

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,clustered 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 describe this program and evaluate the performance by Matrix Multiplication. The result 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、データボード、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、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。

[†] 京大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。

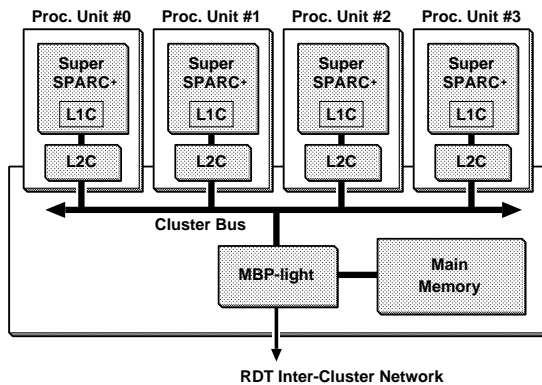


図1 JUMP-1のクラスタ

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-light²⁾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RDT³⁾(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用している。

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次キャッ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 $8b \times 8B$ 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 \times 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異な命令を持つ。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チャンネルの多重化、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ン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 では必要なチャン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せる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で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、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、データレイヤ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、コンテキ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

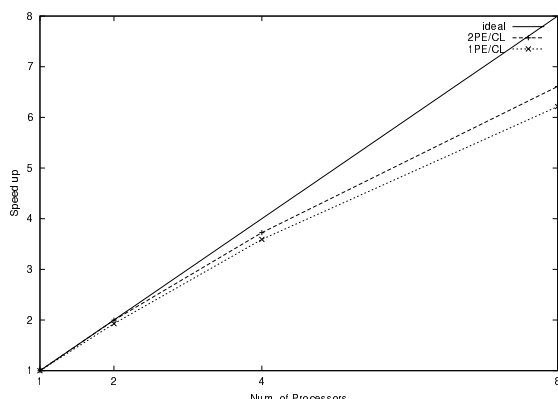


図2 行列積のスピードアップ

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ー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用。

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, *JSP'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, 森真一郎, 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).