Virtual Queue：超並列計算機向きメッセージ通信機構

五島 正裕[†]　池田 泰敏　森 眞一郎
中島 浩　富田 眞治

京都大学 工学部
〒606-01 京都市 左京区 吉田本町
E-mail: {goshima, yikeda, nakasima, tomita}@kuis.kyoto-u.ac.jp

内容梗概

Virtual Queue システムは，通信機構の仮想化とストリームのキャッシングを特長とするメッセージ通信のための専用ハードウェアである．仮想化によって，任意の数の仮想的通信機構を提供でき，通常のキャッシュと同様システム・コールなしで利用できる．このようにメッセージ送受信時のソフトウェア・オーバーヘッドを大幅に削減するので，細粒度の通信に耐える．また，メッセージそれ自体ではなく，メッセージのストリームへの参照の局所性を利用し，よくアクセスされるストリームを物理的通信機構にキャッシングすることによって高速化を図る．物理的通信機構はベクトル・データの転送にも十分なスループットを持つ．

[†] 日本学術振興会特別研究員

Virtual Queue : A Message Communication Mechanism for Massively Parallel Computers

Masahiro Goshima[†]　Yasutoshi Ikeda　Shin-ichiro Mori
Hiroshi Nakashima　Shinji Tomita

Faculty of Engineering, Kyoto University
Yoshida-hon-machi, Sakyo-ku, Kyoto 606-01 Japan
E-mail: {goshima, yikeda, nakasima, tomita}@kuis.kyoto-u.ac.jp

Abstract

The **Virtual Queue** system is a specialized message communication hardware that is marked by the virtualization of the message communication mechanism and the caching of streams of messages. The virtualization provides arbitrary number of virtual communication mechanisms, and enables a user to access the hardware without system call like a usual cache memory. Since these reduce the message handling overhead of software drastically, it withstands the fine grain communication. Moreover the system makes use of the locality of reference, not to the message themselves but to the message stream, and speeds up by caching frequently accessed message streams to physical units. The units provide enough throughput for transmission of vector data.

[†]: Research Fellow of the Japan Society for the Promotion of Science

1　はじめに

ノード間の距離が大きくなる超並列規模の計算機では，メッセージ処理の高速化がますます重要になる．しかし，従来のメッセージ通信の方式はメッセージ処理のオーバーヘッドが大きく，細粒度の通信に適さない．また，通常のキャッシュはメッセージに対しては効果的ではなく，特にベクトル・データの転送に適さない．

本稿では，メッセージ送受信時のオーバーヘッドの低減と，キャッシュに代わるメッセージ向きの高速化方式の実現を目標とするメッセージ通信用ハードウェア，**Virtual Queue** システムを提案する．以降 ***VQ*** と略す．

2　従来のメッセージ通信の問題点

本章では，従来のメッセージ通信方式におけるメッセージ送受信のオーバーヘッドと，通常のキャッシュのメッセージに対する不適合について述べる．

2.1　メッセージ送受信のオーバーヘッド

共有メモリ/メッセージ，両方式において，メッセージ送受信のオーバーヘッドは数百サイクル程もある．このオーバーヘッドは，細粒度のメッセージを頻繁に交換する応用において，性能に対する影響が大きい．

2.1.1　共有メモリ方式のオーバーヘッド

一般に，共有メモリ・システム上でメッセージ通信を行う場合には，遠隔メモリ上のバッファに対するロック操作が必要となる．超並列規模の計算機では，遠隔ノードへのアクセス (ロック操作) のレイテンシは数百サイクルにものぼる．

単なるメッセージの読み出しの場合とは異なり，更新型のキャッシュ一貫性制御やプリフェッチなどでは，ロック操作のレイテンシを隠蔽することはできない．

2.1.2　メッセージ方式のオーバーヘッド

メッセージ方式の計算機では，メッセージ処理を高速化するハードウェアを持つことが一般的となっているが，これらの計算機におけるオーバーヘッドはむしろメッセージ処理のソフトウェア部分で生じる．

一部のソフトウェア処理は最適化やハードウェアの工夫によって省略可能であるが，資源の獲得とアクセス保護のためのシステム・コールは省略することができない．また受信時には，必要なメッセージを検索したり，システム領域に置かれたメッセージをユーザ領域にコピーする必要のあるシステムも存在する．これらのソフトウェア処理のため，500 ~ 1000 サイクル程のオーバーヘッドが生じる [1]．

2.2　メッセージのキャッシング

共有メモリ/メッセージの両方式において，キャッシュ・メモリの果たす役割は大きい．しかし，通常のキャッシュはメッセージに対しては効果的でなく，その問題点は特に数値処理などの大規模ベクトル処理で顕著に現れる．

2.2.1　参照の時間的局所性とキャッシング

メッセージに対する参照はたかだか 1 回程度であることが多く，時間的局所性の観点からは，メッセージをキャッシングする意味は薄い．それにもかかわらず，メッセージをキャッシングすると，本来キャッシングしておくべき他のデータを追い出してしまい，かえって性能を低下させかねない．ベクトル処理などでは，長大で参照回数の少ないデータを交換するのでこの傾向が強くなる．

2.2.2　キャッシュを利用したバッファリング

一方，1 つのメッセージに対する参照は時間的に集中して行われることが多く，バッファリングによるレイテンシの隠蔽が有効に働く可能性が高いと考えられる．

実際，キャッシュをバッファとして利用する方式がいくつか提案されおり，これらの方式では以下のようにしてメッセージ・アクセスのレイテンシを隠蔽する：送信側ではキャッシュ上で生成したメッセージをキャッシュから直接送信し[1][2][3]，受信側ではプロセッサが参照する前にメッセージをキャッシュに移しておく [2][3][4]．

しかしキャッシュをバッファとして利用する方式では，ベクトル処理のようにプロセッサの稼働率とメモリ・アクセスの頻度が高い場合，以下の問題が生じる：

1. メッセージが 2 度ずつキャッシュ・メモリのポートを通過するため，スループットが不足する．
2. プリフェッチの処理量の増加が性能に影響を及ぼす．
3. また，前節で述べた他のデータの追い出しの問題も解決されない．

3　*VQ* システムの設計方針

VQ システムは，前章で述べたの問題点を以下のような方針で解決する．

3.1　オーバーヘッドの低減

共有メモリ方式では，各プロセッサがメッセージのバッファを分散管理しているのでバッファに対するロック操作を避けることはできないが，一方，メッセージ方式では，受信ノードにおいてバッファを集中管理するので明示的なロック操作を必要としない．しかし，メッセージ処理の高速化のためにバッファ管理用のハードウェアを導入すると，その獲得とアクセス保護のためのシステム・コールが必要となる．

VQ システムは，バッファ管理用のハードウェアを導入し，それを仮想化することで OS の介入を避けるアプローチを採る．

さて，キャッシュ・メモリはシステムの共有資源であるにもかかわらず，OS をまったく介することなく利用

できる．それは資源 (キャッシュ・ブロック) の獲得/横取りと記憶保護をハードウェアで実現しているためである．キャッシュと同様に，*VQ* システムは以下のようにして OS を介さないハードウェアの利用を可能にする．

3.1.1 仮想化

ハードウェア・ユニットの処理中の状態を定義するのに十分な情報をそのユニットの コンテクスト と呼ぶ．例えば，サーキュラ・バッファの制御回路は，バッファの書き込み/読み出し位置やエントリ数などのバッファの管理情報を保持するレジスタを持つが，それらのレジスタの内容はその制御回路のコンテクストとなる．

VQ システムでは，通信機構のコンテクストを主記憶上の退避領域に置き，コンテクストを適宜切替えることによって，その通信機構を時分割多重利用する．コンテクストをキャッシュする専用のキャッシュを用意し，通信機構のコントローラがそのキャッシュを使用する形で，これを実現する．

コンテクストをキャッシングすることで，通常のキャッシュと同様に，仮想記憶方式を応用することによって記憶保護が実現でき，また，キャッシュ・ミスの処理をハードウェアで自動的に行うことによって資源 (通信機構) の獲得/横取りを OS を介さずに実行することができる．

一方，通信機構の個々のコンテクストは主記憶上におかれるので，実際上任意の数の仮想的な通信機構をユーザに提供できる．ユーザは，メッセージのストリームごとに仮想通信機構を割り付けることによって，受信時にメッセージの検索やコピーを避けることができる．

このように *VQ* システムでは，従来のメッセージ方式で必要であったソフトウェア処理のほとんどを省略することができる．

3.2 高速バッファ・メモリの小量化と高速化

高速なメモリにデータを移しておくことによってアクセスのレイテンシを短縮することは重要であるが，通常のキャッシュは主に参照の時間的局所性を利用して高速化を図るものであるから，メッセージのようにその性質の弱いデータに対しては効率的に働かない．メッセージに対するアクセスはたかだか一回程度であることが多く，一度アクセスされたメッセージをキャッシュに置いておくことは無駄であることが多い．

3.2.1 FIFO による通信モデル

VQ システムでは，通信モデルを FIFO によるもののみに制限することによって，この問題を解決する．すなわち，*VQ* システムは通常のキャッシュと同様にメッセージを高速なバッファ・メモリに移しておくのだが，メッセージに対する参照の回数を 1 度に限定することによって，古いメッセージの占めていたバッファ・メモリ上の領域の再利用を図る．

FIFO による通信のみを用いる言語も多く，この制限は決して厳し過ぎるものではないと考えられる．プロセッサ間チェイニングによるベクトル処理などにも対応可能である．

また，FIFO による通信モデルを仮定することによって，プリフェッチを自動化できるという効果も得られる．

3.2.2 ストリームのキャッシング

一度アクセスされたメッセージは 2 度とアクセスされないことが多いが，一度アクセスされたメッセージのストリームは近い将来再びアクセスされる可能性が高いと考えられる．*VQ* システムは，メッセージそれ自体ではなく，メッセージのストリームに対する参照の局所性を仮定し，これを利用する．

前述したハードウェアのコンテクストのキャッシングは，よくアクセスされるストリームを高速な物理的通信路に置くことと捉えることができる．コンテクスト・キャッシュはプロセッサごとに 4 ~ 16 エントリ程度しか用意しないが，プロセッサで走行中のスレッドがそれ以下のストリームを使用するならば *VQ* システムへのアクセスはヒットし続ける．

3.2.3 バッファの構成

以上の 2 点によって，バッファリングのために必要なバッファの容量は大幅に削減される．したがって，デュアルポート・メモリを用いてこのバッファを構成し，メッセージのバッファリング専用に用いる．

デュアルポート・メモリを用いてバッファリングを行うので，バッファリングの実施によって系のスループットが低下することはない．また，メッセージのバッファリング専用に用いるので，通常のキャッシュ上の他のデータを追い出すことがない．

4 *VQ* システムの構成

本章では，以下の手順で *VQ* システムについて詳細に説明する．まず 4.1 節では *VQ* システムがプログラムからどのように見えるか示し，プログラムから見える動作を実現するハードウェアについて，4.2 節以降で述べる．

4.1 プログラムに対するインターフェイス

4.1.1 *VQ*

図 1 に，*VQ* システムにおける通信の様子を示す．

VQ は，送信スレッドと受信スレッドの間の共有仮想空間上に設定された仮想的な通信機構である．プログラムは任意の数の *VQ* を設定できるので，通常メッセージのストリームごとに *VQ* を設定する．

VQ は，3 種の仮想空間上の領域—ライト・ポート (**WP**)，サーキュラ・バッファ (**CB**)，および，リード・

ポート (**RP**) を構成要素とする．1 つの VQ は，1 つ以上の WP，および，1 組の CB と RP からなる．

VQ は，数十バイトの ブロック を通信の単位とする．ブロックは不可分に扱われるので，転送経路各部の実効バンド幅を向上させ，管理情報を小量化する他，リモート・プロシージャ・コールなどの応用において有用である．

ブロックの送信は WP へのストアによって，ブロックの読み出しは RP へのロードによって行う．1 つの WP から送信されたブロックは，送信された順序で RP から読み出されるが，複数の WP から送信されたブロック間の順序は非決定的となる．

4.1.2 VQ のアドレス

VQ がプログラムからどう見えるか，次項からより詳細に説明する．

ユーザは，VQ の使用に先だって VQ のデータ構造を設定する必要がある．データ構造は，VQ システムのハードウェア・ユニットがそのコンテクストの位置を特定するため，アクセスされたアドレスとコンテクスト退避領域のアドレスの間の簡単な写像を定義する．

(A) システムのアドレス空間の構成

1 つのプロセス内のスレッドは 1 つの仮想アドレス空間を共有するものとする．ただし，遠隔ノードのメモリに対して直接アクセスが行える必要はない．ノード間にまたがる大域的仮想アドレス空間の構成方法は本稿の範囲を越えるので，ここでは触れない．

VQ システムの実アドレス空間中には，主記憶の物理アドレス空間とは別に，VQ アクセスのためのアドレス空間—ポート・アドレス空間 を設ける (図 2 左)[1]．ポート・アドレスは，主記憶の物理アドレスに対する一種の別名である．ポート・アドレス空間は，主記憶の物理アドレス空間と等しい大きさを持つ．また，ポート・アドレス空間上のページ—ポート・ページ は，それぞれのアドレス空間内での変位が互いに等しい物理ページと対応する．

VQ を使用するユーザ・プロセスは，UNIX では *mmap* に相当するシステム・コールにより，物理ページに対応するポート・ページを獲得し，仮想ページに写像する (図 2 左から中)．OS は，ユーザが対応する物理ページを既に獲得している場合に限り，ポート・ページへの写像を許可する (理由は後述)．

物理アドレス空間に写像された仮想ページを 仮想メモリページ，ポート・アドレス空間に写像された仮想ページを 仮想ポート・ページ と呼ぶ．

これらのページ対の上に，VQ のデータ構造を設定するのだが，一旦ページ対を割り付けられた後には仮想記憶によってアクセス保護が実現されるので，VQ に関するすべてのアクセスはユーザ・モードで行うことができる．

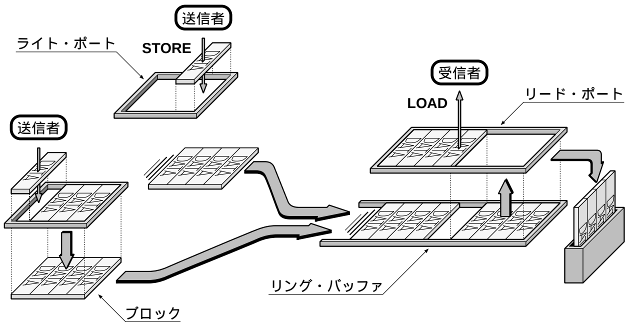


図 1: プログラミング・モデル

(B) VQ のデータ構造

プログラムは，図 2 右 に示すように，VQ のデータ構造を設定する．

VQ の各要素は，アラインされた 1 ブロック・サイズの領域—ブロック枠 を単位とする．

プログラムはまず，仮想メモリ・ページ上に各要素の退避領域と CB を割り付ける．ただし，RP ユニットと CB ユニットのコンテクストは 1 つのブロック枠に詰め，CB はその次のブロック枠からはじまる 1 ページ内の連続領域に割り付ける (図 2 右下)．

WP と CB/RP は，送信先の CB/RP の仮想アドレスを WP のコンテクストとして指定することで結び付けられる (図 2 右下の矢印)．図 2 では，説明の都合上 WP と CB/RP を同一ページに設定していているが，実際には相互の位置関係に制限はない．

仮想ポート・ページ上の，退避領域とページ内変位が互いに等しいブロック枠がそれぞれのポートとなる．ただし，RP は 2 ブロック枠を使用する (図 2 右上)．この領域に対するストア/ロードにより，VQ による送/受信を行う．

ポート・ページと物理ページは，各アドレス空間内での変位が互いに等しい位置にとった (図 2 左) ので，ポートと退避領域の実アドレスは各アドレス空間内での変位が等しい．したがって，VQ システムのハードウェアは，ポートへのアクセスがミスを起こした時，ポートの実アドレスのポート・アドレス空間内の変位から対応する退避領域の物理アドレスを知ることができる．

4.1.3 通信手順

上記のデータ構造を設定した後に，VQ による通信が可能となる．通信は以下のように行う：

1. **送信** 送信者は，WP に対するストアにより，WP 上にブロックを形成する．WP の最後のバイトが変更を受けた時，その時点での WP 上のブロックが CB の末尾に追加される．転送すべきデータが 1 ブロックに満たない場合には，最後のバイトへのダミー・

---

[1] その他にメモリ・マップ I/O のアドレス空間があってもよい．

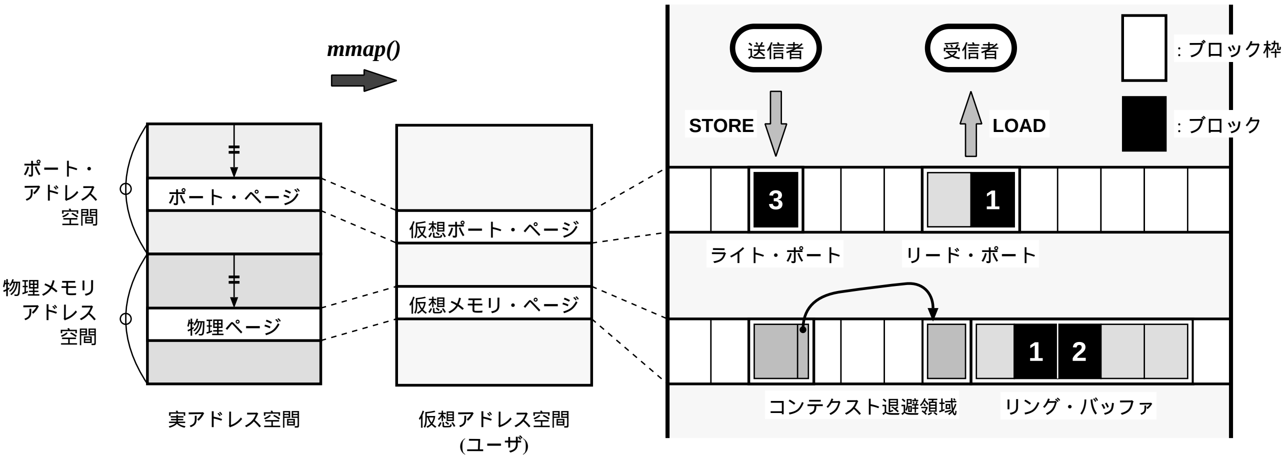


図 2: VQ のデータ構造

ライトを行う．

最後のバイト以外の部分に対する書き込みには，順序，回数などの制限はない．

プログラムは送信処理の完了を確認する必要はない．

2. 受信 RP の 2 つのブロック枠を用いて 2 面バッファリングを行う．2 つのブロック枠のどちらか一方には CB の先頭のブロックが，他方には 2 番目のブロックが写像されている．先頭のブロックへの参照が終り 2 番目のブロックへの参照を開始すると，参照の終ったブロックは自動的に CB からデキューされ，空いたブロック枠には新たに 2 番目となったブロックが写像される．

1 つのブロック枠内のデータに対する参照には，順序，回数などの制限はない．

4.2 ハードウェアの構成

VQ システムは，WP，CB，および RP それぞれに対応するユニットからなり，これらのユニットの協調動作によって上述の通信処理を実現する．

ポートへのアクセスに対して各ユニットは，そのコンテクストを主記憶上の退避領域からコンテクスト・キャッシュにキャッシュし，コンテクスト・キャッシュ上のコンテクストを参照/更新しながら動作する．OS は，ユーザが対応する物理ページを既に獲得している場合に限り，ポート・ページへの写像を許可すると述べた．したがって，ポートへのアクセスが許可されいれば，退避領域へのアクセスの正当性を改めて検査する必要はない．コンテクスト・キャッシュについては，次節で詳しく述べる．

図 3 に，対象計算機の処理ノードの構成 (主要なデータ・パス) を示す．図中には示していないが，通常のメモリ・アクセスのためのバイパス (・バッファ) がある．簡単のため，プロセッサが 1 次キャッシュやストア・バッファを内蔵しない場合について説明する．それらへの対応については 4.5 節でまとめる．

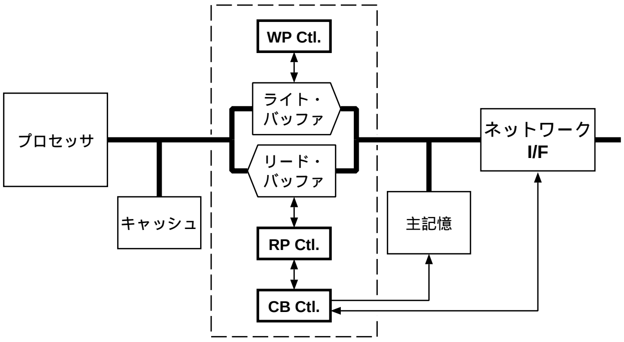


図 3: 処理ノードの構成

システムはデュアルポート・メモリで構成されたライト・バッファ/リード・バッファを備え，ブロックに対するアクセス・レイテンシを隠蔽する．

以下，各ユニットがどのように動作して前述の通信処理を実現するか説明する．同時に，記憶保護の実現について述べる．

(A) WP ユニット

WP に対してストアされるデータは，実際にはライト・バッファに書き込まれ，ライト・バッファ上でブロックにまとめられる．

ライト・バッファの最後のバイトへのストアをトリガとして，保持するブロックを送出する．ブロックは，主記憶をバイパスして，ライト・バッファから直接ネットワーク I/F 部に送られる (図 3 を参照)．

ネットワーク I/F 部では，CB の仮想アドレスから CB のある処理ノードの物理アドレスへの変換が行われ，転送されるブロックには CB の仮想アドレスが付加される．大域的仮想アドレスの具体的な構成方法についてはここでは触れない．

(B) CB ユニット

CB ユニットは，ネットワークからブロックが届いた時，ブロックに付加された CB の仮想アドレスで CB ユ

ニットのコンテクスト・キャッシュにアクセスし，主記憶上のCBの書き込み位置の物理アドレスを求める．したがって，OSが仮想アドレスと物理アドレスの写像を制御することによって，記憶保護が実現される．

書き込みに関するCBの管理情報の更新はCBユニットで行われる．

(C) RPユニット

RPユニットのコントローラは，先頭と2番目のブロックがリード・バッファ上に存在するように制御する．RPに対するロードでは，実際にはリード・バッファから読み出しが行われる．

リード・バッファ上の2ブロック枠を用いて2面バッファリングを行い，主記憶上のCBに対するアクセス・レイテンシを隠蔽する．アクセスが先頭のブロックから次のブロックに移った時に，RPユニットのコントローラはRPの実アドレスでコンテクスト・キャッシュにアクセスし，主記憶上のCBの読み出し位置を求める．CBはRPのコンテクスト退避領域と同一ページ内に割り付けるので，RPへのアクセスが許可されていれば，CBへのアクセスの正当性を改めて検証する必要はない．

読み出しに関するCBの管理情報の更新は，RPユニットで行われる．

4.3 コンテクスト・キャッシュ

*VQ*システムの各ユニットは，それぞれのコンテクスト・キャッシュを持ち，コンテクストはそれぞれ独立にキャッシングされる．

前述のライト/リード・バッファは，実際には，コンテクスト・キャッシュの一部とみなすことができる．

図4にコンテクスト・キャッシュの構成と論理的なデータのながれを示す．また，各コンテクスト・キャッシュのエントリ数，連想度，アクセスするアドレスを，同図中の表に示す．

以下，各ユニットの **(A)** コンテクスト と，**(B)** コンテクスト・キャッシュ について詳細に説明する．

4.3.1 WPユニット

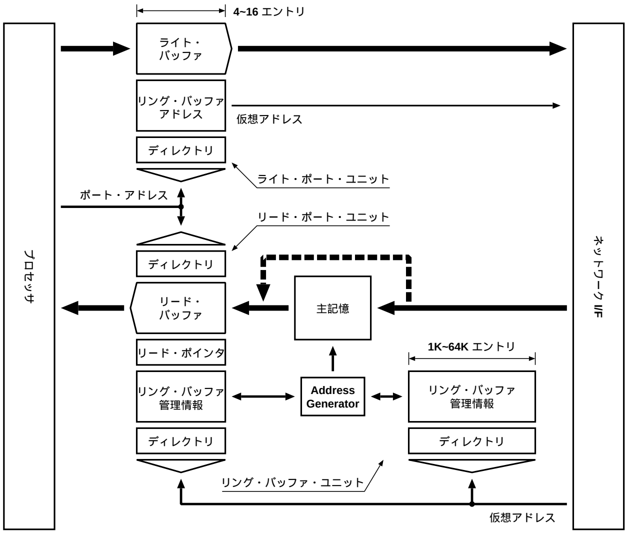
(A) コンテクスト

ライト・バッファ上の生成途中のブロックは，WPのコンテクストとなる．

また，送信先のCBの仮想アドレスもコンテクスト・キャッシュにキャッシュし，送信処理を高速化する．

(B) コンテクスト・キャッシュ

WPユニットのコンテクスト・キャッシュは，通常のプロセッサが持つストア・バッファなどとほぼ同じものと考えてよい．WPの実アドレスがディレクトリの保持するアドレスと比較され，ヒット/ミスが判断される．



ユニット	エントリ数	連想度	アドレス
WP	4 ~ 16	完全	実
CB	1K ~ 64K	1 ~ 4	仮想
RP	4 ~ 16	完全	実/仮想

図4: コンテクスト・キャッシュ構成

4.3.2 CBユニット

(A) コンテクスト

以下のCBの管理情報がコンテクストとなる：

- サイズ (CB末尾の，先頭アドレスからの変位)
- 読み出し/書き込み位置 (先頭アドレスからの変位)
- 現在のブロック数

(B) コンテクスト・キャッシュ

WP，RPへはプロセッサ上で走行中のスレッドのみがアクセスするのに対して，CBユニットへは遠隔の多くのノードから非同期にブロックが届くので，1つのCBユニットは性能上同時に多数のコンテクストをキャッシングする必要がある．したがって，CBユニットのコンテクスト・キャッシュは，RAMチップを用いて構成する．

理由は後述するが，CBに対する書き込みはクリティカルではない．通常のメモリ・アクセスが*VQ*アクセスをバイパスできるとすれば，このキャッシュに対するアクセスのレイテンシを短縮する必要はない．スループットに関して，(このキャッシュのアクセス・サイクル数) = (主記憶のブロック・アクセス・サイクル数) ÷ (主記憶アクセスに占める*VQ*アクセスの割合) 程度であればよい．

複数サイクルをかけてアクセスすることで線幅を縮小できるので，このキャッシュは主記憶と同等のサイクル・タイムを持つSRAMチップ1個程度を用いて構成できる．これは，ハードウェア量としては外部2次

キャッシュのディレクトリなどと同程度である．

4.3.3 RP ユニット

(A) コンテクスト

WPユニットとは異なり，リード・バッファ上のブロックはRPのコンテクストとする必要はない．なぜなら，リード・バッファ上のブロックはCBのコピーであり，CBのコンテクストがRPの状態を完全に指定するからである．

また，リード・ポートの2ブロック枠のうちどちらに先頭のブロックが写像されているかを示すビットが必要であるが，このビットはRPのコンテクストとなる．

(B) コンテクスト・キャッシュ

図4の，RPユニットのコンテクスト・キャッシュのリードバッファから上側は，通常のキャッシュと同じものと考えてよい．

RPユニットのコンテクスト・キャッシュは，CBユニットとは独立にCBの管理情報をキャッシュし，新たに2番目となったブロックをリード・バッファにプリフェッチする時には，ここの情報を用いて高速に主記憶にアクセスする．2面バッファリングを行っているとはいえ，リード・バッファへのプリフェッチは比較的高速に行う必要があるので，大容量で低速なCBユニットのコンテクスト・キャッシュとの共用はしない．

このように，CBの管理情報はRPユニットとCBユニットの2箇所にキャッシュされるため，2つのキャッシュの一貫性を保つ必要がある．RPユニットのコンテクスト・キャッシュの下の部分は仮想アドレスでアクセスできるようになっており，外部からブロックが到着した時には，CBユニットと同時にRPユニットのコンテクスト・キャッシュにもアクセスする．2つのコンテクスト・キャッシュに同一のエントリが存在した時には，RPユニット側を優先して用い，CBユニット側のエントリは無効化する．

また，あるVQに対してプロセッサがスピン・ウェイトしている最中にブロックが到着した場合には，RPユニット側のコンテクスト・キャッシュがヒットするので，ブロックは主記憶をバイパスしてリード・バッファに送ることができる(図3参照)．前項でCBユニットのコンテクスト・キャッシュに対するアクセスはクリティカルでないと述べたが，それはこの理由による．

CBの管理情報を2箇所にキャッシュすることには，プロセッサからのVQアクセスとネットワークからのVQアクセスを並列処理できるというメリットもある．

4.4 アンダーフロー/オーバーフロー

本節では，VQのアンダーフロー/オーバーフローの検査の方法についてまとめる．以下で述べる方法の中には割り込みを用いるものがあるが，紙面の都合により割り込みからの回復の方法に関しては割愛する．

4.4.1 アンダーフローの検査

VQのアンダーフローは，以下の3種の方法によって検査される：

1. **検査用コマンドを使用**　コマンドにより，CB中のブロックの数を検査する．
2. **使用しないパタンを応答**　アンダーフローを起こした読み出しに対して，ヌル・ポインタ値0，あるいは，NotANumberなど，アプリケーションが使用しないパタンをブロックにつめて応答する．

   例えば，次のようにするとよい：送信者はブロックの最終バイトへ0以外の値を書き込むことによって送信を行い，受信者はその位置が0以外であることを確認する．
3. **割り込みを発生する**　アンダーフロー時に(しばらく待って)割り込みを発生する．この方法を採れば，プログラムはVQのアンダーフローを明示的に検査する必要がないので，VQが空にならない可能性が高い場合に有効である．

2，3のうちのいずれの方法を使用するかは，RPのコンテクストとして指定する．1はいつでも使用することができる．

4.4.2 オーバーフロー

プログラムは，原則として，以下のいずれかの方法によってオーバーフローの発生以前に送信側のスレッドと速度を調整するものとする：

- コマンドにより定期的にブロック数を検査する．
- ブロック数がユーザが設定した値になった時点で割り込みを発生する．

それでもオーバーフローを起こした時には，しばらく受信処理を中断した後に，プロセッサに割り込みをかけて知らせる．エンキューできるようになるまで受信処理を中断する方法も考えられるが，デッドロックを起こす可能性があるので採用しない．

4.5 市販のプロセッサへの対応

現在，市販の多くのプロセッサが1次キャッシュやストア・バッファを内蔵している．本節では，それらのプロセッサへの対応について述べる．

4.5.1 プロセッサの条件

VQシステムを実装するためには，要素プロセッサは以下の条件を揃えている必要がある：

1. プログラム・オーダでストアを観測する手段がある．
2. 1次キャッシュ・ブロックが外部から無効化できる．

市販の多くの汎用プロセッサがこれらの条件を満たしている[5][6]．

4.5.2 対応の方法

プロセッサが上述の条件を満たしている場合，*VQ*へのアクセスは内蔵1次キャッシュにまでキャッシュ可能であり，内蔵ストア・バッファによるストアのマージも可能である[7]．ただし，その場合，*VQ*のブロックのサイズは，1次キャッシュのブロックのサイズと等しくする必要がある．

以下，WPとRPにおけるハードウェア，ソフトウェアの変更点について説明する．

(A) WP

WPでは，送信のトリガとなるブロックの最終バイトへのストアが，ブロックの他の部分へのストアを追い越さないようにする必要がある．ストア・バッファにおいてストアのオーダを変更するようなプロセッサでは，ストア・バリア命令の挿入が必要となる[6]．

(B) RP

RPでは，バッファリングを行う時の次のブロック枠への参照の移動が外部から観測できる必要がある．そのため，RPは3ブロック枠で構成し，3面バッファリングを行う．RPのコントローラは先頭のブロックと直前に先頭であったブロックのみが1次キャッシュ上に存在するように制御する．すると，2番目のブロックへのアクセスは必ず1次キャッシュ・ミスを起こすので，参照の移動を外部から観測できる．この1次キャッシュ・ミスに対して新たに先頭となったブロックを応答し，古いブロックを無効化する．

最適に動作した場合，*VQ*に対する読み出しは，メッセージが常に2次キャッシュ上に存在する場合と同等の性能を示す．

5 おわりに

*VQ*システムは，仮想化により任意の数の仮想的通信機構を提供し，メッセージ送受信時のソフトウェア・オーバーヘッドを大幅に削減する．また，FIFOによる通信モデルとストリームのキャッシングにより，ハードウェアを小量化/高速化する．

しかし，これらの特長はプログラムに対して以下の制限を与えた結果として得られるものであり，これらの制限の妥当性を検証する必要がある：

1. **FIFOによる通信モデル**　送信者が一旦送信したブロック，受信者が一旦デキューしたブロックに対する参照は行えない．

   *VQ*システムは，明示的通信向きに設計されており，プロセッサ内で生産されたデータを直接送信する時にその真価を発揮する．逆にいえば，*VQ*システムは，暗示的な通信に対しては最適な通信システムとはいえない．

2. **固定長メッセージ方式**　長いメッセージを不可分に転送すると，その間ハードウェアを横取りすることができず仮想化が困難になるので，可変長のメッセージは扱わない．

   可変長メッセージを扱うためには，送信者が1人である*VQ*を別途生成するなどの処理が必要となる．

本機構の一部は，超並列計算機JUMP-1に実装される[7]．今後は，超並列計算機JUMP-1上で得られた実アプリケーション，および，シミュレーション[8]などで得られたデータを元に，上で示した制限について定量的評価を行う予定である．

謝辞

本研究の一部は，文部省科学研究費補助金 (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」，および，特別研究員奨励費 高速メッセージ通信機構に基づく並列計算機システム」) による．

参考文献

[1] 清水俊幸, 堀江健志, 石畑宏明 :高速メッセージハンドリング機構—AP1000における実現—, in *JSPP '92*, pp. 195–201 (1992).

[2] Byrd, G. T. and Delagi, B. A.: StreamLine: Cache-Based Message Passing in Scalable Multiprocessor, in *ICPP '91*, pp. I–251–I–254 (1991).

[3] 五島正裕, 森眞一郎, 富田眞治 :プロセッサ間通信をサポートするOn-Memory FIFO機構, 情処研報 92–OS–56, pp. 111–118 (1992).

[4] 平木敬他 :超並列計算機プロトタイプJUMP-1の構想, 情処研報 93–ARC–102 (1993).

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

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

[7] 五島正裕他 :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, 情処研報 93–ARC–101, pp. 121–128 (1993).

[8] 五島正裕, 森眞一郎, 中島浩, 富田眞治 :アーキテクチャ・シミュレータのためのC++クラス・ライブラリ, 情報処理学会 第47回 全国大会 論文集 (6) (1993).