並列計算機 JUMP-1 における分散共有メモリ管理の実装

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，リモートに対する大容量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プログラムベースで行う．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，リモート主記憶読み出しのレイテンシは 441 サイクルとなった．またハードウェアのレイテンシに対するソフトウェアオーバヘッドは 143.6%となった．

Implementation of Distributed Shared Memory Management 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 flexibly performed by program. In this paper we discribe this program and evaluate the performance. The resulut shows that it takes 441 cycles to acccess remote memory and the ratio of the software overhead to the hardware latency is 143.6%

1. はじめに

計算機ハードウェアは筐体，マザーボード，チップといった要素により階層化がなされている．階層的な実装はアーキテクチャ，特にメモリシステムの階層化を促す．

このような傾向は，LSI の集積度が向上しても緩和されるものではない．LSI の集積度の向上に従い，ゲートや短い配線の遅延に対する，ワード線やビット線といった長い配線の遅延の差が大きくなるからである．大容量のメモリをプロセッサコアと同等の速度で動かすことはますます困難になる．

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，高コストとなる．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，アーキテクチャは実装の物理的な階層構造を受け入れ，そのための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．

そこで，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と，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．

JUMP-1 では，各クラスタに分散された主記憶の一部を，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する．クラスタ内の要素プロセッサは 1 次，2 次キャッシュを持つため，この主記憶の一部はクラスタ内で共有される 3 次キャッシュとして機能する．

この 3 次キャッシュでは，大容量であることから高いヒット率を期待することができ，ミス時のレイテンシに対する要求を低く抑えることができる．そこで，クラスタ間のキャッシュ制御をソフトウェアで行うことを考える．JUMP-1 では，各クラスタのクラスタ間ネットワークに対するインタフェース部に，コアプロセッサ **MBP Core** を内蔵する **MBP-light** と呼ばれる LSI が配置されている．クラスタ間の処理は MBP Core 上のソフトウェアによって行う．

† 京都大学情報学研究科
Graduate School of Informatics, Kyoto University

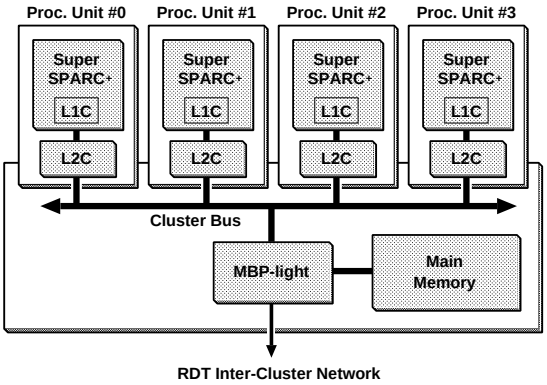

図 1 JUMP-1 のクラスタ
Fig. 1 JUMP-1 Cluster.

本稿では，この MBP Core 上での DSM 管理プログラム設計上の問題点を明らかにし，実装されたプログラムの性能を評価を行う．

まず 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．3 章で DSM 管理における問題点について論じた後，4 章でそれをふまえて実装を行った DSM 管理プログラムについて説明する．最後に 5 章で評価結果について述べる．

2. JUMP-1 の概要

本章では，まず 2.1 節で JUMP-1 全体の構成を説明し，2.2 節で MBP Core について述べる．2.3 節では JUMP-1 の DSM について説明する．

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．

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と呼ばれるバスによって接続され，高バンド幅/低レイテンシの通信を行うことができる．

プロセッサユニットは主に，要素プロセッサ Super-SPARC+ と，プライベートキャッシュからなる．キャッシュはプロセッサ内蔵 1 次と，外部 2 次キャッシュ[1] の 2 階層からなる．

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，MBP-light[2),3)]からなる．MBP-light については次節で詳述する．

クラスタ間は **RDT**（Recursive Diagonal Torus）[4] 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．

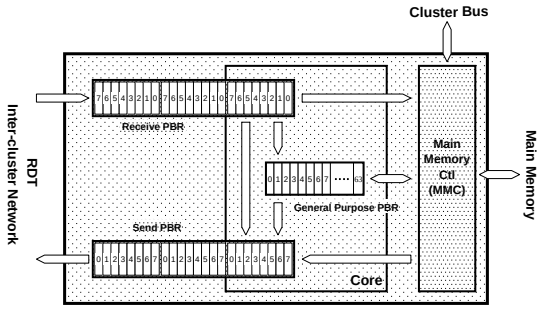

図 2 MBP-light 内部のデータ・パス
Fig. 2 Internal data paths of MBP-light.

2.2 MBP-light

図 2 に MBP-light 内のデータ・パスを示す．MBP-light は主に，コア・プロセッサ **MBP Core** と Main Memory Controller **MMC** からなる．

MBP Core の設計方針

MBP Core は，そのために割けるゲート数が限られていたため，面積効率を第 1 の目標として設計された．ゲートの消費量を抑えるため，基本となるアーキテクチャをシンプルなものとする一方，少ないゲート数で実現できるパケット操作向けの特殊な命令を用意することによって実行サイクル数の削減を図るアプローチをとっている．実際 MBP Core は，ロジックとしては 13,355 ゲートという少ないゲート数で実装されている．

MBP Core は，基本的には，ごくシンプルな 16 bit スカラ RISC プロセッサである．16 本の 16 bit 汎用レジスタ（**GPR**）を持ち，ロード/ストア・アーキテクチャを採用する．

記憶階層もシンプルである．キャッシュ，命令バッファ，ストアバッファなどは持たず，1 サイクルでアクセス可能な外付けの SRAM を主記憶として動作する．この SRAM は MBP Core の**ローカル・メモリ**と呼ばれる．このような構成のため，ロード/ストア実行時には 1 サイクルのストールが発生する．

MBP Core はパケット操作向けの特殊命令を持つ．特殊命令には以下のものがある：

- パケットバッファに対するアクセス命令
- パケット送受信命令
- その他の特殊命令

これらの命令について，以下で詳しく述べる．

PBR

MBP-light は **PBR**（Packet Buffer Register）と呼ぶ，特殊なメモリを持つ．PBR には，送受信用と汎用がある．

送受信用 PBR は，その名のとおり，ネットワークに対するパケットの送受信用バッファとして機能する．送信用/受信用 PBR はそれぞれ，8 bit×8B×8 フリット×3 パケットの語構成となっている．

汎用 PBR は，スクラッチ・パッドとして利用する．そのため，送受信用とは違ってパケットの境界はなく，8 bit×8 Byte×64 フリットの語構成となっている．

PBR アクセス命令

PBR は，GPR ほどではないが，ローカル・メモリよりは小容量である．そのため，2 ポート（1-read/1-write）化されており，Core からのアクセスには，通常考えられるようなメモリ・マップ I/O ではなく，専用の命令を用いる．

PBR は，命令形式上はメモリとして扱われる．すなわち，PBR 番号は，GPR+4 bit 即値によるレジスタ間接で指定される．これは主に，命令の幅が不足しているためである．

命令形式としては，PBR–GPR 間の転送命令のほか，PBR–PBR 間の直接転送命令や，CISC のメモリ–レジスタ演算に似た PBR–GPR，PBR–即値間の演算命令が用意されている．PBR–PBR 間の直接転送命令は，8 bit，16 bit に加えて，1 フリット単位での転送をサポートする．

PBR–GPR，PBR–即値演算命令は，パケットのヘッダを一部修正するのに都合がよい．

パケットの送受

ネットワークから到着するパケットは受信用 PBR に直接格納され，送信用 PBR に置かれたパケットはネットワークへ直接送出される．

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 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対して弱く，クラスタ内部とのパケットのやりとりは十数サイクルかかる．

特殊命令

MBP Core は，そのほかにもパケットの処理向けの特殊な命令を持つ．特に，PBR のパケット・ヘッダのアドレス部分のハッシュ値を得る命令が効果的である．この命令は，通常の RISC の命令セットなら数命令かかる処理を，1 サイクルで実行することができる．

2.3 JUMP-1 DSM

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．

2.3.1 アドレス体系

3 次キャッシュのアドレス体系は SVM[5]に準ずる．SVM ではキャッシングはページ単位でなされる．リモートにあるオリジナルページをキャッシングするには，ローカルのクラスタメモリ上にページ枠を確保し，オリジナルページをコピーする．

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，アクセスされる物理アドレスがオリジナル/コピーにかかわらず，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，物理アドレスを基に行えばよい．ページサイズは SuperSPARC+MMU では 4 KB である．

SVM では，1 つの仮想ページに対し，クラスタごとに自由に物理アドレスを割り当てることができる．そのためクラスタ間では，オブジェクトを一意に特定する大域仮想アドレスを必要とする．このようなアドレスとしては，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．その場合，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が必要となる．

2.3.2 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，32 B のライン単位で一貫性制御を行う．そのため，クラスタメモリ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，クラスタ間の一貫性制御は分散ディレクトリ方式で行う．

ディレクトリ方式

ディレクトリのマップの形式は MBP Core のプログラム次第である．SVM のオリジナルとなるページを持つクラスタに，そのページのディレクトリも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** と呼ぶ．

Home のマッピングもやはり自由なので，クラスタ間の一貫性制御を行う場合には，Home 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

一貫性維持動作としては，無効化型，更新型，その組合せをサポートする．

共有状態の管理

ラインの状態として，アーキテクチャの階層性に合わせて，コピーの存在範囲が，プロセッサ，クラスタ，

システム全体であることに対応する，exclusive，local shared，global shared の 3 つを持つ．

2 次キャッシュコントローラは，スヌープに対して状態を応答し，主記憶アクセスの頻度を軽減する．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 は共有状態を示すタグを読み出し，local であった場合にはハードウェア的に処理する．global であった場合には，MBP Core に処理を依頼する．したがって一貫性制御を行ううえでの MBP Core の役割は，クラスタ間の処理が必要である場合に限定される．

3. DSM 管理

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，基本的には，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，必要ならばパケットを送出することでなされる．ただし，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よいわけではない．

本章ではトランザクションの処理順序に関して議論する．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

トランザクションに関わるクラスタは次のように分類される．2.3.2 項で述べたように，ラインのディレクトリを持つクラスタを Home と呼ぶ．システム内で唯一の有効なラインを所持するクラスタを **Owner**，Home 以外で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．

要求パケットが，①Initiator から Home，②Home から Owner/Renter へ送信され，要求に対する応答パケットが③Owner/Renter から Home，④Home から Initiator に返される．当然，共有状態によっては②③が省略されることもある．

パケットの流れは MBP Core のプログラム次第であるが，本稿では簡単のため，三角通信を行わず，いったん Home を経由するものとする．

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．このとき，Home では，先行トランザクションの③応答パケットが到着しないうちに，①後続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を待たずに，後続パケットの処理を始めるのは面倒であるので，普通トランザクションの逐次化を行う．すなわち，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き，競合を検出する．競合が検出された場合には，先行トランザクションの応答が返るまで，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．

競合に負けた場合には，当該ラインに対する④Home からの応答が返る前に，別クラスタから開始された勝ったトランザクションの②Home からの無効化要求が到着する．この場合，その無効化要求を受け入れ，自コピーを無効化する必要がある．

したがって，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，相撃ちに負けなかっ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，負けていた場合，別クラスタからの無効化要求によって書き込みの内容が失われてしまうからである．

この対処には，負けなかったことが判明するまで，ライトバッファに書き込み内容を保存しておく方法がとられる．負けなかったことが分かるのは，④Home から応答パケットが返されたときであるので，それ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．

したがって，ライト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．JUMP-1 では各プロセッサごとに 3 つまでしか同時に未完了にできない，それを超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ライトバッファが解放されるまで待つことになり，ライトレイテンシが表面化することになる．

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，更新要求パケット自体に書き込み内容が保存されているので，特別な配慮が必要ない．

すなわち，競合に負け，他クラスタからの更新要求によって自コピーが更新された場合でも，自分が送信した更新要求パケットが②Home から戻ってきたときに，パケット内の情報を用いて自コピーを正しく更新することができる．

したがって，パケット送出後，ただちにライトバッ

ファを解放することができる．この場合，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 の 2 次キャッシュコントローラは，更新要求の送出後はただちにバッファを解放する方法をとっており，無制限の更新要求を同時に未完了にすることができる．

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，デッドロックとレイテンシについて注意する必要がある．

3.3.1 デッドロック

パケットを送信する必要が生じたとき，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．以下，それぞれについて説明する．

①チャネルの多重化

3.1 節で述べたように，トランザクションは 4 つのフェーズからなる．したがって，ネットワークには 4 本のチャネルを用意すればデッドロックは防止できる．これに加えトポロジに起因するデッドロックの防止も考慮する必要がある．トポロジのために n 本のチャネルが必要だとすれば，最悪 $4 \times n$ 本のチャネルが必要になる．

ハードウェアコストが過大になるため，JUMP-1 ではネットワークが必要なチャネルを備えていない．

②再試行

再試行では，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．また，再試行を行うためには，送信元に再試行を行えるだけの情報を残しておく必要がある．これは前節で述べた，無制限の要求を同時に未完了にできる更新要求の利点を消すことになり，JUMP-1 では採用できない．

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すべて収容できる，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．

Cenju-4 ではこの方法を採用している．Cenju-4 では，プロセッサあたり 4 個のトランザクションしか未完了にできないので，システム内の要求パケット数の上限およびそれをすべて収容するのに十分なバッファの容量をあらかじめ求めることができる．バッファ本体はノードの主記憶上にあらかじめ確保され，退避/復帰はハードウェアで行われる[6]．

JUMP-1 でも十分長バッファを採用する．前項で述べたように，JUMP-1 では緩和されたメモリ・モデルの利点を引き出すために，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，バッファをあらかじめ確保しておくことはできず，退避/復帰をハードウェアのみで行うのは困難であり，MBP Core がソフトウェアで処理することになる．

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，以下のような非常に長い時間のかかる処理がある．

マルチキャスト Home から Renter に無効化/更新要求を送信する際に，性能上の理由から，ネットワークのマルチキャスト機能を使わず，Renter ごとにユニキャストした方がよい場合がある．

競合 競合した場合には先行する処理の終了まで後続の処理は待たされる．

ページ単位の処理 3 次キャッシュのリプレースはページ単位で行われる．

高機能アクセス Core のプログラムで対応する，複雑な高機能アクセスが定義されている[1]．

平均レイテンシの短縮のためには，Short-Job-First で処理することが望ましい．すなわち，長い時間がかかる処理の最中に後続の要求パケットが到着した場合には，長い時間がかかる処理を中断し，後続のパケットを先に処理する方がよい．

競合に関しては，対処方法によってレイテンシ以外の問題が発生することがある．

競合が発生した場合には，先行する処理 A が終了するまで，後続の処理 B は待たされる．B に時間がかかるのはやむをえないが，問題は，B の後に競合しないパケット C が到着した場合である．C を待たせる理由はなく，理想的には B を待たせ，C を先に処理したい．

B に再試行を要求し，C を先に処理する方法が考えられるが，やはりスターベーションが発生する．

そこで前節と同様に十分長バッファを用意し，B をバッファに退避し C を先に処理する方法が考えられる．しかし JUMP-1 では前節と同じ理由で，あらかじめバッファを確保しておくことができず，ハードウェアで処理するのは困難になる．

Cenju-4 では，十分長バッファに B 以降のすべての要求を退避させる方法をとっている．しかしこのバッ

ファは FIFO であり，C も待たせることになる[1)]．

4. Core における DSM 管理

本章では，前章での議論をふまえて実装した，MBP Core の DSM 管理プログラムについて述べる．まず 4.1 節でプログラムの設計方針について述べ，4.2 節でその実装について述べる．

4.1 DSM 管理プログラムの設計方針

MBP Core 上の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．パケット到着順に処理できれば問題はないが，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 送信バッファが塞がって処理が先に進められないとき，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 長い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，通常の OS などと同様にスレッドの横取り，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 MBP Core プログラムでは，以下のようにスケジューリングを行う．

まず，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．以降，前者を **Long-lived パケット**，後者を **Short-lived パケット** と呼ぶことにする．

Long-lived パケットに対するスレッド間では Round-Robin でスケジューリングを行い，Long-lived パケットと Short-lived パケットの間では，Short-Job-First でスケジューリングを行う．Long-lived パケットの処理中に Short-lived パケットが到着した場合には，処理を中断し，Short-lived パケットの処理を先に実行する．

Short-lived パケットの処理は手続き的に実行する．パケット到着時に実行されていた，Long-lived パケットのスレッドのスタック上で，割込みを禁止し，中断することなく実行する．これにより，スレッド記述子やスタックの割付け，レディーキューの登録などスレッド化のための処理をすべて省略することができる．

4.2 実　　装

Core プログラムは，スレッドの横取り，事象待ちなどをサポートするため，通常の OS と同様のスレッドテーブル，レディーキュー，イベント記述子などのデータ構造を持つ．これらのデータ構造は，Core の主記憶であるローカルメモリに置かれる．

これらのほかに **PTT**（Pending Transaction Table）と呼ばれるテーブルを必要とする．またアドレス変換を高速化するため，ソフトウェア TLB を用意する．

4.2.1 PTT

PTT は，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，主に競合の検出を行うために使われる．

Home が要求パケットを受信したときには，まず PTT を検索する．同一ラインに対する登録がすでに存在すれば，競合が発生していることが分かる．競合が発生すれば，パケットをローカルメモリ上に退避し，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たときには，対応するエントリを削除する．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 1 つを実行可能状態にする．

PTT ではラインごと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．

また競合の検出以外に，応答が返された場合，対応する要求パケットに関する情報，たとえば要求パケットの送信元などを得るためにも使用される．Home 以外のクラスタではこの目的にのみ使われる．

4.2.2 ソフトウェア TLB

2.3.1 項で述べたように，パケットをクラスタ外に送信するときには，物理から大域仮想へ，受信したときはその逆のアドレス変換を行う必要がある．そのためにはページテーブル，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の参照，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，アクセスするためにはMMCに処理を依頼する必要があり，2.2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をキーとする検索であるので統合する．

また，ディレクトリのTLBとPTTはライン単位でのアドレスをキーとする検索であるので，やはり統合する．

4.2.3 Short-lived パケットの処理

Short-lived パケットの処理は以下のように進められる．

パケットの到着

パケットの到着は割込みによって知らされる．Short-lived パケット処理中は割込みを不許可にするので，割込みが発生するの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，このときはコンテクストの退避/復帰も省略できる．

実行準備

Short-lived 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．具体的には，Homeであれば PTT を用いて競合を起こしていないこと，そして，必要な送信バッファが空いていることを確認する．

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などを行う．必要ならばパケットの送信がなされる．すで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．

表 1 リードレイテンシ
Table 1 Read latency.

	read Home	read Owner
→①	99 (85/ 14)	99 (85/ 14)
①→②	— (—/ —)	110 (110/ 0)
②→③	— (—/ —)	159 (128/ 31)
③→④	128 (111/ 17)	114 (100/ 0)
④→	81 (64/ 17)	81 (64/ 17)
クラスタ内 HW	45 (0/ 45)	90 (0/ 90)
クラスタ間 HW	88 (0/ 88)	176 (176/176)
合計	441 (260/181)	829 (501/328)

5. 性能評価

本章では，JUMP-1 DSM の基本的な性能を評価する．

5.1 リードレイテンシ

クラスタ間トランザクションのうち最も基本的な読み出し要求に対して，そのレイテンシを実機上で計測した．計測には，MBP-light の持つモニタ用のカウンタを用いた．

Coreプログラムは，基本的にはC言語を，性能上クリティカルな部分に関してはアセンブリ言語を用いて記述した．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．

計測結果を**表 1** に示す．

①〜④は，3.1節で述べたトランザクションの4つのフェーズを示す．たとえば，①→②は，要求パケットが ①Home に届いてから，②Owner に向けて送信するまでの MBP Core の処理を表す．

左側の列は Home でヒットし②と③が省略できた場合，右側は Home でミスし Owner が応える場合である．

クラスタ内 HW とは，要素プロセッサの命令パイプラインがロード/ストア命令を実行してから MBP Core に割込みがかかるまでの時間と，MBP Core が応答パケットの転送を MMC に依頼してから要素プロセッサの命令パイプラインが再び動き出すまでの時間の合計である．

クラスタ間HWとは，クラスタ間ネットワークRDTにおけるパケットの転送時間を表す．

表中の数値は各フェーズに要した処理サイクル数を表し，括弧内は，そのうちの Core プログラム/それ以外のハードウェアによる処理サイクル数を表す．

要求パケットが到着したとき Core はアイドル・スレッド実行中とした．したがって，コンテクストの切替えの処理は不要である．また，各種ソフトウェア TLB

表 2 読み出し要求処理のレイテンシ
Table 2 Latency to service a read request.

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	23 (9/ 14)
	アドレス変換，home の検索	32 (32/ 0)
	パケット種 判別	10 (10/ 0)
	ヘッダ作成	26 (26/ 0)
	小計	99 (85/ 14)
①→④	送信バッファの確認	7 (7/ 0)
	PTT，ディレクトリ検索	18 (18/ 0)
	アドレス変換	32 (32/ 0)
	パケット種 判別	10 (10/ 0)
	主記憶読み出し	31 (14/ 17)
	ヘッダ作成	30 (30/ 0)
	小計	128 (111/ 17)
④→	送信バッファの確認	8 (8/ 0)
	PTT 検索，アドレス変換	34 (34/ 0)
	パケット種 判別	10 (10/ 0)
	ヘッダ作成，送信	10 (10/ 0)
	MMC へのパケット転送	19 (2/ 17)
	小計	81 (64/ 17)
クラスタ内 HW		45 (0/ 45)
クラスタ間 HW		90 (0/ 90)
合計		441 (260/181)

はすべてヒットするものとした．

5.2 Home に対する読み出しトランザクションのレイテンシ

本節以降では，表 1 に示した，読み出し要求が Home でヒットした場合について述べる．

まず，このトランザクションのレイテンシの内訳を，**表 2** に示す．括弧内は，表 1 と同様，処理サイクル数のうちの Core プログラム/それ以外のハードウェアによる処理サイクル数を示している．表に示すように，このトランザクションのレイテンシは 441 サイクル，50 MHz 動作時には約 8.82 μsec となる．このうち Core の処理時間は 260 サイクルで，ソフトウェア・オーバヘッド，すなわち，Core の処理時間のそれ以外のハードウェア部分に対する割合は 143.6%となる．

5.3 高速化手法の評価

本節では，ソフトウェア TLB と，Short-lived パケットに対してスレッドを生成しないことの効果について述べる．

ソフトウェア TLB の効果

TLB を用いない場合，処理→①と①→④において，それぞれ以下の処理に対してサイクル数が増える．TLB を用いない場合の処理サイクル数も含めて示す．

- →①：
  - 物理から大域仮想へのアドレス変換 64 サイクル
  - Home の検索 43 サイクル
- ①→④：
  - 大域仮想から物理へのアドレス変換 153 サイクル
  - ディレクトリの検索 43 サイクル

TLB にヒットした場合には，アドレス変換は→①と①→④のそれぞれにおいて，32 サイクルで処理することができる．また Home の検索はアドレス変換と統合され，ディレクトリの検索は PTT の検索と同時に行えるので，トランザクション全体では，239 サイクルの高速化が達成されている．

スレッド生成省略の効果

処理 →①，①→④，および，④→ のそれぞれに対してスレッドを生成した場合，Home に対する読み出しトランザクションのレイテンシは 586 サイクルとなる．スレッドの生成を省略することによって，145 サイクルの高速化が達成されている．

5.4 Core の改良

本節ではサイクル数増大の要因となる MBP Core の弱点をあげ，これが解決された場合の Home に対する読み出しトランザクションのレイテンシを見積もる．

MBP Core の弱点として以下があげられる．

①**パケットヘッダフィールド** クラスタバスのプロトコル設計時には，プロセッサによる扱いは想定されていなかったため，パケットヘッダの各フィールドはバイト境界に整列していない．そのため，フィールドを取り出すために，余分なロード/ストア，マスク，シフト操作が必要となっている．

②**16 bit プロセッサ** Core は 16 bit プロセッサであるため，アドレスなどの 16 bit を超えるデータの扱いに時間がかかる．

③**I/D 非分離** ロード/ストア命令は 1 サイクルのストールをともなうため，ローカルメモリ上の各種データ構造へアクセスを行うと，速度が低下する．

パケットヘッダのフィールドがバイト境界に整列され，MBP Core アーキテクチャを，32 bit 化，I/D 分離化したものを考える．Home に対する読み出しトランザクションのレイテンシは上記それぞれにより，①20，②30，③18 サイクル削減され，全体のレイテンシは 373 サイクルとなる．この場合，ハードウェアのレイテンシに対するソフトウェア・オーバヘッドの割合は 106.1%となる．

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とる．

Core プログラムを実装し，評価を行った結果，3 次キャッシュミスにおける，ハードウェアのレイテンシに対するソフトウェアオーバヘッドは 143.6%となった．また Core の 32b 化，I/D 分離化によりこれは，106.1%になることが分かった．

このデータは，たとえノード間の処理のすべてをハードウェアで行ったとしても，レイテンシは 1/2 程度にしかならないことを示している．

JUMP-1 の実装は，.5〜.4 μm 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ヘッドはさら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．

参 考 文 献

1) Goshima, M., Mori, S., Nakashima, H. and Tomita, S.: The Intelligent Cache Controller of a Massively Parallel Processor JUMP-1, *IWIA'97, Int'l Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems*, pp.116–124 (1997).
2) 佐藤　充，天野英治，安生健一郎，周東福強，西　宏章，工藤知宏，山本淳二，平木　敬：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) 安生健一朗，井上浩明，佐藤　充，工藤知宏，天野英晴，平木　敬：超並列計算機 JUMP-1 における分散共有メモリ管理プロセッサ MBP-light，情報処理学会論文誌，Vol.39, No.6, pp.1632–1643 (1998).
4) 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).
5) Li, K.: IVY: A Shared Virtual Memory System for Parallel Computing, *ICPP'88*, pp.94–101 (1988).
6) 細見岳生，加納　健，中村真章，広瀬哲也，中田登志之：並列計算機 Cenju-4 の分散共有メモリ機構，*JSPP'99*, pp.15–22 (1999).

(平成 12 年 8 月 31 日受付)
(平成 12 年 12 月 1 日採録)

小西　将人

1976 年生．1992 年綾部市立八田中学卒業．1995 年京都府立綾部高校卒業．1999 年京都大学工学部情報学科卒業．同年同大学大学院情報学研究科修士課程に進学．並列計算機アーキテクチャの研究に従事．

五島　正裕（正会員）

1968 年生．1992 年京都大学工学部情報工学科卒業．1994 年同大学大学院工学研究科情報工学専攻修士課程修了．同年より日本学術振興会特別研究員．1996 年京都大学大学院工学研究科情報工学専攻博士後期課程退学，同年より同大学工学部助手．1998 年同大学大学院情報学研究科助手．高性能計算機システムの研究に従事．

森　眞一郎（正会員）

1963 年生．1987 年熊本大学工学部電子工学科卒業．1989 年九州大学大学院総合理工学研究科情報システム学専攻修士課程修了．1992 年九州大学大学院総合理工学研究科情報システム学専攻博士課程単位取得退学．同年京都大学工学部助手．1995 年同助教授．1998 年同大学大学院情報学研究科助教授．工学博士．並列/分散処理，計算機アーキテクチャの研究に従事．IEEE，ACM 各会員．

富田　眞治（正会員）

1945 年生．1973 年京都大学大学院博士課程修了，工学博士．同年京都大学工学部情報工学教室助手．1978 年同助教授．1986 年九州大学大学院総合理工学研究科教授．1981 年京都大学工学部情報工学科教授．1998 年同大学大学院情報学研究科教授．計算機アーキテクチャ，並列計算機システムに興味を持つ．著書「並列計算機構成論」,「並列処理マシン」,「コンピュータアーキテクチャI」など．電子情報通信学会，IEEE，ACM 各会員．