計算機アーキテクチャ 101−16
（1993. 8. 20）

細粒度プロセッサ間通信をサポートする高機能キャッシュ・システム

五島 正裕†, 岡田 智明†, 細見 岳生†,
森 眞一郎†, 中島 浩†, 富田 眞治†

†：京都大学 工学部

我々は，一つのアーキテクチャで共有メモリ型およびメッセージ・パッシング型の両方の通信モデルをサポートすることが重要であると考え，スケーラブルな共有メモリ・マルチプロセッサのコヒーレント・キャッシュ・システムとメッセージ通信機能を統合することを試みている．本システムではI-Structure や FIFO などの同期構造体を利用して高速な細粒度メッセージ通信を実現する．本稿ではこのキャッシュ・システムのコヒーレンス制御方式とメッセージ通信機構について述べる．

The Intelligent Cache System for Fine-Grain Inter-Processor Communication

Masahiro Goshima†, Chiaki Okada†, Takeo Hosomi†,
Shin-ichiro Mori†, Hiroshi Nakashima†, Shinji Tomita†

†: Department of Information Science
Faculty of Engineering, Kyoto University
Yoshida-hon-machi, Sakyo-ku, Kyoto 606-01 Japan

E-mail: {goshima, okada, hosomi, moris, nakasima, tomita}@kuis.kyoto-u.ac.jp

Based on an opinion that a single architecture should support communication models of both shared memory type and message passing type, we are trying to integrate message communication mechanism with the coherent cache system on the shared memory multiprocessor. This system realizes fast fine-grain message communication by utilizing the synchronization structure such as I-Structure or FIFO. In this paper, we describe the coherence control method and the message communication mechanism of the system.

1 はじめに

大規模マルチプロセッサのアーキテクチャは大きく2つのクラス——共有メモリ型とメッセージ・パッシング型に分類できる．それぞれのクラスはそれぞれの通信モデルを基礎にしている．

それぞれの型の計算機はそれぞれの通信モデルを強力にサポートするように設計される．もちろんどちらのアーキテクチャ上でも両方のモデルを実現可能であるが，サポートされていないモデルを用いることは深刻な性能の低下を招く．

我々は一つのアーキテクチャで共有メモリとメッセージの両方の通信モデルをサポートすることが重要であると考え，スケーラブルな共有メモリ・マルチプロセッサにメッセージ通信のためのサポートを付加することを試みている[1][2]．

本稿では，両方の通信モデルをサポートする高機能キャッシュ・システムについて述べる．

2章では前提として，本稿で述べるキャッシュ・システムの対象とする計算機の概要と，その機能を利用するためのコマンドの指定法について述べる．

3章では共有メモリ通信モデルのサポートとしてキャッシュ・コヒーレンス機能について述べる．

メッセージ通信モデルのサポートとして本キャッシュ・システムでは I-Strucuture や FIFO などの同期構造体を用いる．4章では同期構造体のサポートについて述べる．

2 前提

2.1 対象とするシステム

本稿で述べるキャッシュ・システムが対象とする計算機の概要について述べる[1][2][3]．

全体構成 対象とする計算機は数K～数十K個のプロセッサを持つ，スケーラブルな共有メモリ型マルチプロセッサである．

対象計算機はクラスタ構造を持つ．クラスタのイメージを図1に示す．

各クラスタは主に，2～4個のプロセッサ，クラスタ・メモリ，ネットワーク・インターフェイス・ユニットなどからなる．それらを共有バスにより結合する．

更に，クラスタ・メモリの近傍にはメモリ管理用のコプロセッサを持つ．コプロセッサがクラスタ間のコンシステンシ制御，データ転送処理などを行う．

最大数十K個のクラスタを一般のネットワークで接続する[4]．

プロセッサ プロセッサには市販の高性能RISCチップを用いる．ただし以下の機能を持つものとする[5][6]．

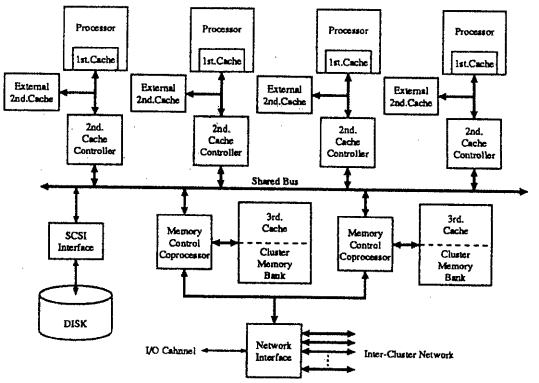

図 1: クラスタのイメージ

- 内蔵1次キャッシュを備え，そのコンシステンシ制御のための最低限の機能を持つ．ライトは外部から可視である．
- いくつかの同期基本命令が用意されており，これらを利用して弱いコンシステンシ・モデルに基づく制御が可能である．

メモリ・アーキテクチャ 対象計算機は共有メモリ方式であり，クラスタ内はスヌープ方式，クラスタ間はディレクトリ方式により，ハードウェアでキャッシュ・コヒーレンスを維持する．

クラスタのメモリは他のクラスタのメモリに対する3次キャッシュとしても機能する．ただし本稿では，主記憶としてのメモリと3次キャッシュが別個に存在し，並列にバスに接続されているとしても差し支えない．したがって，以降では“3次キャッシュ/メモリ”と表記する．

2.2 コマンドの指定法

本節では，以降の節の準備として，システムの提供する各機能を起動するコマンドの指定方法について述べる．

コマンドの指定には，プロセッサから出力される実アドレスの一部を用いる．プロセッサから出力される実アドレスは上下2分割される．実アドレスの下位（およそ30bits）は物理メモリのアドレスに用い，上位（およそ6bits）はその物理アドレスに対するコマンドを指定する．

1つの物理アドレスに対する各コマンドは実アドレスの上位数bitsが異なるため，異なる物理ページにマップされる．したがって，コマンドごとにCacheable/Uncacheableを指定できる．

Example: 例えば，&h000 ～ &hFFF のアドレスを持つ物理メモリを持つシステムにおいて，表1に示す16種類のコマンドを用意するとしよう．表中の第1列の数値がコマンドの番号を表す．

表 1: コマンドの例

#	ロード	ストア
0	通常のロード	通常のストア
1	Fetch & Inc.	Indivisible Add
⋮	⋮	⋮
F	—	プリフェッチ

実アドレス&h1A00 に対して '5' を Store すると，物理アドレス&hA00 に対して '5' を引数に#1 のコマンド——Indivisible Add を発行することになる．物理アドレス&hA00 の内容は不可分に '5' だけ増やされるであろう．

また，ページ・サイズを &h100 Bytes とすると，物理アドレス&hA00 に対する 16 種のコマンド&h0A00 ～ &hFA00 は，異なるページにマップされる．

実アドレス&hFA000 に対する Store はプリフェッチ指示であり，Uncacheable とするのが適当であるが，同時に通常のロード/ストアや Fetch & Inc. などは Cacheable にしてよい．

3 キャッシュ・コヒーレンス制御

共有メモリ型の通信モデルのサポートとして，ハードウェアによるキャッシュ・コヒーレンス制御を行う．

3.1 節では，コヒーレンス制御方式の設計方針に関して述べる．その方針に基づいて設計されたプロトコルについて 3.2 節で一度簡単にまとめる．1 ～3.3 節で，3.1 節で述べた設計方針にしたがって，制御方式の特徴を詳細に説明する．

3.1 コヒーレンス制御方式の設計方針

コヒーレンス制御方式の設計方針は以下の 3 点である．

方針 1. ソフトウェアの指示情報の利用

プロトコルの改良などによって純ハードウェア的にコヒーレンス動作の最適化を図ることには限界がある．

したがって，ソフトウェアによって与えられる指示情報を最大限に利用するとともに，コヒーレンス制御の多くの部分をソフトウェアから直接制御可能とする．

方針 2. 3 次キャッシュ/メモリ負荷の低減

クラスタをまたがるトラフィックはすべて 3 次キャッシュ/メモリを通過するため，この部分が高負荷になる可能性が高い．

一方で，クラスタ構造を意識したアプリケーションでは，クラスタ内に閉じるトランザクションが多いと予想される．

したがって，クラスタ内のトランザクションはできる限りキャッシュ間で処理し，3 次キャッシュ/メモリの負荷の低減を図る．

表 2: 2 次キャッシュ・コピーのステート

Inv	Invalid
Ex-Cln	Exclusive, Clean
LS-Cln	Locally Shared, Clean
GS-Cln	Globally Shared, Clean
Ex-Dty	Exclusive, Dirty
LS-Dty	Locally Shared, Dirty

方針 3. リード時のレイテンシの短縮

プロセッサにとって本質的であるのはライト時のレイテンシではなく，リード時のレイテンシである．

リード時のレイテンシの低減のために Update プロトコルを強力にサポートする．

また，リード・ミス時に最新のデータがメモリに存在しない場合，そのレイテンシは存在する場合の 2 倍程にもなる．クラスタ間のデータの共有を考慮し，メモリができる限り最新のデータを保持するように制御する．

方針 2 と方針 3 は，3 次キャッシュ/メモリの更新に関して相反することを要求している．この解決にソフトウェアによる指示は不可欠である．

これらの方針に基づいて設計された制御機構の特徴について以降で述べる．

3.2 コヒーレンス・プロトコルの概要

詳細について述べる前に，本項ではプロトコルの概要について簡単にまとめておく．

3.2.1 2 次キャッシュのステート

2 次キャッシュ・コピーは表 2に示すの 6 つのステートを持つ．特徴は，以下に示すクラスタ構造を意識した 3 つの共有状態を持つことである．

Exclusive システム中に当該ブロックのキャッシュ・コピーは唯一である．

Locally Shared クラスタ内に複数のコピーが存在する可能性があるが，他のクラスタには存在しない．

Globally Shared クラスタ間にまたがって複数のコピーが存在する．

3.2.2 コヒーレンス・プロトコルの特徴

コヒーレンス・プロトコルの基本的な特徴は以下のようにまとめられる．

- Write-Invalidate/Update はページごとアクセスごとに切替え可能．

表 3: コヒーレンス・コマンド

Inv	全キャッシュ・コピーを無効化する[a].
Inv-Wr	他のコピーを無効化し，ライトする.
Updt-Wr	ライトし，他のコピーを Update する.
Ex-Rd	リードし，他のコピーを無効化する.

[a]Dirty コピーはメモリに書き戻される.

- クラスタ内では Ownership を持たないが，クラスタ間では Ownership を持つ.
- クラスタ内ではキャッシュ間転送を行い，転送時にメモリを更新しないが，クラスタ間ではキャッシュ間転送を行わなず，転送時にメモリを更新する.
- Multi-Level Inclusion を保証する.

3.3 方針 1. ソフトウェアの指示情報の利用

3.3.1 コヒーレンス・プロトコルの切替え

2 次キャッシュでは，コヒーレンス制御に関わるページ属性として以下の 3 種を意識する.

Global クラスタ間のコヒーレンス制御を必要とする.

Local クラスタ間のコヒーレンス制御を必要としない.

Incoherent コヒーレンス制御を必要としない.

コヒーレンス制御を行うページに対しては，2 次キャッシュでは

- Write-Invalidate,
- Write-Update,

の双方のプロトコルをサポートする. これらもページごとに指定が可能である.

3.3.2 コヒーレンス・コマンド

表 3 に示すコヒーレンス・コマンドを用意する. これらのコマンドはページのコヒーレンス・プロトコル属性に関わらず発行することができ，よりきめ細かい動作指示が可能となる.

3.4 方針 2. 3 次キャッシュ/メモリ負荷の低減

3.4.1 クラスタ内キャッシュ間転送

メモリ負荷の低減のため，クラスタ内ではキャッシュ間転送を行い，キャッシュ間転送時に 3 次キャッシュ/メモリを Update しない.

3.4.2 Random Reply

本プロトコルは，クラスタ内では Ownership の概念を持たない. すなわち，バス上のリード・リクエストに対して，最新のコピーを保持しているすべてのデバイス——2 次キャッシュ および 3 次キャッシュ/メモリがリプライを試みる.

クラスタ・バスを獲得したデバイスが実際にリプライを開始する. クラスタ・バスを獲得できなかったキャッシュはリプライをキャンセルする.

本方式は，以下の特長を持つ.

負荷の分散 その瞬間に busy でなかったデバイスがリプライすることになるため，負荷が分散される.

特に 3 次キャッシュ/メモリの負荷を低減する効果がある. キャッシュがコピーを保持している時には，高い確率でキャッシュがリプライを行う.

応答性の向上 その瞬間にもっとも早く応答可能なデバイスが応答するのだから，全体としての平均応答時間が短縮される.

3.4.3 クラスタ間共有状態の管理

2 次キャッシュ・コピーは，クラスタ構造を意識した 3 種の共有状態を持つ (3.2.1 項参照).

こうして 2 次キャッシュでもクラスタ間の共有状態を管理することにより，クラスタにローカルなライトに対する Ack. 処理を省略する. すなわち，キャッシュ・コピーが Locally-Shared である時には，

- キャッシュはクラスタ・バスの獲得をもってライトの完了とみなし，Ack. を待たない. と同時に，
- 3 次キャッシュ/メモリはバス・トランザクションによってライトがローカルであることが分かり，3 次キャッシュ/メモリの更新 および キャッシュ・ディレクトリの検索を省略できる.

3.5 方針 3. リード時のレイテンシの短縮

3.5.1 Update プロトコルに対するサポート

コヒーレンス制御のプロセッサ間通信の側面を意識し，Update 系のプロトコルを強力にサポートする.

Update プロトコルの欠点はトラフィックが増加しがちであることに尽きる. 以下の 3 点により対応する.

I. Update リクエストのマージ

弱いオーダリング・モデルに基づけば，コヒーレンス維持のための要求を即座に出力する必要はない. 外部への Update リクエストを適当にバッファリングしマージすることによって，クラスタ・バスおよびネットワークの実効バンド幅の向上を図る.

表 4: 3 次キャッシュ/メモリ・コピーの状態

キャッシュ・コピーの状態	Ex-Cln	Ex-Dty
方法 1	Dty	Dty
方法 2	Cln	Dty

II. Exclusiveness Check の強化

クラスタ内では，Update リクエストを送出した時に，自コピーの Exclusiveness を検査することにより，それ以降の無用な Update を除去する．Update リクエストをバスに送出した時に，他のデバイスはスヌープのヒット/ミスを Wired-OR 線に出力する．Update のリクエスタはその線をチェックすることにより，自コピーの Exclusiveness を判定する．

III. Invalidate 系のコマンド

3.3.2 項で述べたように，ページ属性として Update プロトコルを選択しているページのブロックに対しても Invalidate 系のコヒーレンス・コマンドを発行可能である．ソフトウェアは，適切なタイミングで無用なコピーを Invalidate することができる．

3.5.2 メモリの Clean 化

クラスタ構造を持つシステムでは，2 次キャッシュ・コピーとメモリ・コピーが異なるクラスタに存在することがある．したがって，キャッシュ・コピーが Ex-Dty である時にはメモリ・コピーは Dty でなければならない．

この時，表 4 に示す 2 つの方法が考えられる．方法 1 では Ex-Cln コピーのリプレースの通知が，方法 2 では Ex-Cln コピーへのライトの通知が，3 次キャッシュ/メモリに対して必要である．

問題は，Ex-Cln である時に他のクラスタからのリード・リクエストが発生した場合のそのレイテンシである．リード・ミス時に最新のデータがメモリに存在しない場合，そのレイテンシは最低でも存在した場合に比べて 1.5~2 倍になってしまう．

2 次キャッシュは Self Cleanup 機能 [7] を持ち，Global 属性 (3.3 節を参照) を持つページに関しては，メモリをできる限り Cln に保つ．

したがって，本システムでは方法 2 を採る．

4 メッセージ通信機能

対象とする計算機は任意のアドレスに I-Structure や FIFO などのような一般的な同期構造体を形成する．これらの同期構造体を利用してメッセージ通信を行う．Ack. と明示的な同期操作が不要となり，共有メモリ空間上で細粒度メッセージ通信を効率よく実現できる．

対象計算機のメモリは full/empty bit を含む数 bits のタグを持つ．メモリ管理用コプロセッサがこの bit を操作することにより，一般的な同期構造体を実現する．同期構造体へのアクセスは基本的にメモリ管理用コプロセッサにコマンドを与えることによって行う [8]．

2 次キャッシュもワードごとに full/empty bit 持ち，2 次キャッシュ・コントローラとメモリ管理用コプロセッサとの協調動作によりこの同期構造体のサポートを行う．2 次キャッシュでは主に，I-Structrure のキャッシング，および，FIFO トップのプリフェッチングにより，リード・アクセス時のレイテンシの短縮を図る．

4.1 節では I-Structure について，4.2 節では FIFO について述べる．そして，パケット単位に FIFO を構成するアクセスについて，4.3 節で述べる．

4.1 I-Structure

対象計算機の各ワードは full/empty bit を持っており，full/empty の状態を変更するライト・コマンドと状態をチェックするリード・コマンドによって I-Structure による通信が可能である．

2 次キャッシュも full/empty bit を備えており，I-Structure をキャッシュすることができる．

4.1.1 I-Structure のコマンド

Full/Empty Bit の解釈

I-Structure のライト・コマンドは full/empty bit を 0 にセットするモードと，1 にセットするモードを持つ．

一方，リード・コマンドには，full/empty bit の 0/1 を full/empty と解釈するモードと，empty/full と解釈する 2 つのモードがある．

一度使用して full となった I-Structure を再利用するには，ライタ／リーダ双方でこのモードを反転する．2 次キャッシュ上の I-Strucure のコピーの再利用が可能である．

リード・コマンドのリターン値

ワードの full/empty の状態に対して表 5 に示す値を返す 5 種のリード・コマンドを定義する．

4.1.2 I-Strucure アクセスの Cacheability

ライト・アクセスの Cacheability

2.2 章で述べたように，I-Strucure に対するライトとリードは異なる物理アドレスにマップされる．したがって，ライト・アクセスを 1 次キャッシュに Cacheable とする意味はない．

2 次キャッシュには Cacheable としてもよい．

表 5: リード・アクセスの返す値

#	full 時	empty 時
0	DATA	DATA
1	full を表す値 (1)	empty を表す値 (0)
2	DATA	NULL ポインタ値 (0)
3	DATA	Not A Number [a]
4	DATA	— [b]

[a] 浮動小数点数として使用しないビット・パタン
[b] 一定時間待って割り込みを発生する

リード・アクセスの Cacheability

empty 時に割り込みを発生するリード・コマンド (表 5 中の#4) 以外のリード・コマンドは 1 次キャッシュまで Cacheable としてよい.

表 5 中のいずれかのアクセスが 1 次キャッシュ上でミスを起こした時にはまず, full/empty 情報付のデータが 2 次キャッシュにブロック転送される. そして, ブロック内のすべてのワードに対してリターン値が生成され, 1 次キャッシュに転送される.

Example: 例えば, 4 ワードからなる 1 次キャッシュ・ブロックに対し, 表 5 の#1 のコマンドが 1 次キャッシュ・ミスを起こしたとしよう. 2 次キャッシュにブロックが転送されてきた時, ブロックの各ワードの状態が full, full, empty, empty であったとすると, 1 次キャッシュには各ワードの値が '1', '1', '0', '0' であるブロックが 1 次キャッシュにキャッシュされる.

full/empty の状態の変化は, ライトによって起こる. したがって, 状態が変化した時には通常のキャッシュ・コヒーレンス動作 (Invalidate あるいは Update) が行われ, プロセッサは新しいデータを取得することができる.

4.1.3 ブロック単位 I-Strucure

リード時にデータが full である確率が非常に高い場合がある. そのような場合には empty 時に割り込みを発生するリード・コマンド (表 5 中の#4) はもっとも有効である.

しかし, empty 時に割り込みを発生するリード・コマンドは, 1 次キャッシュには Cacheable にできない. empty であるワードを含むブロックを 1 次キャッシュまでキャッシュすると, empty であるワードへのアクセスが 2 次キャッシュ・コントローラに不可視となるからである.

そこで, full/empty の管理をブロック単位に行うリード・コマンドを用意する. 使用できる場面は限られるが, 割り込みを発生するアクセスを 1 次キャッシュまで Cacheable にできる.

このコマンドにおいては empty であるワードを含むブロックを empty とみなす. 2 次キャッシュ・コントローラはこのコマンドが 1 次キャッシュ・ミスを起こした時に, ブロックのすべてのワードの full/empty の状態をチェックする. ブロック内のすべてのワードが full であれば, 当該ブロックを 1 次キャッシュに転送し, empty なワードが存在する時には (一定時間待った後に) 割り込みを発生する.

表 6: FIFO のコマンド (一部)

Enq	FIFO の後尾にデータを追加 (Store) する.
Deq	FIFO の先頭データを Load する. 当該データは FIFO の先頭から取り除かれる.
Top	FIFO の先頭データを Load する.

4.2 FIFO

メモリ管理コプロセッサがメモリ上で管理する FIFO をコマンドによって操作する. FIFO の操作はこのコプロセッサがメモリ上で不可分に実行する.

4.2.1 FIFO アクセスのコマンド

表 6 に示すコマンドにより共有アドレス空間上に FIFO を形成する [9]. あるアドレスに対し, Enq コマンドによって Enqueue されたデータは, Enqueue された順序で Deq コマンドによって Dequeue される.

Top コマンドには表 5 に示した 5 種のバリエーションがある. 一方, Deq コマンドには表 5 に示したうちの#1 以外の 4 種のバリエーションがある.

コマンドはメモリ上の 1 次キャッシュ・ブロックの先頭の 1 ワードに対してのみ発行できる. ブロックの残りのワードはメモリ管理コプロセッサが作業領域として使用する.

4.2.2 FIFO の 2 次キャッシュによるサポート

複数のリーダによって Dequeue 操作が行われるような FIFO のキャッシングはさまざまな不具合を生じる. したがって, Multiple-Reader に対する FIFO はキャッシングしない. 2 次キャッシュでは One-Reader の FIFO のみをサポートする.

FIFO のサポートの基本的な方法は, メモリ管理コプロセッサがメモリ上で管理する FIFO トップの数エントリをプリフェッチすることである.

実効バンド幅を向上させるため, メモリ—2 次キャッシュ間では FIFO のエントリはブロックにパックして転送する. 転送されるブロック内の有効なエントリ数は full/empty bit を利用して通知する. 2 次キャッシュ・コントローラは転送されてきたブロックを 2 次キャッシュにキャッシュする.

Example: 例えば, FIFO へのリード・アクセスが 1 次キャッシュ・ミスを起こした時, その FIFO には 2 つのエントリ=ワードがあったとしよう. コプロセッサはその 2 ワードを full 状態

にして1ブロックの先頭からつめ，ブロック内の残りのワードをempty として，2次キャッシュに転送する．

同一のFIFO に対する複数個の Top/Deq コマンドの1つ1つが2次キャッシュ・コントローラに対して可視である必要がある．したがって，FIFO アクセスのコマンドはプロセッサの1次キャッシュには Uncacheable でなければならない．

2次キャッシュ・コントローラは1つ1つのリード・アクセスに対して，2次キャッシュ上のブロックからエントリを1つずつ取り出して，返す．

4.3 パケット FIFO

4.2.2 項で述べたように，ワード単位の FIFO ではシステムの各部でワード単位の転送を行わなければならない．

そこで，トラフィックの増加を避け，各部の実効バンド幅を向上させるため，ワード単位ではなく数ワードからなるパケットを単位に FIFO を構成し転送するアクセスをサポートする [10]．

パケットのサイズはプロセッサの1次キャッシュ・ブロックのサイズと等しい．このアクセスは1次キャッシュまで Cacheable にできる．

4.3.1 パケット FIFO の構成

1つのパケット FIFO は，FIFO の実体であるリング・バッファ，1以上のライト・ポート，1つのリード・ポート，そして，各ポートに付随するパケット・フレームからなる．

すべての要素はプロセッサの1次キャッシュ・ブロックを単位として構成される．

リング・バッファ

FIFO の実体はリード・ポートのあるクラスタのメモリ上にアロケートされたリング・バッファである．

リング・バッファの管理はやはりメモリ管理コプロセッサが行う．

ライト・ポート/リード・ポート

ワード単位 FIFO では，Enqueu と Dequeu は同一物理アドレスに対して行われた．しかし，パケット FIFO では Enqueu/Dequeu はそれぞれ異なる物理アドレスに対して行う．Enqueu/Dequeu を行う対象となる物理メモリ上のブロックをそれぞれライト/リード・ポートと呼ぶ．ライト・ポートは2ブロック，リード・ポートは1ブロックからなる．

1つの FIFO に対し，ライト・ポートは複数あってよいが，リード・ポートは1つでなければならない．パケット FIFO は Multiple-Reader をサポートしない．

各ポートには以下に示す管理情報がセーブされる．

表 7: パケット単位 FIFO のコマンド (一部)

Pack	パケット・フレームにデータを Store する．
Unpk	パケット・フレーム内のデータを Load する．
Enq-P	FIFO の後尾にパケット・フレームの内容を追加する．
Deq-P	パケット・フレームの内容を次のパケットに置き換える．

ライト・ポート

- 生成中のパケット (詳細は後述)
- リード・ポートへのポインタ

リード・ポート

- リング・バッファの管理情報

パケット・フレーム

各ポートには，対応する仮想的なブロックが存在する．このブロックをパケット・フレームと呼ぶ．

パケット FIFO におけるデータの転送はパケット・フレームを通じて行われる．

パケット・フレームは対応するポートへのコマンドによりアクセスされる．

4.3.2 パケット FIFO アクセス

4.3.2.1 パケット FIFO アクセスのコマンド

パケット FIFO は表7 に示すコマンドで操作する．Unpk コマンドには表5 に示した5種のバリエーションがある．

パケット FIFO によるデータの授受はパケット・フレームを介して行われる．ライト側はパケット・フレーム上にパケットを生成し，Enqueu する．リード側はパケット・フレーム上に FIFO の先頭パケットを展開し，パケット・フレームに対してリードを行う．

このようにパケット FIFO では，1つ1つの Store/Load を Enqueue/Dequeu とみなさない．あるパケットへのアクセスが完了し次のパケットへのアクセスを開始する時には，Enq-P/Deq-P コマンドにより明示的に指示する．

NOTE: 前項で，ライト・ポートとリード・ポートは異なる物理アドレスを持つと述べた．それは以下の理由による．

パケットの生成は複数回の Pack コマンドからなり，不可分には行えない．Enqueue が同一のアドレスに対して行われるとすると，1つの FIFO に対する複数の並列アクティビティの Pack コマンドがマージされてしまう可能性がある．したがって，1つの FIFO には複数のパケット・フレームと複数のライト・ポートが必要であり，ライト・ポートとリード・ポートは同一アドレスにできない．

4.3.2.2 パケット FIFO の操作

パケット FIFO による通信処理は以下のように行われる．

ライト側の操作

Pack コマンドにより，Enqueu すべきパケットをパケット・フレーム上に生成する．

パケットが生成されたら，Enq-P コマンドを発行する．パケット・フレームの内容は Enqueu される．

リード側の操作

リード・ポートのパケット・フレーム上には FIFO の先頭のパケットが (FIFO が空でなければ) 常に存在している．パケット・フレームの内容は Unpk コマンドで参照する．

当該パケットへの参照が完了したら，Deq-P コマンドを発行する．パケット・フレームの内容は次のパケットに置き代わる．

4.3.3 パケット FIFO の実装

4.3.3.1 パケット FIFO のハードウェア

2 次キャッシュ・コントローラは内部にパケット・フレーム上のパケットをキャッシュする極小容量のバッファを 2 種類持つ．1 つはライト・バッファであり，もう一方はプリフェッチ・バッファである．これらのバッファによりアクセス・レイテンシの短縮を図る．

ライト・バッファ Pack コマンドによって Store されるデータはこのバッファにキャッシュされる．

プロセッサが備えるようなマージ機能付ライト・バッファ[5] と同様のハードウェアである．ただし，Enq-P コマンドが発行されるまで，内容を送出しない．リプレース時には，バッファの内容はライト・ポート内のセーブ領域にセーブされる．

プリフェッチ・バッファ リード時のレイテンシを隠蔽するために，パケット・フレーム上にあるパケットの次のパケットをプリフェッチしておくためのバッファである．

4.3.3.2 パケット FIFO のハードウェアの動作

パケット FIFO のハードウェアは以下のように動作する．

ライト側の動作

Pack コマンドによって Store されるデータはライト・バッファに Store-in される．

そして，当該パケットへの Enq-P コマンドによって，ライト・バッファの内容はリード側へ転送され，メモリ上のリング・バッファの後尾に書き込まれる．パケットの転送処理，リング・バッファへの追加処理はメモリ管理コプロセッサが行う．

リード側の動作

Unpk コマンドの 1 次キャッシュ・ミスにより，先頭のパケットが (メモリ上から) 1 次キャッシュまでキャッシュされる．と同時にその次のパケットがプリフェッチ・バッファにプリフェッチされる．

Deq-P コマンドで 1 次キャッシュ上のコピーは Invalidate される．先頭パケットは廃棄され，既にプリフェッチ・バッファにプリフェッチしてあったパケットが新しく先頭になる．

次回の Unpk コマンドのミスヒットでは，パケットはこのバッファから 1 次キャッシュに転送される[1]．

5 おわりに

共有メモリとメッセージ・パッシングの両方の通信モデルをサポートする高機能キャッシュ・システムを提案した．

今後は，実際に設計／製作し実現可能性を検証するとともに，完成した実機上での定量評価を行う予定である．

謝辞

日頃から御指導御助言頂く富田研究室の諸氏，特に池田泰敏，神嶌潔 両君に感謝する．

並列計算機 AP-1000 の実行環境を御提供頂いた (株) 富士通研究所に感謝する．

本研究の一部は文部省科学研究費 (重点領域研究 (1)「超並列ハードウェア・アーキテクチャの研究」) による．

参考文献

[1] 文部省重点領域研究「超並列原理に基づく情報処理基本体系」シンポジウム予稿集 (1992), (in Japanese).

[2] 文部省重点領域研究「超並列原理に基づく情報処理基本体系」第 2 回シンポジウム予稿集 (1993), (in Japanese).

[3] Matsumoto, T. and Hiraki, K.: Distributed Shared-Memory Architecture Using Memory-Based Processor, in *JSPP '93*, pp. 245–252 (1993), (in Japanese).

[4] Yang, Y. and Amano, H.: An interconnection network for massively parallel computers, Technical Report 92–ARC-96, IPS Japan (1992), (in Japanese).

[5] Degital Equipment Corp., : *DECchip^{TM} 21064 Microprocessor Hardware Reference Manual* (1992).

[6] Texsus Instruments Inc., : *SuperSPARC User's Guide* (1992).

[7] Mori, S., Omori, Y., Nakashima, H. and Tomita, S.: Proposal on Self Cleanup Write-Back Cache, Technical Report 93–ARC–100, IPS Japan (1993), (in Japanese).

[8] Matsumoto, T. and Hiraki, K.: A Shared-Memory Architecture for Massively Parallel Systems, Technical Report CPSY92, IEICE (1992), (in Japanese).

[9] Goshima, M.: Inter-Processor Communication, Internal Report (1992).

[10] Goshima, M., Mori, S. and Tomita, S.: The On-Memory FIFO Mechanism for Inter-Processor Communication, Technical Report 92–OS–56, IPS Japan (1992), (in Japanese).

[1] プロセッサの 1 次キャッシュが Update プロトコルをサポートしていれば，1 次キャッシュ上のパケット・フレームを次のパケットに Update する．