超並列計算機プロトタイプ JUMP-0.5 における分散共有メモリ管理手法

山本考伸† 秤谷雅史† 津田健†
舟本一久† 五島正裕† 森眞一郎†
中島浩†† 富田眞治†

CC-NUMA 型の並列計算機では、分散共有メモリ管理を行うコントローラをプロセッサを用いて実現することがある。この場合、柔軟なプロトコル制御が可能になる一方、処理速度の低下という問題が生じる。本稿では、共有メモリ管理プロセッサとして市販の DSP を用いた場合の共有メモリ管理プログラムの構成について述べ、その性能を評価する。その評価をもとに、性能向上に向けて考えられるハードウェア ・サポートについて述べる。本稿で述べたハードウェア ・サポートのうち一部の機能は、既に開発中の並列計算機 JUMP-1 の共有メモリ管理プロセッサ MBP-light に実装されているが、更に全てのハードウェアサポートを実装した場合には、最大 6 割程度の性能向上が期待できる。

A Distributed Shared Memory Management Method on A Massively Parallel Computer Prototype JUMP-0.5

Takanobu Yamamoto,† Masashi Hakariya,† Takesi Tsuda,†
Kazuhisa Funamoto,† Masahiro Goshima,† Shin-ichiro Mori,†
Hiroshi Nakashima†† and Shinji Tomita†

Some CC-NUMA type multiprocessors of which the distributed shared memory controller is built up of a processor are developed. In these cases, the method of the memory management may become more flexible, but the speed might become slower. In this paper, we describe organization of the shared memory management program on a off-the-shelf DSP, and evaluation of performance of it. Some hardware supports considered from the evaluation result are also given. Some of these have already implemented on MBP-light, which is the memory management processor on a shared memory parallel computer JUMP-1. But when all of these hardware supports are implemented, the performance may improve by about 60 percent at the most.

1. はじめに

CC-NUMA 型の並列計算機では、分散共有メモリの管理を行うコントローラの実装方式が、システムを設計する上で重要な位置を占める。DASH などのようにこのコントローラを完全にハードワイアードで実現すると、高速性は得られるものの、ハードウェアの設計コストが非常に大きくなり、また、共有メモリの管理方式の柔軟性を損う。そこで、現在我々が他大学と共同で開発を進めている並列計算機 JUMP-1 をはじめ、FLASH[1)]、NUMA-Q など、最近の CC-NUMA 型並列機では、このコントローラをプロセッサを用いて実装するようになっており、共有メモリ管理はソフトウェア化される方向にあるようにみえる。

このアプローチによれば、たしかに共有メモリ管理方式の柔軟性は確保されるし、またハードウェアとソフトウェアの同時開発によって開発期間が短縮される可能性もある。しかし単に処理をソフトウェア化しただけでは、必要とされる性能を得ることは難しい。

本論文では、このような CC-NUMA 型並列計算機における共有メモリ管理プロセッサの、共有メモリ管理プログラムの構成と、考えられるハードウェア ・サポートについて述べる。以下、2 章で対象となる並列計算機のモデルについて述べる。JUMP-1 のプロトタイプである JUMP-0.5 では、共有メモリ管理プロセッサとして、市販の DSP を用いており、共有メモリ管

---

† 京都大学大学院工学研究科情報工学専攻
Division of Information Science, Graduate School of Engineering, Kyoto Univ.
†† 豊橋技術科学大学情報工学系
Department of Information and Computer Science, Toyohashi University of Technology

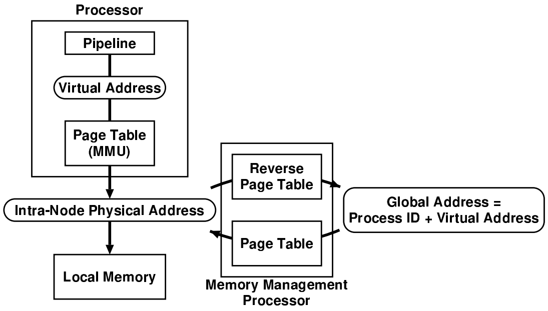

図 1　対象とする並列計算機の構成

理はほとんどソフトウェアのみによって実現される。3 章では、DSP 上の共有メモリ管理プログラムの構成について述べる。続く 4 章では、その場合の性能評価について述べる。その評価結果をふまえて 5 章では、このような共有メモリ管理プロセッサにおいて有効と思われるハードウェア・サポートについて述べる。

2. 対象とする並列計算機のモデル

本章では、対象とする CC-NUMA 型並列計算機のモデルについて述べる。このモデルは、JUMP-1 の構成を基に、多少の一般化を施したものである。

2.1　対象とする並列計算機の構成

図 1に、対象とする並列計算機の処理ノードの構成を示す。ノードは、主に、1 つまたは複数のプロセッサ、主記憶、そして、本稿での話題の中心であるノード間の共有メモリ管理プロセッサからなる。ノード間は、一般のネットワークで接続される。

2.2　メモリ・システムの構成

2.2.1　主記憶のキャッシュとしての利用

各ノードに分散配置された主記憶は、各ノードにローカルな主記憶として使用するだけでなく、リモート・メモリへのアクセスに対するキャッシュとしても利用する。

キャッシングの手法としては、Shared Virtual Memory (SVM) を基にした方式を採用する。すなわち、リモート・メモリのデータのキャッシングは、ページ共有に類似した方法で行う。ただしデータの有効性の管理と転送は、ページより小さいブロックを単位として行い、false sharing による性能の悪化を防ぐ。例えばこの方式では、リモート・メモリ上にあるデータをはじめてキャッシングする場合には、まずローカル・メモリ上の物理ページを 1 ページ確保し、そこへリモート・メモリから 1 ブロックをコピーする。その後、同一ページ内の別のブロックにアクセスがあった場合には、ブロックのコピーのみを行うことになる。

またこの方式では、キャッシュ・ディレクトリなどのアドレス・マッピングに関わるテーブルは、ページ単位に構成することになる。一方、有効ビットなどの一貫性制御のための管理情報は、メモリの各ブロックに対して付与される。

2.2.2　SVM におけるメモリ・アクセス

前出の図 1には、システムのアドレス体系の概略を同時に示してある。

SVM におけるローカル・メモリ・アクセス

SVM 方式では、キャッシュとして使用される物理ページへアクセスにも、主記憶として使用される物理ページに対するのと全く同じアドレス変換機構を用いることになる。仮想アドレスから物理アドレスへのマッピングはページ・テーブルによって管理され、アドレス変換はプロセッサの持つ MMU を利用して高速に実行することができる。

SVM におけるリモート・メモリ・アクセス

一方この方式では同一のページの物理アドレスがノード間で異なるため、ノード間の共有メモリ管理を行う時にはローカル・メモリの物理アドレスから大域的なアドレスへの変換が必要となる。グローバル・アドレスの構成法には任意性があるが、各プロセスの仮想アドレスを大域的なプロセス ID で拡張したものなどを用いることができる。その場合、ローカル・メモリの物理アドレスからグローバル・アドレスへの変換には逆索きページ・テーブルを、グローバル・アドレスからローカル・メモリの物理アドレスへの変換にはページ・テーブルを用いることになる。

2.3　共有メモリ管理方式

共有メモリ管理は、関係するノード間でパケットを交換することによって行われる。メモリ・アクセスに伴うノード間のパケットの転送及び一連の状態遷移をトランザクションと呼ぶ。

ノードの状態

ノード間の共有メモリ管理は、ディレクトリ方式で実現する。ディレクトリは、各アドレスに対して定められた **Home** ノードに置く。

各ノードはトランザクションの対象アドレスのコピーの保有状態によって以下のように分類される。

Renter　Home 以外の、有効なコピーを保持するノード。

Owner　Renter のうち、読み出し要求に対して応答する責任をもつノード。
コピーを排他的に保持するノードは Owner である。Renter が複数存在する時には、Home にも有効なコピーをおくこととし、Home が Owner を兼ねる。

トランザクションの対象となるアドレスに関して、Renter でも Home でもないノードは、そのトランザクションには関わらない。

トランザクションの流れ

典型的なトランザクションにおけるパケットの流れを図 2に示す。

(1)　トランザクションを開始したノードを **Initia-**

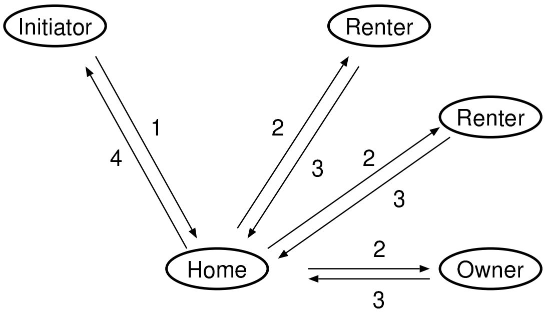

図 2　パケットの流れ

tor と呼ぶ。Initiator は、まず Home に要求パケットを送る。

(2) Home は、読み出し要求であれば Owner に転送し、書き込み要求 (無効化/更新) であれば Renter に転送する。

(3) Owner は読み出し要求に対する応答を、Renter は書き込み要求に対する応答 (ack) を、Home に返す。簡単のため、三角通信は行わず、応答は Home ノードを経由するものとする。

(4) Home は、Initiator に応答を返す。

3. DSP 上の共有メモリ管理プログラムの構成

超並列計算機 JUMP-1 のプロトタイプである JUMP-0.5 では、共有メモリ管理プロセッサとして、汎用の DSP、TI 社の TMS320C40[7)]を用いて実現した。本章では、DSP 上の共有メモリ管理プログラムの実現方法について述べる。

3.1　共有メモリ管理プログラムの基本構造

共有メモリ管理プロセッサでの処理は、基本的に、パケットの到着によって起動される。共有メモリ管理プロセッサでの処理は、その実行順序に関して以下のことなどを考慮する必要がある：

- 共有メモリ管理のための処理は、普通 長くとも数十サイクル程度で終了する。したがってそれらの処理は、割り込みなどがかからないようにして、一気に実行してしまう方がよい。
- 共有メモリ管理プロセッサは、ページ単位の処理など、通常の共有メモリ管理の処理に比べて非常に長い時間のかかる処理を同時に扱わなければならない。そのような処理に対しては、割り込みをかける必要がある。
- 資源や共有メモリ管理プロトコル上の制約によってすぐには処理を開始できない場合がある。安直に実行可能になるまで待つような実装では、性能上の問題はともかく、デッドロックに陥る可能性がある。

これらの問題点に対して今回は、到着したパケットの処理をプロセスとして実現することで対応した。それによって、上述の問題点をプロセス・スケジューリングの問題として統一的に扱うことができる。したがって共有メモリ管理プログラムは、プロセスの管理を行う下位層と、プロセスとして動作し実際に共有メモリ管理を行う上位層の、2 階層のシステムとして実現される。

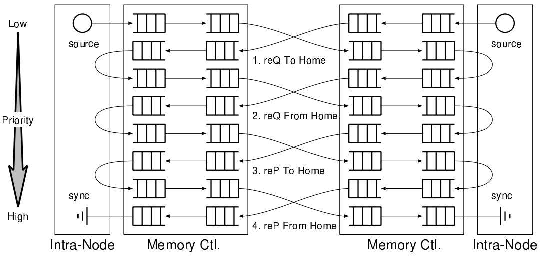

図 3　トランザクションに関する資源要求グラフ

本稿では、主に、下位層について述べる。以下、3.2 節と 3.3 節ではまず、デッドロックの防止とアドレス変換について述べ、そして 3.4 節ではプロセスの管理について、説明する。

3.2　デッドロックの防止

各ノード間でトランザクションの資源要求にループがあると、デッドロックを生じる。2 章で述べた共有メモリ管理方式では、デッドロックを防止するため、図 3に示すように、4 段階の優先順位を設ける必要がある。そして、低い優先順位の処理によって資源が占有されないように制御しなければならない。ノード間を接続するネットワークにも、トポロジに関するデッドロック防止のためとは別に、4 本のチャネルが必要になる。

3.3　アドレス変換

前述したように、SVM では、ノード内のローカル・メモリをアクセスするためにはローカル・アドレスを用い、ノード間を流れるパケットにはグローバル・アドレスを用いる。したがって、パケット送信時にはローカル・アドレスをグローバル・アドレスに、到着時にはグローバル・アドレスをローカル・アドレスにそれぞれ変換する必要がある。

また、OS によるページ・エイリアスを考慮し、共有メモリ管理プロセッサ内部の処理は、基本的に、ローカル・アドレスを用いて行う必要がある。

グローバル・アドレスとして各プロセスの仮想アドレスを大域的プロセス ID で拡張したものを用いると、この変換に必要となるテーブルには、要素プロセッサが参照するページ・テーブル、および、OS が管理するであろう逆引きページ・テーブルを、そのまま用いることができる。ただし、大域的なプロセス ID とプロセッサの認識するプロセス ID の間の変換が別途必要となる。

ローカル・メモリ上にあるページ・テーブルの検索

には何十サイクルもかかるため、DSP から高速アクセス可能なメモリ上にハッシュ・テーブルを設け、ページ・テーブルの一部をキャッシングすることによって高速化を図る。このハッシュ・テーブルの内容はローカル・メモリ上にあるページ・テーブルのコピーであるので、ハッシュ値が衝突した時には古いエントリは廃棄してしまってかまわない。

3.4 プロセス管理

3.4.1 プロセスの生成

ハードウェアが用意するパケットの受信キュー、および、ノード間のネットワークが前述のデッドロック防止条件を満たしている場合には、これらのキューは各優先順位に対応するプロセスの実行可能キューとみなすことができる。

条件を満たしていない場合には、以下のようにする必要がある：プログラムで各優先順位に対応する実行可能キューを別途用意し[☆]、受信キューに到着したパケットに対してプロセスを生成し実行可能キューに移す。移す処理自体は、最高の優先順位で実行する必要がある。

3.4.2 プロセスの横取り

前述したように、共有メモリ管理プロセッサは、通常の共有メモリ管理に関する処理と、長い時間のかかる処理を扱う必要がある。したがってプロセスの優先順位としては、基本的には、以下の 2 レベルを用意する：

高位　通常の共有メモリ管理に関する処理は、割り込み/横取り不可とする。
　　このレベルの各プロセスには、前述のデッドロックの防止条件で述べた優先順位順位がそれぞれ与えられる。

低位　その他の処理は割り込み/横取り可とし、処理中に新たなパケットが到着したときには、割り込まれ、横取りされる。
　　プログラムは、横取りされたプロセスのための実行可能キューを用意する。このキューには最低の優先順位が割り当てられ、共有メモリ管理のプロセスが存在しない場合にのみディスパッチされる。このレベル内の各プロセスの間のスケジューリング方式には任意性がある。

3.4.3 プロセスの中断と再開

中断の要因としては以下の 3 つがある：

(1) 送信キュー・フル
(2) トランザクションの競合
(3) 応答待ち

ただし、1 および 2 では、プロセスは実行の開始以前に中断状態に陥る．

送信キュー・フル

ある送信キューがフルである場合には、対応する実行可能キューは中断プロセスのキューと見なせる。送信キューが空けば、直ちに対応するキューを実行可能キューと見なせばよい。

トランザクションの競合

各ノードにおいて、あるパケットに対する処理が終了する前に、同一アドレスに対する別の要求パケットが到着することがある。これをトランザクションの競合と呼ぶ。

このような場合、一般には、後続の要求パケットに対する処理は、未終了の処理が終了するまで開始しないようにする。逐次的に処理することにより、プロトコルが大幅に簡略化される[☆☆]。待たせることによる性能への影響も小さいと考えられる。

応答待ち

既に述べたように各トランザクションにおいて要求パケットを送信した後は、一般に、応答パケットを待つ。要求パケットを送信したプロセスのデータの一部は、応答パケット到着時に必要となる。

応答パケット到着時に必要となるデータを保存しておく方法としては、必要なデータのみをそのためのテーブルに保存しておく方法の他に、要求パケットを送信したプロセスそのものを中断しておく方法が考えられる。要求パケットを送信し、中断したプロセスは、対応する応答パケットの到着イベントに対して再開されるようにする。

この方法では、保存しておくデータ量が多いが、応答パケットの到着に対してプロセスを新たに生成する処理を省略することができる。ただし後で述べるようにプロセスの生成処理をハードウェアで高速に行うような共有メモリ管理プロセッサでは、応答パケットに対して新たなプロセスを生成する方法の方が効率が良いことがあるだろう。

3.4.4 ペンディング・トランザクション・テーブル

前述のように、要求パケットを送信したプロセスは、応答待ちと、トランザクションの競合の検出のために、そのアドレスとともに記憶しておく必要がある。このためのテーブルをペンディング・トランザクション・テーブル (**PTT**) と呼ぶ。PTT は、ローカル・アドレスをアクセス・キーとする連想テーブルである。PTT の構造を、図 4に示す。

要求パケットに対応するプロセスは、以下のように処理する：

(1) 対応する送信キューが空いていることを確認する。
(2) 次に PTT を検索する。対象アドレスが PTT に登録されていない場合には、処理を開始する。
(3) プロセスは要求パケットを送信すると同時に当該プロセスを PTT に登録し、中断する。

[☆] 厳密には、これらのキューは長さが無限でなければならない。

[☆☆] 現在の処理内容と後続の要求の組合わせによっては、開始してよい場合もある。

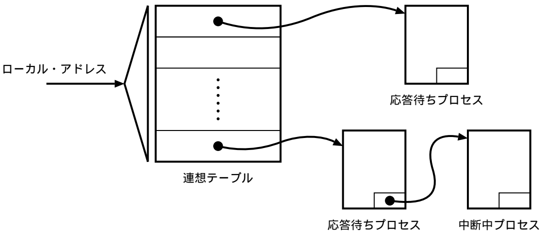

図 4　ペンディング・トランザクション・テーブル

(4)　応答パケット到着時に、PTT を調べ、対応するプロセスを再開する。

(5)　プロセス終了時に、PTT から削除する。

要求パケット到着時に、すでに対象アドレスに関するプロセスが PTT に登録されていた場合には、当該プロセスを中断し、既に登録されている応答待ちプロセスに対して中断したプロセスをフックする。登録されている応答待ちプロセスが終了する時に、フックされているプロセスを再開する。3 つ以上のプロセスが同時に競合を起こした場合には、この処理を再帰的に繰り返せばよい。

4. DSP による実装の評価

本章では、前章で述べた DSP 上での共有メモリ管理プログラムの性能評価を行う。共有メモリ管理プログラムのアセンブリ・コードから、処理にかかるサイクル数を手で求めた。

さて今回使用した DSP は、分岐命令の遅延スロットが 3 と多いなど、そもそもこのような処理に適したものではない。

また、プログラムは C 言語を用いて記述し、TI 社の提供する最適化コンパイラを用いてコンパイルを行ったのみで、有効なインライニングなどの最適化が不十分である。

以上から、サイクル数が大き目に表れていることを断っておく。評価においては、各処理にかかる実行サイクル数の比に注目する。

要求パケットを送信して応答待ちにプロセスにかかるサイクル数を計算した。表 1 にプロセスの生成から中断までの、表 2 にプロセスの再開から終了までの、処理内容とそれぞれの処理にかかるサイクル数を示す。

DSP 上の共有メモリ管理プログラムにおいて特に時間がかかっている処理について以下にまとめる：

割り込み処理　表中には、DSP がアイドル状態での割り込み処理にかかるサイクル数を示した。処理中での割り込み処理には約 70 ステップを要する。これは主に、割り込まれたプロセスのコンテクストの保存と実行可能キューへの移動に費やされる。

プロセス生成、及び終了　要求パケット到着時にはスタックを確保し、プロセス終了時にはスタックを解放する。これらの処理が非常に遅くなっている。

パケットの内容のコピー　到着したパケットの内容をプロセスのスタック上に保存してから各プロセスの処理を開始している。この処理にパケットの容量に比例する時間が費やされる。

ただし、到着パケットに対応する送信キューが空いており、システム自体がそのパケット処理をすぐ開始できるときには、このコピーを省略することができる。

ハッシュ・テーブルの検索　アドレス変換表および PTT は、ハッシュ・テーブルを用いて実現してある。このためハッシュ表への登録及び検索は、ハッシュのあふれがない時で十数サイクル、ハッシュのあふれが起った時には数十サイクルの処理が必要となる。

表 1　プロセスの処理時間（生成から中断まで）

処理	サイクル数
割り込み処理	15
プロセス生成	25–33
アドレス変換	16
実行開始可能判定	8–19
処理内容の実行	X
アドレス逆変換	0/16
要求パケット生成、送信	10–26
PTT への登録、中断	10
計	84–135+X

表 2　プロセスの処理時間（再開から終了まで）

処理	サイクル数
割り込み処理	15
アドレス変換	16
中断中のプロセスの再開	19–27
処理内容の実行	Y
応答パケット生成、送信	10–26
中断プロセスの有無判定	6
プロセス終了	38
計	104–128+Y

5. ハードウェア・サポート

本章では、4 章で述べた DSP 上の共有メモリ管理プログラムの性能評価を元に、考えうるハードウエアサポートについて述べる。

高速な割り込み処理　プロセス実行中におけるパケット到着による割り込み処理が非常に遅いが、これは主にレジスタの待避などをプロセスのスタック上に行っているためである。

コンテキストの切り替えをハードウェアで実行すれば、割り込み時の処理を数十サイクル削減で

きる。

プロセス・スタック　到着パケットに対して生成されるプロセスのスタック領域の数を制限し、空きスタックの管理をハードウェアによって行う。これによってプロセス生成及び終了の大半の処理時間を占めているスタック操作が高速に実現できる。

スタック領域の空きがない時は、新たにプロセスを生成しないようにしても、図3に示した優先順位を守っていればデッドロックは発生しない。

パケットのコピー　プロセス生成時、及び応答パケット到着時のプロセス復帰時において到着パケットの内容のコピーをプロセスのスタック上に行っているため、プロセスの生成に時間を費やしている。

パケットのコピーをスタック上ではなく特殊なレジスタ上に行うことによって1サイクル程度でパケットのコピーを実現することができる。

PTTのハードウェア化　ペンディング中のプロセスの管理をハードワイヤードな連想記憶で実現する。未終了になるプロセス数は多くはないと予想されるため、この連想記憶はフルアソシアティブで構成可能である。このPTTでは、プロセスの実行開始判定が1サイクル程度で終了し、応答パケット待ちになったプロセスのPTTへの登録も1–2サイクル程度で可能である。

この場合 実行開始判定は、プロセスの生成処理と並列に実行可能である。

PTTの管理をハードウェアのみで行う場合には連想記憶のあふれのサポートは多少難しくなるが、PTTがフルの時には応答パケット待ちになるプロセスは実行開始できないようにしても、図3に示した優先順位を持っているときにはPTTエントリ待ちによるデッドロックは発生しない。

アドレス変換テーブル　グローバル・アドレスとローカル・アドレスの変換用のTLBを用意する。TLBにヒットしたときにはアドレス変換は1サイクル程度で可能になる。

受信側のTLBの十分な容量は予測が困難であるが、送信側のTLBの容量は要素プロセッサのMMUのTLBと同程度で良い。

6. おわりに

本稿では、JUMP-0.5におけるDSP上の共有メモリ管理プログラムの実装と性能評価について述べ、また、考え得るハードウェア・サポートについての考察を行った。

JUMP-0.5はJUMP-1のプロトタイプであり、JUMP-1では、共有メモリ管理プロセッサとして、MBP-lightと呼ぶ専用に開発されたプロセッサが用いられている2)。

本稿で述べたハードウェア・サポートのうち、高速な割り込み処理、アドレス変換TLB、及びパケットのレジスタへのコピーはMBP-lightにおいて既に実装されている。

更にPTTのハードウェア化及びスタックのハードウェア管理を実装した場合には、最大で6割程度の高速化が期待される。

謝辞

日頃から貴重な御意見を頂いている文部省重点領域研究共同研究者各位に感謝します。

また、メンター・グラフィックス・ジャパン株式会社のHigher Education Programの一環として製品とサービスをご提供頂いたことに感謝します。また、日本テキサス・インスツルメント株式会社の加藤賢二氏には、DSPユニバーシティプログラムの一環としてDSP開発環境をご提供頂いたことに感謝します。

なお本研究の一部は、文部省科学研究費補助金（基盤研究(C)課題番号09680334、奨励研究(A)課題番号09780268）による。

参考文献

1) J.Kuskin and al. et :The Stanford FLASH Multiprocessor, In Proc. The 21st ISCA,pp.302-313,1994.
2) 佐藤 充, 他 :超並列マシンJUMP-1のための分散共有メモリ管理プロセッサ, 並列処理シンポジウムJSPP'97予稿集,pp.265-272,1997.
3) 安生 健一朗, 他 :分散共有メモリを持つWSクラスタJUMP-1/3, 並列処理シンポジウムJSPP'97予稿集,pp.321-328,1997.
4) 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予稿集,pp.245-252,1993.
5) 平木 敬, 他 :JUMP-1共有バス仕様書,1995.
6) 五島 正裕, 他 :JUMP-1 CBus仕様書,1994.
7) Texas Instruments TMS320C4x User's Guide.