計算機アーキテクチャ 126－14
（1997．10．29）

低コストハードウェア分散共有メモリ

田中清史† 松本尚†
対木潤†† 平木敬†

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分散共有メモリが必須である。その際、遠隔データアクセスのレイテンシを小さくするためにキャッシュを行なうことが有効である。プロトコル制御をハードウェアで実現することにより、キャッシュのコンシステンシ管理に付随するオーバーヘッドを削減することが可能である。

本稿では効率の良いコヒーレンス制御とスケーラブルなディレクトリを備えた分散共有メモリを、タグ情報のための特別なSRAMチップを使用せず、メモリコントローラに簡単な回路を付加することによって低コストで実現する方式を提案する。更に並列計算機プロトタイプお茶の水5号に本機構を実装した。

Low Cost Hardware Distributed Shared Memory

KIYOFUMI TANAKA,† TAKASHI MATSUMOTO ,† JUN TSUIKI†† and KEI HIRAKI†

Distributed shared memory system is essential to provide a general and convenient programming environment. On a distributed system, caching of remote data is effective for minimization of the access latency. Hardware implementation of protocol management mechanism reduces runtime overheads which accompany with consistency management of cache coherence.

This paper describes a low cost method to implement a distributed shared memory system with efficient coherence management mechanism and scalable directory structure, by equipping a memory controller with a simple circuit instead of employing extra SRAM chips for tag information. We implemented this mechanism on a parallel computer prototype system, OCHANOMIZ-5.

1. はじめに

分散／並列処理においてデータアクセスの静的な予測が困難な場合は、データ通信を静的にスケジューリングできないため、他のプロセッシングノードのメモリ内のデータを参照／更新する機能が何らかの方法で実現されなければならない。この遠隔メモリアクセス機能の実現の方法として共有メモリ機構がある。共有メモリを実現する場合、大規模なシステムではリモートデータ参照のレイテンシで大きく性能が低下する可能性があるため、それらのキャッシュを行なう手法が主流となりつつある。このキャッシュを行なう際に、データのコピーの作成、キャッシュミスやヒットの判定およびコヒーレンス管理を要素プロセッサのソフトウェアで行なう選択肢がある。しかしこの場合、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ーヘッドが大きい。

† 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
†† 富士通株式会社
FUJITSU Ltd.

最近よく用いられる方式に、IVY [1] 流の仮想共有メモリが挙げられる。この方法は遠隔メモリアクセスをページトラップソフトウェアによって起動し、ハードウェアのページ管理機構を利用したキャッシングを行なうことで、キャッシュヒット時はオーバーヘッドがない。しかしこの方法でも共有されるデータが多くなると共有データのコンシステンシをとるためのソフトウェアオーバーヘッドが増えることと、データ転送単位が基本的にページ単位の大きさになる欠点がある。

一方、ノードにキャッシュ管理を行なうハードウェアを用意することにより、これらのオーバーヘッドを削減可能である。ただし、ソフトウェアによる方法が、市販のPCやワークステーションで実現可能であるのに対して、ハードウェアの実装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 [2] や最適化コンパイラ [3] 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。従ってハードウェア分散共有メモリシステムを構築する場合は、低いコストで実現可能なことが課題となる。

我々はハードウェア制御の分散共有メモリを低コスト

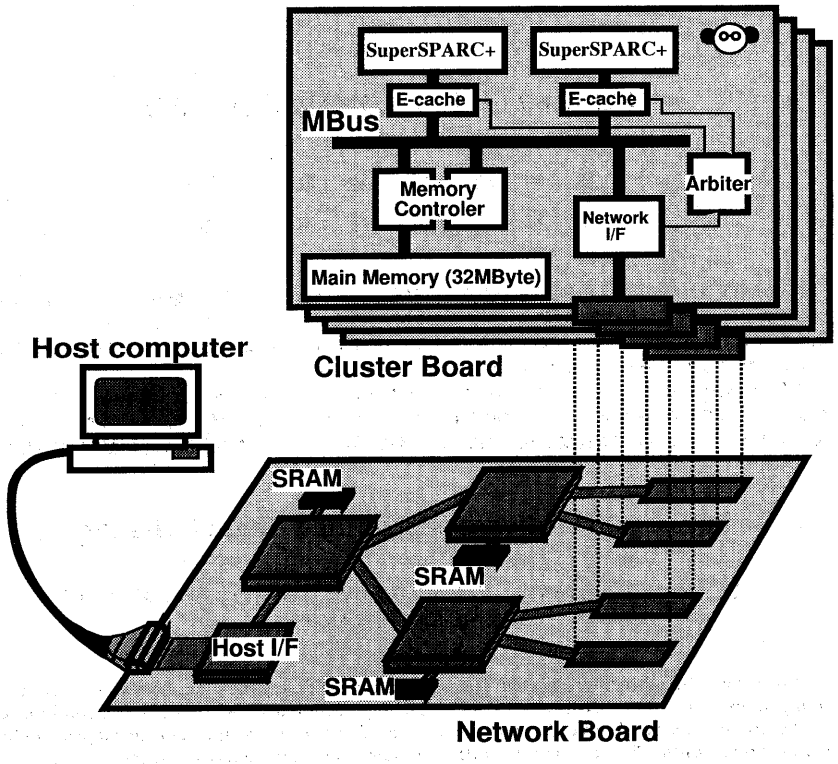


図1 お茶の水5号の構成

で実現することを本研究の目標とした。具体的にはリモートデータキャッシュのためのタグ情報の格納場所として高価なSRAMを使用せずに、DRAMで構成される主記憶上に確保する。メモリコントローラがプロセッサからのメモリアクセスリクエストに対してタグの参照、更新およびリプライを一括して管理する。これにより、基本的な分散メモリ型システムに新たな構成要素を付加することなしにCC-NUMAが実現可能となる。我々はメモリコントローラに汎用デバイスであるSRAMベースのFPGA(Field Programmable Gate Array)を使用した。開発効率の利点に加えて将来的なコストダウンが見込まれるため、カスタムチップを製造するのに対してはるかにコストを低くできる。

本稿では並列計算機プロトタイプお茶の水5号に実装したハードウェア分散共有メモリの詳細を述べる。

2. お茶の水5号

お茶の水5号(OCHANOMIZ-5:Omnipotent Concurrency-Handling Architecture with Novel OptiMIZers-5) [7] [8]は東京大学平木研究室における並列処理プロジェクト[6]の第5号並列計算機プロトタイプとして開発された。

2.1 お茶の水5号の構成

お茶の水5号は2個のPE(SuperSPARC+*)とメモリを共有バス(MBus:SPARC専用バス)結合したものをクラスタとしている。DRAMモジュールによる主記憶はクラスタ毎に32Mbyteあり、その他にメモリコントローラ、バス使用権を調停するアービター、ネットワークへのインターフェースのそれぞれにXilinx社のSRAMベースFPGAを使用している。なお、共有メモリのキャッシュ管理はメモリコントローラ内部のhardwired回路によって高速に行なう。

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1)。ネットワークの各内部スイッチングノードをFPGAおよびSRAMで構築することにより、一般化されたコンバイニング[11]をサポートする高機能ネットワークを実現している。

3. 実　　装

今回実装した分散共有メモリ機構は基本的に[10]に従ったが、FPGAの容量的問題があるためInvalidateプロトコルのみの実装など、更なる簡単化を行なった。

* CPU動作クロック60MHz、マルチキャッシュコントローラ内蔵モジュール。

3.1 コヒーレンス制御方式の設計方針

コヒーレンス制御のオーバーヘッドを削減し、多数のクラスタ間の共有を適切なコストで扱うことを目標とするために、以下のことが挙げられる。

- スケーラブルなディレクトリ方式
  ディレクトリのビット数を節約するために、共有情報を階層距離で管理する。この方法は疑似フルマップディレクトリ [9] のサブセットである。
- マルチキャスト／コンバイニング機構の利用
  クラスタ間ネットワーク内の各階層でマルチキャスト／コンバイニング [9] を行なうことにより、一回のコヒーレンス制御あたりのネットワークの負荷を軽減させる。

また、ハードウェア制御を容易にするために、以下のように簡単化する。

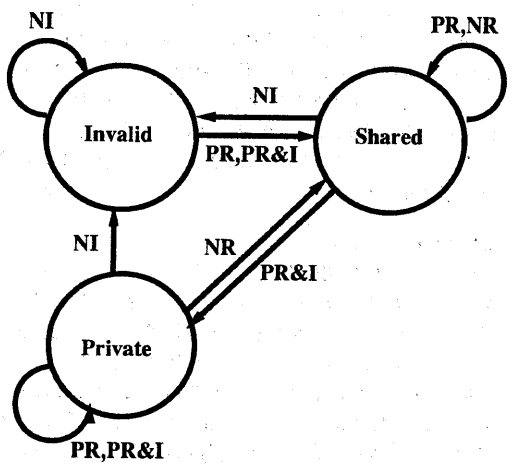
- キャッシュライン単位の管理
  共有データのクラスタレベルキャッシュのラインサイズをプロセッサキャッシュと同サイズにし、ライン単位で管理を行なう。またこれにより、false sharing を削減することが可能である。
- ホームの固定
  ディレクトリを持つクラスタはキャッシュブロックに対して静的に固定される（Home）。
- 状態／OwnerID／ディレクトリの Home での一括管理
  Invalidate プロトコルで一時的に Owner が Home 以外の他のクラスタに移る場合、Owner クラスタ ID を Home に記録する。
- リクエストは Home へ転送
  Invalid な場合の Read、Shared な場合の Invalidate リクエストは Home へ転送する。Read リクエストに関して、Home が Owner ならば Reply、Invalid ならば Owner クラスタへリクエストをフォワードする。Invalidate に関して、Home は Shared 領域への invalidate を行ない、その後 Acknowledge(以下 Ack) を返送する。

3.2 クラスタレベルキャッシュの状態

クラスタキャッシュの状態遷移を図 2に示す。（中間状態 Pending は省かれている。）ブロックが shared 状態になるときに、Home を必ず clean 状態にすることにより、shared-dirty を回避する。これによりディレクトリは Home にのみ用意すればよい。

以下に各状態の説明を述べる。

- **Private**
  該当ブロックが他のクラスタ（プロセッサ内キャッシュを含む）にキャッシュされていない状態。各ブロックの Home クラスタにおける初期状態、および外部に Invalidate リクエストを発行したブロックの状態は Private になる。



PR: Read request from PE
PR&I:Read&Invalidate request from PE
NI:Invalidate request from Network
NR:Read request from Network

図2 クラスタキャッシュの状態遷移

- **Shared**
  該当ブロックが複数のクラスタ間で共有されている状態。リモートクラスタからの Read 要求を受けた Home あるいは Owner、および Invalid 状態から Read 要求を発行したクラスタの状態は Shared になる。
- **Invalid**
  クラスタ内キャッシュに Valid なエントリがない状態。Home 以外のキャッシュの初期状態、および Invalidate を受けたブロックの状態は Invalid になる。
- **Pending**
  Invalid 状態から、read 要求を発行してリプライが返ってくるまでの状態、および Invalidate 要求を発行して Ack が返ってくるまでの状態。この状態で他のリクエストを受け取った場合、Home ならば Nack を返す。Home でない場合、外部からの Invalidate に対してのみ強制的に Invalidate される。その他は Nack を返す。

3.3 共有最大距離によるディレクトリ

共有最大距離によるディレクトリは、疑似フルマップディレクトリ [9] の簡易版である。ディレクトリとしてブロックを共有するクラスタの Home からの最大距離を記録する。ここで、距離とは 2 つのクラスタ間でパケットを転送する際に通過するネットワークの階層の段数である。Invalidate パケットは共有最大距離内の全てのクラスタへマルチキャストされる。（共有最大距離だけ階層ネットワークをさかのぼったノードを root

とする部分木へのブロードキャスト。）この方法は、Stanford大学のDASH [4] のフルマップディレクトリに比べて大幅にメモリ使用量が少なく、ネットワークのマルチキャスト機能を利用することによりチェインドディレクトリ方式よりも効率が良い。また、ネットワーク上でのコンバイニングによりAckメッセージの収集を効率良く行なう。

3.4 メモリマップ

各クラスタの32Mbyteの主記憶のうち、8Mbyteを共有メモリのデータキャッシュ領域に割り当て、2Mbyte*をクラスタキャッシュの状態、Owner、ディレクトリを保持する領域に割り当てる。メモリマップを図3に示す。各クラスタに対応した2Mbyteのキャッシュ領域に関してHomeかどうかで特に区別がない点においてCOMA [5] に類似しているが、Owner、ディレクトリはHomeノードのみが管理する。Ownerは2ビットで、HomeがInvalid時にValidなブロックを持つクラスタIDを保持する。ディレクトリは1ビットで、一階層のShared状態ならば

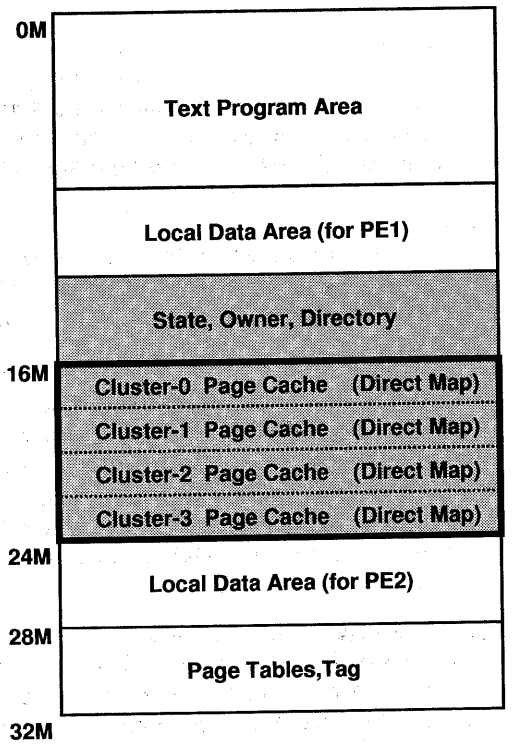


図3 メモリマップ

☆ 8Mbyteのキャッシュ空間に対して256Kbyteで十分であるが、回路内の結線を簡単にするためにキャッシュブロック毎に8byteでアラインした。

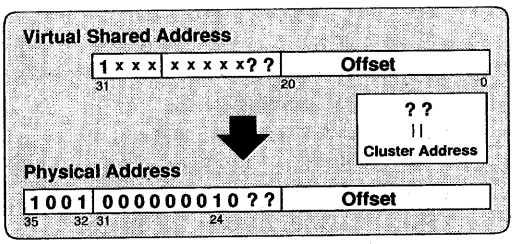


図4 アドレス変換

0、二階層のSharedならば1の値をとる。その他、各クラスタノードは全てのキャッシュブロックに関して、状態を表すValid/Invalid、Shared/PrivateおよびPendingの3ビットを保持する。

3.5 アドレス変換

共有アドレスはクラスタアドレス2ビット＋クラスタ内アドレス21ビットで表される。共有アドレスから物理アドレスへの変換は図4のように1対1対応となる。論理アドレスの最上位ビットが1であるとき、共有空間へのアクセスを意味する。物理アドレスの上位4ビットのビットパターンは、メモリコントローラが共有データアクセスであることを認識するためのものである。

3.6 メモリコントローラの基本動作

メモリコントローラはバスリクエストを受け取り、共有アドレスに対するデータ／Invalidate要求ならば対応するタグ情報を読み出す。キャッシュの状態によって要求元にデータ供給、あるいはリトライで返答する。リトライの場合、必要ならば同時にネットワークインター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させる。その後、必要ならばタグの更新を行なう。

共有アドレス以外への通常のMBusトランザクションと比較すると、終了応答までにタグの読み出し／デコードの時間が付加されるが、これは高々4バスクロックサイクルである。一度プロセッサのキャッシュに読み込まれ、その後高い確立でヒットすればこのタグ参照のコストは無視できるほど小さくなる。

3.7 メモリトランザクション処理

リモートメモリアクセスに伴うクラスタ間のデータ転送はパケット通信によって行なう。ネットワークトランザクションのパケットの種類は以下の7種類である。

- Read
- Reply
- Invalidate request to Home (InvReq)
- Acknowledge to InvReq requester (InvReqAck)
- Invalidate multicast (InvMul)
- Acknowledge to Home (InvMulAck)
- Nack

一方、プロセッサの共有データに対するMBus上のメモリトランザクションは以下の4種類である。

- Coherent Read
- Coherent Invalidate
- Coherent Read & Invalidate
- Write

以下、MBusに発行されたリクエストと起動されるネットワークパケットとの関係、およびメモリコントローラによる状態、Owner、ディレクトリビットの処理を示す。

- **プロセッサによるCoherent Read**
  メモリコントローラは状態を調べ、Pendingならば R&R(Relinquish and Retry)でパスを解放させて終了する。Pendingでない、かつValidならばデータを返して終了する。Invalidならば Home(Homeクラスタ自身ならば、Owner)クラスタへReadを発行し、R&Rで終了する。同時にPendingをセットする。クラスタ内の他方のプロセッサが$\overline{\text{MIH}}$*をアサートした場合は、メモリコントローラは処理を中断する。

- **プロセッサによるCoherent Invalidate**
  状態を調べ、PendingならばR&Rで終了する。Pendingでない、かつPrivateならばそのまま終了する。Sharedならば、自分がHomeでない場合はHomeクラスタへInvReqを発行し、R&Rで終了する。Homeクラスタ自身ならばOwnerビットに自分自身をセットしてディレクトリの値でInvMulを発行し、R&Rで終了する。同時にPendingをセットする。

- **プロセッサによるCoherent Read & Invalidate**
  状態を調べ、PendingならばR&Rで終了する。Pendingでない、かつInvalidならばHome(Homeクラスタ自身ならば、Owner)クラスタへReadを発行し、R&Rで終了する。同時にPendingをセットする。Validかつ Privateならばデータを返して終了する。このとき$\overline{\text{MIH}}$がアサートされたら処理を中断する。Sharedならば、自分がHomeでない場合はHomeクラスタへInvReqを発行し、R&Rで終了する。Homeクラスタ自身ならばOwnerビットに自分自身をセットしてディレクトリの値でInvMulを発行し、R&Rで終了する。同時にPendingをセットする。

- **プロセッサによるWrite**
  通常のメモリライトを行なう。プロセッサがこのリクエストを発行するのは、プロセッサのdirtyなキャッシュブロックを置換するとき、つまりクラスタキャッシュはPrivate状態のときである。

- **外部からのRead**
  ネットワークインターフェースは外部からのReadリクエストを受け取ると、MBusに対してRead & Invalidateを発行する**。メモリコントローラは状態を調べ、PendingならばNackを返送して終了する。Pendingでない、かつInvalidならばOwnerクラスタへReadを発行し、かつ要求元にNackを返送して終了する。同時にPendingをセットする。Validで$\overline{\text{MIH}}$がアサートされない場合はReplyを返送して終了する。同時にSharedをセットし、かつHomeならばディレクトリを適切な値にセットする。$\overline{\text{MIH}}$がアサートされた場合は、その後バスに流れるデータで主記憶上のキャッシュブロックを更新し、Sharedおよびディレクトリをセットする。

- **外部からのReply**
  主記憶上のキャッシュブロックを更新して終了する。同時にValid、Sharedをセットし、Homeならばディレクトリを適切な値にセットする。

- **外部からのInvReq**
  状態を調べ、PendingならばNackを返送して終了する。Pendingでないならばディレクトリの値でInvMulを発行して終了する。同時にPendingをセットし、Ownerビットに要求元のクラスタIDをセットする。

- **外部からのInvReqAck**
  終了と同時にValidおよびPrivateをセットし、Pendingをリセットする。

- **外部からのInvMul**
  InvMulAckを返送して終了する。同時にPendingかどうかにかかわらずInvalidをセットする。このときPendingビットの値は変化させない。

- **外部からのInvMulAck**
  Ownerビットを調べ、その値が自分自身ならば終了と同時にValid、Privateをセットし、Pendingをリセットする。Ownerビットの値が自分で

* SuperSPARCはdirtyなブロックのownerであるときパス上に$\overline{\text{MIH}}$をアサートし、自分がトランザクションのslaveとなってデータを供給する。

** クラスタ間でShared状態、かつプロセッサキャッシュがownerであるとき、外部へのInvalidateが必要であるにもかかわらずプロセッサがslaveとなってトランザクションを終了させる可能性がある。これを避けるためにRead & Invalidateに変換する。

ないならば、Ownerビットが表す最初の要求元にInvReqAckを返送して終了する。同時にInvalidをセットし、Pendingをリセットする。

- 外部からのNack
  終了と同時にPendingをリセットする。その他の状態ビットは変化させない。

3.8 ネットワークスイッチングノードにおけるコンバイニング処理

InvReqおよびInvMulAckのコンバイニングをサポートする[11]。InvReqのコンバイニングはfalse sharingが起こった場合のネットワークの負荷およびHomeでの処理を軽減するものである。スイッチングノードは各キャッシュブロック毎に、InvReqのSourceクラスタIDの2ビットとその到着を表す1ビットをそれぞれ2方向について保持する。またInvMulAckの到着を表す1ビットとあわせて、計7ビットのエントリで処理を行なう。InvReq、InvReqAck、InvMulおよびInvMulAckがネットワークスイッチングノードを通過する際に以下のような処理を行なう。

(1) InvReqがスイッチングノードに到着したとき、該当キャッシュブロックに対応するエントリに自分のSource IDと到着ビットをセットする。他方の到着ビットが0ならばInvReqをフォワードする。1ならばフォワードしない。

(2) InvMulが到着したとき、該当エリア(共有距離内)にマルチキャストする。ただし共有エリアのルートノードのときは該当エントリのAck到着ビットに1をセットする。

(3) InvMulAckがスイッチングノードに到着したとき、エントリのAckの到着ビットを参照し、0ならば1をセットする。1ならば0にリセットしてInvMulAckをHomeの方向へフォワードする。

(4) InvReqAckが到着したとき、該当エントリのInvReqの到着ビットを参照し、1の値を持つSource IDの方向へフォワード(2方向の到着ビットがいずれも1の場合はマルチキャスト)し、到着ビットを0にリセットする。

4. おわりに

本研究において低コストハードウェアで分散共有メモリ機構を実現する方式を示した。これはメモリコントローラ内にタグ情報操作のための簡単な回路を付加することで実現される。更に本機構を並列計算機プロトタイプお茶の水5号に実装した。現在本機構は動作を開始している。今後は実際のプログラムを実行して性能評価を行なう予定である。

謝辞

本研究は通商産業省RWCプロジェクトの一環として行われた。お茶の水5号の開発に当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) K.Li: IVY: A Shared Virtual Memory System for Parallel Computing. Proc. of the 1988 ICPP, pp.94–101 (Aug. 1988)
2) P.Keleher, A.L.Cox and W.Zwaenepoel: Lazy Release Consistency for Software Distributed Shared Memory. Proc. of the 19th ISCA, pp.13–21 (May. 1992)
3) D.J.Scales, K.Gharachorloo and C.A.Thekkath: Shasta: A Low Overhead, Software-Only Approach for Supporting Fine-Grain Shared Memory. Proc. of 7th Int. Conf. on ASPLOS, pp.174–185 (Oct. 1996)
4) D.Lenoski, et al.: Design of Stanford DASH Multiprocessor. Technical Report CSL-TR-89-403, Stanford Univ. (Dec. 1989)
5) E.Hagersten, A.Landin and S.Haridi: DDM – A Cache-Only Memory Architecture. Computer, Vol.25, No.9, pp.44–54, (Sep. 1992)
6) 平木 敬, 松本 尚, 稲垣 達氏, 大津 金光, 戸塚 米太郎, 中里 学: 細粒度並列計算機お茶の水1号 — 基本構想 —. 第47回情報処理学会全国大会講演論文集(6), pp.55–56 (Oct. 1993)
7) 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 (Jul. 1995).
8) 対木潤, 田中清史, 松本尚, 平木敬 : スケーラブル並列計算機プロトタイプ : お茶の水5号. 情報処理学会研究報告, ARC Vol.95, No80, pp.25–32 (Aug. 1995)
9) 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, CPSY92-26. pp.47–55 (Aug. 1992)
10) 対木 潤, 田中 清史, 松本 尚, 平木 敬: 軽いハードウェアによる分散共有メモリ — お茶の水5号の分散共有メモリ機構. 電子情報通信学会技術研究報告, CPSY 96-54. pp.55–62 (Aug. 1996).
11) 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.13,pp.31–36 (Jan. 1996)