマルチキャストによる通信の増加が問題となる．

プロトコルプロセッサは実現に要するハードウェア量がコストを引き上げるとともに，SMPシステムではすべてハードウェアで実現されるメモリ制御にプロセスの起動，命令実行を介在させるために大きなオーバヘッドを引き起こす．また，プロトコルプロセッサは構造が複雑であるために長い開発期間を要する．並列計算機で市販のプロセッサを使用する場合には，プロトコルプロセッサをはじめ他のシステム構成要素の設計がプロセッサの外部インタフェースに大きく依存する．プロセッサの高速化および世代交替が急速であることを考慮すると，システムの開発期間が長くなることはコスト上昇に匹敵する障害となる．

したがって，ハードウェアDSMシステムの構築には，低コストで短期間に実現可能な方式を求めることが課題である．我々は以下の要素を取り除くことにより，軽いハードウェアによるDSMを実現する．

(1) プロセッサ数に比例した量のディレクトリメモリ
(2) ディレクトリ，タグおよび状態を格納する専用メモリ
(3) 専用プロトコルプロセッサ

本論文では，プロトコルプロセッサを含まない軽いハードウェアによる分散共有メモリ方式を提案する．提案する方式は共有するプロセッサへの距離を用いたスケーラブルなディレクトリ構成を持ち，また，ディレクトリ，タグおよび状態を主記憶内に格納することにより，分離した高速なSRAM要素の使用によるコストの上昇を避ける．また専用プロトコルプロセッサを使用しないことによってプログラム実行の介在しない低オーバヘッドの制御および低コスト化を実現する．ただし，提案する方式ではディレクトリ情報を簡素化し，プロトコルプロセッサを省略する代償としてコンシステンシ維持管理に必要なネットワーク通信量が増大するが，動的な階層マルチキャスト/コンバイニング機構の利用により通信の増加を補い，効率の良い共有メモリを実現する．本方式は基本的な分散メモリ型並列計算機に対して，メモリコントローラとネットワークのスイッチングノード内に少量の論理回路を付加す

ることによってCC-NUMAの実現を可能とする．

お茶の水5号は，クラスタ化し階層構造を持つ並列計算機アーキテクチャを評価するための汎用テストベッドである[27]．お茶の水5号は様々な方式の評価を行うために，メモリコントローラやネットワークノードに大規模FPGAを使用している．本論文では，お茶の水5号に一般化したコンバイニング機構と提案するDSM方式を実装し評価した．

本論文は次の構成をとる．2章では，軽いハードウェアによるDSMアーキテクチャを提案する．3章では，軽いハードウェアによるDSMの実装を述べ，4章で評価する．さらに，5章で関連研究を示し，6章でまとめる．

2. 軽いハードウェアによるDSMアーキテクチャ

本論文で提案する方式では粗い共有管理方式である階層最大共有距離ディレクトリを採用する．これは疑似fullmapディレクトリ[25]における共有情報を管理する方式をさらに簡略化したものである．階層最大共有距離ディレクトリ方式では，ディレクトリに必要なメモリ量は要素プロセッサ数の対数オーダーとなる．また，ディレクトリ，メインメモリに相当するレベルのキャッシュタグおよび状態は別個の専用ディレクトリメモリではなく主記憶内に格納し，それらの管理をメモリコントローラが担当する．すなわちメモリコントローラはプロセッサあるいは外部からのリクエストに従って共有情報を操作する．この方法は従来のようなプロトコルプロセッサを必要とせず，キャッシュ管理においていかなるプログラム実行も介在しない．

このような管理方法の簡略化は，通信パケット数の増加を引き起こすが，ネットワークによる動的な階層マルチキャスト/コンバイニングを使用することで共有操作（コンシステンシ維持操作）の逐次通信化を避けることにより，通信量の削減および高速化を実現する．

2.1 階層最大共有距離ディレクトリ方式

ネットワークとして木構造が埋め込み可能なものを仮定する．各メモリブロックに対して静的にHomeプロセッサ☆を割り当てる．Homeプロセッサは“最大共有距離”によって共有情報を記録する．最大共有距離はHomeプロセッサとメモリブロックのコピーを持つ最も遠い要素プロセッサとの間の距離である．言い換えると，キャッシュしているすべての要素プロセッサを含む最小部分木の高さである．図1に階層最大共有距離ディレクトリを示す．図中の黒色の葉はコピーをキャッシュしている要素プロセッサを示し，灰色の空間は共有空間を示す．

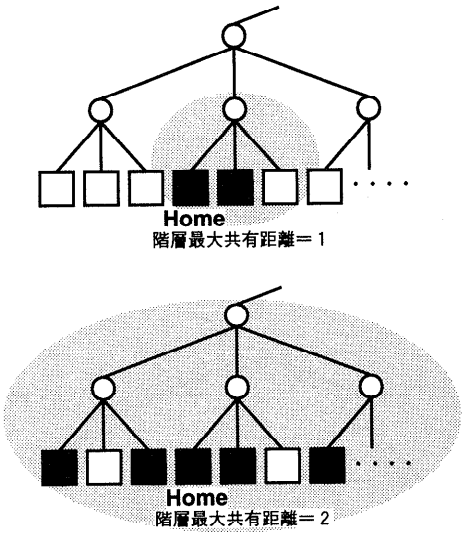


図1 階層最大共有距離ディレクトリ
Fig. 1 Hierarchical coarse directory.

ディレクトリに必要なメモリ量は要素プロセッサ数の対数オーダーである．Limitedディレクトリに代表されるコピー数の制限を設けることなしに少ないメモリ量を実現する．各メモリブロックにつき16ビットワードを用意することで1万台を超える超並列システムに対応できる．

また，ディレクトリの幅が小さいことから，ディレクトリ参照に多段アクセスの必要がないために，ディレクトリを主記憶（DRAM）に格納することによるメモリアクセスオーバヘッドを隠蔽することが可能である．

本方式では，最大共有距離が示す共有空間内のすべての要素プロセッサがキャッシュコピーを持っていると見なしてコヒーレンス処理を行う．コピーのインバリデート処理の際に，Homeプロセッサは共有空間内の全要素プロセッサにメッセージをマルチキャストするため余分な通信が生じる．実際にコピーを持っていない要素プロセッサ（図中の白色の葉）がメッセージを受け取った場合はダミーのacknowledgementメッセージを返送する．次にこのような余分な通信が実行効率を妨げない方法を述べる．

2.2 階層マルチキャスト/コンバイニング機構

キャッシュコンシステンシ管理において，インバリデートまたはアップデートメッセージに代表される管

☆ Homeプロセッサに相当するものが，バス結合の計算機クラスタである場合は，Homeクラスタと読み替える．

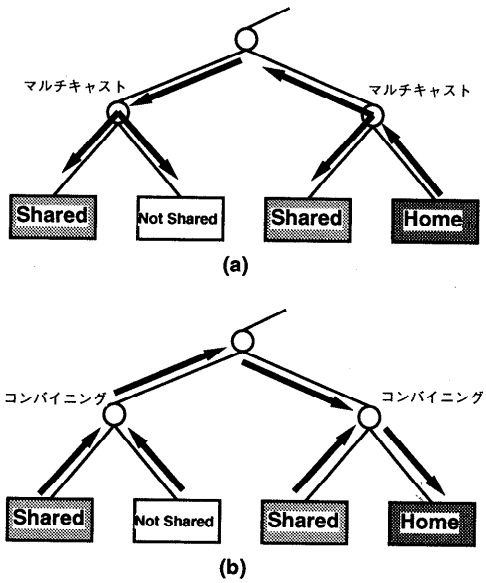


図2 マルチキャスト/コンバイニング
Fig. 2 Multicasting and combining.

理メッセージを，対象であるメモリブロックを共有する複数かつすべての要素プロセッサへ通信する必要がある．その際，階層マルチキャストを利用することにより通信効率を上げることが可能である．たとえば，共有コピーのインバリデートを行うとき，Homeプロセッサはインバリデートメッセージを1回だけ発行する．ネットワーク内の各スイッチングノードはこのメッセージを受け取った場合それを共有空間内のすべての方向へマルチキャストする．**図2**(a)に最大共有距離が2のときのマルチキャストを示す．

一方，複数のメッセージが同一の要素プロセッサに送られるときは，ネットワーク内でそれらを1つにコンバイニングする．これによりメッセージを受け取る要素プロセッサにおいて逐次に処理する必要がなくなる．たとえば，前述のインバリデートメッセージを受け取った要素プロセッサはプロセッサ内のインバリデートを完了した後acknowledgementメッセージ（コピーを持っていない場合はダミーのacknowledgement）をHomeプロセッサへ返送する．各スイッチングノードはマルチキャストしたすべての方向からこのacknowledgementメッセージが返ってきたとき，1つのメッセージをHomeプロセッサの方向へフォワードする．この操作を図2(b)に示す．

階層マルチキャストおよびコンバイニングはHomeプロセッサでの逐次的処理の必要性を除去する．Homeプロセッサは共有数にかかわらず，共有空間内の最も離れたプロセッサとの間の1回のround-tripレイテンシで共有空間のコヒーレンス処理を完了する．また，階層構造を考慮して近傍による局所性を利用した最適化/スケジューリングと組み合わせることにより，粗いディレクトリ情報による損失を補う．共有数が大きい場合でもディレクトリのビット幅が小さいこと，およびマルチキャスト/コンバイニングの利用により，インバリデート型だけでなくアップデート型プロトコルの使用が現実的に可能となる．

acknowledgementメッセージのコンバイニングについては，一般化されたコンバイニング機構[28]のサポートにより実現する．

2.3 一般化されたコンバイニングのサポート

階層距離ディレクトリ構成，プロトコルプロセッサの除去，あるいはキャッシュライン単位の管理といったハードウェアの簡略化はネットワーク内の通信量を増大させ，ホットスポット[17]を引き起こす原因となる．コンバイニング[6]はホットスポットを軽減する方法として研究されてきた．同一ターゲットへの同一リクエストどうしがネットワーク内で衝突した場合に，動的にそれらを1つのリクエストに置き換えることによってネットワーク内の通信量を削減する．しかし文献6)で提案されたコンバイニング方式はリクエストどうしがスイッチングノード上で衝突するという，きわめて偶発的な事象を前提としている．また，スイッチングノード内にキューの段数と同じ数の比較器を用意するため，必要ハードウェア量が大きい[17]．

我々は従来のコンバイニングを拡張することによって，柔軟なコンバイニングを実現する一般化されたコンバイニングを提案した[28]．一般化されたコンバイニングは到着条件の一般化，演算機能の一般化およびマッチング条件の一般化の3つにより構成されるが，軽いハードウェアDSMの実現において効果的なサポートとなる到着条件の一般化を以下に示す．

到着条件の一般化

スイッチングノードにおけるメッセージの到着時間に対して，マッチングが起こる時間は“過去”，“現在”，“ノード上での遅延時間内”および“未来”の4つの場合に分類される．過去はメッセージがノードに到着したときに，そのメッセージに関するマッチングがすでに完了していることを意味する．現在は到着と同時にマッチングが行われることを意味する．ノード上での遅延時間内は，メッセージが到着してからノードから出ていくまでの間に後続メッセージとマッチングが行われることを意味する．従来のコンバイニングはすべてこの遅延時間内のマッチングに属する．未来はメッセージがノードを通過した後でマッチングが行われる

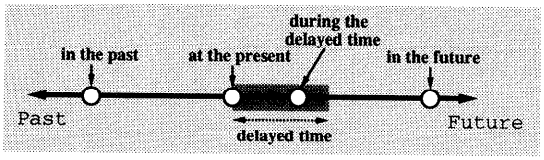


図3 到着条件の一般化—マッチングポイント
Fig. 3 Generalization of arrival requirement – matching point.

ことを意味する．図3に4種類のマッチングポイントを示す．

遅延時間について，ゼロから無限大まで設定可能であるとする．ゼロの場合，メッセージは後続する到着メッセージを待つことなくノードを通過する．無限大の場合は対象となるメッセージがすべて到着するまでノード上で待機する．ゼロから無限大の間の時間を設定した場合は，最大その時間が経過するまで後続メッセージを待つ．

到着条件の一般化により，メッセージがノード上に存在する間のみ，言い換えると，メッセージがコンバイニングキューを通過する間のみにコンバイニングが起こりうるという制限は除去される．すなわち，メッセージどうしの到着時間のずれに対応可能で，コンバイニングの成功率を上げることが可能である．

2.4 実現方式

遠隔メモリブロックを参照する場合，プロセッシングノードの主記憶内にコピーを生成する．これをクラスタレベルキャッシュと呼ぶ．各メモリブロックに対して静的にHomeプロセッサが割り当てられるが，主記憶上でそのメモリブロックがオリジナルであるかコピーであるかは区別しない☆．クラスタレベルキャッシュの状態はメモリコントローラが管理する．軽い制御を実現するため，コヒーレンス処理単位はキャッシュブロックサイズとし，状態は次の3状態に限定する．

- **private**
  メモリブロックの有効なコピーはそのプロセッシングノードのみに存在する．
- **shared**
  複数のプロセッシングノードが有効なコピーを持つ．sharedのときは必ずHomeプロセッサは有効なコピーを持つ．
- **invalid**
  そのプロセッシングノードはすべての階層のキャッシュ（プロセッサキャッシュ，クラスタレベルキャッシュ）内で有効なコピーを持たない．Homeプロセッサの状態がinvalidの場合は，他の唯一のプロセッシングノードにおいてprivateなコピーが存在する．

☆ 以下，主記憶上のメモリブロックの実体をコピーと呼ぶ．

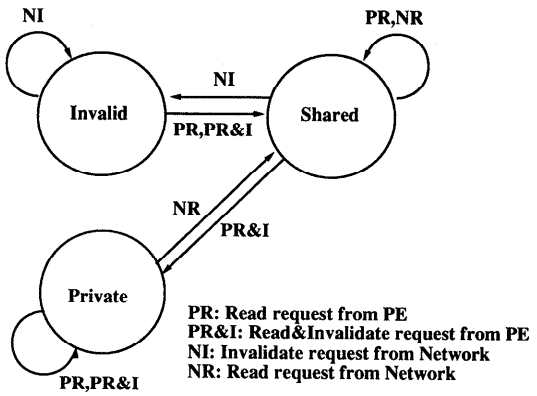


図4 クラスタレベルキャッシュの状態遷移
Fig. 4 State transition diagram of cluster-level cache.

図4にクラスタレベルキャッシュの状態遷移を示す．

物理メモリアドレス空間を上位アドレス（4ビット）によって局所メモリ空間と共有メモリ空間に分割する．メモリコントローラはクラスタ内バスに発行されるリクエスト（物理アドレス）を受け取り，上位アドレスを調べて，それが共有アドレスのときは対応するタグ/状態を読み出し，その内容に従って要求者に対してデータの返送あるいはリトライを返す．リトライの場合は必要に応じてネットワークインタフェースに通知してネットワークトランザクションを起動し，同時に状態を書き替える．図5にメモリコントローラの動作を示す．

到着条件の一般化の実現のために，ネットーワークメッセージを次の4種類で分類する．

- **レベル0**
  遅延時間（待ち時間）はゼロであり，スイッチングノード内のコンバイニングユニットを通過しない．すなわち，コンバイニングの対象とならないメッセージである．
- **レベル1**
  待ち時間はゼロであるが，スイッチングノード内のコンバイニングユニットを通過する．メッセージがリクエストタイプで，かつ該当アドレスへの最初のリクエストならばウェイトバッファ内にエントリをセットし，メッセージをフォワードする．最初ではない場合はフォワードせず，コンバイニングの完了を示すようにエントリを更新する．リプライが到着したときに，エントリの情報に従ってマルチキャストする．

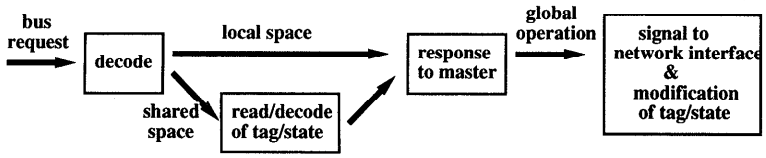


図5 メモリコントローラの基本動作
Fig. 5 Control sequence of the memory controller.

- **レベル2**
  メッセージがスイッチングノード上で指定された時間待機することを意味する．実際の待ち時間は4段階で指定可能とする．
- **レベル3**
  待ち時間が無限大であることを意味する．コンバイニングの対象となるメッセージがすべて到着した時点で1つのメッセージがフォワードされる．予測可能型コンバイニングを実現する．

レベル1のメッセージに関して，コンバイニングが成功するためにメッセージどうしがスイッチングノード上で衝突する必要はない．予測不可能型のコンバイニングに対する，低オーバヘッドかつ到着時間の差に対する制約が最も緩和された方法である．

ネットワークトランザクションの例として，shared状態のブロックに対するインバリデートの処理を取り上げる．2つのプロセッサが同一メモリブロックに対してインバリデートリクエストを発行した場合を示す．以下がステップである（図6）．

(1) CPUが共有ブロックへのコヒーレントインバリデートを発行した場合，そのクラスタノードは該当ブロックのホームノードへレベル1のインバリデートリクエストを発行する（図6(a)）．

(2) スイッチングノードにこのリクエストが到着したとき，ウェイトバッファを探索する．先行する同じリクエストがなかったらウェイトバッファにエントリをセットし，(レベル1であることより）リクエストを要求先にフォワードする（図6(b)）．すでに先行リクエストのエントリがあった場合，コンバイニングを行ってリクエストをフォワードしない（図6(c)）．このとき最終的なAcknowledge（以下Ack）メッセージをマルチキャストすることを示すためにエントリを更新する（図6(d)）．

(3) ホームノードはこのインバリデートリクエストを受け取ると，該当エリア（共有距離内）にレベル0のインバリデートメッセージを1つだけ発行する（図6(e)）．

(4) スイッチングノードにこのインバリデートメッセージが到着したとき，該当エリアにマルチキャスト（図6(f)）〔ただし共有エリアのルートならば，同時にウェイトバッファにレベル3のダミーAckエントリを挿入（図6(g)）〕する．

(5) このメッセージを受け取ったクラスタノードは，クラスタ内でインバリデート処理を行い，レベル3のAckメッセージを返送する．実際にはそのブロックコピーが存在しないクラスタや最初の要求元クラスタは，ダミーAckを返送する（図6(h)）．

(6) スイッチングノードにこのAckが到着したときウェイトバッファを探索し，先行Ackのエントリがなければエントリを挿入する（図6(i)）．エントリがあれば更新し，かつすべてのAckメッセージの到着が確認された場合のみホームノードの方向へフォワードし，エントリをクリアする（図6(k)）．

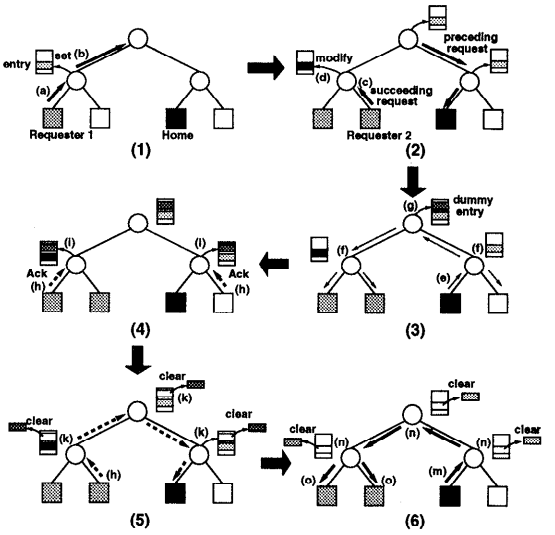


図6 インバリデートトランザクション
Fig. 6 A series of operations during invalidate transaction.

(7) ホームノードは該当 Ack が返送されてきたら，最初の要求元にレベル 1 の Ack メッセージをリプライとして返送する（図 6 (m)）．

(8) スイッチングノードにこの Ack が到着したとき，要求元へフォワード（必要ならばマルチキャスト，図中の Requester2 には NACK が送られる）し，エントリをクリアする（図 6 (n)）．

(9) 要求元に Ack が到着し，トランザクションが終了する（図 6 (o)）．

3. 軽いハードウェアによる DSM の実装

並列計算機プロトタイプ，お茶の水 5 号（OCHANOMIZ-5）[27]は，クラスタ化し階層構造を持つ並列計算機アーキテクチャを評価するための汎用テストベッドである．2 台の SuperSPARC+ [21]，32 Mbyte の主記憶，メモリコントローラ，バスアービターおよびネットワークインタフェースによってクラスタを構成する☆．クラスタ内は共有バス型マルチプロセッサを構成している．全体としては 4 つのクラスタを 2 進木階層構造の相互結合網で接続した構成をとる．**図 7** にお茶の水 5 号の構成を示す．

軽いハードウェア制御の DSM を実現するために，メモリコントローラおよびネットワークスイッチングノードに専用のロジックを実装した．

メモリコントローラの実装

メモリコントローラは主記憶に格納された共有情報の管理を担当する．メモリブロックごとのディレクトリ，状態タグは以下のように 8 台構成に特化したものを使用した．

- 最大共有距離 … 1 bit（0 のとき 1 段の共有，1 のとき 2 段の共有）
- valid/invalid ビット … 1 bit（0 のとき invalid，1 のとき valid）
- shared/private ビット … 1 bit（0 のとき private，1 のとき shared）
- owner ビット … 2 bit（Home が invalid の場合に，0〜3 番クラスタのどれが private かを示す）
- pending ビット … 1 bit（状態遷移の途中であることを表す）

また，プロセッサが局所アドレスへ 32 byte メモリブロックのリクエストを発行した場合，11 バスクロックサイクルで完了する．共有アドレスの場合はタグ/状態をデコードするのにさらに 4 サイクル必要となる．

☆ クラスタ内でメモリコントローラ，ネットワークインタフェースが別ユニットになっているが，現在利用可能な FPGA を用いると 1 チップでの実装は十分可能である．

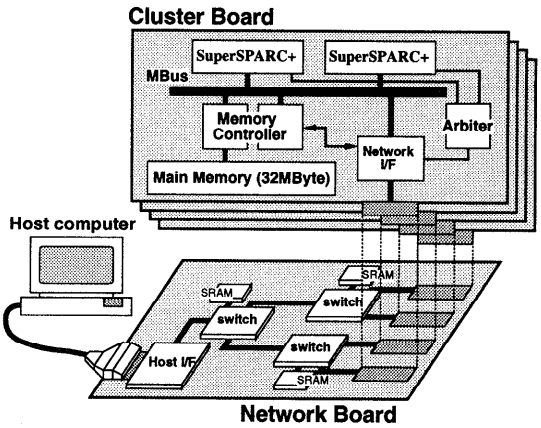


図 7 お茶の水 5 号の構成
Fig. 7 Block diagram of OCHANOMIZ-5.

表 1 メモリコントローラのデバイス使用率
Table 1 Device utilization of a memory controller.

	non-cache	cache DSM	max available
CLB	211	274	400
F&G FG	217	305	800
H FG	42	82	400
CLB FF	135	157	800

最初のミス時にデータがプロセッサキャッシュ内に読み込まれ，その後高い確率でヒットすればこのタグ/状態参照のコストは無視できるほど小さい．

メモリコントローラには FPGA [24]（XC4010：10,000 ゲート相当）を使用した．**表 1** に FPGA 内部の使用率を示す．表中で non-chache がキャッシュ管理を行わないメモリコントローラ，cache DSM が行うコントローラである．chache DSM は non-cache に対して 30〜40%多くのゲートを必要とするが，non-cache 自体が約 4,000 ゲートの小さい回路で構成されているのでこの増加分はわずかである．

ネットワークスイッチングノードの実装

提案した DSM 方式において，ネットワークスイッチングノードは階層マルチキャストおよび一般化されたコンバイニングを担当する．お茶の水 5 号のネットワークは以下のコンバイニングを行う．

- acknowledgement メッセージのコンバイニング [25]
- 階層化された Elastic Barrier [26]
- Fetch&Add [6]，Test&Set リクエストのコンバイニング
- Read/Invalidate リクエストのコンバイニング

各メッセージタイプとレベルの対応を**表 2** に示す．ネットワークの内部スイッチングノードは，コンバ

表2 メッセージレベル
Table 2 Levels of messages.

Level	Waiting time	Purpose
0	zero	Invalidate multicast
1	zero	Read, Invalidate, Test&Set
2	specified time	Fetch&Add
3	eternity	Acknowledgement, Barrier

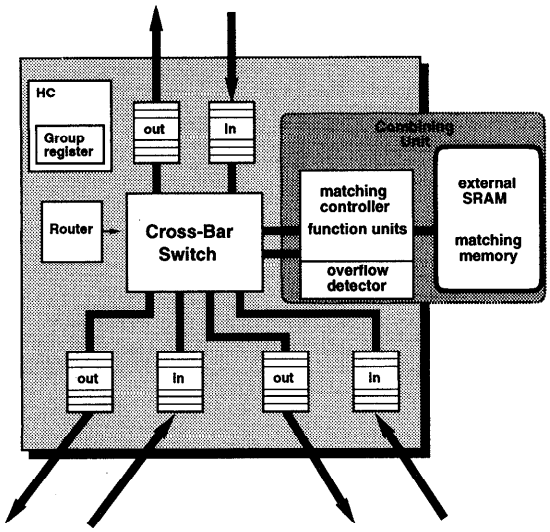


図8 スイッチングノードの構成
Fig. 8 Block diagram of switching node.

イニング機構を実現するために以下の要素から構成される（図8）.

- スイッチ要素およびそのコントローラ（Router）
  ノードの親，2つの子，およびコンバイニングユニットの間で，4×4 クロスバースイッチを構成する.
- 外部メモリで構成されるマッチングメモリ
  ウェイトバッファを構成するメモリ空間を外部のSRAMによって確保する．なお，バッファ探索のオーバヘッドを削減するためにハッシュ表を構成する.
- 演算器とコントローラ
  スイッチからコンバイニングユニットに入ってくるデータをレジスタにセットし，レジスタ-メモリ演算を行う.
- 溢れ検出回路
  各ハッシュキーに割り当てられた空間ごとに溢れを検査する．溢れを検出した場合は，割り込みパケットを生成し，近傍のプロセッサへ送る.
- 階層コントローラ（HC），グループレジスタ
  階層化されたElastic Barrierを実現するために，階層コントローラと同期グループを指定するグループレジスタを持つ.

表3 スイッチングノードのデバイス使用率
Table 3 Device utilization of a switching node.

	no-combine	combine	max available
CLB	400	670	1024
F&G FG	682	963	2048
H FG	74	159	1024
CLB FF	184	294	2048

我々のネットワークスイッチングノードはコンバイニングキュー[6]を含まず，コンバイニング機構はスイッチ内の簡単なwired-logicとウェイトバッファのためのメモリのみで構成される．スイッチングノードの実装はFPGA（XC4025：25,000ゲート相当）を使用した．**表3**に内部使用率を示す．表中のno-combineはコンバイニングユニットを含まないスイッチであり，combineは含むものである．コンバイニングユニットを含まないスイッチのハードウェア量は7,000ゲート程度であるのに対し，含むものは50%程度の増加がある．Pfister[17]らの見積りによれば，コンバイニングキューをスイッチ内に実装することによりハードウェアサイズ/コストは6~32倍になる．これに対し我々の方式は回路規模の点から低コストであるといえる.

4. 性能評価

4.1 基本通信性能

お茶の水5号における動作周波数はCPUが60 MHz，他のシステムが22 MHzである．ネットワークはcut-through方式で，データ転送幅は8ビットである．遠隔メモリアクセスのレイテンシおよびバンド幅をそれぞれ**表4**，**表5**に示す．表4の距離は通信の始点，終点のクラスタを含む最小木の高さを示している．サイクル数はプロセッサがリクエストを発行してから通信が完了までにかかるシステムクロック数である．コンバイニングの対象となるメッセージとならないメッセージについてそれぞれ示す．また，InvはHomeクラスタのプロセッサがインバリデートを発行してから，共有エリアのインバリデートを行い，Private状態になるまでのことを意味する.

表5は，指定された数のプロセッサ（PE）が隣（距離＝1）のクラスタに32 byteブロックのwriteおよびreadを連続して発行した場合の結果を示している．4台の実行は4つのクラスタ内でそれぞれ1つのPEを実行させた．writeに関して，4台のPEの実行が8台の実行より良い結果を得たが，これは各クラスタ内でのバスの競合によるものである．すなわち，4台

表4 遠隔メモリアクセスレイテンシ
Table 4 Latency of remote memory access.

距離	タイプ	サイズ (Bytes)	Combining (ON/OFF)	Cycle 数	μsec
1	Write	1–8	OFF	33	1.50
			ON	43	1.95
	Write	32	OFF	60	2.73
			ON	70	3.18
	Read	1–8	OFF	45	2.05
			ON	65	2.95
	Read	32	OFF	74	3.27
			ON	94	4.27
	Inv	32	ON	76	3.45
2	Write	1–8	OFF	41	1.86
			ON	71	3.23
	Write	32	OFF	68	3.09
		32	ON	98	4.45
	Read	1–8	OFF	61	2.77
			ON	121	5.50
	Read	32	OFF	88	4.00
		32	ON	148	6.73
	Inv	32	ON	138	6.27

表5 遠隔メモリアクセスのバンド幅
Table 5 Bandwidth of remote memory access.

Number of PEs	Type	Bandwidth (Mbyte/sec)
1	Write	15.2
	Read	7.3
4	Write	58.3
	Read	24.5
8	Write	47.8
	Read	30.8

表6 リードリクエストコンバイニングの効果
Table 6 The impact of combining read requests.

Combining ON/OFF	Bandwidth (Mbyte/sec)
OFF	19.3
ON	30.0

の場合は各クラスタ内でバスマスターになるのは，1つのPEとネットワークインタフェースの2つであるが，8台の場合はさらに1つのPEが加わる．

次にネットワークがread，Test&Setのコンバイニングを行うことによって得られる通信性能の向上を示す．**表6**に8台のPEが繰り返し同一クラスタの同一アドレスのメモリブロックを読み出した場合の結果を示す．この結果はコンバイニングが最大限に成功した場合を示しており，コンバイニングによる通信性能向上の限界値である．ネットワーク内で競合が起こらない場合（表5）と同程度のバンド幅が得られている．

Test&Setを使用してクリティカルセクションのロックの獲得を行った結果を示す．各PEは繰り返しロックを獲得し，セクション内でカウンタの値を1増やす．

表7 実行時間—ロック獲得ループ
Table 7 Execution time of lock acquisition loop.

Combining ON/OFF	Total time (seconds)
OFF	119.1
ON	85.7

以下がプログラムモデルである．

```
for (i=0; i<LOOP_COUNT; i++) {
  while(Test&Set(LockVar) != 0); // Lock Get
  counter += 1;           // Critical Section
  LockVar = 0;            // Unlock
}
```

表7に2^{20}回繰り返し実行した場合の実行時間を示す．コンバイニングにより，実行時間が28%短縮した．

4.2 コンバイニングの評価

数値計算を実行した場合のコンバイニングの効果を示す．行列$A[N][N]$のn乗（$N=128$, $n=32$）を計算するプログラムを実行した．

各PEは要素番号（cNO）を獲得し，その番号に対応する要素の計算を行う．ここで，$A[i][j]$の要素番号は$i \times N + j$ $(0 \leq eNO < N \times N)$である．要素番号は共有変数を使用し，クリティカルセクションで保護される．冪乗の各ステップの始めと終わりでバリア同期を行う．ここでソフトウェアバリアとハードウェアバリアを使用した．ソフトウェアバリアはクリティカルセクションと共有カウンタ変数で実現される．以下にアルゴリズムを示す．

```
shared double A[N][N];
local  double AA[N][N]; /* copy of matrix A */
for(k=0; k<nthpower; k++) {
  counter = 0;
  copy(A,AA);             /* copy from A to AA */
  barrier();
  while(1) {
    while(Test&Set(LockVar) != 0);
    eNO = counter;
    counter = eNO + 1;
    LockVar = 0;
    if (eNO < N*N) {
      double d = 0.0;
      for (i=0; i<N; i++)
        d += AA[eNO/N][i]*AA[i][eNO%N];
      A[eNO/N][eNO%N] = d;
    }
    else {
      barrier();
      break;
    }
  }
}
```

共有変数（$A, counter, LockVar$）はnon-cacheableアクセスとし，行列$A[N][N]$の実体はクラスタ0番に配置した．各プロセッサはループの始めで$A[N][N]$

表8 実行時間—行列の冪乗計算
Table 8 Execution time (seconds) of the power of a matrix.

Number of PEs	NC	HC	HLC	HLRC
1	19.9	19.9	19.9	19.9
2	10.1	9.9	9.9	9.9
4	7.3	6.5	6.6	6.7
8	6.5	6.4	6.3	6.0

のコピーをローカル（cacheable）メモリ内に作成する（$copy(A, AA)$）．この部分でreadリクエストのコンバイニングの効果を得る（32バイトのブロック転送リクエストを使用した）．

PEの台数が1，2，4，8のときの実行時間を**表8**に示す．2台の実行は，1つのクラスタ内の2つのPEを使用し，4台の実行は，2つのクラスタ内の4つのPEを使用した．表中のNCはコンバイニングを使用しない場合であり，HCはバリア同期にハードウェアバリアを使用した場合である．HLCはハードウェアバリアとTest&Setのコンバイニングの使用，HLRCはハードウェアバリア，Test&Set，readリクエストのコンバイニングを使用した場合である．4台の実行について，Test&Setおよびreadリクエストのコンバイニングは高速化に貢献していない．この場合，配列および他の共有変数を1つのクラスタ内のメモリに配置したために，通信方向は1方向のみである．このことからスイッチングノード上で複数の方向からのリクエストどうしのコンバイニングが生じない．また，通信パケットがスイッチングノード内のコンバイニングユニットを通過するために付加レイテンシを受けていることから，逆に実行時間が大きくなっている．8台の実行においては，HLRCが最も良い結果を得た．レイテンシの増加よりもコンバイニングの効果が大きいためである．8台の実行では，HLRCがNCに対して実行時間が7.7%短縮した．

4.3 ハードウェアDSMの評価

我々のハードウェアDSM方式を使用した結果を示す．評価プログラムとして，SPLASH-2[23]のLU-Contigを使用した．LU-Contigは密行列のブロック分割を用いたLU分解を実行する．行列のサイズは512×512で，ブロックのサイズは16×16で行った．PEの台数が1，2，4，8のときの実行時間を**表9**に示す．2台の実行は，1つのクラスタ内の2つのPEを使用し，4台の実行は，2つのクラスタ内の4つのPEを使用した．また，プログラム中のバリア同期には，ハードウェアバリアを使用した．表中のCache OFFはすべての階層のキャッシュ（プロセッサキャッシュ，クラスタレベルキャッシュ）を使用しない場合の実行を示している．Cache ONはすべてのキャッシュを使用した場合である．2台の実行が1台のときよりも2倍以上速くなっているが，これはトータルデータサイズが1台のPEのプロセッサキャッシュの容量よりも大きいことに起因したスーパーリニア効果のためである．全体として，Cache ONはCache OFFに対して実行時間を短縮し，また8台の実行が1台の実行に対して81%実行時間を短縮し，台数効果を得た．

表9 実行時間—LU分解
Table 9 Execution time (seconds) of LU-Contig.

Number of PEs	Cache OFF	Cache ON
1	149.36	10.57
2	64.39	4.96
4	72.32	2.71
8	47.18	1.97

5. 関連研究

ハードウェアDSMシステムの構成は専用プロトコルプロセッサの有無により大別される．プロトコルプロセッサを持たないシステムとして，DASH[14]，Alewife[1]，KSR1[9]があげられる．DASHはfullmapディレクトリを採用し，リモートデータのキャッシュおよびディレクトリのための専用のメモリを主記憶のほかに備えている．AlewifeはLimitLESSディレクトリ方式を採用し，共有コピー数が大きい場合にはfullmapのエミュレートをプロセッサが実行する．また，ディレクトリのために主記憶と分離したメモリを持っている．KSR1はCOMAによるハードウェア共有メモリを提供する計算機であるが，階層的なディレクトリを持つため，各階層でディレクトリ探索が必要になりアクセスコストが大きい．

プロトコルプロセッサを持つシステムとして，FLASH[13]，Typhoon[19]がある．FLASHはディレクトリ情報を主記憶に格納することによって他の専用メモリを持たない構成となっている．プロトコル処理を専用プロトコルプロセッサがハンドラを実行することにより行うが，プロトコルプロセッサを含むユニットであるMAGICはキャッシュを含むため大規模な構成となっている．TyphoonはプロトコルプロセッサにSPARCコアを内蔵しており，ソフトウェアによってLimitLESSディレクトリを実現している．

本論文で提案したDSM方式は，ディレクトリを主記憶内に格納し，かつコヒーレンス制御をメモリコントローラ内部のhardwiredロジックによって行うという点で上記のすべてのシステムと異なる．

6. まとめ

本論文において，階層化されたコヒーレンス管理方法と一般化されたコンバイニング技法による軽いハードウェアによる分散共有メモリ方式を提案した．本方式における階層最大共有距離ディレクトリは，メモリ使用量がプロセッサ数の対数に比例することから，ディレクトリの幅が短く，主記憶に格納することによるメモリアクセスオーバヘッドを隠蔽することが可能である．共有数が大きい場合にもビット幅が小さいことから，無効化型だけでなく，更新型プロトコルの使用を現実的に可能とする．また，他の共有に関する情報を不完全に持つディレクトリ方式との併用，たとえば，Limitedディレクトリにおいて共有数がlimitを超えた場合における共有管理方法として有効である．

さらに，本方式はコヒーレンス管理をメモリコントローラに担当させることにより，専用プロトコルプロセッサの必要性を除去し，低レイテンシな通信を実現する．管理方法の簡略化は通信パケット数の増加を引き起こすが，ネットワークによる動的な階層マルチキャスト/コンバイニング技法により逐次通信化を避け，通信量の削減および高速化を実現する．

テストベッド計算機お茶の水5号上でプログラムを実行して性能測定を行った．行列の冪乗計算において，一般化されたコンバイニングの使用により，8台の実行時に最大7.7%の性能向上を得た．またSPLASH-2のLU-Contigプログラムを実行した結果，軽いハードウェアによる分散共有メモリおよびコンバイニング機構の使用により8台の実行が1台の実行に対して81%実行時間を短縮した．本研究において，実際のハードウェアを通して軽いDSMが実際に実現可能であることを実証した．

謝辞 本研究の一部は通商産業省RWCプロジェクトの一環として行われた．お茶の水5号の開発にあたり，協力していただいた日本サンマイクロシステムズ株式会社ならびにデジタルテクノロジー株式会社に深く感謝の意を表します．また，Mentor Graphics社とSynopsys社のUniversity Programを用いた．両者に深く感謝します．

参考文献

1) Agarwal, A., Bianchini, R., Chaiken, D. and Johnson, K.: The MIT Alewife Machine: Architecture and Performance, *Proc. ISCA*, pp.2–13 (1995).
2) Agarwal, A., Simoni, R., Hennessy, J. and Horowitz, M.: An Evaluation of Directory Schemes for Cache Coherence, *Proc. ISCA*, pp.280–289 (1988).
3) Censier, L. and Feautrier, P.: A New Solution to Coherence Problems in Multicache Systems, *IEEE Trans. Comput.*, pp.1112–1118 (1978).
4) Chaiken, D., Kubiatowicz, J. and Agarwal, A.: LimitLESS Directories: A Scalable Cache Coherence Scheme, *Proc.International Conference on ASPLOS*, pp.224–234 (1991).
5) Charlesworth, A., Aneshansley, N., Haakemeetster, M., Drogichen, D., Gilbert, G., Williams, R. and Phelps, A.: The Starfire SMP Interconnect, *Proc. Supercomputing 97*, CDROM (1997).
6) Gottlieb, A., Grishman, R., Kruskal, C., McAuleffe, K., Rudolph, L. and Snir, M.: The NYU Ultracomputer-Designing and MIMD Shared Memory Parallel Computer, *IEEE Trans. Comput.*, pp.175–189 (1983).
7) Hagersten, E., Landin, A. and Haridi, S.: DDM – A Cache-Only Memory Architecture, *Computer*, pp.44–54 (1992).
8) Iftode, L., Dubnicki, C., Felten, E. and Li, K.: Improving Release-Consistent Shared Virtual Memory using Automatic Update, *Proc. 2nd International Symposium on HPCA* (1996).
9) III, H.B., Frank, S. and Rothnie, J.: Overview of the KSR1 Computer System, Technical Report, KSR-TR-9202001, Kendall Square Research (1992).
10) Inagaki, T., Niwa, J., Matsumoto, T. and Hiraki, K.: Supporting Software Distributed Shared Memory with an Optimizing Compiler, *Proc. ICPP*, pp.225–234 (1998).
11) James, D., Laundrie, A., Gjessing, S. and Sohi, G.: Distibuted-Directory Scheme: Scalable Coherent Interface, *Computer*, pp.74–77 (1990).
12) Keleher, P., Cox, A. and Zwaenepoel, W.: Lazy Release Consistency for Software Distributed Shared Memory, *Proc. ISCA*, pp.13–21 (1992).
13) Kuskin, J., Ofelt, D., Heinrich, M., Heinlein, J., Simoni, R., Gharachorloo, K., Chapin, J., Nakahira, D., Baxter, J., Horowitz, M., Gupta, A., Rosenblum, M. and Hennessy, J.: The Stanford FLASH Multiprocessor, *Proc. ISCA*, pp.302–313 (1994).
14) Lenoski, D., Laudon, J., Gharachorloo, K., Gupta, A. and Hennessy, J.: The Directory-Based Cache Coherence Protocol for DASH Multiprocessor, *Proc. ISCA*, pp.148–159 (1990).

15) Li, K.: IVY: A Shared Virtual Memory System for Parallel Computing, *Proc. ICPP*, pp.94–101 (1988).
16) Lilja, D.J.: Cache Coherence in Large-Scale Shared-Memory Multiprocessors: Issues and Comparisons, *ACM Computing Surveys*, Vol.3, No.25, pp.303–338 (1993).
17) Pfister, G. and Norton, V.: Hot Spot Contention and Combining in Multistage Interconnection Networks, *IEEE Trans. Comput.*, pp.943–948 (1985).
18) Protic, J., Tomasevic, M. and Milutinovic, V.: Distributed Shared Memory: Concepts and Systems, *PDT*, Vol.4, No.2, pp.63–71 (1996).
19) Reinhardt, S., Larus, J. and Wood, D.: Tempest and Typhoon: User-Level Shared Memory, *Proc. ISCA* (1994).
20) Scales, D., Gharachorloo, K. and Thekkath, C.: Shasta: A Low Overhead, Software-Only Approach for Supporting Fine-Grain Shared Memory, *Proc. 7th International Conference on ASPLOS*, pp.174–185 (1996).
21) Sun Microsystems, Inc.: *SuperSPARC & MultiCache Controller User's Manual* (1994).
22) Thapar, M. and Delagi, B.: Distributed-Directory Scheme: Stanford Distributed Directory Protocol, *Computer*, pp.78–80 (1990).
23) Woo, S., Ohara, M., Torrie, E., Singh, J. and Gupta, A.: The SPLASH-2 Programs: Characterization and Methodological Considerations, *Proc. ISCA*, pp.24–36 (1995).
24) XILINX, Inc.: *The Programmable Logic Data Book* (1994).
25) 松本 尚，平木 敬：超並列計算機上の共有メモリアーキテクチャ，電子情報通信学会技術研究報告，CPSY, Vol.92, No.173, pp.47–55 (1992).
26) 松本 尚，平木 敬：拡張されたSnoopy Spin Waitと階層化されたElastic Barrier，第47回情報処理学会全国大会論文集（4），pp.43–44 (1993).
27) 田中清史，対木 潤，松本 尚，平木 敬：並列計算機プロトタイプお茶の水5号，第3回FPGA/PLD Design Conference & Exhibit応用技術論文集，pp.505–514 (1995).
28) 田中清史，対木 潤，松本 尚，平木 敬：一般化されたコンバイニング機構，情報処理学会研究報告，ARC, Vol.96, No.10, pp.31–36 (1996).

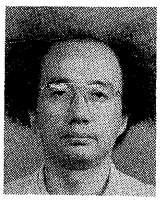
(平成10年9月7日受付)
(平成11年3月5日採録)

田中 清史

1971年生．1997年東京大学大学院理学系研究科情報科学専攻修士課程修了．同年4月から同大学理学系研究科情報科学専攻博士課程に在籍．並列計算機/共有メモリ機構に関する研究に従事．ほかに最適化コンパイラ，並列分散オペレーティングシステムに興味を持つ．

松本 尚（正会員）

1962年生．1985年東京大学工学部計数工学科卒業．1987年大阪市立大学大学院理学研究科物理学専攻修士課程修了．日本アイ・ビー・エム（株）東京基礎研究所研究員を経て，1991年11月より東京大学大学院理学系研究科情報科学専攻助手．並列計算機アーキテクチャ，並列分散オペレーティングシステム，最適化コンパイラに関する研究に従事．ほかに数値計算による制約解消系，グラフィックス，ニューラルネットワーク等に興味を持つ．電子情報通信学会，日本ソフトウェア科学会，ACM各会員．



平木 敬（正会員）

1976年東京大学理学部物理学科卒業．1982年同大学大学院理学系研究科物理学専攻博士課程修了．理学博士．1982年通商産業省工業技術院電子技術総合研究所入所．1988年より2年間IBM社T.J. Watson研究センタ客員研究員．1990年より東京大学理学部情報科学科（現在大学院理学系研究科情報科学専攻）に勤務．現在，超並列アーキテクチャ，超並列超分散計算，並列オペレーティングシステム，ネットワークアーキテクチャ等の高速計算システムの研究に従事．日本ソフトウェア科学会会員．