計算機アーキテクチャ 144−23
（２００１．７．２７）

並列計算機 JUMP-1 の性能評価

小西将人† 額田匡則† 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，行列積による評価を行った．その結果，2プロセッサで1.93，4プロセッサで3.72の台数効果を得ることができた．

Evaluation of the JUMP-1 Multiprocessor

Masahito Konishi,† Masanori Nukata,† 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 by Matrix Multiplication. The resulut shows that the rate of speed-up is 1.93 with 2 processors system, and 3.72 with 4 processors.

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管理について述べ評価を行う．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DSMの概要について述べる．次に第3章でクラスタ間DSM管理におけるパケット処理順序について述べ，これを踏まえて実装したDSM管理プログラムについて第4章で述べる．最後に第5章で実機上で評価を行う．

2. JUMP-1の概要

本章では，まず2.1節でJUMP-1の物理的構成についてまとめた後，その後，JUMP-1 DSMの概要について述べ，さらにJUMP-1 DSMにおいて重要な役割を果たすMBP-light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．

† 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.

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．

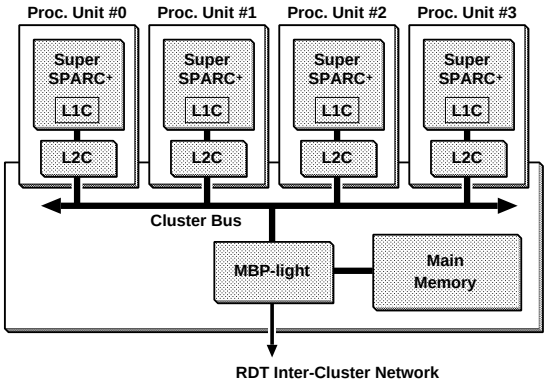

図 1　JUMP-1 のクラスタ

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 2.3節で詳述する．

JUMP-1 ではクラスタを **RDT**[3](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　JUMP-1 DSM

JUMP-1 では，クラスタメモリの一部，例えば半分程度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ことで，平均メモリアクセスレイテンシの削減を図る．要素プロセッサは，SuperSPARC+ 内蔵の 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．

この大容量 3 次キャッシュでは高いヒット率を見込むことができるため，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．JUMP-1 におけるキャッシュの一貫性制御は，クラスタ内に関してはハードウェアで高速で行う一方，クラスタ間では MBP Core 上で動作するソフトウェアで柔軟に行う方式を採っている．

JUMP-1 における一貫性維持操作の流れは以下のようになる．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，クラスタメモリアクセスの頻度を軽減する．キャッシュコントローラが処理できない場合に限り，MMC がクラスタメモリの状態を調べ，クラスタ内に閉じた処理になる場合にはそのまま MMC がハードウェア的に処理を行う．クラスタ間処理が必要な場合には，MBP Core にパケットを受け渡し，以降 MBP Core 上で動作するソフトウェアで処理が行われる．

2.2.1　アドレス体系

3 次キャッシュのアドレス体系は SVM(Shared Virtual Memory)[4] に準ずる．SVM ではキャッシングはページ単位でなされる．リモートクラスタにあるオリジナルページをキャッシングするには，自身のクラスタメモリ上の 3 次キャシュ領域にページ枠を確保し，オリジナルページをコピーする．

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．要素プロセッサ内の仮想アドレスから物理アドレスへの変換は，アクセスされる物理アドレスがオリジナル/コピーに関わらず，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う．

SVM では，1 つの仮想ページに対し，クラスタ毎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．これをネットアドレスと呼ぶ．

クラスタ間で一貫性制御を行う際，パケット送信時には物理アドレスからネットアドレス，受信時にはその逆の変換が必要となる．これは MBP Core によって行われる．物理アドレスとネットアドレスの対応は自由であるが，このような大域仮想アドレスは，仮想アドレスにプロセス ID を付加したものとするのが普通である．この場合にはアドレス変換には，逆引き/正引きページテーブルを参照する必要がある．

2.2.2　ディレクトリ

一貫性制御はクラスタ内のスヌープ方式に対して，クラスタ間では分散ディレクトリ方式で行う．

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．縮約階層ビットマップ形式を用いると，クラスタ間ネットワーク RDT のマルチキャスト/コンバイニング機能を利用することができる[3]．

ディレクトリの，クラスタへのマッピングは自由であるが，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．

2.3　MBP-light

MBP-light は 2.2節で述べたように，クラスタ内に

閉じた一貫性制御をハードウェアで行うMMC，クラスタ間の制御をその上で動作するソフトウェアで行うMBP Coreから主に構成される．この他にパケットを扱うため，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．

PBRは命令形式上はメモリとして扱われ，GPR+4bオフセットによるレジスタ間接で指定される．

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

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．

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ち，後続の到着パケットを受け付けなければ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デッドロックを防止するため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，JUMP-1では必要なチャネルを備えていない．また②

再試行はスターベーションを引き起こすためエイジングと組み合わす必要がありプロトコルが複雑になる．

したがって，JUMP-1では③十分長バッファを採用する．具体的には，システム内に存在可能な要求パケットの全てを収容できる，十分な長さのバッファを用意し，デッドロックの危険性が高まった時にパケットをバッファ本体に退避し，後続の到着パケットを受け付ける．

3.3.2 平均レイテンシの短縮

MBP Core は通常のライン処理と比較して，非常に長い時間がかかるものが存在する．このような長いトランザクション処理が MBP Core の処理を占有することによって，後続の通常ライン処理が処理されないことになると，平均レイテンシの悪化につながる．

また，3.2節で述べたように，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，競合する後続処理 B は待たされる．しかし，B のために A を終了を待つことのみに MBP Core を占有されると，さらに後に届く競合しないパケット C が処理されないことになり，やはり平均レイテンシの悪化を招く．

以下では，長いトランザクションと競合に関して，平均レイテンシ短縮のためにパケット処理順序における注意点を述べる．

長いトランザクション

長いトランザクションの例は，ページの書き戻し処理がある．2.2.1項で述べたように，ページフォールト時に 3 次キャッシュが溢れ新たなページ枠が確保できない場合には，リプレース処理を行う必要がある．ページ単位の処理には，通常のキャッシュライン単位の処理と比較すると明らかに長い処理時間を要する．

平均レイテンシの短縮ためには，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，後続パケットを先に処理する方がよい．

競　　合

競合については，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．できれば，競合した B を待たせたままで，後続の競合しない C を先に処理することが望ましい．この対処には再試行，十分長バッファが考えられる．

前者では B に再試行を要求しおいて，C の処理を行う．しかし，スターベーションの問題が残る．

十分長バッファでは，B をバッファに退避しておき，C を先に処理する方法が考えられる．このバッファは，前項で述べた十分長バッファと同様に，システム内に存在可能な要求パケットをすべて収容できる容量が必要である．

4. DSM 管理プログラム

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．

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　競合が検出された場合，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，到着時に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．

4.1 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を持つ．

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，以下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．

4.2 パケットの処理

到着したパケットに対する処理の流れがどのようになるかを説明する．

(1) パケット到着
パケットが到着すると，実行中のスレッドが横取りされ，コンテクストの退避が行われる．ただし，アイドルスレッド実行中であった場合にはこの処理は省略される．

(2) 実行準備
パケット実行に必要な情報を準備する．
具体的には，アドレス変換，Home の検索，パケット種の判別，ディレクトリの検索，競合の検出，送信バッファの確認を行う．
当然 Home 以外では，ディレクトリの検索と競合の検出は省かれる．
競合を起こしている，もしくは必要な送信バッファが塞がっている場合には，ローカルメモリにパケットを退避し，PTT，もしくはバッファの空きを示すイベント記述子に登録し待たされる．これらの場合，ここでパケット処理には移らず直前に走っていたスレッドを再開し，後続のパケットを受け付けることになる．
この 2 つが確認できたならば，実行に移る．

(3) 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い，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．

(4) 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

この章では，実装した DSM 管理プログラムを実機上で動作させ，評価を行う．5.1節では基本的なトランザクションの処理時間を計測する．また 5.2では，行列積プログラムを実行し評価を行う．

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，gcc-2.8.1 を MBP Core 向けにポーティングしたものを用いた．

5.1 基本性能

基本性能として，読み出し要求の処理時間を計測した．

計測結果を 1に示す．

表中 () 内は，Core プログラム/それ以外のハードウェア の処理サイクル数を表す．

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．例えば，①→② は，要求パケットが ① Home に届いてから，② Owner 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 の主記憶でヒットし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．Home の主記憶でヒットするとは，ディレクトリ検索の結果，共有状態が global shared であり，主記憶に有効なデータが存在すると判断した場合である．一方，Owner はクラスタメモリ上に有効なデータが存在することは確認できないため，クラスタバスに要求を転送する必要がある．表中③→ は Home からのパケットを受け取ってからクラスタバスへの転送，→④はクラスタバスから応答を受け取ってから Home に送信するまでの処理を表す．

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 に割り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．Owner が応える場合は，これに Core が要求をクラスタバスに送信してから，その応答が Core に割り込みをかけるまでの時間が加わる．

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．

各クラスタでの Short-lived パケットの処理に 100〜150 サイクルかかる結果となっている．

Home，Owner が応える場合のそれぞれおける Core の処理時間は310，609 サイクルで，ソフトウェア・オー

	Home read	Owner read
→①	109 (92/ 17)	109 (92/ 17)
①→②	— (—/ —)	110 (110/ 0)
②→	— (—/ —)	105 (88/ 17)
→③	— (—/ —)	128 (94/ 34)
③→④	142 (125/ 17)	149 (132/ 17)
④→	110 (93/ 17)	110 (93/ 17)
クラスタ内HW	45 (0/ 45)	92 (0/ 92)
クラスタ間HW	102 (0/102)	208 (0/208)
合計	508 (310/198)	1011 (609/402)

表 1 読み出し要求処理時間

バーヘッド，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 156.6%，151.5%となる．

5.2 行 列 積

本節では，JUMP-1 で倍精度 512 の行列積を動作させ，実行性能を調べた．

評価は，クラスタあたりの使用プロセッサ数を 1,2 とし，最大 8 プロセッサまでのシステムで行った．

データは各クラスタに均等に分割して配置されている．また行列積プログラムは Owner-Computes Rule で計算を行うため，起こりうるクラスタ間トランザクションは 5.1で評価した Home read のみである．また，各階層でのキャッシュヒット率を向上させるために多重のタイリング[5]を施してある．

行列積の実行時間，MBP Core1 台あたりの平均稼働時間，3 次キャッシュヒット率を表2に示す．評価環境の列は，(クラスタあたりの使用プロセッサ数)×(使用クラスタ数) を示している．Core の稼働時間は，クラスタ間処理をソフトウェアで行うことがどの程度，全体性能に影響を与えているかの指標を与える．

また，スピードアップのグラフを図2に示す．PE/CL とは，クラスタあたりの使用プロセッサ数を示している．

評価環境 (PE×CL)	実行時間 ($10^6 cycle$)	Core 稼働時間 ($10^6 cycle$)	3rd hit(%)
1 × 1	676.63	0	100(%)
2 × 1	338.73	0	100(%)
1 × 2	351.06	13.216	99.953(%)
2 × 2	181.64	13.386	99.905(%)
1 × 4	188.41	15.914	99.858(%)
2 × 4	102.37	16.527	99.716(%)
1 × 8	108.90	17.134	99.669(%)

表 2 行列積の実行時間，Core 稼働時間，3 次キャッシュヒット率

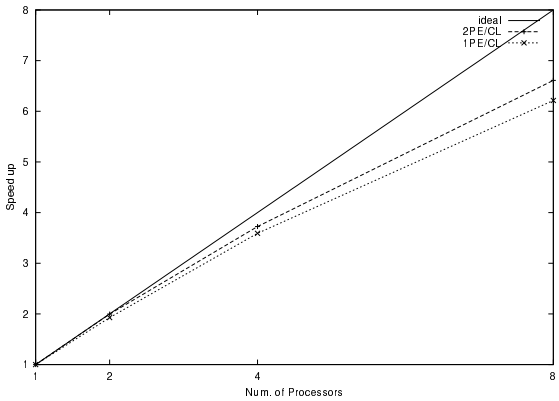

図 2 行列積のスピードアップ

クラスタ数が 2 の場合には，1 プロセッサ×2 クラスタの場合で逐次の 1.93 倍，2 プロセッサ×2 クラスタの場合で 3.72 倍の性能が得られている．全実行時間に対する Core の稼働時間の割合は，それぞれ 3.7%，7.3%である．

1 プロセッサ×4 クラスタの場合には，逐次の 3.59 倍であり，Core の稼働時間の割合は 8.4%である．

3 次キャッシュヒット率が高いため，ソフトウェアオーバヘッドの影響は少ないといえる．

しかし今回のデータサイズでは，プロセッサ数が 8 以上になると，計算時間に対する Core の稼働時間の割合が大きくなり，速度向上が伸びなやむ結果となっている．

6. おわりに

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。

MBP 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 508 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 150%程度となった．

また，512 × 512 の行列積の実行時間を計測した結果，使用プロセッサが 4 以下の場合には，ソフトウェア・オーバーヘッドの影響が少なく良い速度向上を得ることができた．

今後，より問題サイズが大きく実際的なアプリケーションで評価をとる予定である．

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) 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).

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

5) 津田健, 山本孝伸, 田中利彦, 五島正裕, 森眞一郎, 富田眞治: メモリ・アクセスの局所性を最適化するループ再構成手法, 情処研報 99–ARC–132, 99–OS–80, 99–HPC–75, pp. 133–138 (1999).