超並列計算機 JUMP-1 における分散共有メモリシステム

Distributed Shared Memory System of the Massively Parallel Computer JUMP-1

秤谷 雅史

平成 12 年 2 月 9 日

内容梗概

ある程度の規模をもつシステムを構成するとき、システムの多階層化は必然である。また、システム構成を反映してメモリ・システムも多階層化される傾向がある。JUMP-1 はシステムの物理的構成だけでなく、メモリ・システムも多階層化された並列計算機である。

JUMP-1 では主記憶レベルの DRAM をリモートメモリに対する 3 次キャッシュとして利用する。このとき 3 次キャッシュのコントローラの実装方式が大きな問題となる。高速性を重視して全て布線論理で構成してしまうのは処理の複雑さから考えても非現実的である。そこで我々は専用のプロセッサ MBP(Memory Based Processor)-light を開発し、3 次キャッシュヒットを布線論理で高速に処理し、ミスヒットをソフトウェア・ベースで処理を行うというハイブリッドなアプローチをとった。

MBP-light はヒット時の処理を行う MMC(Main Memory Controller) とミスヒット時の処理を行う MBP-Core に分かれる。本稿ではこの MBP-Core 上で実行される DSM(Distributed Shared Memory) 管理プログラムを実装した。

DSM 管理プログラムは競合の検出や優先度の制御をするためにパケットの処理をプロセスとしてとらえ、パケットの処理を行うプロセスとプロセスの管理を行うカーネルの 2 階層から構成される。プロセスは MBP-Core の占有を防止するために割り込みの可否によって 2 つのレベルに分割して優先度の制御を行っている。プロセス間の競合は PTT(Pending Transaction Table) と呼ばれる特別なテーブルを用意して、その順序づけを行なう。また、デッドロックを防止するために 6 本の無限長のキューやバッファを用意した。

キャッシュ上のデータの再利用性を高めメモリ・アクセスの最適化を行った場合、3 次キャッシュがミスヒットする確率は極めて低い。行列積の場合では 3 次キャッシュのヒット率は 99.9%以上の値となる。シュミレータを用いて 3 次キャッシュ・ミスヒット次のレイテンシを計測したところ 1878 サイクルであった。さらにシステム全体での平均レイテンシを計算したところ 1.1282 サイクルと良い結果を残すことができた。

Abstract

It is often necessary to simplify the design of large scale system to implement a hierarchical structure. According to this structure, the physical memory of such systems tends also to be hierarchically designed. The Massively Parallel Processor JUMP-1 has a hierarchical memory system composed of three layers. JUMP-1 uses a part of the main memory as the level 3 cache for remote memory.

With such structure, the main problem is how to implement the level 3 cache management controller. A hardware implementation may allow to decrease the cost of cache miss, but makes the design very difficult. To simplify this design problem, we implemented a remote memory cache controller, called Memory Based Processor (MBP-light), with an hybrid approach. If level 3 cache access is successful, the wired-logic device replies directly accesses. Miss-hit accesses are replied using a program running on a dedicated processor.

In this report, we describe the implementation of the level 3 cache management program and its evaluation. The program is similar to a classical OS kernel, remote memory accesses are managed as processes, allowing flexibly in request management. When memory accesses to the same address are requested at the same time, the kernel uses a special table called Pending Transaction Table (PTT) to serialize the accesses. To prevent long processes from monopolizing the ded micated core processor, each request process is assigned a priority. Long processes priority are decreased, and new requests processes are allowed to interrupt (preempt) low priority processes. To avoid deadlock and starvation, six infinite length ready queues and an infinite number of buffers are used.

We have measured using a simulator that the latency of main memory read operations is 1878 cycles and that the average memory access latency with an application optimized to increase cache hit ratio is 1.1282 processor cycles.

目 次

1 はじめに 2

2 JUMP-1 の概要 3
- 2.1 JUMP-1 のシステム構成 3
- 2.2 メモリ・システム 4
- 2.3 3 次キャッシュの制御 4
- 2.4 MBP-light 5
  - 2.4.1 MBP-light の構成 5
  - 2.4.2 MBP-Core 5
  - 2.4.3 C コンパイラ 7

3 DSM 管理 7
- 3.1 MBP-light の役割 7
- 3.2 パケットの転送 7
- 3.3 ページの管理 8
- 3.4 トランザクション間の競合の検出及び優先順位の制御 8
  - 3.4.1 競合の検出 8
  - 3.4.2 優先順位の制御 9
- 3.5 デッドロック及びスターベーションの回避 9
  - 3.5.1 デッドロックの回避 9
  - 3.5.2 スターベーションの回避 . . 10
- 3.6 DSM 管理プログラムが満たすべき要件 10

4 DSM 管理プログラム 10
- 4.1 DSM 管理プログラムの概略 10
- 4.2 カーネルの概略 11
  - 4.2.1 カーネルが扱うデータ構造 . 11
  - 4.2.2 通常の OS のカーネルとの違い 11
  - 4.2.3 カーネルの動作の特徴 . . . 12
- 4.3 プロセスの状態遷移 13
- 4.4 データ構造 14
- 4.5 詳しい動作 16
- 4.6 パケット・バッファの管理 17
  - 4.6.1 概略 17
  - 4.6.2 実装 18

5 性能評価及び考察 18
- 5.1 評価環境 18
- 5.2 読み出し要求に要するサイクル数 . 18
- 5.3 考察 20
  - 5.3.1 ソフトウェア・オーバーヘッド 20
  - 5.3.2 メモリ・システムの平均レイテンシ 21
  - 5.3.3 布線論理で構成した場合 . . 21

6 おわりに 22

1 はじめに

計算機ハードウェアは普通、物理的な階層構造を持つ。例えば現在の計算機は、筐体、マザー・ボード、ドータ・ボード、プロセッサ・カートリッジ、

MCM(Multi-Chip Module)、そして、LSIチップといった要素によって階層的に実装されている。

このような階層的な実装は、やはり階層的なアーキテクチャの採用を促進する原動力となる。階層的なアーキテクチャとは、構成要素が多階層化、かつ/または、クラスタリングされたアーキテクチャを指す。

この傾向は、1チップに多くの構成要素が集積されることによって緩和されるという観測も以前にはあった。特に、主記憶まで1チップに集積することによって、主記憶に対するアクセスに時間を要してしまうというメモリ・ウォール(Memory Wall)が緩和/解消されるという予測も聞かれた。

しかし、実際にはそうなってはいない。LSIが微細化されるにつれて、ゲート遅延に対して配線遅延が相対的に増大し、ローカルな部分とグローバルな部分の速度差はますます広がっていく。メモリ・ウォールは、以前には、主に、プロセッサ・チップとメモリ・チップの速度差の問題として捉えられてきた。LSIの集積度が向上すると、ゲート遅延やローカル配線遅延と、ワード・ライン、ビット・ラインといったグローバル配線の遅延の差に置き換わる。したがってこの傾向は、LSIの微細化が進むにつれてますます顕著になる。

例えば、最近では1MB近くのSRAMがプロセッサ・チップに集積できるようになってきたが、それは1次キャッシュの増加にではなく、2次キャッシュの集積に充てられている。当然のことであるが、1MBものSRAMをプロセッサ・コアと同等のスピードで動かすことは困難だからである。

たとえシステム全体が1チップに集積されるようになったとしても、チップの内部における多階層化とクラスタリングは避けることができないであろう。結局のところ、小さなものは速く大きなものは遅いということは、実装技術の進歩によらない普遍的な原理なのだと考えられる。

このような傾向に逆らって、単階層で強力なメモリ・システムやフラットなアーキテクチャを採用することは、不可能ではないにしても、高コストとなる。コストの増大は、近い将来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階層性を受け入れ、そのために生じる不具合をソフトウェアで解消するというアプローチが将来的に重要になると考える。その場合、そのようなソフトウェア技術として、プログラムが内包するデータ参照の局所性—特に時間局所性を抽出し、データの再利用性を高める最適化技術が必須となる。

このような背景から、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、階層的なアーキテクチャ上のソフトウェアの研究に対するテストベッドの提供を、大きな目標としている。

JUMP-1は、クラスタ構成をとり、階層的なアーキテクチャを持つ。また、クラスタ上のメイン・メモリの一部をリモート・メモリに対するキャッシュとして利用し、3階層キャッシュを持つ分散共有メモリ(**DSM**: Distributed Shared Memory)を持つ。クラスタ間でのメモリ・アクセスは、専用のプロセッサ**MBP-light** [9](Memory Based Processor)がプログラム・ベースで実行する。なぜならばクラスタ間におけるDSM管理は様々な複雑な制御を行う必要があるためである。本稿では、このMBP-light上のDSM管理プログラムについて述べる。

以下、2では、まずJUMP-1とMBP-lightについて概説する。次いで, 3でクラスタ間におけるDSM管理の要件について述べた後、4でDSM管理プログラムの実装について説明する。そして最後に5で開発したプログラムの評価結果について述べる。

2　JUMP-1の概要

本章ではまずJUMP-1の物理的構成及びメモリ・システムについて説明し、次にMBP-lightの構成について述べる。

2.1　JUMP-1のシステム構成

JUMP-1は汎用の並列計算機であり、クラスタ構造を採用している。

各クラスタはRecursive Diagonal Torus [8](以下**RDT**)と呼ばれる結合網で接続されている。最大で256クラスタがRDTによって結合される。RDTは最大60MHzで動作し、パケット交換方式でトーラスを基本とした直接結合網である。ネットワークの多重化は行っていないため、アクセス

の種別によらずノード間の経路は同一である。

JUMP-1のクラスタは1枚のクラスタ・ボードと4枚のプロセッサ・カードから構成される。クラスタ・ボード上にはクラスタ・バス・チップ、主記憶であるクラスタ・メモリ及び**MBP-light**(Memory Based Processor)が搭載される。クラスタの構成の模式図を図1に示す。

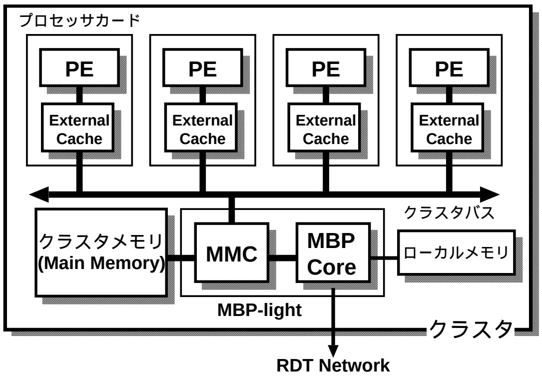

図1: JUMP-1のクラスタ構成

1枚のプロセッサ・カードは市販のプロセッサであるSuperSPARC+と外部2次キャッシュから構成される。SuperSPARC+は1次キャッシュとして20KByte命令キャッシュと16KByteデータキャッシュを内蔵する。2次キャッシュは1MByteの容量を持ち、専用に開発したCache Controller[7](以下CC)によって制御される。CCはアドレス空間を拡張して利用し、更新と無効化のプロトコルの切り替えなど様々な高機能を提供する。クラスタ・バスは64ビット幅のバスであり、4つのCCとMBP-lightが接続されている。

MBP-lightはクラスタ・メモリ及びクラスタ間にまたがる共有メモリを管理するプロセッサである。詳細は2.4節で述べる。

クラスタ・メモリは16MByteの容量を持つ。クラスタ・メモリはクラスタ内での主記憶として使用するだけでなく、クラスタ間でのリモート・メモリ・アクセスに対する3次キャッシュとしても利用する。クラスタ・メモリにはキャッシュ・ライン(32Byte)単位にSRAMで構成された有効性を示すタグが設けてあり、MBP-lightによってそのヒット判定を行う。3次キャッシュについては次の2.2節で詳細を説明する。

最後に実物のクラスタの写真を図2に示す。

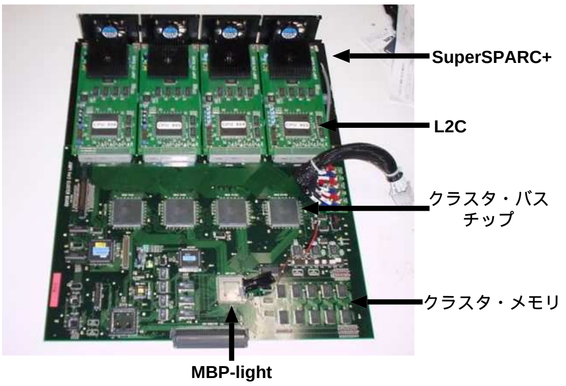

図2: JUMP-1のクラスタの写真

中央より下半分がクラスタ・ボード、上半分が4枚のプロセッサ・カードである。クラスタ・ボードは下部にあるコネクタによってRDTネットワークであるバックプレーンに接続される。SuperSPARC+は図2の上方部に位置するヒート・シンクが装着されているチップである。その下にある正方形のチップが外部2次キャッシュである。図2中央にある4つのチップはクラスタ・バス・チップであり、クラスタ・ボードの下方中央に位置するTBGA(Tape Ball Grid Array)のチップがMBP-lightである。

2.2 メモリ・システム

JUMP-1ではパケットを交換することによってメモリ・アクセスを行う。パケットとはデータ・アクセスの1つの単位であり、1つのパケットはヘッダ部分のみもしくはヘッダ部分と1キャッシュ・ライン(32Byte)に相当するデータ部分からなる。

JUMP-1では物理的な構造を反映してメモリ・システムも階層化されている。1次、2次キャッシュだけでなく、クラスタ・メモリの一部をリモート・メモリに対する3次キャッシュとして利用する。

JUMP-1はSVM(Shared Virtual Memory)[3]を採用し、仮想共有メモリ空間を提供する。すなわちリモート・クラスタのデータを自クラスタのメモリにコピーすることによって仮想的な共有空間を実現する。具体的には3次キャッシュにおけるキャッシングをアドレスによって分散配置されるHomeクラスタの物理ページをコピーすることに

よって実現する。したがって各クラスタ・メモリはHomeとしての領域と3次キャッシュ領域に分割して利用する。図3にその模式図を示す。キャッシングする際には、False sharingを防止するためにコピー領域はページ単位(4KByte)ごとに確保するが、ページ全てをコピーせずにキャッシュ・ライン単位(32Byte)で必要な部分だけコピーする。

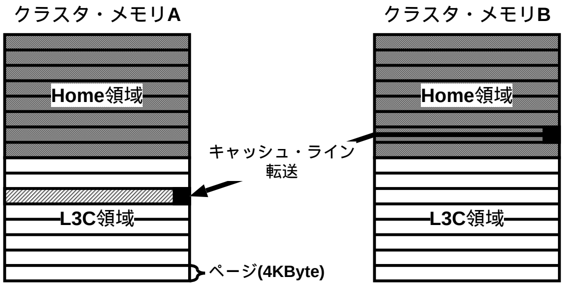

図3: 3次キャッシュにおけるキャッシング

本稿ではこの3次キャッシュの管理に主眼を置いた。

2.3 3次キャッシュの制御

前節で述べたようにJUMP-1では1次、2次、3次キャッシュ及びメイン・メモリと階層性のあるメモリ・システムをもつ。ここで考えるべきことは、各階層のキャッシュ・システムをハードウェアとソフトウェアのどちらで管理するかということである。

1次キャッシュと2次キャッシュはそのアクセスの頻度から高速性が要求されるため、すべてを布線論理で実装した。

一方3次キャッシュの管理をどちらで行うかの選択は難しい問題である。なぜならばそのコントローラには高速性もさることながら多くのクラスタで共有される可能性があるために、複雑な処理が要求されているからである。

コンパイラやプログラミング手法によって各階層でのキャッシュ・ヒット率を向上させるような最適化を施すことができれば、3次キャッシュへのアクセスはほとんど起こらない。さらに大容量な3次キャッシュ・レベルではかなり高いヒット率を期待することができる。例えばキャッシュ上のデータの再利用性を高めるタイリングという方法がある。行列積の計算では階層的なタイリングを用いて最適化を施すと、3次キャッシュ・レベルでは99.9%以上[2]のヒット率を出すことが可能である。したがってヒットした際にはある程度の高速性が求められる。

一方ヒット率が高い場合にはミスヒットはあまり発生することはなく、ミスヒットした場合には次の3で述べるが複雑な処理が必要となるために柔軟性が求めれらる。また、高速に処理が行えるように布線論理で構成したとしても、ネットワークにまで高速性を求めるにはコストが高くついてしまう。そこでJUMP-1ではミスヒット時の処理は高速性より柔軟性に重きをおくこととした。ミスヒット時の処理をソフトウェアによって行うメリットは以下のようなものが挙げられる。

1. 複雑な処理でも柔軟に行える
2. 全てをハードウェアでやるのは設計コストが非常に高い
3. ハードウェアとソフトウェアの同時開発が行える
4. 様々な実装を試すことができる

以上からJUMP-1では

- 3次キャッシュにヒットしたときはハードウェアによって応答
- ミスヒットしたときはソフトウェアによって柔軟に処理

というハイブリッドなアプローチをとった。そして我々は3次キャッシュ・コントローラとして**MBP-light**を開発した。本稿ではミス・ヒット時にMBP-light上で動作するDSM管理プログラムの実装について述べる。MBP-lightの詳しい構造については次の2.4節で述べる。

2.4 MBP-light

2.4.1 MBP-lightの構成

MBP-lightは全て布線論理で実装された**MMC**(Main Memory Controller)とコア・プロセッサである**MBP-Core**の2つのユニット

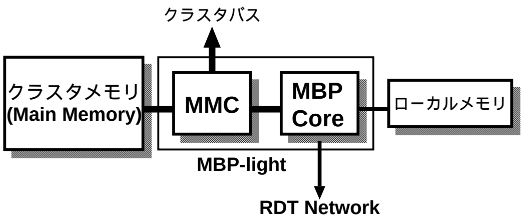

図4: MBP-lightの構成

から構成されている[9]。図4にMBP-lightの構成とその外部との接続状況を示す。

- MMC :クラスタ・メモリのコントローラ。3次キャッシュにヒットした場合は布線論理で高速に応答
- MBP-Core :専用のプロセッサ。3次キャッシュにミスヒットした場合にはその上で実行されるプログラムによって処理

MBP-Coreについては次の2.4.2項で詳しく述べる。

MMCはクラスタ・メモリ・アクセスを制御するユニットである。

2.1で述べたようにクラスタ・メモリにはキャッシュ・ライン単位にSRAMで構成された有効性を示すタグが設けてあり、クラスタ・バスからのアクセスがあるとMMCはそのヒット判定を行う。アクセスがヒットしていればMMCは自動的に応答を返し、ミスヒットであるとMBP-Coreに処理を依頼する。本稿ではミスヒット時のプログラムについて述べるので、これ以降はMMCについての説明は行わない。

2.4.2 MBP-Core

MBP-CoreはMBP-lightのコア・プロセッサであり、命令長21bit、データ長16bitのRISC型プロセッサである。外部には21bit×64Kエントリのアドレス空間を持つローカル・メモリ(図4)を備えている。

MBP-Coreは汎用のプロセッサと異なり、DSM管理を効率よく行うために次のような特徴を持っている。

1. パケットの到着による割込み機構を備える
2. クラスタ・メモリへの直接アクセスが可能
3. パケット格納用レジスタを用意
   (**PBR**: Packet Buffer Register)
4. 特殊命令をサポート

これらの特徴をDSM管理プログラムがうまく利用することによって、柔軟にかつある程度の高速性をもって処理を行うことができる。

以下それぞれの特徴について説明する。プログラムによる具体的な利用については4で述べる。

パケットの到着による割込み機構　MBP-Coreは3次キャッシュ・ミスによるMMCから依頼及びRDT側からパケットが到着すると割込みが発生する機構を備える。詳しくは後述するが、DSM管理プログラムは様々な長さの処理を行う必要があるために、処理の優先度をこの割込み機構を利用することによって区別している。

クラスタ・メモリへの直接アクセス　MBP-Coreからは、MMCが参照するキャッシュ・ライン単位に設けられたタグへのアクセスが可能である。プログラムはクラスタ間でのキャッシュの一貫性を保つために、クラスタ・メモリの無効化などクラスタ内でのデータの有効性を更新する必要がある。

また、クラスタ・バスを介さずにMMCを経由して直接クラスタ・メモリにアクセスすることができる。したがってクラスタ・メモリ上のデータの有効性が保証されていれば、プログラムによってクラスタ・バスを利用せずに応答を作成することが可能である。

PBR:Packet Buffer Register　JUMP-1ではパケットを用いてメモリ・アクセスを行うために、MBP-CoreはPBRと呼ばれるパケット格納用のバッファを持つ。図5にPBRの模式図を示す。

PBRは汎用が64本とRDTの送受信用に8本を1組としてそれぞれ3組ずつ用意されている。RDTへの送信は送信用PBRにパケットをコピーし、専用の送信命令を発行することによって行われる。パケットの受信は、パケットの到着による割込みがかかり、受信用PBRの先頭の1組を参照することによって行う。図5に示すようにRDT送受

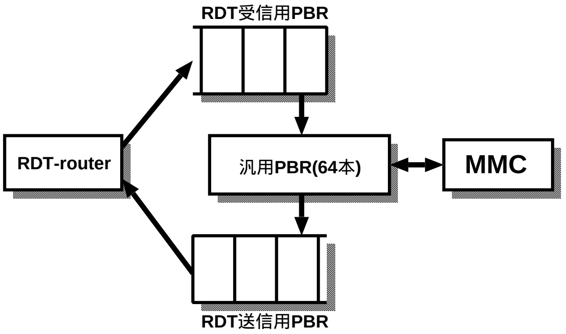

図 5: PBR

信バッファはFIFO構造となっており、MBP-CoreからはRDT送信用PBRの末尾の1組、受信用PBRの先頭の1組しか見えない。したがってこれらのPBR上にパケットを保存することはできず、プログラムは送信及び受信時にはパケットのコピーを行う必要がある。

RDT側の送受信とは異なり、クラスタ・バス側の送受信は汎用のPBRによって行われるためにコピーを行う必要はない。

PBRは名前ではレジスタとされているが、メモリと同様にレジスタ間接によってアクセスする。ただしアクセスには専用命令を使用する。メモリと異なる点としてはPBR間のコピーがGPR(General Purpose Register)を介さずに直接可能となっていることが挙げられる。

特殊命令　主にPBR操作命令とGPRに対するビット操作命令の2種類に分類される。

PBR操作命令は前述したPBRに対するロード・ストアだけでなく、プログラムによってパケットのヘッダ部の作成が容易にできるよう即値のセットや論理演算なども用意されている。

また、詳しくは後述するが、プログラムは同一アドレスに対するアクセスを検出する必要がある。このときJUMP-1で用いられる物理アドレス空間は28bit幅であるために、PBR上のアドレス部分を複数回に分けてGPRにロードしなければならない。したがってその比較を行うには多くの時間が必要となる。しかしMBP-lightではアドレスが格納されたPBRのINDEXを指定すると、アドレスのハッシュ値を求める命令が用意される。この命令を利用することによって、まずはハッシュ値を比較することによって同一であるかどうかの検出を行うことができる。

GPRに対するビット操作命令は主にビット・マップ方式のディレクトリ操作を高速に行うために利用する。プログラムではPBRの管理などでビット・マップ方式のディレクトリを扱うことがある。このディレクトリに対して、使用/未使用を表すためにビットのセット/リセットをするだけでなく、現在未使用のものを探し出すという操作を行わなければならない。この操作を通常のRISC型プロセッサに用意された命令セットのみで行うと「シフト演算・論理積・比較」というサイクルを繰り返すことによって行う必要がある。しかし、MBP-lightでは最下位ビットから0となっているビットを探し出し、その番号を返す命令が用意されており、わずか1命令で行うことができる。

2.4.3　Cコンパイラ

これまでプログラムの開発はアセンブリ言語を用いて開発されており、十分な最適化を行うことは困難であった。そこでMBP-light用にGNU Cコンパイラをポーティングし、プログラムはC言語によって記述している。特殊命令は関数として表現する。

コンパイラによって生産性が向上するだけでなく、コードの最適化を自動で行うことができるために複雑なプログラムでも高速な実行が期待できる。特にMBP-lightではローカル・メモリ・アクセスに1サイクルのストールを伴なうため、インライン展開によってレジスタ保存のためのローカル・メモリ・アクセスの削減を狙っている。

3　DSM管理

MBP-lightの役割はクラスタ間にまたがるDSMの管理である。本章ではMBP-lightによるDSM管理においてDSM管理プログラムが満たすべき要件について述べる。

3.1　MBP-lightの役割

メモリ・アクセスに伴なうクラスタ間のパケットの転送及び一連の状態更新をトランザクション

(Transaction) と呼ぶ。MBP-light の役割はクラスタ間でのキャッシュの一貫性制御のためにトランザクションを実行することである。

MBP-light は同時に複数のトランザクションを実行することがある。このとき、詳しくは後述するが

- トランザクション間の競合の検出
- トランザクション間の優先順位の制御

などの処理も行わなければならない。また、トランザクションを実行するだけでなく、デッドロック (Deadlock) 及びスターベーション (Starvation) を回避する必要がある。つまり MBP-light はトランザクションの実行だけでなく、様々な複雑な制御も行わなければならない。MBP-light はこれらの制御をソフトウェアで行う。

本章ではまず、複雑な制御についての説明を行うために、パケットの転送とページの管理について述べる。次にトランザクション間の競合の検出及び優先順位の制御について述べ、さらにデッドロック及びスターベーションの回避について説明する。最後に DSM 管理を行うにあたっての DSM 管理プログラムに求められている要件についてまとめる。

3.2 パケットの転送

JUMP-1 ではクラスタ間にまたがるメモリ・アクセスは必ず Home クラスタを経由する。2.2節にて述べたように Home クラスタは 3 次キャッシュ・ディレクトリを所有し、クラスタ間の共有状態を管理する。Home クラスタ以外で有効なデータを所有しているクラスタを Renter クラスタと呼ぶ。したがって Home クラスタでは Renter クラスタの管理を行う。

キャッシュ・ラインは、Modified、Exclusive、Shared、Invalid の 4 状態で管理する。管理を簡単にするために、ラインの状態が Shared であるときは必ず Home クラスタのメイン・メモリを Clean な状態とする。

有効なデータを持つ唯一のノードを Owner クラスタと呼ぶ。ラインの状態が Shared である場合には Home クラスタは必ず有効なデータを持つので Home クラスタを Owner クラスタとする。

典型的なパケット転送の例を図 6に示す。メモリ・アクセスを開始するクラスタを Initiator クラスタと呼ぶ。

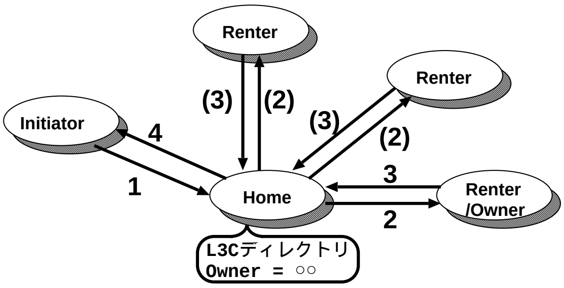

図 6: クラスタ間でのパケットの流れ

1. Initiator クラスタにおいて 3 次キャッシュがミスヒットすると Home クラスタに対して要求パケットを送信
2. Home クラスタでは要求に対してディレクトリを検索して Owner クラスタに要求パケットを転送
3. Owner クラスタは応答パケットを Home クラスタに送信
4. Home クラスタは応答パケットを Initiator クラスタに転送

ソフトウェアで制御するため、様々な転送方法を実装することができる。例えば NEC 社の Cenju-4 のように Owner クラスタから Initiator クラスタへの直接転送 [4] などの実装も考えられる。しかし、今回は簡単のために応答も必ず Home クラスタを経由してから Initiator クラスタへと転送されることとした。

3.3 ページの管理

JUMP-1 ではマルチ・ユーザ/マルチ・タスクを実現する基盤として、仮想共有空間を提供する。JUMP-1 でのアドレス変換を図 7に示す。

SuperSPARC+に内蔵される MMU(Memory Management Unit) はクラスタ・メモリ上のページテーブルを参照する。SuperSPARC+が扱う論理アドレスは MMU を利用することによって物理

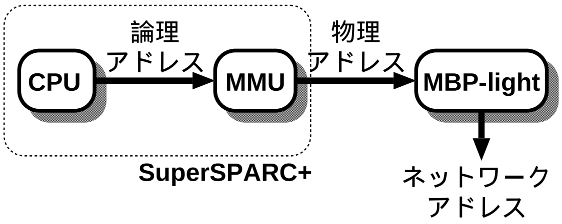

図 7: アドレス変換

アドレスに変換される。MBP-light はこの物理アドレスをさらにネットワークアドレスに変換することによって、システム全体のアドレス空間を管理する。

SuperSPARC+が自クラスタにマッピングされていないアドレスを参照した場合ページフォールトが発生する。このとき MBP-light はそのアドレスを含むページ領域を確保し、ページテーブルを整備する必要がある。

ネットワークアドレスである論理アドレスを得るために、MBP-light は物理アドレスから論理アドレスへの変換テーブルを所有する。以後このテーブルのことを逆引きページテーブルと呼ぶこととする。また、ネットワーク経由で到着したアドレスを物理アドレスに変換するために、MBP-light はページテーブルの参照もする。つまり、MBP-light は物理アドレス⇔ 論理アドレスの相互の変換をし、それぞれの変換に利用するページテーブルと逆引きページテーブルを管理する。

3.4 トランザクション間の競合の検出及び優先順位の制御

MBP-light は同時に複数のトランザクションを実行することがあるために、トランザクション間での競合が起こる場合がある。例えば Home クラスタでは同時に複数のクラスタからの同一アドレスに対する要求パケットが到着する場合がある。このとき競合が発生する。クラスタ間でのキャッシュの一貫性を保つためには競合するトランザクションの待合わせを行う必要がある。

また、競合が検出されなくともトランザクション間に優先順位を設けなければ、トランザクションとして重要度の低いものが MBP-light を独占してしまう可能性があるためこれを防止しなければならない。

本節ではそれぞれの問題を解決するための要件を挙げる。

3.4.1 競合の検出

同一アドレスへのアクセスは、必ず同一の Home クラスタを経由する。したがって Home クラスタでは、同一アドレスへのトランザクションが複数存在する場合がある。しかし、ひとつのトランザクションが終了していないにもかかわらず競合するトランザクションを実行してしまうと、状態の更新が終了していないためにクラスタ間での一貫性を保つことができなくなる。したがって競合を検出してトランザクションのシリアライズをしなければならない。

以上から MBP-light ではアドレスが同一であるトランザクションを検出し、競合がある場合はその待合わせを行わなくてはならない。

3.4.2 優先順位の制御

MBP-light は様々な長さの処理を行う。例えば 3.3節で述べたようにページフォールトが発生し、3 次キャッシュの Modified なページのリプレースが起こると、メイン ・メモリへの書き戻しを行う必要がある。この処理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。すなわち MBP-light が処理すべきものには、大きく分けると短時間で処理が終了するトランザクションと長時間必要なものとの 2 種類が存在する。

長いトランザクションが MBP-light を占有することによって後続の短いトランザクションが実行されないことがあると、システム全体の性能低下につながる恐れがあるためにこれを防止しなければならない。したがって長いトランザクションの優先度を下げ、長いトランザクションを実行中でも短いトランザクションを優先的に処理する必要がある。

以上から MBP-light は長いトランザクションを実行中に、短いトランザクションの実行優先的に実行する。また、トランザクション実行中に短い

トランザクションの要求を検出する機構を用意しなければならない。

3.5 デッドロック及びスターベーションの回避

MBP-light はトランザクションを実行するだけでなく、デッドロック及びスターベーションを回避しなくてはならない。本節ではそれぞれの回避方法について述べる。

3.5.1 デッドロックの回避

各クラスタ間のトランザクションにおいて、資源要求グラフにループが形成されるとデッドロックが生じる。3.2節で述べたクラスタ間通信を実装した場合の資源要求グラフを図 8に示す。MBP-Core 間の矢印上の数字は図 6のそれと対応する。

図 8では Request to Home から Request from Home への点線で示された矢印があるが、これは有効なデータが Home クラスタ上に存在している場合である。Reply to Home から Reply from Home の矢印も同様の理由である。

CC や MMC には Request 用と Reply 用の送受信バッファがそれぞれ用意されているので、クラスタ内部でのデッドロックは起こらない。

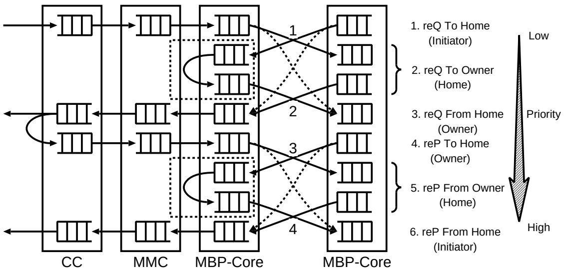

図 8: 資源要求グラフ

図 8の点線で囲まれた部分は Home クラスタの RDT 側の送受信キューである。これらは一時的に保存するバッファを省略して考えると 1 つにまとめることができる。したがって MBP-Core はデッドロックを防止するために合計 6 本のキューが必要となる。

6 本のキューには優先度を設け、優先度に従ってサービスを受けることができる。優先度の低い方から順に

1. reQ to Home
2. reQ to Owner
3. reQ from Home
4. reP to Home
5. reP from Owner
6. reP from Home

という 6 つのレベルとする (図 8右)。

3.5.2 スターベーションの回避

2.1節で述べたように JUMP-1 ではアドレス空間を拡張して利用することによって更新と無効化のプロトコルの切り替えをすることができる。無効化プロトコルではプロセッサの outstanding 要求数からシステム全体での最大数を静的に決定することができる。しかしながら更新プロトコルでは際限なく更新要求を発行することができるために、要求の最大数を静的に決定することができない。

MBP-light ではここで再試行を要請することはしない。なぜならばスターベーションが発生してしまう可能性があるからである。また、ネットワークのトラフィックが増大し、システム全体としての性能が低下する可能性も考えられる。再試行を要請しないために、MBP-light はローカル・メモリなどを利用して無限長のバッファを用意し、全てのメモリ・アクセスをバッファリングしなければならない。

次に前項で述べた各キューの深さについて考える。

MBP-light の負荷状態によっては低レベルのキューはサービスを受けることができずに、キューの深さが有限であると溢れてしまう場合が考えられる。ここでキューの溢れを理由に受信を拒否して再試行を要請すると、やはりスターベーションが発生する危険性がある。さらに 2.1節で述べたように RDT はすべてのレベルのパケットが同一のネットワークを使用するため、優先度の高いパケットを受信できなくなるとデッドロックが発生する可能性さえ出てくる。デッドロックを回避す

るためにはネットワークを多重化する方法も考えられるがコストが高くついてしまう。したがってこれらの問題を解決するためにはキューの深さに上限を設けない必要がある。つまりMBP-lightでは無限の深さを持つキューを用意しなくてはならない。

また、万が一RDTが詰ってしまった場合にはタイマーによってそれを検出し、受信バッファからパケットを取り出してリカバーする。

以上からMBP-lightはスターベーションを防止するためには無限の深さのキューを用意する必要がある。

3.6 DSM管理プログラムが満たすべき要件

本章ではMBP-lightによるDSMの管理における複雑な制御について説明した。そしてMBP-lightが満たすべき要件をいくつか挙げた。その要件についてまとめると

1. アドレスが同一であるトランザクションの検出とその待合わせを行う
2. 長いトランザクションを実行中でも、短いトランザクションを優先的に実行する
3. デッドロック及びスターベーションを防止するための優先順位をもつ6本の無限の深さのキューを用意する

の3つをMBP-lightは行う必要がある。MBP-lightはこれらの複雑な制御をハードウェアではなく、ソフトウェアによって行うというアプローチをとった。その具体的な実装については次章で述べる。

4 DSM管理プログラム

本章では作成したDSM管理プログラムについて述べる。

4.1 DSM管理プログラムの概略

DSM管理プログラムは前章で述べた複雑な制御を行うために、各クラスタでのトランザクションの実行を1つのプロセスとしてとらえ、プロセスとプロセスの管理を行うカーネルの2階層から構成される。トランザクション間の競合や優先度の制御は全てプロセス間の問題に置き換えて考える。つまりプロセスのスケジューリングの問題として扱った。したがって基本的には、パケットの到着によってプロセスを生成し、そのプロセスを実行するOS(Operating System)である。

プロセスはパケットの転送や3次キャッシュ·ディレクトリの更新などを行う。3次キャッシュ·ディレクトリはページ単位とキャッシュ·ライン単位とに分けて管理する。

- クラスタ間での共有状態 :ページ単位
- Ownerクラスタ :キャッシュ·ライン単位

更新要求や無効化要求は各Renterクラスタにも放送しなければならない。このときページ単位のディレクトリを利用する。

また読み出し要求の場合にはそのブロックのOwnerを検索することによって転送先を決定する。例えばHomeクラスタに読み出し要求が到着した場合にはディレクトリからOwnerクラスタを検索しパケットを転送する。

カーネルについては次節以降で詳しく説明する。

4.2 カーネルの概略

カーネルの基本的な動作は、パケットの到着によって割込みが発生するとプロセスを生成した後、プロセスのスケジューリングをし実行することである。そして実行が終了するとプロセスを消去し、次のパケットの到着を待つ。

本節ではまずカーネルが扱うデータ構造について簡単に説明し、次にカーネルの概略として

1. 通常のOSとの違い
2. カーネルの動作の特徴

について述べる。1では前章の要件を満たすために、通常のOSとの異なる点について明らかにする。2ではカーネルの基本的な実装方針について述べる。

4.2.1 カーネルが扱うデータ構造

カーネルが扱うデータ構造は以下の5つがある。

1. プロセス・テーブル
2. レディキュー
3. TLB (Translation Lookaside Buffer)
4. PTT (Pending Transaction Table)
5. パケット・バッファ・ディレクトリ

1と2は通常のOSと同様のものである。3はSuperSPARC+などに備えられたTLBをソフトウェアによって実装したものである。4と5はDSM管理プログラムに特有なものである。パケット・バッファとはパケットの保存場所のことを指し、PBRなどから構成されている。詳しい説明は4.6節で述べることとし、ここでは特別なメモリと理解して欲しい。

以下、5つのデータ構造について簡単に述べる。

プロセス・テーブル プロセスの情報を保持しているテーブル。主にパケットの種別やパケット・バッファへのポインタなどを登録する。

レディキュー プロセス・テーブルのリストで無限の深さをもつ。短いプロセス用に6本と長いプロセス用に1本の優先順位を持つ合計7本のキュー。 3.6節で挙げた3つめの要件を満たす。

TLB (Translation Lookaside Buffer) 3.3節で述べたようにMBP-lightは物理アドレス⇔論理アドレスの相互の変換をする。ページテーブル及び逆引きページテーブルはクラスタ・メモリ上にあり、複数回のアクセスを必要とするためにその変換に時間がかかってしまう。そこでページテーブルと逆引きページテーブルをそれぞれキャッシングする。

PTT (Pending Transaction Table) 通常のOSではイベントキューに相当する。アドレスをキーとした連想テーブルで、MBP-light用に最適化している。プロセス間の競合の検出やプロセスの検索に利用する。

パケット・バッファ・ディレクトリ 3.6節において3つめの要件に挙げたように、パケットの保存をするための無限長バッファを管理するディレクトリである。

各データ構造の詳細は4.4節で述べる。

4.2.2 通常のOSのカーネルとの違い

通常のOSのカーネルとの違いは主に2つ挙げられる。

1. プロセス間の競合の検出
2. 長短2種のプロセスの扱い

3.6節において1つめの要件に挙げたように、MBP-lightはプロセス間の競合の検出や優先順位の制御を行わなければならない。競合する場合にはプロセスの待合わせを行う。前述したようにカーネルは競合の検出及び待合わせをするために、PTT(Pending Transaction Table)と呼ばれる特別なテーブルを用意した。具体的には4.4節で述べることとする。

また3.6節において2つめの要件に挙げたように、MBP-lightは長短2種のプロセスを扱わなくてはならず、コア・プロセッサの占有を防止するために長いプロセスの優先度を下げる必要がある。つまり長いプロセスの実行中でも短いプロセスを優先的に処理しなければならない。したがって長いプロセスの実行中に新たなパケットの到着を検出する必要がある。

一般的にパケットの到着を検出する機構としては

1. 受信バッファのポーリング
2. パケットの到着による割込み

の2つが考えられる。MBP-lightではどちらの方法も実装が可能である。

ポーリングによる実装では長いプロセスの処理を行っていると、新たに到着しているパケットの検出ができない。ポーリングをある一定間隔で行う方法も考えられるが、そのオーバーヘッドがどうしても生じてしまう。

一方、割込みによる実装のオーバーヘッドはコンテキストの保存である。しかし短いプロセスの

処理は割込みを不許可にして一気に実行し、割込みを許可するのはidle時と長いプロセスを実行している場合のみとすることができる。基本的に割込みが発生するのはidle時であって、その場合にコンテキストの保存をする必要はない。したがって通常の割込みでは割り込みハンドラに相当するものの処理は極めて軽い。また、多重割込みは行わない。

以上より今回のプログラムではパケットの到着による割込みという実装を行い、

- 短い処理は割り込み不許可で一気に処理
- 長い処理は割り込みを許可

とすることによって長短2種のプロセスの優先度の制御を行った。

4.2.3 カーネルの動作の特徴

通常、カーネルは多くのプロセスを扱うことはないと考える。また、負荷が高い場合でも複雑な制御を行わず、スループットを重視する方針をとった。したがってプロセス実行の優先度はレディキューに関してのみ行う。

スループットを向上させるために、カーネルの動作の特徴としては次の3つが挙げられる。

1. サスペンドを行わない
2. スケジューリング中の割込みを許可
3. 要求の処理が終わるとプロセスはsleep

この3つの特徴について述べる。

サスペンドを行わない 通常のOSが扱うプロセスとは異なりプロセスの実行内容はある程度既知である。例えばInitiatorクラスタではHomeクラスタへパケットを転送するためにRDTの送信バッファを必ず利用する。したがってあらかじめ送信バッファを確保しておくことによって、途中でサスペンドすることなくプロセスを一気に実行することができる。そこでスケジューラはプロセス実行前に必要な資源を確認し、確保できた場合のみ実行にうつす。ただし短いプロセスの場合に限る。

スケジューリング中の割込みを許可 プログラムは新たなパケットの到着による割込みを受け付けなけらばならない。ここで問題となるのはいつ割込みを許可するかということである.

前述のようにスケジューラはレディキューにプロセスが存在すると、プロセスの実行が可能でないか資源を確認しながらループしている。このとき割込みを受け付ける方法としては、通常のOSのように、割込みを許可したidleプロセスを実行する方法が考えられる。ところがidleプロセス実行中にレディキューにつながれているプロセスが必要とする資源の確保が可能となる場合もある。しかしidleプロセスの終了を待っていてはレスポンス・タイムが低下してしまう。

以上よりスループットを向上させ、なおかつレスポンス・タイムを低下させないために次の2つの要件を両立させたい。

1. 新たなパケットの到着をできる限り受け付ける。
2. 資源の確保ができれば直ちにプロセスの実行を開始する。

そこでスケジューリング中の割込みを許可した。しかし、常に許可するとスターベーションを起こしてしまい、実行可能プロセスがいつまでも実行されない可能性がある。したがって新たにプロセスがレディキューにキューイングされた直後のスケジューリングは割込みを不許可とする。

資源が確保でき実行を開始する際には割込みを不許可にする。

要求の処理が終わるとプロセスはsleep パケットの転送などを行い要求の処理が終了すると、プロセスはその応答を待たなければならない。このときプロセスをDeleteせずにSleep状態に遷移しておくことで、プロセスCreate/Deleteのオーバーヘッドを削減する。

詳しい遷移については次の4.3節で述べる。

4.3 プロセスの状態遷移

基本的にプロセスは要求が到着した際に生成され、その応答の処理が終了すると消滅する。本節ではプロセスの状態遷移について述べ、プロセス

が競合している場合や応答を待っている際にどのような遷移を行うかについて説明をする。

プロセスの状態には通常のOSと同様に**Ready**、**Running**、**Waiting**の3状態がある。

各状態への遷移を図9に示す。Arrivedは要求がMBP-lightには到着しているが、まだ割込みが発生しておらずプログラムによってサービスが開始されていない状態である。

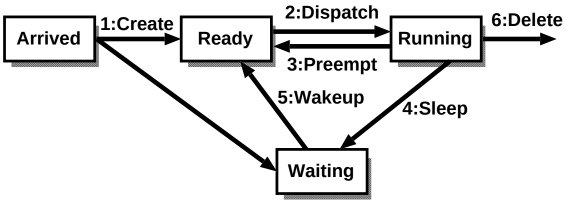

図9: プロセスの状態遷移

通常のOSと異なる点としては次の2つがある。

- PTTでの競合
- 長いプロセスは横取り(Preempt)される

プロセスをCreateするとPTTによって競合を検出し、検出されなければReady状態となる、競合が検出されればWaiting状態となり、解消されると即座にWakeupする。

4.2.2項にて述べたように長いプロセスは割込みを許可するために実行中に割込みが発生する。すなわちPreemptは長いプロセスのみに起こる遷移であり、短いプロセスでは起こらない。

各遷移について説明する。

1. **Create**

   要求パケットがMBP-lightに到着し、割込みによってそのサービスを開始するとプロセス・テーブルを用意する。プロセス・テーブルには各種の情報を登録する。PTTによって競合をチェックし、競合が検出されなければレディキューにキューイングされReady状態に、検出されれば競合が解消されるまでWaiting状態へと遷移する。

2. **Dispatch**

   スケジューラにより選択されるとプロセスの実行を開始する。Preemptされた長いプロセスを実行する場合ではコンテキストの復帰を行う。

3. **Preempt**

   優先度の低い長いプロセスは、優先度の高い短いプロセスに横取りされる。

4. **Sleep**

   要求を送信したプロセスはその応答が返ってくるまでWaiting状態となる。

5. **Wakeup**

   応答が到着した、または競合が解消されたプロセスはWakeupし、レディキューにエンキューされる。

6. **Delete**

   応答の処理が終了したプロセスは消滅する。プロセス・テーブルはフリーリストに返却され、1つのトランザクションが終了する。

前節で特徴として挙げたように応答待ちとなったプロセスはDeleteせずにSleepする。したがって通常の状態遷移としてはArrived → Ready → Running → Waiting → Ready → Running → endとなる。

応答待ちのプロセスと競合が検出されたプロセスはどちらもWaiting状態となる。基本的に両者は応答待ち状態である。なぜならば後者も競合しているプロセスが終了するのを待っているからである。したがって同じ状態として扱うこととした。

4.4 データ構造

カーネルは次の5つのデータ構造を扱う。

1. プロセス・テーブル
2. レディキュー
3. TLB
4. PTT
5. パケット・バッファ・ディレクトリ

本節では1～4のデータ構造について詳しく述べる。特に4のPTTについては、プロセスの待合わせ方法について詳しく述べる。5のパケット・バッファ・ディレクトリについては4.6節で述べる。

プロセス・テーブル　前述のようにプロセスは応答待ちとなるとSleep状態へと遷移する。このときコンテキストとしてアドレス及びSource Device IDなどを記憶しておく必要がある。そこで通常のOSと同様にプロセスの様々な情報を登録するテーブルを用意する。

メンバーとしては

- プロセス・レベル：対応するレディキューのレベル
- アドレス：PTTの検索時に利用
- パケットの種別
- Source Device ID：応答の宛先
- パケット・バッファへのポインタ
- スタック・ポインタ

などがある。プロセスが生成されるとフリーリストから用意する。

レディキュー　優先順位を持つ7つのキューから構成される。3.6節で挙げた3つめの要件を満たすために、プロセス・テーブルのリニア・リストで構成された6本と長いプロセス用の1本からなる。その構造を図10に示す。

図10の左に示してあるように優先順位は低い方から順に

1. Page Operation
2. reQ To Home
3. reQ To Owner
4. reQ From Home
5. reP To Home
6. reP From Owner
7. reP From Home

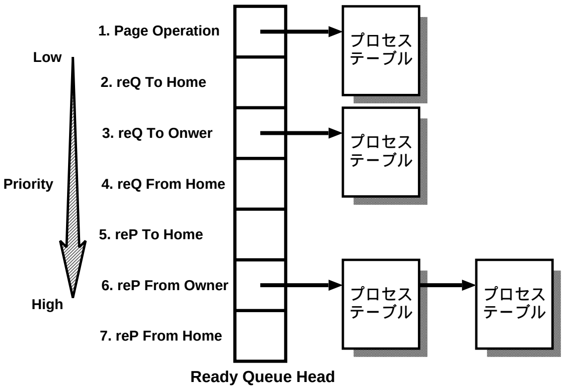

図10: レディキュー

とする。1は長いプロセス用、それ以外は図8で示した各レベルに対応し、短いプロセス用のものである。

キューには競合が検出されなかったプロセスや、応答パケットが到着してWakeupしたプロセスがつながれる。

TLB (Translation Lookaside Buffer)　3.3節で述べたようにMBP-lightは物理アドレス⇔論理アドレスの相互の変換を行う必要がある。具体的にはクラスタ・バス側からパケットが到着すると物理アドレスから論理アドレスへの変換が行われ、パケットをクラスタ・バスへ送信する場合及びデータの読み出しなどクラスタ・メモリへ直接アクセスする場合には、論理アドレスから物理アドレスへの変換を行う。

ページテーブルはクラスタ・メモリに配置される。プログラムから直接ページテーブルを読み出して変換することが可能であるとはいえ、各プロセスを実行する度にクラスタ・メモリへのアクセスが必要となれば著しい性能低下の原因となる。そこでMBP-lightではSuperSPARC+などのプロセッサと同様に、ページテーブルをキャッシュングするTLBをソフトウェアによって用意した。

PTT (Pending Transaction Table)　3.6節において1つめの要件に挙げたように、クラスタ間でのキャッシュの一貫性を保つためには、アドレスが同一であるプロセスは同時に1つしか実行してはいけない。従ってHomeクラスタでは競合

するプロセスを検出し、その順序付けを行う。

また、各クラスタでは応答パケットの到着時にプロセスのコンテキストが必要であるため、sleep 状態のプロセスの中からの対応するプロセス·テーブルの検索を行う。

プログラムはこれらの機能を有する Pending Transaction Table(以下 PTT) と呼ばれるテーブルを用意した。PTT はその名前が示すように処理中のトランザクションの管理を行うテーブルである。PTT の構造を図 11に示す。アドレスをキーとするチェーンド·ハッシュ·テーブルで、プロセス·テーブルがリストでつながれている。

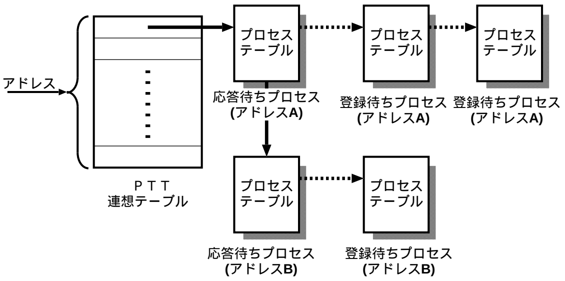

図 11: Pending Transaction Table

PTT へのアクセスは、まずアドレスをハッシュ関数にかける。ハッシュ値を求めるには MBP-light に用意された特殊命令を利用した。次にハッシュ値をキーとしてテーブルを引き、すでに登録がある場合にはプロセス·テーブルに登録されたアドレス同士を比較して衝突の判定を行う。図 11中のアドレス A とアドレス B はハッシュ値が同一であるが、アドレスが異なっている。

PTT の機能は前述したように

1. プロセス間の競合の検出
2. Sleep 状態のプロセスの検索

である。したがって PTT の操作は登録·削除·検索の 3 つがある。3 つの操作について説明する。

1. 登録

   要求パケットが到着しプロセスを生成すると、登録中のプロセスが存在するか PTT を検索する。登録中のプロセスが存在しなければプロセス·テーブルを登録し (図 11の実線矢印)、Ready 状態へと遷移する。登録中のプロセスが存在する場合には、そのプロセス·テーブルにフックし登録待ちとなる (図 11点線矢印)。プロセスは Waiting 状態へと遷移する。すでに登録待ちプロセスが存在する場合には、さらにそのプロセス·テーブルにフックする。この操作を再帰的に行うことで複数の競合するプロセスが存在する場合にでも正しく順序付けを行うことができる。

2. 削除

   登録中のプロセスの処理が終了すると PTT からプロセス·テーブルを削除する。このとき登録待ちプロセスがフックされていると PTT に登録され、そのプロセスは Ready 状態へと遷移する。

3. 検索

   応答パケットが到着すると PTT を検索して、対応するプロセス·テーブルを探し出す。プロセスは Wakeup し、Ready 状態へと遷移する。

4.5 詳しい動作

基本的には通常の OS と同様にスケジューラによってプロセスが選択され実行される。本章ではパケットの到着による割込みからその実行までのカーネル全体の詳しい動作について説明する。なお、本節では説明を簡単にするために短いプロセスの場合のみを扱う。長いプロセスの場合はプロセスが Dispatch と Preempt されるときにコンテキト·スイッチが行われる。

カーネルの動作のフローチャートを図 12に示す。基本的な流れとしては次のようになる。

1. スケジューラの割込みが許可され、パケットの到着による割込みが発生
2. プロセスに対する操作 (Create/Wakeup 等) が行われ、再びスケジューラへと制御が移る
3. スケジューラによって選択実行され、実行後に再びプロセスに対する操作 (Sleep/Delete 等) が行われる

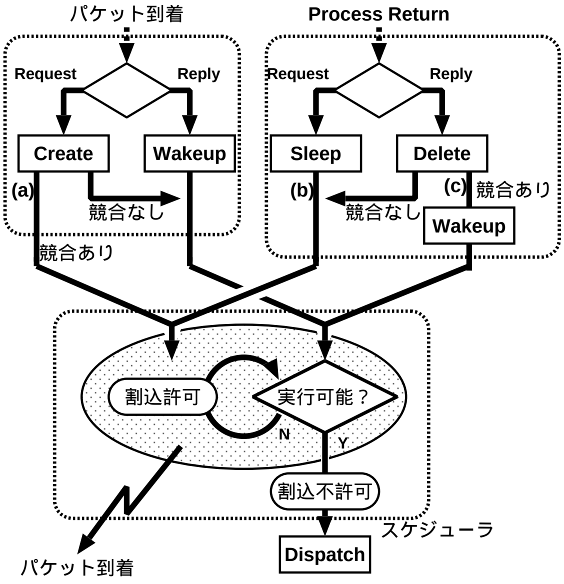

図 12: カーネルの動き

したがって主に次の 3 つの動作にわけることができる。

- スケジューリング (図 12下部)
- パケットの到着後の処理 (図 12左上部)
- プロセス実行後の処理 (図 12右上部)

3 つの動作について詳しく述べる。

スケジューリング　4.2節で述べたように DSM 管理プログラムのスケジューラの特徴は、資源の確保を行ってからプロセスの実行を開始することと、スケジューリング中の割込みを許可することである。新たにプロセスが Ready 状態になった場合のみ割込みを許可せずにスケジューリングが始まる。

スケジューラは優先順位に従ってキューを検索する。そしてキューにプロセスがつながれている場合はそのレベルに必要な送信バッファをチェックする。

送信バッファの確保ができた場合には割込みを不許可にしてプロセスの実行を開始する。できなかった場合には次のレベルのキューをチェックする。

キューにプロセスが存在しない、または資源の確保ができず実行できるプロセスが存在しない場合には割込みを許可し、スケジューラはレディキューを検索し続ける。

パケットの到着　パケットが到着すると、クラスタ・バス側からの割込みであればアドレス変換を行い、その後にパケットが要求か応答かの判別を行う。RDT 側からのパケットであれば変換を行う必要がないため、すぐに要求か応答かの判別を行う。

要求であった場合にはプロセスを Create し、各種情報をプロセス・テーブルに登録した後に、PTT によって競合のチェックをする。競合がない場合にはレディキューにエンキューされプロセスは Ready 状態へと遷移する。制御は割込み不許可でスケジューラへと移る。

競合している場合はプロセスは Sleep 状態に遷移し、制御は割込みを許可してスケジューラへと移る (図 12の (a))。

応答であった場合には Sleep 状態になっているプロセスを PTT から検索する。プロセスを Wakeup させ、割込み不許可でスケジューラへと制御が移る。

プロセス実行後　要求の処理であった場合にはプロセスは Delete せずに Sleep させ (図 12の (b))、スケジューラへと制御が移る。

応答の処理であった場合には PTT からエントリを削除する。プロセス・テーブルをフリーリストへ返還し、プロセスは Delete される。PTT から削除する際に競合しているプロセスが存在すると、Delete されたプロセスにフックしていたプロセスは直ちに Wakeup し (図 12の (c))、レディキューにエンキューされる。

4.6　パケット・バッファの管理

パケット・バッファは、3.6節において 3 つめの要件に挙げたパケットを格納する無限長バッファである。本章ではその具体的な構成と利用方法について述べる。

4.6.1　概略

2.4節で述べたように MBP-Core には PBR と呼ばれるパケット格納専用のバッファが用意されている。RDT やクラスタ・バスへのパケットの転

送命令はPBR上のパケットに対してのみ発行できる。

PBRは汎用のものが64本用意されており、表1のように利用する。

表1: PBRのアドレス・マップ

Index	用途
0-39	パケット保存用(8パケット分)
40-43,48	ロード専用(1パケット分)
44-47,49	テンポラリ(1パケット分)
50-63	テンプレート

プログラムは、一貫性制御用に新たなパケットの生成を高速に行えるよう、あらかじめパケットの種類、データのサイズ及びキャッシングの可否などのパケットのヘッダ部のフィールドが設定されているテンプレートを用意している。代表的なものとしてクラスタ・メモリ・アクセス用パケットや無効化要求パケット、読み出し応答パケットなどがある。

テンポラリはクラスタ・メモリのデータ及びタグをプログラムが読み出す場合などに利用する。また、一貫性制御用に新たにパケットを生成する場合、テンプレートをテンポラリにコピーし、アドレスなどを設定することによってパケットを生成する。

ロード専用PBRについては後で詳しく説明する。

これらのPBRを除く残りのPBRはパケット保存用に利用するが、3.6節において3つめの要件として挙げたように、MBP-lightは理論上無限個のパケットを扱う必要があるにもかかわらず、表1に示すように最大で8パケット分しかない。これを解決するためにローカル・メモリ上にパケット用バッファ領域を確保した。基本的にはPBRを利用することとし、PBRの空きがなくなった場合にはメモリ上のバッファに退避させる。メモリ上に退避されたパケットは転送命令などを直接発行することができないために、処理の実行を行う際にはPBR上にロードしなければならない。したがってパケット・バッファへの操作は確保と解放の他にロードという操作がある。各操作の概略を述べる。

1. 確保

   パケットが到着し割込みが発生するとバッファを確保し、受信バッファからのコピーを行う。

   PBRに空きがある場合 :普通に保存用PBRにコピー

   PBRに空きがない場合 :テンポラリのPBRにコピーし、すぐにメモリ上のバッファに退避

2. ロード

   パケットがローカル・メモリ上に存在する場合はディスパッチ時にPBRにロードする。

3. 解放

   プロセスがDeleteまたはSleep時にはPBRを解放する。

詳しい動作については次項で述べる。

4.6.2 実装

通常は8パケット分のPBRが溢れることはあまりないと考え、基本的にはPBRのみを利用するよう最適化した。またここでもスループットを重視し、複雑な優先順位の制御は行わない。したがってプロセスの種類が要求/応答、Initiator/Homeなどにかかわらず公平にPBRが割り当てられる。

PBRはビットマップ方式のディレクトリで、メモリ上バッファはフリー・リストで管理している。管理すべきPBRは8個しかないために、PBRを管理するディレクトリの操作はMBP-lightの特殊命令を使用した。

先に挙げた3つの操作について詳細を説明する。

確保 パケットが到着すると保存用PBRに空きがあるかチェックする。空きがある場合には保存用PBRに受信バッファからコピーする。空きがない場合には直接ローカル・メモリに退避することができないために、テンポラリにコピーしてからメモリ上に退避させる。このときリプレースの対象を決定する手間をはぶくために、保存用PBR上のパケットをリプレースすることはしない。パケットを格納したロケーションはプロセス・テーブルに登録される。

ロード プロセスの実行を開始する際にプロセス・テーブルを参照する。パケットがメモリ上にある場合はロード専用PBRへロードし、メモリ上バッファを解放する。ロード後は保存用PBR上のパケットと同様に扱うことができる。

解放 プロセスの実行が終了しSleep/Deleteへと状態が遷移する場合、保存用PBRを利用していればそれを解放する。このとき空きができてもプリロードは行わない。ロード専用PBRを利用していた場合には特になにもしない。

最後にロード専用のPBRについて説明する。

短いプロセスは前述のように割込み不許可で一気に実行され、プロセスの状態がSleepやDeleteに遷移すると必ずPBRを解放する。なぜならば必要なコンテキストは全てプロセス・テーブルに保持されているからである。この特徴を利用し、ロード専用のPBRを用意してリプレースを省略することで、そのオーバーヘッドを削減できる。ロード専用PBRはプロセスの実行が終了すると必ず解放される。

ロード専用のPBRがあるために、プロセスの処理が終了してPBRに空きができてもメモリ上に退避されているパケットのプリロードは行わない。空いたPBRは次に到着したパケットが利用する。

ただし長いプロセスがロード専用PBRを利用している際にPreemptされると、コンテキストとしてPBR上のパケットを保存する必要がある。

5 性能評価及び考察

本章では作成したDSM管理プログラムを用いた基本性能の評価結果について述べ、3次キャッシュ・コントローラをソフトウェア・ベースで開発したアプローチについての検証をする。

5.1 評価環境

VHDLで論理合成可能なように記述されたJUMP-1の設計データを用い、Mentor Graphics社のVHDLシミュレータqhsim v8.5_4.6cを用いてDSM管理プログラムの動作シミュレーションを行った。したがってサイクル数は実際のものと同じである。

クラスタ数は4とし、無負荷状態での読み出しにかかるレイテンシを計測した。

仮想メモリは現在実装段階であり、ページテーブル及び逆引きページテーブルの管理は行っていない。変換用に用意するTLBにヒットすると考えると、変換にあまり多くのサイクル数を要するとは思えないが、本稿で挙げるサイクル数にはその時間は加算されていない。

5.2 読み出し要求に要するサイクル数

読み出し要求のReplierが1次、2次、3次キャッシュ、Homeクラスタ及びHomeクラスタではない他のクラスタ(図6参照)である各場合のレイテンシを表2に示す。単位はサイクルである。

表2: 読み出し要求処理時間

Replier	時間(サイクル)
1次キャッシュヒット	1
2次キャッシュヒット	5
3次キャッシュヒット	29
Home(メイン・メモリ)	1878
not Home	3757

Homeクラスタへの読み出しには1878サイクルかかることがわかった。

次に読み出し要求のReplierがHomeクラスタである場合(表2の4段目)の、Homeクラスタ上での処理を詳細に調べた結果を表3に示す。

Homeクラスタではクラスタ・メモリ上に有効なデータが存在しているとき、そのデータを直接読み出して応答パケットを作成する。したがって表3中の合計値はHomeクラスタにパケットの到着による割込みが発生してから、その応答をInitiatorクラスタに送信するまでのサイクル数である。サイクル数はコンパイラによって作成されたコードを手で数えた。

プロセスのCreate、PTTへの登録及びディスパッチャはコンパイラによってインライン展開されておらず、レジスタ保存のためのスタック操作のサイクル数も含んでいる。

表 3: Home クラスタでの処理時間の詳細

	処理の種別	サイクル数
カーネル	パケット種の判別	25
	パケット・バッファ	47
	プロセスの Create	116
	PTT	73
	キューイング	12
	スケジューラ	105
	ディスパッチャ	63
	その他	43
プロセス	3 次キャッシュ・ディレクトリ検索	24
	クラスタ・メモリ・タグの読み出し	22
	メイン・メモリからの読み出し	24
	応答パケットのヘッダ作成	14
	パケットの送信	65
	その他	24

いくつかの項目について補足する。

- パケット・バッファ

  2.4節で述べたように受信用 PBR は 3 組用意されているものの、MBP-Core からは先端のものしか扱うことができない。しがたって RDT からのパケットは保存用 PBR を確保した後に、受信用 PBR からコピーを行わなければならない。

- スケジューラ

  レディキューを検索するサイクル数だけでなく、送信バッファの確認も含まれる。この処理の場合は reQ to Owner のレベルに相当し、送信バッファとしては RDT 送信バッファだけでなくクラスタ・バス側の送信バッファもチェックする。なぜならば Owner クラスタが Home クラスタ自身、つまり Home クラスタ内の 2 次キャッシュ上にのみ有効なデータが存在する場合があるからである。

- 3 次キャッシュ・ディレクトリ検索

  Owner クラスタを検索し転送先を判別する。この場合は Home クラスタが Onwer クラスタである。

- クラスタ・メモリ・タグの読み出し

  クラスタ・メモリ上に有効なデータが存在するかの確認をする。MMC がヒット判定に利用するタグを直接読み出し、メモリ上のデータの有効性を判断する。有効であればデータをクラスタ・メモリから直接読み出し、有効でない場合には 2 次キャッシュ上にあるものと判断しクラスタ・バスへパケットを転送する。

- メイン・メモリからの読み出し

  2.4節で述べたように MBP-Core はクラスタ・バスを介さずにクラスタ・メモリを読み出すことができる。その際にはテンプレートを利用して読み出し用のヘッダを生成し、専用の命令によってアクセスする。

- パケットの送信

  パケットを送信用 PBR へコピーし、送信命令を発行する。ただし RDT ネットワークのヘッダ部作成の時間も含まれる。送信先はプロセス・テーブルを参照する。

カーネル部ではプロセスの Create とスケジューラのサイクル数が大きな値となった。プロセス・テーブルやレディキューはローカル・メモリ上に

存在するために、どちらの処理もメモリ・アクセスを頻繁に行う。結果としてローカル・メモリ・アクセスに1サイクルのストールを伴なってしまうというMBP-lightの弱点がそのまま値として現れた。

詳細は割愛するが、パケットが到着してから転送を行うまでのサイクル数を平均すると約600サイクル程度かかることがわかった。したがってOwnerクラスタがHomeクラスタではない場合は

1. Initiatorクラスタから Homeクラスタへ
2. HomeクラスタからOwnerクラスタへ
3. Ownerクラスタのクラスタ・バスへ
4. OwnerクラスタからHomeクラスタへ
5. HomeクラスタからInitiatorクラスタへ
6. Initiatorクラスタのクラスタ・バスへ

の合計6回の転送が必要(図6参照)となり、MBP-Core上の処理のみで約3600サイクル必要である。表2の値はさらに通信などの時間を加えたものと考えることができる。また、Homeクラスタ上のCCに有効なデータが存在している場合は約2400サイクル程度であった。

5.3 考察

本節ではまずソフトウェアで3次キャッシュ管理を行ったオーバーヘッドについて考察し、次に前節の結果を用いてJUMP-1のメモリ・システムについて考察を行う。

5.3.1 ソフトウェア・オーバーヘッド

ソフトウェアで行うオーバー・ヘッドについて考察してみる。

基本的に各処理は全て行う必要があり、純粋にソフトウェアで行うために増えてしまう処理はない。

3.5節で要件として挙げたが、JUMP-1では更新プロトコルをサポートするために、プロセスの管理やパケット・バッファの管理は理論上無限の数を相手にしなければならい。これらの処理をハードウェアによる実装で行うことはかなり困難である。その一方でPTTは連想記憶などを用いれば比較的容易にハードウェア化できよう。また、アドレス変換に用いるTLBやMMUなどは既存のものをそのまま流用することが可能である。

スケジューラの主なオーバーヘッドは資源の確認である。MBP-CoreにはRDT送信バッファに空きがあれば割込みがかかる機構が存在する。しかしながら、クラスタ・バス側のバッファに対しては同様の機構が備っておらず、片手落ちと言える。この機構が備わっていれば、スケジューラは必要なバッファに関する割込みがかかるように設定し、idleプロセスを実行すればよいだけである。このような実装を行うことができた場合には、スケジューリングにかかるサイクル数は半分以下になるものと思われる。

また、クラスタ・メモリ・タグ及びデータの読み出しは、読み出し用パケットを作成する必要がある。専用の命令を用意することによって、これらのサイクル数は容易に削減することができるであろう。

以上から、DSM管理において定型的な処理を行う部分はハードウェアによってさらにサポートし、柔軟性を必要とするプロセスの管理などだけをソフトウェアによって行うべきである。

5.3.2 メモリ・システムの平均レイテンシ

次に5.2節の結果を用いてJUMP-1のメモリ・システムについて考察を行う。

簡単のため、Homeクラスタではない他のクラスタからデータを獲得しなければいけない場合(表2最下段のデータ)については考慮しないものとする。

1次、2次、3次キャッシュ及びメイン・メモリへの読み出しアクセスにかかるレイテンシがA、B、C、Dサイクルであるシステムを考えたときに、1次、2次及び3次キャッシュのヒット率を$\alpha_1\%$、$\alpha_2\%$、$\alpha_3\%$とすると、平均レイテンシLは次のような式で与えられる。

$$L = A \times \alpha_1 + B \times (1-\alpha_1)\alpha_2$$

$$+ C \times (1-\alpha_1)(1-\alpha_2)\alpha_3 + D \times (1-\alpha_1)(1-\alpha_2)(1-\alpha_3)$$

各階層のヒット率が十分に高い場合には$(1-\alpha_1)(1-\alpha_2)(1-\alpha_3)$の項が極めて小さな値となり、メイン・メモリへアクセスはほとんど生じない。

この式を用いて読み出しの平均レイテンシを計算し、JUMP-1 のメモリ・システムについての評価を行った。ヒット率を向上させるために階層的なタイリングを施して、メモリ・アクセスの最適化を行った行列積 [2] を計算した際の平均レイテンシ L を表 4に示す。

表 4: 読み出し平均レイテンシ

	α_1 (%)	α_2 (%)	α_3 (%)	L(サイクル)
1	100.0	-	-	1.0000
2	96.875	100.0	-	1.1244
3	96.875	99.609	100.0	1.1279
4	96.875	99.609	99.902	1.1282
5	96.875	99.609	99.988	1.1280
6	96.875	99.609	0.0	1.3536
7	90.0	93.0	95.0	2.2152

各データは以下の場合である。

1. 1 次キャッシュにすべてヒット
2. 2 次キャッシュまでですべてヒット
3. 3 次キャッシュまでですべてヒット
4. メイン・メモリへのアクセスあり[1]
5. 3 次キャッシュの容量が 64MByte[2]
6. 3 次キャッシュを利用しない
7. ヒット率が下った場合

ヒット率が十分に高ければ 3 次キャッシュにすべてヒットする場合と比較 (表 4の 3 と 4) してもわずか 0.0195%程度悪化するだけで、ミスヒット時の処理をソフトウェアで行ってもその影響は軽微である。

また、3 次キャッシュの容量を大きくしてもあまり改善は見られず (表 4の 4 と 5)、今回のようにヒット率が高い場合には容量はあまり求められていないことがわかる。

さらに 3 次キャッシュを利用することによって、利用しない場合と比べて (表 4の 4 と 6) 平均レイテンシが 20%程度削減される。各ヒット率が高いために 3 次キャッシュまでアクセスする場合はほとんど稀であるにもかかわらずかなりの効果があることがわかった。

しかしながらヒット率が低下すると平均レイテンシは加速度的に増大するために、キャッシュ・ヒット率を向上させる技術はかなり重要となる。

5.3.3 布線論理で構成した場合

3 次キャッシュ・コントローラを全て布線論理で構成した場合について考察してみる。

リモート・メモリ・アクセスを全て布線論理で制御した場合のレイテンシは約 300 サイクル程度であると予想する。この値は JUMP-1 と同様にクラスタリングを行っている Sun Microsystems 社の WildFire[5] を参考にした。

このときの平均レイテンシ $L = 1.1279$ となり JUMP-1 と比べてもほんの僅かしか改善がみられない。ただし、リモート・メモリアクセスに必要なサイクル数以外のパラメータは JUMP-1 と同じ値を用いた。しかし、この結果は 3 次キャッシュのヒット率が極めて高い場合であり、ヒット率が低下するにしたがってその差は広まっていく。

階層構造をもつシステムとしてキャッシュのヒット率というものは重要度はかなり高い。ヒット率を高くできるような技術が今後さらに出現してくるのであれば、複雑な制御を全てハードウェアで実装するための開発の効率やコストを考えるとソフトウェアで制御するアプローチは有効であると考える。

6 おわりに

本稿では JUMP-1 における MBP-light 上で動作する DSM 管理プログラムの実装と、それを用いた性能評価の結果について述べた。

JUMP-1 は物理的構造だけでなくメモリ・システムも階層性を持つ並列計算機である。ある程度

---

[1] メイン・メモリの半分 (8MByte) を 3 次キャッシュとして利用

[2] クラスタ・バスのアドレス空間の最大である 64MByte の容量を持つクラスタ・メモリが仮に存在し、そのほとんどを 3 次キャッシュとして利用

の規模をもつ計算機は今度ますますこのような階層構造を持つものとなってくるであろう。

JUMP-1 ではメイン・メモリの一部をリモート・メモリの 3 次キャッシュとして利用する。MBP-light は 3 次キャッシュを管理する専用のプロセッサであり、管理方法としてはハードウェアとソフトウェアによるハイブリッドなアプローチを採った。すなわち 3 次キャッシュにヒットした場合には MMC と呼ばれる wired-logic が高速に応答し、一方ミスヒットした場合には MBP-Core 上で動作するプログラムをベースとして処理を行った。

DSM 管理プログラムはトランザクションの実行を行うために、様々な複雑な制御も行わなけらばならない。例えばトランザクション間の競合の検出や、デッドロック及びスターベーションを回避するために無限のキュー及び無限のバッファを用意する必要がある。プログラムではこれらの制御を行うためにカーネル部とプロセス部の 2 階層に分割される。そしてコア・プロセッサの占有を防止するために割り込みの可否によってプロセスを 2 つに分けて、その優先度の制御を行った。また、プロセス間の競合は PTT というテーブルにプロセスを登録することによって、その検出を行った。さらにデッドロックフリー/スタベーションフリーなシステムを構成するためにソフトウェアによる 6 つのレディキューやローカル・メモリを利用したパケット・バッファを用意した。

シミュレータ上でこのプログラムを実行することにより、SuperSPARC+からのメモリ・アクセスのレイテンシを計測した。リモートのメイン・メモリ・アクセスには 1878 サイクルかかることがわかった。また、ヒット率の向上を考慮したアプリケーションを用いて平均レイテンシを計算したところ、1.12822 サイクルと良い結果を収めることができた。高いヒット率を持つアプリケーションでは、3 次キャッシュやソフトウェア・ベースで管理することの有効性が示された。しかし、高いヒット率を出さなくては高い性能を得ることができないこともわかった。

近年プロセッサの性能とメモリの性能の差が相対的に広がっていることを背景に、キャッシュのヒット率向上の研究などが行われている。今後はそのような技術の研究がよりいっそう重要となってくると言える。さらに数値計算などだけではなく、多くの実用的なアプリケーションに対してもそのような技術の研究が必要となるであろう。

謝辞

本研究の機会を与えてくださり、適切な御指導を賜わった富田 眞治教授に深甚な謝意を表します。

また、貴重なご助言をいただいた森 眞一郎助教授、中島 康彦助教授、五島正裕助手に深く感謝致します。

さらに，共同研究者である小西 将人氏、齋藤 康二氏をはじめとして，日頃から御鞭撻下さったコンピュータ工学講座計算機アーキテクチャ分野の諸兄に感謝致します。

また、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致します。

参考文献

[1] 秤谷 雅史：「超並列計算機 JUMP-1 における分散共有メモリ管理の実装とその評価」、SWoPP ’98

[2] 津田 健：「メモリ・アクセスの局所性を最適化するループ再構成法」、HOKKE ’99

[3] Kai Li, Paul Hudak :Memory Coherence in Shared Virtual Memory Systems, In the ACM Transactions on Computer Systems, Vol7, No.4, November 1989, pp321-359

[4] 細見 岳生：「並列計算機 Cenju-4 の分散共有メモリ機構」、並列処理シンポジウム JSPP ’99, pp. 15-22.

[5] Erik Hagersten. and Michael Koster. : ”WildFire:A Scalable Path for SMPs”, *The Fifth International Symposium on High-Performance Computer Architecture*, IEEE (1999).

[6] Kenjiro Taura, Kunio Tabata, and Akinori Yonezawa. ”Stackthreads/MP : Integrating Futures into Calling Standards.” *In Proceedings of ACM SIGPLAN Symposium on*

Principles & Practice of Parallel Programming, 1999

[7] Goshima, M., Mori, S., Nakashima, H. and Tomita, S.: The Intelligent Cache Controller of a Massively Parallel Processor JUMP-1, *IWIA'97*, IEEE, pp. 116–124 (1997).

[8] Yang, Y., Amano, H., Shibamura, H. and Sueyoshi, T.: Recursive Diagonal Torus: An Interconnection Network for Massively Parallel Computers, *The 5th IEEE Symp. on Parallel and Distributed Processing*, IEEE (1993).

[9] 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, pp. 265-272.