計算機アーキテクチャ 79－3
（1989. 11. 20）

可変構造型並列計算機のキャッシュ・システム

岩田英次　森 眞一郎　村上和彰　福田 晃　富田眞治
（九州大学大学院総合理工学研究科）

『可変構造型並列計算機（Reconfigurable Parallel Processor）』と呼ぶ汎用／多目的の高速中規模マルチプロセッサ・システムを開発している．本システムは，128台のプロセッシング・エレメント（PE）を128×128のクロスバー網で相互結合したMIMD型のマルチマイクロプロセッサ・システムである．

本システムの物理的なメモリ構成自体は，各PEにローカルメモリ（容量4MB）を持たせる“分散メモリ構成”となっており，全PEにとって均等な距離に位置するグローバルメモリは存在しない．しかしながら，各ローカルメモリを自PEに単に専有させるのではなく，他PEからもクロスバー網を経由してアドレスにより直接アクセス可能とする“ローカル／リモート・アーキテクチャ（分散共有メモリ構成）”を採用している．

さらに，各PEはプライベート・キャッシュを有する．よって，“キャッシュ－メモリ間”の一貫性を保証しなければならない．さらに，キャッシュには，ローカルメモリのみならずリモートメモリのコピーを持つことを許す．したがって，あるメモリのコピーが任意のPEのキャッシュに存在し得ることから，“キャッシュ－キャッシュ間”の一貫性を保証しなければならない．また，本システムは仮想記憶を採用しており，キャッシュは仮想アドレス・キャッシュとして動作する．したがって，多義語問題および同義語問題が生じることから，“キャッシュ内”の一貫性もまた同様に保証する必要がある．

本稿では，本キャッシュ・システムの設計方針，その構成および動作，コヒーレンス処理，性能について述べる．

Cache System of the Kyushu University Reconfigurable Parallel Processor
(in Japanese)

Eiji IWATA, Shin-ichiro MORI, Kazuaki MURAKAMI, Akira FUKUDA, and Shinji TOMITA
Department of Information Systems
Interdisciplinary Graduate School of Engineering Sciences
Kyushu University
6-1, Kasuga-koen, Kasuga-shi, Fukuoka, 816 Japan
e-mail : iwata@kyu-is.is.kyushu-u.ac.jp

The reconfigurable parallel processor system has been developed at Kyushu University. It is an MIMD-type multiprocessor which consists of 128 processing-elements (PEs) fully connected by a 128×128 crossbar network.

The system employs a distributed memory organization by providing 4M bytes of local memory in each PE, and therefore there is no global memory situated at a location equally accessible to all processors. To exploit the memory reconfigurability, however, the system allows more than one processor to share a local memory. A processor of each PE can directly access the local memories of the other PEs, which memories are called "remote memories", via the crossbar network. The memory architecture is referred to as "local／remote architecture".

Furthermore, there is a private cache in each PE, so it is necessary to maintain coherence between cache and main-memory. In addition, the system allows a cache to hold copies of data from remote-memory, as well as copies of data from local-memory. Therefore, such copies can be placed at any cache in different PEs. Then, "inter-cache coherency", must be ensured. In the other hand, as our system supports the virtual memory and adopts the virtual address cache, we must maintain also "inner-cache coherency" to solve homonym and synonym problems.

In this paper, we describe the design philosophy and hardware mechanism of our cache system, and a protocol for cache coherence control.

1. はじめに

我々は，実行すべき並列アルゴリズムの構造に対して，柔軟に計算機構成を適応させる『可変構造型並列計算機（Reconfigurable Parallel Processor）』の開発を進めている．[1]～[3] 本システムは，128台のプロセッシング・エレメント（PE）を128×128のクロスバー網で相互結合したMIMD型のマルチマイクロプロセッサ・システムであり，相互結合網およびメモリを次のように可変構造化した“ダイナミック・アーキテクチャ”を採用している．[1],[2]

① 可変構造型ネットワーク・アーキテクチャ：プロセッサープロセッサ結合およびプロセッサ－メモリ結合として任意の結合形態を提供する．

② 可変構造型メモリ・アーキテクチャ：任意のPE間でのメモリの共有／非共有を可能とする．

本システムの物理的なメモリ構成自体は，各PEにローカルメモリ（容量4MB）を持たせる“分散メモリ構成”となっており，全PEにとって均等な距離に位置するグローバルメモリは存在しない．しかしながら，各ローカルメモリを自PEに単に専有させるのではなく，他PEからもクロスバー網を経由してアドレスにより直接アクセス可能とする“ローカル／リモート・アーキテクチャ（分散共有メモリ構成）”を採用している．[3] あるPEのプロセッサから見たとき，クロスバー網を経由してアドレスにより直接アクセス可能な他PEのローカルメモリを“リモートメモリ”と呼ぶ．

さらに，各PEにはプライベート・キャッシュを設ける．[4],[5] よって，“キャッシュ－メモリ間”の一貫性（coherence）を保証しなければならない．さらに，キャッシュには，ローカルメモリのみならずリモートメモリのコピーを持つことを許す．したがって，あるメモリのコピーが任意のPEのキャッシュに存在し得ることから，“キャッシュ－キャッシュ間”の一貫性（multicache consistency）を保証しなければならない．[6],[7] また，本システムは仮想記憶を採用しており，キャッシュは仮想アドレス・キャッシュとして動作する．したがって，多義語問題および同義語（synonym）問題が生じることから，“キャッシュ内”の一貫性もまた同様に保証する必要がある．[5]

本稿では，まずキャッシュ・システムの設計方針をまとめたあと，その構成および動作，コヒーレンス処理，性能について述べる．

2. 設計方針

可変構造型並列計算機をそのキャッシュ・システムに着目した場合の構成を図1に示す．PEに設けるマイクロプロセッサとして，RISCプロセッサのSPARCを用いる．RISCプロセッサの性能を最大限に引き出すためには，高速大容量のキャッシュは不可欠である．表1にキャッシュの諸元を示す．

キャッシュの設計にあたっては，次の項目について検討を行い，以下に述べる設計方針をたてた．[8]

① アクセス・アドレス：仮想アドレス・キャッシュvs実アドレス・キャッシュ

② 連想度：ダイレクト・マッピングvsセット・アソシアティブ

③ ブロック・サイズ

④ キャッシュ－メモリ間コヒーレンス（メモリ更新アルゴリズム）

⑤ キャッシュ－キャッシュ間コヒーレンス

⑥ キャッシュ内コヒーレンス

2.1 アクセス・アドレスおよび連想度

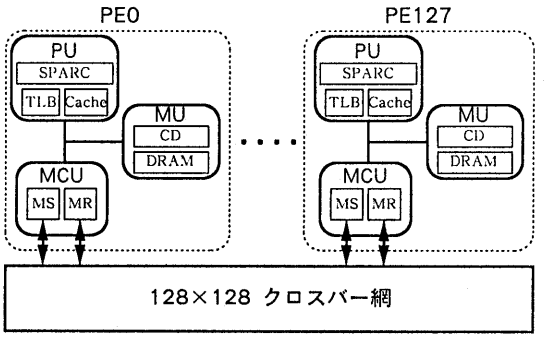
仮想記憶を採用した場合，キャッシュへのアクセスを仮想アドレスで行うか（仮想アドレス・キャッシュ），あるいは，実アドレスで行うか（実アドレス・キャッシュ）が問題となる．[10] この選択肢により，次の点に影響を与える．

① キャッシュ・アクセス時間

② キャッシュ容量

③ ハードウェア・コスト

④ ヒット率



PU：プロセッサ・ユニット　CD：コヒーレンス・ディレクトリ
MU：メモリ・ユニット　MS：メッセージ・センダ
MCU：メッセージ通信ユニット　MR：メッセージ・レシーバ

図1. 可変構造型並列計算機のキャッシュ・システム

まず，③については，実アドレス・キャッシュの場合，キャッシュ・アクセスに際し仮想アドレスから実アドレスへのアドレス変換が必要なため，アドレス変換バッファ（TLB）の装備が不可欠となる．また，④に関しては，TLBヒットがキャッシュ・ヒットのための前提条件となるため，キャッシュ・ヒット率はTLBヒット率に依存する．

さらに，上記の4点は，キャッシュの連想度（ウェイ数）を1とするか（ダイレクト・マッピング方式），あるいは，2以上とするか（セット・アソシアティブ方式）により，以下のように影響を受ける（表2参照）．

a）仮想－ダイレクト：最もアクセス時間が短く，かつ，セット数を増やすことで容量を容易に大きくできる．さらに，ハードウェア・コストも最も小さい．

b）仮想－セット：方式aに比べて，ウェイ選択時間（T_{mpx}）分だけアクセス時間が長くなる．容量を大きくするには，方式a同様にセット数を増やす場合は容易だが，ウェイ数を増やす場合はそれだけハードウェア・コストが大きくなる．

c）実－ダイレクト－パラレル：アクセス時間は方式aとほぼ同程度である．容量については，セット数が仮想記憶のページサイズ（本システムでは4KB）により制限されるため，ページサイズ以上には大きくできない．

d）実－ダイレクト－シリアル：方式cと異なり，容量は仮想記

表1. キャッシュの諸元

		初期仕様	現仕様
キャッシュ	アクセス・アドレス	仮想アドレス／実アドレス	仮想アドレス
	Unified／Split	Unified（単一キャッシュ）	
	連想度	2	1 (Direct Mapping)
	メモリ更新アルゴリズム	ストア・スルー	
	リプレースメント・アルゴリズム	LRU	
	アドレス・ブロック・サイズ	32B	32B／64B
	トランスファ・ブロック・サイズ	8B	32B／64B
	コヒーレンス・ブロック・サイズ	32B	32B／64B
	セット数	1024／128	2048
	キャッシュ・サイズ	64KB／8KB	64KB／128KB
RTB	連想度	2	4
	リプレースメント・アルゴリズム	LRU	
	セット数	1024	512

表2. アクセス・アドレスおよび連想度に着目したキャッシュの諸方式の比較

方式	アクセス・アドレス	連想度	TLBアクセス	キャッシュ・アクセス時間	容量	ハードウェア・コスト
A	仮想アドレス	1（DM方式）		$T_{tag}+T_{comp}$	S×B	H_c
B		2以上（SA方式）		$T_{tag}+T_{comp}+T_{mpx}$	S×B×W	$H_c \times W$
C	実アドレス	1（DM方式）	パラレル	$\max\{T_{tag}, T_{TLB}\}+T_{comp}$	P	H_c+H_{TLB}
D			シリアル	$T_{TLB}+T_{tag}+T_{comp}$	S×B	$H_c \times W+H_{TLB}$
E		2以上（SA方式）	パラレル	$\max\{T_{tag}, T_{TLB}\}+T_{comp}+T_{mpx}$	P×W	$H_c \times W+H_{TLB}$

DM方式：ダイレクト・マッピング方式
SA方式：セット・アソシアティブ方式
H_c：キャッシュの1ウェイ当りのハードウェア・コスト
H_{TLB}：TLBのハードウェア・コスト

T_{tag}：タグ・アクセスに要する時間　S：セット数
T_{comp}：比較に要する時間　B：ブロック・サイズ
T_{mpx}：ウェイ選択に要する時間　W：ウェイ数
T_{TLB}：TLBアクセスに要する時間　P：ページ・サイズ

憶のページサイズに限定されないが，TLBアクセスとキャッシュ・アクセスとがシリアルに行われるため，アクセス時間が方式aのほぼ2倍程度と長くなる．

e）実－セット－パラレル：アクセス時間は方式bとほぼ同程度である．容量を大きくするには，方式c同様セット数が仮想記憶のページサイズにより制限されるため，ウェイ数を増やすしか方法がない．

これら5方式の選択肢の中で，まず方式cではキャッシュ容量が4KBしかなく不十分であり，また，方式dではSPARC（マシンサイクル60ns）に対してno-waitで命令およびオペランドを供給することが不可能である．残りの3方式a，b，eに関しては，アクセス時間はほぼ同程度であり，同じ容量であれば連想度が増すにつれてヒット率は一般に向上する．

したがって，初期仕様では，2ウェイ・セット・アソシアティブ方式を採用し，仮想アドレス・キャッシュ（方式b）および実アドレス・キャッシュ（方式e）のいずれでも動作可能な2モード・キャッシュとしていた．[5] しかし，大容量（本システムでは64KB／128KB）のキャッシュにおいては連想度のヒット率への影響は小さいことが報告されており，[12],[13] また，セット・アソシアティブ方式にしたことによるハードウェア・コスト増加はかなり大きくなる．よって，現仕様ではダイレクト・マッピング方式の仮想アドレス・キャッシュ（方式a）としている．

2.2 ブロック・サイズ

キャッシュ管理の単位となるブロックとしては，以下の3種類がある．[14]

① アドレス・ブロック（AB；ライン）：タグ・アレイにおいて各アドレスタグに対応する単位
② トランスファ・ブロック（TB）：キャッシュ－メモリ間またはキャッシュ－キャッシュ間における転送単位．
③ コヒーレンス・ブロック（CB）：キャッシュ－メモリ間およびキャッシュ－キャッシュ間の一貫性を保証する単位

これらのブロック間の大小関係は任意に定められる．初期仕様においては，

TB（8B）＜AB（32B）＝CB（32B）

としていた．これにより，以下の効果をねらった．[5]

i）TB＜ABとすることで，リモート・メモリからのクロスバー網を経由するライン・フェッチにおいて，TBの未着を許す（回線中途切断への対処）．その結果1個のAB内部でTBの不在が起こり得て，次のライン・フェッチ時には当該TBのみを転送することで転送時間を短縮する．
ii）AB＝CBとすることで，コヒーレンス処理の際のタグ・アレイへのアクセス回数を1回のみとする．

しかし，TB＜ABとした場合，TB対応に存在ビットが必要となり，さらに，キャッシュ・ミス時の処理が煩雑となる．よって，現仕様では，一般的なキャッシュと同様，

TB＝AB＝CB

としている．

このとき，ブロック・サイズの大小によるヒット率の変動はプログラム参照パターンに依存し一概に特定できないが，16～64B程度が望ましいとの報告がある．[11] そこで，3.1節で述べるように，ブロック・サイズとしては，32Bおよび64Bを選択可能としている．

2.3 キャッシュ－メモリ間コヒーレンス

キャッシュ－メモリ間コヒーレンス処理（メモリ更新アルゴリズム）としては，ストア・スルー方式およびストア・イン方式が一般に用いられているが，コヒーレンス処理方式を詳細に分類すると表3のようになる．

本来なら各方式について個別に検討を行う必要があるが，本稿ではキャッシュ－キャッシュ間コヒーレンス処理との関係にのみ着目して，これらを次の2つのタイプに分けて検討する．

① clean型：常にメモリに最新の更新結果が存在し，ライン・リプレース時にキャッシュ内のコピーをメモリに書き戻す必要がない．すなわち，コヒーレンス・ブロックの状態としてdirty状態が存在しない．
② dirty型：メモリに最新の更新結果が存在するとは限らず，ライン・リプレース時にキャッシュ内のコピーをメモリに書き戻す必要が生じ得る．すなわち，コヒーレンス・ブロックの状態としてdirty状態が存在する．

5.3節で詳述する検討の結果，現仕様ではclean型を採用し，メモリ更新アルゴリズムとしてストア・スルー方式を採用している．

2.4 キャッシュ－キャッシュ間コヒーレンス

キャッシュ－キャッシュ間コヒーレンス処理には，大別して次の2つのアプローチがある．

（1）静的コヒーレンス処理

表3. メモリ更新アルゴリズム

	ストア・ヒット		ストア・ミス		アルゴリズム
	自キャッシュ	メモリ	自キャッシュ	メモリ	
clean型	Update	Update	No	Update	ストア・スルー
			A&U	Update	SANF
			F&U	Update	
dirty型		No	No	Update	
			A&U	No	
				Update	
			F&U	No	ストア・イン
				Save	
				Update	ライト・ワンス

No：何も行わない　U（Update）：ストア・データで更新
A（Allocate）：ライン作成　F（Fetch）：ライン・フェッチ
Save：更新前のラインをメモリに書き戻す

データの属性（非共有，read-only共有，read-write共有）をユーザが明示的に宣言するか，あるいは，コンパイラがデータ依存解析により自動判定することで，データのキャッシング可／不可を“静的”に決定する．この情報に基づいて，

① ページテーブル・エントリに，仮想ページ単位でキャッシング可／不可制御情報を持たせる．

② メモリにディレクトリを設け，メモリ・ブロック単位にキャッシング可／不可を指定する．

③ オブジェクト・プログラムにキャッシュ制御コードを挿入する．

などの手法が用いられる．

(2) 動的コヒーレンス処理

複数キャッシュ間の一貫性を保証するために，コヒーレンス・ブロックの状態遷移制御をハードウェアにより“動的”に行う．一般に“キャッシュ・コヒーレンス処理”と言った場合これを指す．これには，状態遷移要求をどこが発行するかにより，さらに次の2方式に分類される．

① キャッシュ側発行：各々のキャッシュが他キャッシュのメモリ・トランザクションを常に監視しており，自キャッシュ内のブロック状態に影響を及ぼすトランザクションを検出すると，当該ブロックの状態遷移を行う．スヌーピング・キャッシュ方式がこれに相当する．[15]

② メモリ側発行：各メモリ・ブロックのコピーがどのキャッシュに存在しているかを登録するディレクトリをメモリに設け，メモリ・トランザクションにより状態遷移の必要なブロックを有するキャッシュに対し状態遷移要求を発行する．グローバル・ディレクトリ方式がこれに相当する．[18]

以上の処理方式と本システムとの親和性について検討を行った結果，[7] 以下の方針を採ることとした．

i) 本システムのように100台を越えるようなマルチプロセッサ・システムでは，動的コヒーレンス処理にのみ頼った場合，そのオーバヘッドによる性能低下は無視できない．そこで，静的コヒーレンス処理の考え方を全面的に取り入れる．方式としては，上記の①～③の全てを可能としている．

ii) 動的コヒーレンス処理としては方式②を採り，グローバル・ディレクトリを各ローカル・メモリに分散させる“分散グローバル・ディレクトリ方式”とする．これは，方式①の場合メモリ・トランザクションを全PEにブロードキャストする必要があり，クロスバー網という相互結合網の性質上，ブロードキャストに伴う通信オーバヘッドにとても耐えられないからである．

2.5 キャッシュ内コヒーレンス

仮想アドレス・キャッシュを用いる場合，以下の2つの問題を解決する必要がある．

(1) 多義語問題

多重仮想空間環境下では，同じ仮想アドレスでも仮想空間が異なっていれば一般に異なる実アドレスと対応する（多義語）．よって，仮想アドレス・キャッシュでは，多義語の仮想アドレスにより誤ってヒットすることが起こる．これを防ぐには，

① 仮想空間スイッチ時にキャッシュをフラッシュする．

② タグに仮想空間識別子を持たせ，これで仮想空間を識別する．

の2つの方法がある．

方法①は，小容量キャッシュ，あるいは，空間スイッチの間隔が充分長い場合に有効である．しかし，本システムのような大容量キャッシュにおいて，空間スイッチが頻繁に起きる場合，そのヒット率の低下は無視できない．

そこで，①と②の両方法を採用している．すなわち，キャッシュのフラッシュコマンドを用意するとともに，8ビットの仮想空間識別子（空間ID）をタグに含め256空間まで識別可能としている．空間IDの管理はOSが行い，また，空間スイッチ時にキャッシュをフラッシュするか否かもOS次第である．

(2) 同義語問題

あるデータを共有する場合，異なる仮想アドレスが同じ実アドレスに対応する（同義語）ことがある．このとき仮想アドレス・キャッシュでは，当該共有データのコピーがキャッシュ内に複数存在し得るので，これらコピー間の一貫性を保証する必要がある．

この同義語問題は，2.3節で述べたキャッシュ－キャッシュ間コヒーレンスと類似している．すなわち，後者が複数キャッシュに存在する同一メモリ・ブロックのコピー間の一貫性を保証するのに対し，前者は1キャッシュ内に存在する同一メモリ・ブロックの複数コピー間の一貫性を保証する．

同義語問題の解決策としては，

① 同義語が生じないよう，データの共有形態に制限を設ける．すなわち，1仮想空間内でのデータ共有を禁止する．また，異なる仮想空間間でデータを共有する場合，その仮想アドレスは同一にする（異なる場合，空間スイッチ時にキャッシュをフラッシュ）．

② 実アドレスを仮想アドレスに変換する逆変換バッファ（RTB）を設ける．これにより，実アドレスによっても仮想アドレス・キャッシュへのアクセスを可能とし，対応するメモリ・ブロックのコピー間の一貫性を保証する．

の2つの方法がある．[10]

5.2節で述べるように，動的キャッシュ－キャッシュ間コヒーレンス処理においても，実アドレスから仮想アドレスへの逆変換が必要なためRTBは不可欠である．したがって，同義語問題の解決策としても，RTBを用いた方法②を採ることができる．[14] よって，方法②を採用する．これに基づくキャッシュ内コヒーレンス処理は4章で述べる．

なお，方法①は純粋にOSの構成および制御に依存しており，方法①を採るか否かはOS次第である．

3. キャッシュの構成と動作

3.1 デュアル・ディレクトリ

キャッシュの構成を図2に示す．2.1節で述べたようにダイレクト・マッピング方式の仮想アドレス・キャッシュであるが，コヒーレンス処理の際に実アドレスでもアクセス可能なように，次の2つのディレクトリを備えたデュアル・ディレクトリ・キャッシュ[14]構成となっている．

① Vタグ・アレイ：仮想アドレス・タグ等を保持するディレクトリ

② Rタグ・アレイ：実アドレス・タグ等を保持するディレクトリ（RTB）

また，2.2節で述べたように，ブロック・サイズとして32Bおよび64Bのいずれかが選択可能であるが，説明を簡単にするため以下32Bの場合について述べる．

(1) Vタグ・アレイ

セット数2048のダイレクト・マッピング方式のディレクトリである．通常は32ビットの仮想アドレス（VA31－0）中の11ビットのセット・アドレス（VA15－5）でアクセスされる．

Vタグ・アレイの各エントリ（Vタグ）は，以下の4つのフィールドからなる．

① 仮想アドレス・タグ：アドレス比較用の上位仮想アドレス（VA31－16）およびライン有効（V）ビット．

② 空間ID：8ビットの仮想空間識別子で，ディスパッチされている仮想空間の番号（0～255）との比較に用いる．

③ Rタグ・ポインタ（RTP）：当該仮想アドレス・タグと対応関係にある実アドレス・タグを登録しているRタグへのポインタ（RA13－12およびウェイ番号）．

④ 記憶保護：ページテーブル・エントリの記憶保護フィールドのコピーで，読出し権および実行権のレベル（スーパバイザ／ユーザ）検査に用いる．

(2) Rタグ・アレイ

セット数512，ウェイ数4のセット・アソシアティブ方式のディレクトリであり，逆変換バッファ（RTB）として動作する．通常は32ビットの実アドレス（RA31－0）中の9ビットのセット・アドレス（RA13－5）でアクセスされる．Rタグ・アレイのエントリ数はVタグ・アレイ同様2048であるが，その連想度は，

3.5節で述べる“衝突（collision）”によるヒット率低下の影響を軽減するため4ウェイと増やしている．

Rタグ・アレイの各エントリ（Rタグ）は，以下の2つのフィールドからなる．

① 実アドレス・タグ：アドレス比較用の上位実アドレス（RA31－14）およびエントリ有効（V）ビット．

② Vタグ・ポインタ（VTP）：当該実アドレス・タグと対応関係にある仮想アドレス・タグを登録しているVタグへのポインタ（VA15－12）．

3.2 データ・アレイ

幅4B，深さ32K，容量128KBのSRAMである．ブロック・サイズが32Bの場合は，半分の64KBのみ使用する．選択されたブロック・サイズにより，アクセス・アドレスは次のように定まる．

① 32Bのとき：VA15－2（14ビット）

② 64Bのとき：VA16－2（15ビット）

本キャッシュは単一（unified）キャッシュであり，命令／オペランド，および，スーパバイザ／ユーザの区別を行うことなく，いずれのメモリ・ブロックのコピーも本データ・アレイに存在し得る．また，非共有キャッシュであり，データ・アレイへのアクセス要求はSPARCプロセッサのみ行える．

3.3 通常動作

本キャッシュへのアクセス要求元には，次の2つがある．

① SPARCプロセッサ：32ビットの仮想アドレスでVタグ・アレイおよびデータ・アレイにアクセスし，命令フェッチおよびオペランド・ロード／ストアを要求する．

② MCU（メッセージ通信ユニット）：キャッシュ－キャッシュ間コヒーレンス処理の一環として，32ビットの実アドレスでRタグ・アレイにアクセスし，コヒーレンス・ブロックの無効化を要求する．

上記②の場合の動作については，5.2節で述べる．ここでは，①の場合の通常動作について述べる．SPARCプロセッサからのアクセス要求に対する応答として，以下の3種類がある．

① 読出しヒット：命令フェッチおよびオペランド・ロード時にヒットした場合，直ちに当該データをSPARCに返す．

② 読出しミス：命令フェッチおよびオペランド・ロード時にミスヒットした場合，ライン・フェッチ要求をバスに送出し，ライン・リプレース処理に入る．ライン・フェッチに必要な実アドレスは，TLBから得る．また，ライン・リプレース処理の一環として，キャッシュ内コヒーレンス処理（4章参照）を行う．当該ライン・フェッチが完了するまで他のアクセス要求は受け付けない（ブロッキング・キャッシュ方式）．

③ 書込み：オペランド・ストアの場合はヒット／ミスヒットに関わらず，メモリ書込み要求をバスに送出する（ストア・スルー方式）．実アドレスは，TLBから得る．また，読出しミス時と同様，キャッシュ内コヒーレンス処理（4章参照）を行う．

3.4 無効化機能

プログラムがキャッシュ全体あるいは一部を無効化するための手段として，以下の3種類のコマンドを用意する．

① フラッシュ・コマンド：Vタグ・アレイおよびRタグ・アレイ内のすべてのタグを無効化する．

② Vタグ・パージ・コマンド：Vタグ・アレイに仮想アドレス（および空間ID）を与えてヒットした場合，当該Vタグ，および，それとリンクしているRタグを無効化する．

③ Rタグ・パージ・コマンド：Rタグ・アレイに実アドレスを与えてヒットした場合，当該Rタグ，および，それとリンクしているVタグを無効化する．

3.5 Vタグ－Rタグ間ダブルリンク

デュアル・ディレクトリ・キャッシュにおいては，有効なVタグとRタグ同士は互いにリンク（doubly linked）していなければならない（図3（a）参照）．[14] リンクは，セット・アドレスのページ・オフセット部（A11－5：仮想アドレスと実アドレスとの共通部分）とタグ・ポインタ（VTPおよびRTP）とを次のように連結したものである．

① Rタグへのリンク：RA13－12｜A11－5およびウェイ番号

② Vタグへのリンク：VA15－12｜A11－5

キャッシュ内コヒーレンス処理およびキャッシュ－キャッシュ間コヒーレンス処理において，これらのリンクを用いてRタグ・アレイないしVタグ・アレイをアクセスする．

さて，有効なVタグ－Rタグ間にダブルリンクが必要であることから，Rタグ・アレイの連想度次第では，ライン・リプレース時にRタグ間の“衝突”が起きる可能性がある．[14] この様子を図3（b）に示す．

① まず，VタグaとRタグaの組が有効である．ここで，仮想アドレスbによりミスヒットを起こし，ライン・リプレース処理に入る．

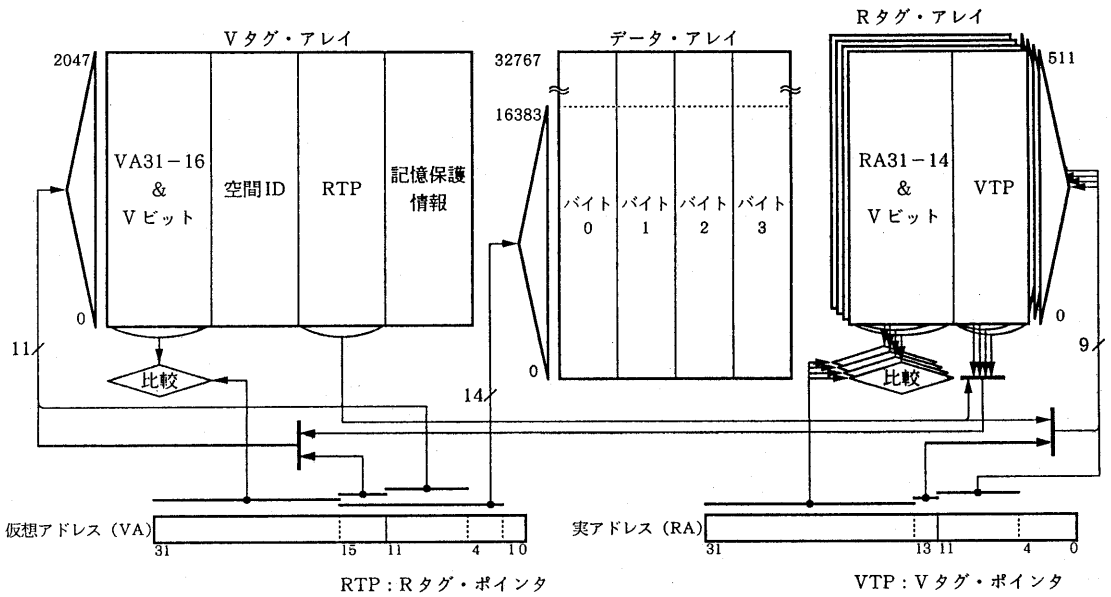


図2. キャッシュの構成（ブロック・サイズが32Bの場合）

② ライン・リプレースに伴い，Vタグ・アレイおよびRタグ・アレイに，VタグbおよびRタグbをそれぞれ登録しなければならない．このとき，Rタグbを登録すべきエントリに，すでにRタグaが登録されていた．すなわち，RタグaとRタグbが衝突した．

③ Rタグbを登録するためには，Vタグa－Rタグa間のダブルリンクを解く必要がある．その結果，もともと有効であったVタグaが無効となる．つまり，Vタグaは，ライン・リプレースによって生じた“犠牲者（victim）”といえる．

④ ライン・リプレース処理が完了したとき，Vタグb－Rタグb間にダブルリンクが張られ，これらの組が有効となる．

この衝突現象は，Rタグ・アレイの連想度をあるウェイ数以上にすれば生じない．そのウェイ数は，キャッシュ容量とページ・サイズの比で求まる．すなわち，本キャッシュの場合，キャッシュ容量64KB（ブロック・サイズ32Bのとき）およびページ・サイズ4KBであるから，同一ページ・オフセットを持つVタグとRタグがそれぞれ16個ずつ存在し得る．よって，これらVタグ－Rタグ間で衝突を起こすことなくダブルリンクを張るためには，少なくとも16ウェイあればよいことになる．

しかし，3.1節で述べたように，本キャッシュのRタグ・アレイは4ウェイであり衝突は起こり得る．この衝突によるヒット率の低下については，6章で考察する．

4. キャッシュ内コヒーレンス処理

2.5節で述べた同義語問題をRタグ・アレイを用いて解決する手順について述べる．このキャッシュ内コヒーレンス処理は，3.3節で述べたように，ライン・リプレース時およびオペランド・ストア時（ミスヒット時）に行う．

① フェッチするラインの実アドレス，または，オペランド・ストア先の実アドレスをTLBから得る．この実アドレスを用いて，Rタグ・アレイをアクセスする．ミスヒットした場合，処理完了．

② ヒットした場合，Rタグ中のVTPを用いてVタグ・アレイをアクセスする．そして，当該Rタグからリンクされ ているVタグを無効化する．

5. キャッシュ－キャッシュ間コヒーレンス処理

5.1 コヒーレンス処理支援機構

2.4節で述べた各種の静的および動的コヒーレンス処理を支援するために，Rタグ・アレイに加えて以下の機構を用意する．

(1) ページ・テーブル

静的コヒーレンス処理の方式①（2.4節参照）を可能とするために，ページ・テーブル・エントリ中にCA（CacheAble）ビットを設けた．このビットにより，仮想記憶空間の各仮想ページ単位にキャッシング可／不可を指定できる．

(2) コヒーレンス・ディレクトリ（CD）

各ローカル・メモリに分散したグローバル・ディレクトリの各々をコヒーレンス・ディレクトリ（CD）と呼ぶ．CDは，一貫性を保証すべきメモリ・ブロック（CB：コヒーレンス・ブロック）のコピーがどのPEのキャッシュに存在するかを記録する．CDの各エントリは，コヒーレンス・ブロックに対応する．CDの構成にあたっては，以下の点を考慮した．

①CBサイズ：CDのエントリ数は，メモリ容量とCBサイズの比となる．CBを小さくすることできめ細かなコヒーレンス処理が可能だが，エントリ数の増加をもたらす．よって，2.2節で述べたように，現仕様ではライン・サイズと同じ大きさの32Bないし64Bとしており，エントリ数は128Kないし64Kとなっている．

②PE数：CDの1エントリに登録できるPE数で，キャッシュに存在し得るコピーの数が決まる．理想的には128台まで登録できるのが望ましいが，エントリのビット数の増加をもたらす．現仕様では1PEのみ登録可能とし，登録に必要なビット数を7ビットに抑えている．その結果，一時には1台のPEのキャッシュにしかコピーを許さないという“single copy”の制限がつく．

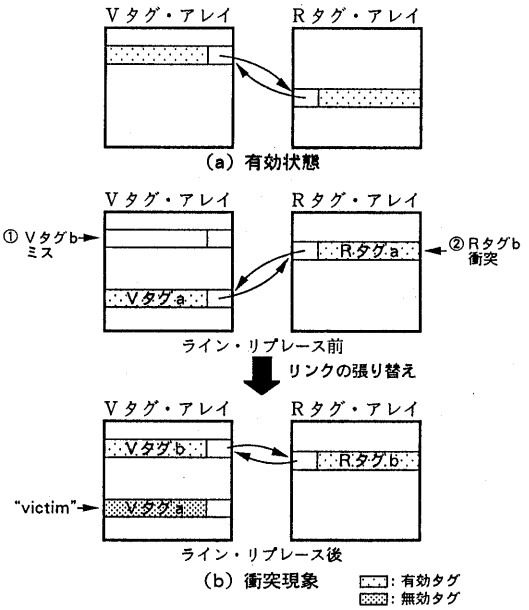


図3. Vタグ－Rタグ間リンク

CDエントリのフォーマットを以下に示す．

CC	CA	PE番号（7 ビット）

ここで，CC（Coherence Check）およびCA（CacheAble／CAched）ビットは次の意味を持つ．

①CC＝0，CA＝0：キャッシング禁止．

②CC＝0，CA＝1：キャッシング許可，動的コヒーレンス処理禁止（multiple copies許可状態）．

③CC＝1，CA＝0：キャッシング許可，動的コヒーレンス処理許可，どのPEにもコピーが存在しない（no copy状態）．

④CC＝1，CA＝1：キャッシング許可，動的コヒーレンス処理許可，登録されたPEにのみコピーが存在する（single copy状態）．

これらのビットは実行前ないし実行中の任意の時点でプログラムにより設定／変更可能であり，静的コヒーレンス処理の方式②（2.4節参照）を可能とする．

5.2 動的コヒーレンス処理

2.4節で述べたように，分散グローバル・ディレクトリ方式に基づく動的コヒーレンス処理を行う．5.3節で述する理由により，キャッシュ－メモリ間コヒーレンス処理（メモリ更新アルゴリズム）にはclean型（2.3節参照）のストア・スルー方式を用いる．また，コヒーレンスの取り方には，

①無効化型：キャッシュ内コピーを無効化する．

②更新型：キャッシュ内コピーを最新状態に更新する．

の2方式が可能だが，現仕様では無効型を採る．

動的コヒーレンス処理が必要となるのは，コヒーレンス・ディレクトリ（CD）エントリのCC＝1（動的コヒーレンス処理許可）であるコヒーレンス・ブロック（CB）に対して，次のいずれかのメモリ・トランザクションが発生したときである．以下，その手順を示す．

a) CA＝0（no copy状態）で，PEiからライン・フェッチ要求が到着した．このときは，

①CA＝1としてPEiをCDエントリに登録する．

②当該CBのコピーをPEiに返す．

b) CA＝1（single copy状態）で，登録されているPE（PEi）と

同じPE（PEi）からライン・フェッチ要求が到着した．このときは，直ちに当該CBのコピーをPEiに返す．

c）CA＝1（single copy状態）で，登録されているPE（PEi）とは異なるPE（PEj）からライン・フェッチ要求が到着した．このときは（図4（a）参照），

①PEiに対し当該CBの無効化を要求するメッセージを送出する．

②PEjをCDエントリに登録する．

③当該CBのコピーをPEjに送出する．

という手順をとり，当該CBのコピーが唯一であること（single copy）を保証する．

d）CA＝1（single copy状態）で，登録されているPE（PEi）とは異なるPE（PEj）からメモリ書込み要求が到着した．このときは，

①PEiに対し当該CBの無効化を要求するメッセージを送出する．

②CA＝0とする．

上記c，dにおいて，無効化要求メッセージを受け取ったPE（PEi）では，以下の処理を行う．

①メッセージ中に含まれている実アドレスを用いて，Rタグ・アレイをアクセスする．ミスヒットした場合，処理完了．

②ヒットした場合，Rタグ中のVTPを用いてVタグ・アレイをアクセスする．そして，当該Rタグからリンクされ ているVタグを無効化する．

5.3 clean vs dirty型動的コヒーレンス処理

2.3節で述べたように，キャッシュ－メモリ間コヒーレンス処理としては，clean型とdirty型の2種類のタイプがある．5.2節で述べた動的コヒーレンス処理はclean型を前提としている．ここでは，同一システム構成におけるdirty型を前提とした動的コヒーレンス処理を検討し，clean型と比較した場合の問題点を明らかにする．

以下，clean型とdirty型で処理が大きく異なる状況c（5.2節参照）におけるdirty型の動的コヒーレンス処理を考察する．このとき，要求されるCBのコピーを有するPEiにおいて，当該CBを更新している可能性がある．すなわち，当該CBの最新内容はPEiにのみ存在し（dirty），メモリには存在しないかも知れない．よって，ライン・フェッチ要求を行っているPEjには，PEiに存在する最新内容のコピーを転送しなければならない．この転送方法には，以下の2方式が考えられる．

（1）キャッシュ間直接転送方式

メモリを介さずにPEiからPEjへ直接転送する（図4（b）参照）．このときの手順は，次のようになる．

①メモリはPEjに対し，当該CBのコピーを持つPEの番号（i）を通知する．

②PEjはPEiに対し，コピーの転送を要求する．

③転送要求メッセージを受け取ったPEiは，コピーをPEjに転送する．

④転送が完了したら，PEjはその旨をメモリに通知する．

⑤PEjをCDエントリに登録する．

この方式では，PEjからの転送完了通知（④）が到着するまで，当該CBのコピーがPEiあるいはPEjのいずれに存在するか不明である．よって，この間に当該CBに対する新たなメモリ・トランザクションが発生しても，完了通知が到着するまでそれを待たせなければならない問題がある．

（2）メモリ経由転送方式

メモリがPEiからPEjへの転送を仲介する（図4（c）参照）．このときの手順は，次のようになる．

①メモリはPEiに対し，当該CBのコピーの返送を要求する．

②PEiはメモリにコピーを返送する．

③PEjをCDエントリに登録する．

④コピーをPEjに送出する．

この方式では，PEjがPEkに（図4（c）の状況），PEkがPEiに，および，PEiがPEjに対して同時にライン・フェッチ要求を出すとデッド・ロックが生じる点が問題となる．

6. 性能予測

トレース駆動シミュレーションにより，本キャッシュの性能（ミスヒット率）を測定した．トレースデータは，DhrystoneをSun－4（SPARC）上で実行したものである．

図5に，Vタグ・アレイのセット数，ブロック・サイズおよびRタグ・アレイの連想度（ウェイ数）をパラメータとした測定結果を示す．前提条件は，次の通りである．

①動的キャッシュ－キャッシュ間コヒーレンス処理による無効化は起きない．

②同義語（シノニム）は生じない．

③仮想アドレスと実アドレスとをランダムに対応付ける．よって，衝突は起き得る．

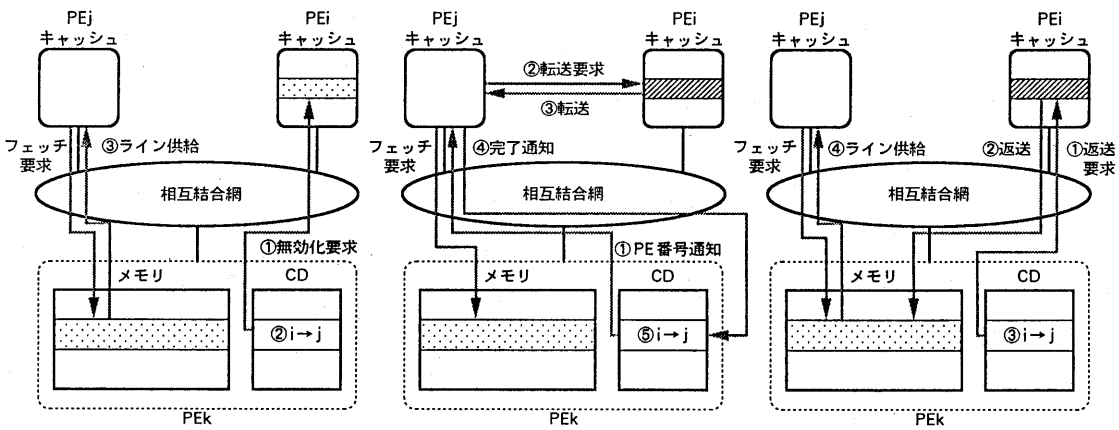
④ストア・スルー方式を採る．よって，ストア・アクセスはすべてミスヒットとする．ただし，ストア・バッファを用いる．

図5より，Rタグ・アレイの連想度が性能に大きく影響することがわかる．特に，連想度を1（ダイレクト・マッピング）としたときのヒット率低下は著しい．

測定結果に基づいて，以下の4つの場合における実効アクセス時間を算出した．

①LC：全メモリ参照がローカル・アクセスで，かつ，キャッシュが存在する場合

②RC：全メモリ参照がリモート・アクセスで，かつ，キャッシュが存在する場合



（a）clean型　（b）dirty型（キャッシュ間直接転送）　（c）dirty型（メモリ経由転送）

CD：コヒーレンス・ディレクトリ

図4. 動的コヒーレンス処理

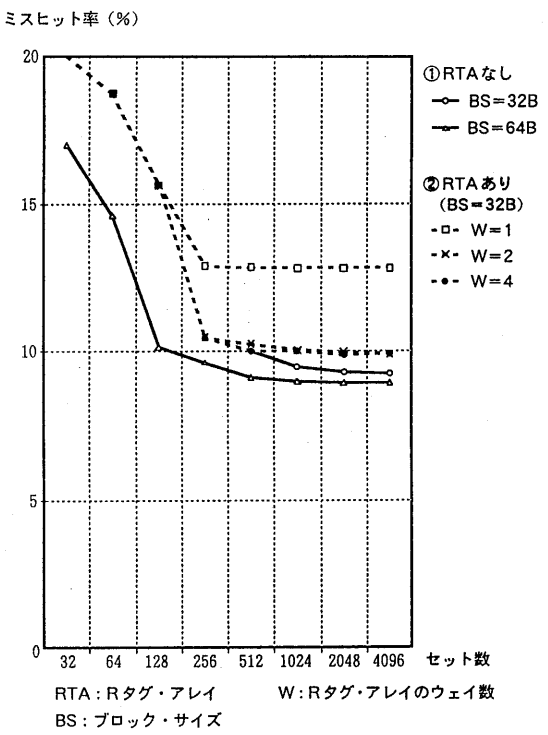


図5. 本キャッシュの性能

③LM：全メモリ参照がローカル・アクセスで，かつ，キャッシュが存在しない場合
④RM：全メモリ参照がリモート・アクセスで，かつ，キャッシュが存在しない場合

これらの実効アクセス時間の比は，

LC：RC：LM：RM＝1.0：1.2：4：16

となる．ただし，キャッシュ，ローカル・メモリおよびリモート・メモリへのアクセス時間は参考文献［4］に基づく．この結果から，リモート・アクセスの頻度が高くなるに従い，キャッシュの効果が大きいことが分かる．

7. おわりに

以上，可変構造型並列計算機のキャッシュ・システムとキャッシュ・コヒーレンス処理について述べた．

現在，ハードウェア開発と並行して，可変構造型並列計算機のための並列／分散OS，および，並列プログラミング言語を含んだ並列プログラミングの開発を進めている．[17],[18]

謝辞

本キャッシュ・システムの初期設計に携わった濱口一正氏（現　キャノン（株）），ならびに，現在我々とともに設計・開発を行っている蒲池，廣谷，福澤，甲斐，草野，恒冨，上野，杉山の各氏，および，日頃ご討論いただく富田研究室の皆様に感謝いたします．

クロスバーLSI開発においては（株）東芝・総合研究所・情報システム研究所の小柳滋氏，PE開発においては富士通（株）本体事業部・スーパーコンピュータ開発部の内田啓一郎氏ならびに高村守幸氏，および，京セラ（株）システム機器開発課の重村慎二氏，システム実装設計においてはアジアエレクトロニクス（株）の作田信彦氏ならびにシステム機器事業部，および，ニシム電子工業（株）の和田勲夫氏に，ご助言ご協力を戴いている．謹んで感謝いたします．

また，“衝突問題”に関しては，米Wisconsin大学Madison校のProf. M.D.Hillに貴重な御意見を頂いた．記して感謝します．

参考文献

[1] 村上ほか：可変構造型並列計算機のシステム・アーキテクチャ，情報処理学会「コンピュータ・アーキテクチャ」シンポジウム論文集，pp.165-174（1988年5月）．

[2] K.Murakami, et al.: The Kyushu University Reconfigurable Parallel Processor－Design Philosophy and Architecture－, Proc. IFIP 11th World Computer Congress, pp.995-1000, September 1989.

[3] K.Murakami, et al.: The Kyushu University Reconfigurable Parallel Processor－Design of Memory and Intercommunication Architectures－, Proc. 1989 Int'l. Conf. on Supercomputing, pp.351-360, June 1989.

[4] 森ほか：可変構造型並列計算機のPE間メッセージ通信機構，情報処理学会「並列処理シンポジウムJSPP'89」論文集，pp.123-130（1989年2月）．

[5] 濱口ほか：可変構造型並列計算機のプロセッサ・ユニット，情報処理学会第38回全国大会講演論文集，3T-1（1989年3月）．

[6] 蒲池ほか：可変構造型並列計算機のメモリ・アーキテクチャ，情報処理学会第38回全国大会講演論文集，3T-2（1989年3月）．

[7] 岩田ほか：可変構造型並列計算機におけるキャッシュ・コヒーレンス処理，情報処理学会第39回全国大会講演論文集，5X-2（1989年10月）．

[8] 濱口一正：可変構造型並列計算機のプロセッサ・ユニット，九州大学大学院総合理工学研究科修士論文（1989年2月）．

[9] 岩田英次：可変構造型並列計算機のメモリ・ユニット，九州大学工学部卒業論文（1989年2月）．

[10] A.J.Smith: Cache Memories, ACM Computing Surveys, vol.14, no.3, pp.473-530, September 1982.

[11] A.J.Smith: Line (Block) Size Choice for CPU Cache Memories, IEEE Trans. on Computers, vol.C-36, no.9, pp.1063-1075, September 1987.

[12] M.D.Hill: A Case for Direct-Mapped Caches, IEEE Computer, vol.21, no.12, pp.25-40, December 1988.

[13] M.D.Hill: Aspects of Cache Memory and Instruction Buffer Performance, PhD dissertation, Tech. Report 87／381, Computer Science Dept., Univ. of California, Berkeley, Calif., November 1987.

[14] J.R.Goodman: Coherency for Multiprocessor Virtual Address Caches, Proc. 2nd Int'l. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS II), pp.72-81, October 1987.

[15] J.Archibald and J.-L.Baer: Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model, ACM Trans. on Computer Systems, vol.4, no.4, pp.273-298, November 1986.

[16] J.Archibald and J.-L.Baer: An Economical Solution to the Cache Coherence Problem, Proc. 11th Int'l. Symp. on Computer Architecture, pp.355-362, June 1984.

[17] 福田ほか：可変構造型並列計算機の並列／分散オペレーティング・システム，情報処理学会オペレーティング・システム研究会資料，OS－43－8（1989）．

[18] 福澤ほか：可変構造型並列計算機の並列／分散オペレーティング・システム，情報処理学会オペレーティング・システム研究会資料，OS－43－8（1989）．