並列計算機 JUMP-1 における分散共有メモリシステムの性能評価

小西将人[†] 五島正裕[†]
森眞一郎[†] 富田眞治[†]

計算機システムは階層的な構造を持ち、その傾向は今後さらに助長されると考えられる。JUMP-1は、多階層/クラスタリングされたアーキテクチャを持つ分散共有メモリ型並列計算機である。クラスタに分散配置された主記憶の一部を、リモートに対するキャッシュとして利用することで、平均メモリアクセスレイテンシの短縮を図る一方、クラスタ間のメモリ制御はその処理の複雑さからプログラムベースで柔軟に行う方式をとる。分散共有メモリ管理プログラムの実装を行い、実機上で性能測定を行った。その結果、リモートの主記憶読み出しのレイテンシは519サイクルであることがわかった。

Evaluation of Distributed Shared Memory System of the JUMP-1 Multiprocessor

Masahito Konishi,[†] Masahiro Goshima,[†] Shin-ichiro Mori[†]
and Shinji Tomita[†]

The JUMP-1 multiprocessor has hierarchical,clusterd architecture with Distributed Shared Memory. For purpose of reducing average memory access latency, JUMP-1 uses a part of main memory distributed among clusters as a cache for remote clusters. And inter-cluster consistency control is complex,so this control is flexibly performed by program. In this paper we discribe this program and evaluate the performance. The resulut shows that it takes 519 cycles to acccess remote memory.

1. はじめに

計算機ハードウェアは普通、階層的な構造を持つ。例えば現在の計算機を例にあげると、筐体、マザーボード、ドータボード、プロセッサ・カードリッジ、MCM、チップといった階層化がなされている。

このような傾向は、システムの大部分がチップに集積されるようになっても、緩和されるものではない。集積度の向上に従い、チップ内で配線遅延の差が支配的になり、チップ内部でもクラスタリングを考えなければならないからである。

このような傾向に逆らって、単階層なアーキテクチャを採用することは、高コストであり、また、将来そのコストの増大は受け入れ難いものになるであろう。したがって、アーキテクチャは階層性を受け入れ、不具合をソフトウェアで解消するというアプローチが重要になると考えられる。

そのような傾向を背景として、本稿で述べる並列計算機JUMP-1は、階層的なアーキテクチャの検証、及びそのようなシステム上のソフトウェア研究のテストベッドの提供を目標の1つとして開発された。JUMP-1は、階層的な実装に合わせたクラスタ構造を持つ、分散共有メモリ(DSM)型並列計算機である。

本稿では、JUMP-1のDSM管理について述べる。JUMP-1ではDSMも階層化されている。各クラスタに分散された主記憶の一部をリモートの主記憶に対するキャッシュとして利用することで、リモートへのアクセスの軽減を図る一方、クラスタ間の処理はソフトウェアで柔軟に行うというアプローチを採る。

まず第2章ではJUMP-1の物理的構成とメモリシステムについて述べる。次に第3章でDSM管理における問題点について述べた後、第4章で実装したDSM管理プログラムについて述べる。最後に第5章で実機上での計測結果を示す。

2. JUMP-1の概要

本章ではJUMP-1の物理的構成と、DSMについて述べる。

2.1 物理的構成

JUMP-1はクラスタ構造を採用しており、複数クラスタをクラスタ間ネットワークで接続した構造を持つ。クラスタは主に4つのプロセッサユニットとメモリユニットから構成されている。

† 京都大学大学院情報学研究科通信情報システム専攻
Division of Communications and Computer Engineering, Graduate School of Informatics, Kyoto Univ.

図 1にクラスタの構成を示す。4 つのプロセッサユニットはクラスタバスで接続され、高速な通信を行うことができる。

プロセッサユニットは主に、1 次キャッシュを内蔵した要素プロセッサ SuperSPARC+と、外部 2 次キャッシュ[1)]から成る。

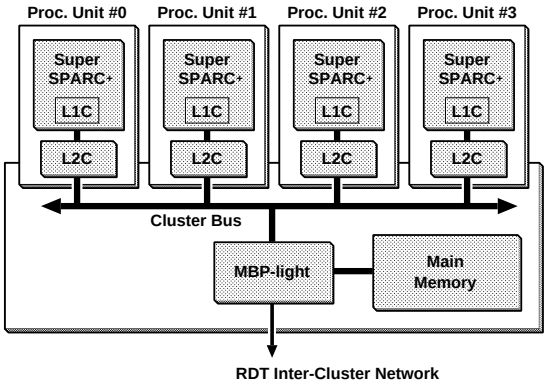

図 1　JUMP-1 のクラスタ

メモリユニットは、各クラスタに分散された主記憶であるクラスタメモリと、そのコントローラである **MBP-light**[2)]から成る。さらに、MBP-light はコア・プロセッサ **MBP Core** と Main Memory Controller(**MMC**) に分割される。MBP-light については 4.1節で詳述する。

JUMP-1 ではクラスタを **RDT**[4)](Recursive Diagonal Torus) と呼ばれる結合網で接続している。RDT は基本のトーラス構造の上に目の粗いトーラスを 45 度ずつ傾けながら再帰的に積み上げた、階層的な構成を持つ。

2.2　アドレッシング・システム

クラスタメモリの一部は、リモートのクラスタの主記憶に対するキャッシュとして利用される。前節で述べたように、要素プロセッサは、SuperSPARC+内蔵 1 次キャッシュと、外部 2 次キャッシュを持つことから、このキャッシュは 4 つの要素プロセッサで共有される 3 次キャッシュにあたる。

2.2.1　アドレス体系

アドレス体系は SVM[3)] に準じており、キャッシングはページ単位でなされる。

SVM では、クラスタ内では、通常の仮想記憶システムを用いることができる。アドレス変換は、アクセスしようとする物理アドレスがオリジナル/コピーに関わらず、要素プロセッサの MMU で高速に変換することができる。クラスタ内キャッシングは、その物理アドレスをを基に行えばよい。ページサイズは SuperSPARC+ MMU では 4KB である。

クラスタ間ではオブジェクトを一意に特定する大域仮想アドレスを用いる。このようなアドレスとして、普通、プロセスの仮想アドレスを大域的なプロセス ID で拡張したものを用いる。

クラスタ間での一貫性制御を行う際、送信時には物理から大域仮想へ、受信時には大域仮想から物理へのアドレス変換が必要となり、逆引きページテーブル/ページテーブルを用いた変換をしなければならない。

2.2.2　一貫性制御

JUMP-1 では、false sharing を軽減するために、ページ単位ではなく、32B のライン単位で一貫性制御を行う。そのため、主記憶にはライン単位でタグが付加されている。

クラスタ内のスヌープ方式に対して、クラスタ間ではディレクトリ方式で一貫性制御を行う。SVM でのオリジナルページを持つクラスタに、そのページに対するディレクトリを持たせることが自然である、このクラスタを、そのページの Home クラスタと呼ぶ。

Home のマッピングは自由であるので、クラスタ間の一貫性制御を行う場合には、Home クラスタを求めるためのテーブル検索が必要となる。

一貫性維持動作として、JUMP-1 は無効化型、更新型及びその組み合わせをサポートする。

一貫性維持動作は、クラスタ内とクラスタ間で以下のように行われる。

クラスタ内　2 次キャッシュがスヌープに対して、状態を応答し、主記憶アクセスを軽減する。2 次キャッシュが処理できない場合に限り、MMC が主記憶アクセスを行う。MMC はタグを読み出し、当該ラインの状態を確認し、クラスタ内で閉じた処理ならば、MMC がハードウェア的に処理する。ここまで全てハードウェアで高速に行なわれる。

クラスタ間　MMC がタグの状態により、クラスタ間処理が必要であると判断した場合、Core にパケット処理を依頼する。Core は、依頼されたパケットをソフトウェアで柔軟に処理する。

3.　DSM 管理

本章では、DSM 管理を行う上で注意すべき事柄を挙げる。

メモリアクセスに起因する、一連の一貫性維持動作をトランザクションと呼ぶ。システムの各部でのトランザクションの処理は、基本的に、到着したパケットに対してデータアレイとディレクトリの更新を行い、必要ならば他クラスタへのパケットを送出することでなされる。

トランザクションの処理順序に関して、単にパケットが到着した順序で行えばいいわけではない。

以下、まず 3.1節でトランザクションの流れについて説明し、3.2節で競合と呼ばれる状況について述べる。トランザクションの処理順序に関しては 3.3節で述べる。

3.1 トランザクションの流れ

すべてのトランザクションは Home クラスタを経由するものとし、三角通信は考えないものとする。

システム内で唯一のラインを保持しているクラスタを Owner と呼ぶ。Home 以外にラインを共有しているクラスタを Renter と呼ぶ。また、トランザクションを開始するクラスタを Initiator と呼ぶ。

トランザクションにおけるパケットの流れは、基本的に次のようになる。

① Initiator が要求パケットを Home へ送信
② Home はディレクトリを検索し、Owner/Renter へ要求パケットを転送
③ Owner/Renter は要求に対する応答パケットを Home へ送信
④ Home は Initiator へ応答パケットを転送

当然、共有状態によっては②③が省略されることもある。

3.2 トランザクションの競合

同一ラインに対するトランザクションが、異なるクラスタからほぼ同時に開始されることがある。この時 Home では、先行トランザクションの応答パケットが到着しないうちに、後続要求パケットが到着することになる。この状態を競合と呼ぶ。

先行トランザクションの応答が返ってこないうちに、後続パケットの処理を始めるのは、一般に面倒である。したがって、Home では、応答待ちのトランザクションの情報を記憶しておくことで競合を検出、競合を起こした後続パケットの処理を何らかの方法で遅延させる。

無効化型トランザクションの競合

競合のうち、無効化トランザクションの相撃ちについて注意を払う必要がある。

無効化型書き込みでは、ラインの状態を不可逆的に更新するのは、応答パケットが返されトランザクションが完了したことが判明した後でなければならない。それ以前に更新を行うと、競合が発生した場合に無効化の相撃ちが発生し、書き込みの内容が失われてしまうからである。

したがって、応答パケットが返されるまで、書き込み情報をバッファに保存しておく必要があるが、そのバッファの数によって、同時に未完了にできる書き込み要求の数に制約が生じることになる。それを越える数の書き込みを行った際には、1 つ目の無効化が完了しバッファが解放されるまで待つことになり、レイテンシが隠蔽できなくなる。

更新型トランザクションの競合

一方、更新型書き込みでは、パケットに書き込みデータが保存されているので、パケット送出後、すぐにバッファを解放することができる。したがって、バッファ資源によって同時に未完了にできる要求パケット数が制約を受けることがなく、レイテンシが表面化することがない。緩和されたメモリ・モデルの利点を十分引き出すことができる。実際 JUMP-1 では、この方法を採用している。

3.3 トランザクションの処理順序

トランザクションの処理順序に関して、デッドロックと平均レイテンシの短縮について注意する必要がある。

3.3.1 デッドロックの回避

パケットを送信する必要が生じた時、ネットワークへの出口が塞がっていると、そのパケットの処理を進めることができなくなる。ここで、ただ出口が空くのを待つとデッドロックが生ずる。

デッドロックの対処には、①チャネルの多重化、②再試行、③十分長バッファが考えられる。

①チャネルの多重化 : デッドロックを防止するのに十分なチャネルを用意するのはハードウェアコストが過大になるため、この方法は JUMP-1 では採用していない。

②再試行 : この方法では、スターベーションを引き起こすのでエイジングと組み合わせなければならず、プロトコルが複雑になる。

③十分長バッファ : システム内に存在しうる要求パケットが全て収容できる、十分な長さの受信バッファを用意できれば、資源競合が発生しないので、デッドロックは起こらない。

十分長バッファが最も現実的な解と考えられるが、前節で述べたように更新プロトコルでは発行できる更新要求の数に制限がない。したがって、バッファの長さは無限長となり、退避/復帰をハードウェアのみで行うのは困難である。

3.3.2 平均レイテンシの短縮

パケットの処理の中には、通常のラインの処理と比較して、非常に長い時間のかかる処理がある。例えば、ページのリプレースが発生したときの、メインメモリへの書き戻しなどである。

後続の通常のライン処理を、このような長い時間のかかる処理の終了まで待たせると、平均レイテンシが著しく悪化する可能性がある。

また、3.2節で述べたように、競合発生時、先行処理の A が終了するまで、競合を起こした後続処理 B は待たされる。B が待たされるのはやむを得ないが、問題はその後に競合しないパケット C が到着した場合である。これを待たせる合理的な理由はなく、待たせると、やはり平均レイテンシが悪化する可能性がある。

この競合の問題に対しても、前節の①チャネルの多重化、②再試行、③十分長バッファが考えられる。

理想的には、B を待たせ、C を先に処理したいが、①チャネルの多重化では、この場合、ライン毎に多重化する必要があり、ハードウェアで行うことは困難である。また、B に②再試行を要求し、C の処理を先に行う方法も考えられるが、やはりスターベーションの

問題が生ずる。

③十分長バッファでは、B をバッファに退避して待たせておき、C を先に処理できれば理想的であるが、このような処理を全てハードウェアで行うのはやはり困難である。

4. Core における DSM 管理

本章では、前章での議論を踏まえて実装した、MBP Core の DSM 管理プログラムについて述べる。まず 4.1節で、MBP Core の説明を行い、4.2節で実装したプログラムについて述べる。

4.1 MBP Core

MBP-light は、MBP Core、MMC の他、パケット格納用のメモリ **PBR**(Packet Buffer Register) を内蔵する。

MBP Core

MBP Core は、基本的には、シンプルな 16b RISC プロセッサであり、16 本の 16b 汎用レジスタ (GPR) を持つ。外部のローカルメモリと呼ばれる SRAM を主記憶として動作する。キャッシュ、命令バッファなどは持たず、ローカルメモリに対するロード/ストアには 1 サイクルのストールを伴う。

PBR

PBR は 8b × 8B で、パケットフリットに 1 本割り当てられる。PBR にはネットワーク送受信用と汎用とがある。

ネットワーク送信用/受信用 PBR は、それぞれ 8 フリット × 3 パケットずつ用意されている。

汎用 PBR はスクラッチパッドとして利用され、64 フリット用意されている。

命令形式としては、PBR-GPR 間転送、PBR-PBR 間転送、PBR-GPR/PBR-即値演算命令がある。PBR-PBR 間転送では 8,16b 単位に加え、1 フリット単位での転送をサポートする。

パケット送受

ネットワークから到着するパケットは受信用 PBR に格納され、送信用 PBR に置かれたパケットは直接ネットワークに送出される。

一方、クラスタ内部とのパケットの送受信は、MMC 内部のバッファと PBR との間でパケットをコピーする必要がある。Core と MMC 内部との接続はネットワーク側に比較して弱く、このコピーには十数サイクルかかる。これは、MMC 内部は、クラスタバスと主記憶との接続に最適化されているからである。

特 殊 命 令

Core は、パケット処理向けの特殊な命令を持つ。特に、パケットのアドレス部分のハッシュ値を得る命令が効果的であり、通常ならば十命令以上かかる処理を 1 サイクルで実行できる

4.2 DSM 管理プログラムの設計方針

Core 上の DSM 管理プログラムの仕事は基本的に、到着したパケットに対して、データアレイやディレクトリを更新し、必要ならばパケットをクラスタ内外に送信することである。ただし、前章の議論から以下の点に注意する必要がある。

競合 競合を検出された場合、後続パケットの処理を先行パケットの処理が終了するまで待たせる必要がある。

デッドロックの回避 送信バッファが塞がって処理が先に進められない時、処理を中断し、パケットをバッファに退避する必要がある。

平均レイテンシの短縮 長時間かかるパケットの処理中に、後続パケットが到着したときには、処理を中断し後回しにするのが望ましい。

また、競合発生時にも、後続の競合を起こさないパケットは処理できることが望ましい。

これらの要求を満たすために、プログラムをマルチスレッド化する。到着したパケットに対して、それを処理するスレッドを生成し、スケジューリングの問題として対処する。プログラムは、スレッドの中断や再開、事象待ちなどをサポートする。

ただし、Core が行う処理には、長時間かかる処理はたしかに存在するが、ほとんどが中断の必要がなく短時間で終了する。短時間で処理が終了するパケットに対してもスレッドを生成すると、スレッド化のオーバーヘッドが大きくなり性能が悪化する可能性が高い。

そこで、パケットを 2 種類に分ける。長時間かかる処理、もしくは中断の必要な処理に関しては、スレッドを生成する一方で、短時間で終了する処理は、手続き的に実行する。また、短かい処理に対しては、高度なスケジューリングを行わず、割り込まれることなく一気に実行してしまう。

4.3 実 装

前節で述べた方針に従って、Core プログラムの実装を行った。スレッドの横取り、事象待ちなどをサポートするため、レディキュー、スレッドテーブルやイベント記述子等のデータ構造を持つ。

これらの他に **PTT**(Pending Transaction Table) と呼ばれるテーブル、各種ソフトウェア TLB なるデータ構造を持つ。

4.3.1 PTT

PTT は、クラスタの物理アドレスをキーとするハッシュテーブルである。未完了のパケットに関する情報を管理し、次のようにして競合の検出を行うために使われる。

パケットが到着すると、PTT を検索する。同一ラインに対する登録が既に存在すれば、競合が発生していることがわかる。競合が発生すれば、パケットをローカルメモリに退避し、PTT の対応エントリにキューイングし、先行パケットの終了を待たせる。

先行パケットの応答が返され処理が終了すると、対応するエントリが消去される。この際、競合を起こしていたパケットがキューイングされていれば先頭のものを実行可能状態にする。

ライン毎にキューイングがなされるので、競合発生時にも別のラインに対するパケットは処理することができる。

4.3.2 ソフトウェア TLB

2.2.1項で述べたように、Core では物理と大域仮想アドレスの変換を行う必要があり、そのためにページテーブル、逆引きページテーブルを参照しなければならない。また、Initiator では Home を求めるためのテーブル、Home ではディレクトリの検索を行う必要がある。これらのテーブルはクラスタメモリ上に置かれているが、クラスタメモリアクセスにはMMC に処理を依頼する必要があり、4.1節で述べたように、時間がかかる。

これらのテーブル参照を高速化するために、ソフトウェア TLB を実装する。クラスタメモリ上のテーブルの内容を、ローカルメモリ上のハッシュ表にキャッシングする。

この際、ページテーブルと Home を求めるテーブルの TLB は、両者ともページ番号に対する検索であるので統合する。

4.3.3 短いパケットの処理

処理時間の短いパケットの処理の流れがどのようになるかを説明する。

処理時間の短いパケットは、スレッド化のための処理をすべて省略し、割り込んだスレッドのスタック上で手続き的に処理される。この時割り込みを不許可として、中断することなく一気に実行する。

パケットの到着

パケットの到着は割り込みによって知らされる。短いパケットの処理中は割り込みを不許可にするので、割り込みはスレッド実行中である。無負荷状態はアイドル・スレッドが実行されており、この時はコンテクストの退避は省略できる。

実行準備

短いパケットの処理は中断なく行う。そのために、処理を始める前に実行準備をしておく。即ち、PTT を用いて競合を起こしていないことと、必要な送信バッファが空いていることを確認しておく。送信バッファが塞がっていればローカルメモリにパケットを退避し、送信バッファの空きを示すイベント記述子にキューイングする。このキューが十分長バッファとして機能する。

実行

パケットの種類に従って、データアレイやディレクトリの更新等を行う。必要ならばパケットの送信がなされる。既に必要な送信バッファの空きが確認されているので中断なく実行できる。

終了

実行の結果、応答パケット待ちになったならば、競合検出のために PTT に情報を登録しておく。また、応答パケットの実行が終了したのならば、PTT の対応する登録を消去する。

これらの処理が終了したら、直前に走っていたスレッドが再開される。

5. 性能評価

本章では実装した Core プログラムを実機上で動作させ、JUMP-1 DSM の基本的な性能を評価する。

5.1 計測結果

Core プログラムはアセンブリ及び C で記述し、C コンパイラとしては gcc-2.8.1 を Core 向けにポーティングしたものを用いた。

基本性能として、読み出し要求が 3 次キャッシュでミスし、Home でヒットした場合のレイテンシを計測した。

トランザクションは次の 3 つからなる。

① Initiator で 3 次キャッシュ・ミスし Core に割り込みがかかる。Core は Home に要求を転送する。

② Home はディレクトリを検索し主記憶が有効であることを確認、MMC に依頼して主記憶を読み出す。Initiator に応答を返す。

③ Initiator は応答パケットをクラスタ内に転送する。

計測は無負荷状態で行った。即ち、パケット到着時 Core はアイドル・スレッド実行中であり、また送信バッファが塞がっていることはない。また各種 TLB は全てヒットするとした。

表1に、リードレイテンシの計測結果を示す。表中、処理の項目の番号は上記のトランザクションの内訳と一致する。またサイクル数の () は、Core プログラム/それ以外のハードウェア、即ち MMC による処理時間を表す。

全体のレイテンシは 519 サイクルとなり、そのうち Core による処理は 338 サイクルとなった。Core の処理の、ハードウェア部分に対する割合、ソフトウェアオーバーヘッドは 186.7% となる。

5.2 考　　察

Core による処理時間の増加の要因として、以下のものが挙げられる。

(1) Core が 16b プロセッサであるため、16b を越えるデータの扱いに時間がかかる。

(2) パケット・ヘッダのフィールドがバイト境界になく、扱う際にはシフトやマスク操作を行う必要がある。

(3) ロード/ストア命令は、1 サイクルのストールが伴うため、各種データ構造のアクセスに時間がかかる。

(1)(2) は特に、ページテーブル検索、PTT 検索、

ディレクトリ検索に影響が大きい。これらテーブル検索はアドレスのうち、ページ番号、ブロックアドレスの部分をキーとする。これらのデータは 16b を越え、しかもパケット・ヘッダのバイト境界にない。Core がこのようなデータの扱いに時間がかかることは、DSM 管理プロセッサとして大きな弱点であるといえる。

(1)(2) が改善された場合のプログラムを記述し、手でサイクル数を求めたところ、全体で 56 サイクル削減できるとの結果を得た。

また (3) に関しては、キャッシュ・命令バッファを装備することで、さらに最大 30 サイクル削減できることが期待できるとの結果を得た。

したがってこれらの改良により、全体のレイテンシは 433 サイクルまで削減できることが期待できる。また、Core による処理は 252 サイクル、ソフトウェアオーバーヘッドは 139.2% となる。

処理内容		サイクル数
①	送信バッファの確認	8 (8/ 0)
	MMC からのパケット転送	24 (10/ 14)
	アドレス変換, home の検索	26 (26/ 0)
	パケット種判別	23 (23/ 0)
	ヘッダ作成	31 (31/ 0)
	小計	112 (98/ 14)
②	送信バッファの確認	13 (13/ 0)
	アドレス変換	26 (26/ 0)
	PTT 検索	17 (17/ 0)
	パケット種判別	10 (10/ 0)
	ディレクトリ検索、状態判定	30 (30/ 0)
	主記憶読み出し	27 (10/ 17)
	ヘッダ作成	35 (35/ 0)
	小計	158 (141/ 17)
③	送信バッファの確認	8 (8/ 0)
	アドレス変換	26 (26/ 0)
	PTT 検索	29 (29/ 0)
	パケット種判別	24 (24/ 0)
	ヘッダ作成	10 (10/ 0)
	MMC へのパケット転送	19 (2/ 17)
	小計	116 (99/ 17)
クラスタ内パケット転送		45 (0/ 45)
クラスタ間パケット転送		88 (0/ 88)
合計		519 (338/181)

表 1　読み出し要求処理時間

6. おわりに

本稿では JUMP-1 の DSM 管理について述べた。

JUMP-1 では、3 次キャッシュによってメモリアクセスの平均レイテンシの短縮を図る一方で、ノード間の処理は MBP Core のプログラムによって柔軟に行うというアプローチを採る。

Core プログラムを実装し、実機上で Home 主記憶アクセスのレイテンシを計測した結果 519 サイクルになるとの結果を得た。また、ソフトウェアオーバーヘッドは 186.7%、Core の自明な改良により 139.2%になると分かった。

JUMP-1 の実装は、.5〜.4μm のテクノロジの時代ものであり、このデータは多少古いものとなっている。現在、もしくは将来のテクノロジを用いればネットワークの遅延に対して DSM 管理を行うプロセッサの性能は著しく向上するため、このソフトウェアオーバーヘッドは更に小さくなると考えられる。

したがって DSM におけるノード間の処理をソフトウェアで柔軟に行うことは、将来その重要性がさらに増していくと考えられる。

今後より実際的なアプリケーションを動作させ評価を行う予定である。

謝　辞

JUMP-1 の基本的アイディアは、松本尚氏による。本プロジェクトに携わった方々には、幾多の御助言、御教示を賜った。また JUMP-1 の開発にあたっては、多くの企業の御賛助を賜った。ここに深甚なる謝意を表したい。

なお, 本研究の一部は文部省科学研究費補助金 (基盤研究 (B)(2) 課題番号12480072 ならびに 12558027) による。

参考文献

1) Goshima, M., et al.: The Intelligent Cache Controller of a Massively Parallel Processor JUMP-1, *IWIA'97*, pp. 116 –124 (1997).

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, *JSPP'97*, pp. 265–272 (1997).

3) Li, K.: IVY: A Shared Virtual Memory System for Parallel Computing, *ICPP'88*, pp. 94–101 (1988).

4) 西村克信他: 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).