オペレーティング・システム 56−15
(1992. 8. 19)

プロセッサ間通信をサポートする On-Memory FIFO 機構

五島正裕†, 森 眞一郎†, 富田眞治†
†：京都大学 工学部

プロセッサ間通信をいかに高速に行うかは物理的に分散されたメモリを持つマルチ・プロセッサのデザインにおける最重要事項の1つである。プロセッサ間の通信においては FIFO が本質的な役割を果たす。従来システムでも FIFO を利用した通信方法は存在したが、それらはさまざまな制約のために十分な性能が得られなかった。本稿で提案する On-Memory FIFO 機構では、共有アドレス空間上に設けられた任意個の FIFO をハードウェアによって半自動的に制御する。本機構では、システム・コールやプロシージャ・コールを介すことなく、プロセッサ間の通信がキャッシュからキャッシュへのメッセージ・パッシングとして高速に実現される。

The On-Memory FIFO Mechanism for Inter-Processor Communication

Masahiro GOSHIMA†, Shin-ichiro MORI†, Shinji TOMITA†

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

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

How to realize fast inter-processor communication is one of the most important issues in designing the multi-processor. The FIFO plays an essential role in inter-processor communication. Some systems have been developed which use the FIFO for communication, but they did not work well because of various constraints. In this paper, We introduce the On-Memory FIFO mechanism which communicates through arbitrary number of hardware controlled semi-automatic FIFOs on the shared address space. This mechanism realizes inter-processor communication as the fast cache-to-cache message passing without system calls or procedure calls.

1 はじめに

複数プロセッサ間のデータのやりとり，プロセッサ間通信をどのように実現するかはマルチプロセッサ・システムの設計における最重要課題の1つである．

プロセッサ間通信の方法としては，メッセージ・パッシングによるものや共有メモリを介するものなどが考えられ，実現されてきた．しかし，それらは結局全く同一の処理を実現するための手法の1つに過ぎない．すなわち，異プロセッサに跨るデータ・フローの実現である．本稿では，メッセージ・パッシングや共有メモリといった概念にとらわれず，純粋にプロセッサ間通信を高速に行うためのプロセッサ間通信機構を考える．

2章では，プロセッサ間通信の本質について考察し，それを高速に行うための必要条件を導き出す．3章では，2章で導かれた条件に基づき，共有アドレス空間上に設けられたFIFOを使用してプロセッサ間通信を行うOn-Memory FIFO機構を提案し，4章でその簡単な性能の予測を行う．

2 プロセッサ間通信機構としてのFIFO

2.1 プロセッサ間通信の本質とFIFOの親和性

プログラムの実行はデータ・フロー・グラフのエミュレーションであると考えられる．このとき，プロセッサ間通信はグラフのアークが異プロセッサ間に跨る時に発生する．マルチプロセッサ上でのプログラムの並列実行においては，プロセッサ間通信に対応するアークにおいて，定義すべきデータがまだ定義できない状況や，参照すべきデータがまだ存在していないという状況が発生し，その部分での待ちがプログラムの実行を遅らせる要因となっている．

このような待ちを最小化するためのプログラムの最適化は重要である．例えばデータの定義はできる限り早く，参照はできる限り遅く行なうなどがそれである．しかし，プロセッサ間通信のあり方そのものにも注意を払わねばならない．

現実のプログラムを考えた場合，実行のほとんどはループなどの繰り返し部分に費やされており，1つの識別子に対して繰り返しデータが定義／参照されている．逐次プログラムとは異なり，マルチプロセッサのプログラムでは，1つのデータに対する定義と参照の発生順序は動的にしか分からない．このため，共有メモリ方式において，通信用のデータを格納する空間が，逐次の場合と同様にそのデータ1つ分しかとられないような方法では以下のような問題が生じる．即ち，定義側は定義したデータが参照され終ったことを確認するまで再定義できず，参照側は定義され終ったことを確認するまで再び参照できなくなる．同期とはその確認のための操作と考えられる．この確認操作のため，定義と参照の間の処理は常に最も遅

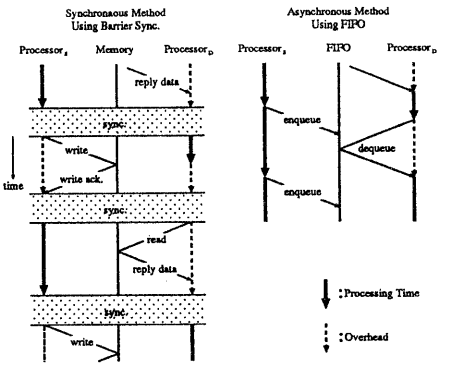

図 1: 同期とオーバーヘッド

いものによってバウンドされ，本来無用の待ちが発生し，不必要な情報がネットワークをながれる（図1左を参照）．

ところが，プログラムの実行はデータ・フロー・グラフのエミュレーションであるという観点でプロセッサ間通信を考えると，プロセッサが処理を中断しなければならないのは，原理的には参照すべきデータが未定義の場合のみである．その場合にのみ，ちょうど命令パイプラインのインターロックのように，参照側の処理をサスペンドする機構があれば良いことがわかる．

それを実現するために，データ・フロー・グラフのプロセッサ間通信に対応するアークの1つ1つに，ハードウェアによって半自動的に動作する識別子付きのFIFOを用意することを考える．このFIFOは，リード／ライト・ポインタや長さなどの管理情報やFIFOに対する相互排除の制御などをハードウェアで実現したものである．このようなFIFOを用いることで，定義と参照の順序関係に関する情報が，1つのFIFOの内部にencapsulateされる．この抽象化されたFIFOにより，

1. 大域的な同期操作をすることなく，定義と参照の順序の局所的な管理が可能となり，
2. データ・フロー実現のための必要最小限のデータのみがネットワークに流れるようになる．

この2点において，データ・フローの最適なエミュレーションが可能となる．

2.2 FIFOに求められる条件

前節ではハードウェアで半自動的に管理される識別子付きFIFOを介して通信を行なうと，通信におけるデータ・フローが最適に実現できることを示した．本節では，通信を更に高速化するためにFIFOをどのように扱ったら良いかを考る．

FIFOの位置 FIFOを介して通信を行なう時には，定義する側はそれによって処理に遅れを生じないが，一方，参照する側は参照のためのコマンドの発行から実際に値

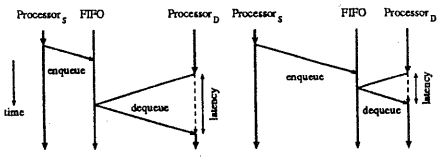

図 2: FIFO の位置と参照のレイテンシとネットワークをながれるデータ量

を得るまでにレイテンシを生じる（図2）．参照のレイテンシを最小化するために，**FIFO の実体はできる限り参照側のプロセッサに近い位置に配置すべきである**．また，そうした方が参照のためのコマンドがネットワークをながれる量も少ない．FIFO をプロセシング・ノード内に配置した場合，参照のレイテンシとネットワークをながれるデータの量は共に最小化される．メッセージ・パッシング型の計算機はその例であり，その意味では最適な形態であるといえる．

セキュリティと共有アドレス　しかしメッセージ・パッシング型の計算機の中には，1つのプロセッサに届くすべてのメッセージが1つのバッファに入れられてしまうものがある [1]．そのような構成ではメッセージの検索に大量のプロセッサ時間が消費され，またマルチ・ユーザで使用する時には，セキュリティのため，システム・コールを介してしか通信できなくなる．

マルチプロセッサにおけるプロセッサ間通信は決して特殊なイベントなどではなく，通常のロード／ストアと同じくらい普通に行われるものである．ゆえに，プロセッサ間通信はロード／ストアと同様にシステム・コールやプロシージャ・コールを介さずに行われなければならない．その点で，共有メモリ型のシステムは優れた資質を持っている．それはアドレス変換機構を記憶保護に利用していることによる．**FIFO の ID も共有アドレスとし，通常のロード／ストア命令で通信を可能とするべきである**．そうすることによって，通常のロード／ストアと同じ記憶保護機構を利用できる．

通信とキャッシュ・メモリ　最新のプロセッサの使用を考えるとキャッシュ・メモリは必須である．通信においてもその扱いを考えるべきである．

参照のレイテンシを最小化するために，送信するデータは定義されるとすぐに送信すべきである．またアプリケーション・プログラムはメッセージを書き換えて再度送信することが多い．送信するデータはキャッシュ上に載っている可能性が非常に高く，キャッシュから直接送信が可能なら更に高速に通信できる．[1] また，受信側に届くデータは近い将来に参照される可能性が高い．ゆえに，届くデータをキャッシュに直接書き込んでおけるなら，さらに低レイテンシの通信が期待できる．届くデータをキャッシュに直接書いておくことは，プリフェッチの1手法と捉えることができる．**通信は，キャッシュからキャッシュへの転送として実現すべきである**．

既にキャッシングされているデータのリプレースを防ぐという理由で，届くデータをキャッシュに直接書き込まないマシンもある．[1] しかし，届くデータの定義と参照の時間間隔がクリティカルであれば，キャッシュに直接書き込んだとしても，すぐに参照される．参照し終ったデータを適切に無効化すれば，到着するデータによって必要なデータが大量にリプレースされる可能性は低い．無論，定義と参照の間隔が長いデータはキャッシュに書き込まなければ良い．

3 On-Memory FIFO 機構

3.1 On-Memory FIFO 機構の概要

On-Memory FIFO とは，通信を行う2以上のプロセッサ間の共有アドレス空間上に，送信側のプロセッサは入力ポート，受信側のプロセッサは出力ポートのみをインタフェースとして提供する仮想的な FIFO メモリであり，その実体を主記憶上に作成しそれを管理するのが On-Memory FIFO 機構である．

3.2 On-Memory FIFO 機構の設計方針

プロセッサ間通信機構として On-Memory FIFO 機構を実現するうえでの具体的な要件ならびにそれに対する設計方針を以下に示す．

1. FIFO の数と大きさの仮想化　個々の FIFO を主記憶上に配置し，共有アドレス空間上に割り当てられた，メモリ・マップド I/O として On-Memory FIFO 機構を実現することで，任意の数の FIFO を提供可能とする．アドレス空間上に配置された FIFO は，当該 FIFO へのアクセスポート識別子 **(FIFO ID)** によって識別する．また，アドレス空間にマッピングされた個々の FIFO には1エントリ分のアドレス空間のみを提供し（制御領域は除く），FIFO の長さに関しては完全に仮想化する（制限を与えない）．

2. FIFO 管理の隠蔽　FIFO から／への1ワードの読み出し／書き込みを FIFO ID で示すアドレスへの通常のロード／ストア命令のみで可能にするため，ポインタ等の FIFO 管理を全てハードウェアで実現する．また，FIFO への読み出し／書き込みを行うプロセッサが異る場合，プロセッサ間のデータ転送が必要となるが，このための通信処理はプロセッサとは独立して専用のハードウェアにより実現する．なお，これらのハードウェアを総称して，以下では **On-Memory FIFO 機構 (OMF)** と呼ぶ．

3. FIFO へのアクセスの高速化　FIFO へのアクセスの高速化のため，On-Memory FIFO の実体は受信側のプロセッサに設けなければならない．しかしながら，受信側

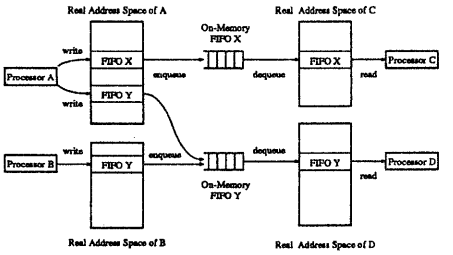

図 3: On-Memory FIFO 機構のプログラミング・モデルの概念図

プロセッサと送信側プロセッサの並列動作を可能とするために，FIFO を受信側と送信側の 2 つの FIFO に分割しネットワークを介してそれらを接続する．このような FIFO 分割の影響はすべて OMF が隠蔽する．また，送信側 FIFO の長さは 1 エントリ分とする．

さらに，FIFO 操作の高速化のため，On-Memory FIFO 専用のキャッシュ（FIFO キャッシュと呼ぶ.）を設け，On-Memory FIFO 機構と プロセッサ ならびに ネットワーク とのインタフェースをこの FIFO キャッシュ上に設ける．これにより，ネットワークから／へのデータを直接キャッシュ上の On-Memory FIFO に書き込むことが可能となり，メモリへのバッファリングという冗長な処理をなくすことができる [1]．また，FIFO キャッシュはプロセッサからのアクセスとネットワークからのアクセスを同時に実行可能とするため，デュアルポート・メモリを用いて実現する．

3.3 On-Memory FIFO 機構のプログラミング・モデル

On-Memory FIFO 機構がユーザに対して提供するプログラミング・モデルを図 3 に示す．On-Memory FIFO 機構のユーザ・インタフェースは，物理的な FIFO メモリを用いた場合とほぼ同一で，FIFO の入力ポートと出力ポートのみである．ただし，必要であれば full/empty 等の状態を知ることができる．また，メモリ上に動的に FIFO を作成する関係上，FIFO の生成／削除のための手続きは必要となるが，一旦 FIFO をメモリ上に割り当て，対応するアドレス (FIFO ID) を得れば，それ以後は FIFO ID への読み書きで FIFO 操作が可能となる．なお，FIFO ID はメッセージ・ハンドラがそれを FIFO ID として識別可能なものであれば，仮想アドレスでも実アドレスでもかまわない．以下では，簡単のため実アドレスとして考える．

On-Memory FIFO 機構のユーザに対する制限としては，出力ポートへアクセス可能なプロセッサが 1 台だけであり，それが FIFO の生成時に指定されることである．

On-Memory FIFO 機構では one-to-one 型通信，one-to-many 型通信，many-to-one 型通信をサポート可能であるが，以下では，1 対 1 通信を前提として議論を行う．

3.3.1 FIFO の生成／削除

FIFO の実体をメモリ上に割り当てるため，当該 FIFO にアクセスを行うプロセッサは事前に FIFO 生成の手続きを行わなければならない．具体的には，FIFO ID 毎の管理情報を，FIFO 管理テーブル (FMT: FIFO Management information Table) として各プロセッサの主記憶上に作成することが，その手続きである．FIFO 生成時のテーブル作成に関して，指定する必要のあるデータは以下の 4 種である．

- 受信側プロセッサ番号（受信側では不要.）
- 主記憶上に FIFO の本体を格納するためのベース・アドレス
- FIFO の長さ（長さを 1 とすると，OMF は送信側であると判断する）
- FIFO の属性（3.7.2受信．送信側では不要.）

3.3.2 FIFO 操作

FIFO を作成し，FIFO ID を獲得したあとの FIFO 操作としては，enqueue ならびに dequeue 操作がある．FIFO の 1 エントリが，プロセッサのロード／ストア命令で扱うことのできるデータサイズ以内であれば，FIFO ID への書き込み／読み出しを enqueue ／ dequeue 操作とみなすことができるが，FIFO の 1 エントリがある程度の大きさを持つ場合（本 On-Memory FIFO 機構はその例である.），当該エントリへ数回のアクセスが必要となる．そのため，enqueue ／ dequeue 操作はそれぞれ，以下の 2 フェーズで実行される．なお，この際の enqueue ／ dequeue フェーズは特定アドレス（個々の FIFO 対応に割り当てられた FIFO 制御用のアドレス）への 1 回のアクセスで実現する．

- enqueue操作:書き込みフェーズ + enqueue フェーズ
- dequeue操作:読み出しフェーズ + dequeue フェーズ

3.4 On-Memory FIFO 機構のハードウェア構成

図 4に On-Memory FIFO 機構の概要を示す．本機構は，FIFO の管理を行う FIFO 管理機構，FIFO のエントリ単位でのキャッシングを行う FIFO キャッシュ，ならびに，ネットワーク・インタフェース（以後 NIF と呼ぶ.）からなる．

3.4.1 FIFO 管理機構

FIFO 管理機構は，プロセッサならびに NIF が FIFO ID によって On-Memory FIFO へアクセスするための一種のアドレス変換機構である．FIFO ID が与えられると，主記憶上の FIFO 管理テーブル (FMT) の対応するエントリの内容をもとに，当該 FIFO の実体へのポインタを作成

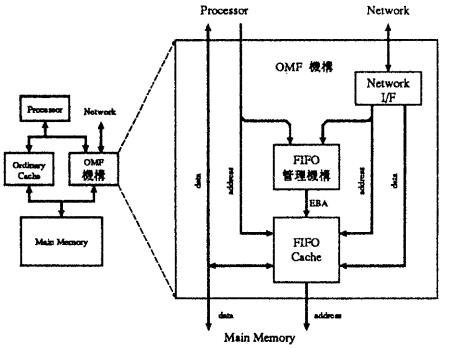

図 4: On-Memory FIFO 機構

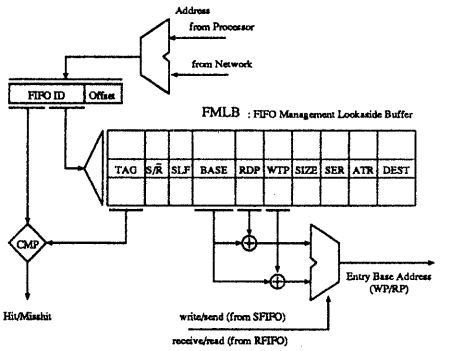

図 5: FIFO 管理機構

する．アクセス元がプロセッサか NIF か，ならびに当該 FIFO が送信側 FIFO(SFIFO) か受信側 FIFO(RFIFO) かの情報により，当該 FIFO へのリード・ポインタ (RDP) あるいはライト・ポインタ (WTP) を作成する．

このようなアドレス変換の高速化を図るため，FMT の一部をキャッシングするアドレス変換バッファ(FMLB：FIFO Management Lookaside Buffer) を設ける．図 5に FIFO 管理機構の概要を示す．

本 FIFO 管理機構は，プロセッサと NIF との共有資源である．したがって，プロセッサと NIF が同時に On-Memory FIFO にアクセスを行うと競合が発生する．しかしながら，後述するように，FIFO のエントリ・サイズが 32 バイトであり，同一の FIFO エントリにアクセスしている間は，先行するアクセスに対して得られた FIFO のポインタを使うことが可能であるため，FIFO 管理機構での競合は少ないと考えられる．

3.4.2 FIFO キャッシュ

図 6に FIFO キャッシュの構成を示す．FIFO キャッシュは，主記憶上の On-Memory FIFO 機構のエントリのみのために設けられたキャッシュである．FIFO キャッシュのデータ・アレイ部は，プロセッサと NIF からの同時アクセスを可能とするため，デュアルポート・メモリを用いて実現する．図に示すように，TAG 部は，シングル・ポートとなっている．TAG へは FIFO 管理機構からのエントリ・ベース・アドレス (EBA) でアクセスを行うが，前述の通り FIFO 管理機構でプロセッサからのアクセスと NIF からのアクセスをシリアライズしているため TAG 部へは一時には 1 つの EBA が与えられるのみである．したがって，TAG 部のデュアル・ポート化は不要である．しかしながら，一旦 EBA が与えられ，当該 EBA に対するエントリがキャッシュ上に割り当てられると，当該エントリに対する以後のアクセスでは，FIFO 管理機構や TAG 部のアクセスが不要となる．したがって，FIFO キャッシュへの，プロセッサと NIF からの同時アクセスが可能となる．また，FIFO 管理が正確に行われている限り，データアレイでのアクセス競合は発生しない．このような同時アクセスを可能とするために，FIFO キャッシュでは 2 組のアドレス生成回路を設けている．この回路は，1 つ前のアクセスに対するポインタを格納しておき，次のアクセスが同一 FIFO へのアクセスであれば，FIFO 管理機構や TAG 部を通さずに，このポインタを使って直接データ・アレイへアクセスを行う．前回のアクセス以降，そのポインタに対応する FIFO の RDP あるいは WTP の変更があった場合，ならびに FIFO キャッシュでリプレース処理が行われた場合は，アドレス生成回路のポインタを無効にし，次のアクセス時に FIFO 管理機構からのポインタ (EBA) を格納する．

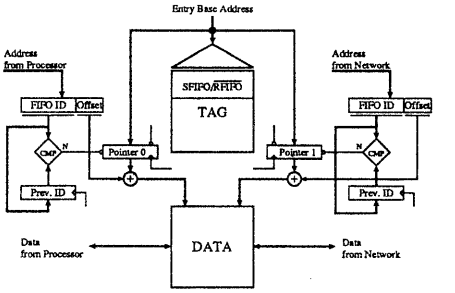

図 6: FIFO キャッシュ

FIFO のエントリ・サイズは，主記憶上の FIFO から 1 回の転送で 1 エントリ分のデータをキャッシング可能とするため，FIFO キャッシュのライン・サイズと同一にする．また，主記憶とのデータ転送のためのインタフェースをオーディナリ・キャッシュと合わせるため，FIFO キャッシュのライン・サイズもオーディナリ・キャッシュと同じにする．現在，エントリ・サイズとしては 32 バイトを考えている．

FIFO キャッシュのミス・ヒット処理は，FIFO の特性を活用し以下の方針に従って行う．ミス・ヒット処理の概要を表 1にまとめる．

表 1: ミスヒット処理

Access Type		Miss	Procedure		Remarks
Target	Action	status	Alloc.	Fetch	
RFIFO	Receive	Empty	Yes	No	方針 1
		Used	No	No	方針 2,4
	Refer	Empty	Yes	Yes	
		Used	No	No	方針 3,4
SFIFO	Send	—	Yes	Yes	方針 4
	Write	—	Yes	Yes	方針 4

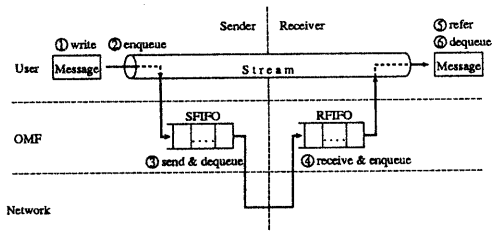

図 7: On-Memory FIFO 機構の通信処理のながれ

方針 1 RFIFO における NIF からのアクセスは，対応するキャッシュ・ライン内の全データに書き込みを行うので，ライン・アロケート時にフェッチの必要がない．

方針 2 RFIFO において，現在キャッシュ上に存在する FIFO エントリは，後から届いた FIFO エントリ（FIFO ID が同じである必要はない）よりも先に参照される可能性が高い．したがって，後続の FIFO エントリを格納すべきキャッシュ・ラインが空き状態である場合を除いて，NIF からのデータは直接主記憶に書き込む．

方針 3 一般に FIFO のあるエントリへの受信は 1 回限りである．したがって，当該エントリを格納すべきキャッシュ・ラインが空き状態である場合を除いて，主記憶からのフェッチを行わない．この場合，当該エントリへのアクセスは直接主記憶に対して行う．

方針 4 SFIFO は一度アクセスが起こると，引続きアクセスされる可能性が高い．したがって，ミスヒット時には必ずフェッチする．また，可能な限りリプレースしない．

3.5 一連の処理のながれ

図 7に従って，On-Memory FIFO 機構による通信処理のながれを概説する．

1. **write** 送信するメッセージを作成する．

2. **enqueue** メッセージが完成すると，プロセッサは enqueuing を指示する．これは各 SFIFO ごとに設けられた特定のアドレスへのライトによる．

3. **send & dequeue** OMF はプロセッサからの指示を認識し，当該 SFIFO のメッセージを送信し，削除する．

4. **receive & enqueue** ネットワークからメッセージが届くと，OMF はそのメッセージをメッセージ・ヘッダの指示する FIFO に enqueue する．

5. **refer** メッセージを参照する．

6. **dequeue** メッセージを参照し終ったら，enqueue 時と同様，特定アドレスへのライトにより，dequeuing を指示する．OMF はそれを認識し，当該 RFIFO の先頭メッセージを削除する．

3.6 On-Memory FIFO 機構の動作

通信処理における On-Memory FIFO 機構の動作は以下のようにまとめられる．

1. プロセッサからの操作に伴う動作

(a) enqueue 操作 enqueue 操作は，対応するエントリを busy 状態とし，送信側 FIFO の WTP を更新する．On-Memory FIFO 機構は busy 状態となったエントリを FMLB の DEST フィールドで示されるプロセッサへ送出する．なお，enqueue 時（書き込みフェーズ）に送信側 FIFO が full の場合（busy 状態である場合を含む），On-Memory FIFO 機構はプロセッサへの例外を発生するか，空きができるまで待つかのいずれかの処理を行う．

(b) dequeue 操作 dequeue 操作では，受信側 FIFO の RDP の更新とともに，プロセッサのオンチップ・キャッシュにキャッシングされた FIFO エントリ（ライン）の無効化を行う．純粋な FIFO 動作のみを考えると，次に RDP が当該ラインを指す以前に必ずデータが更新されるので，FIFO キャッシュの当該エントリの無効化は不要である．しかしながら，キャッシュ・ミスヒット時のリプレースによって不必要なライト・バックを起こさないため FIFO キャッシュの無効化も行う．なお，dequeue 時（読み出しフェーズ）に受信側 FIFO が empty の場合，On-Memory FIFO 機構は当該 FIFO の属性指定（3.7.2節参照）に応じ，例外を発生するか，あるいはメッセージの到着までサスペンドする．

2. NIF からの操作に伴う動作

(a) send & dequeue 操作 busy 状態になった送信側 FIFO のエントリは，メッセージとして組み立てられ，受信側プロセッサへ送出する．なお，当該エントリが既にリプレースされていた場合は，再びキャッシングした後に，送出を開始する．送出が終了すると当該エントリを busy 状態から解放し，送信側 FIFO の RDP を更新する．

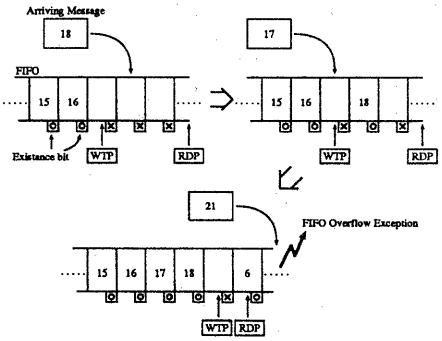

図 8: メッセージの間飛ばしエンキューイングと FIFO オーバーフロー例外の発生

(b) receive & enqueue 操作 メッセージを受信すると On-Memory FIFO 機構は，当該メッセージのヘッダに 示された FIFO ID に対応する FMLB の WTP が示すエントリにデ ータを格納する．全データの格納終了後 WTP を更新する．enqueue 時，受信側 FIFO が full の場合の処理は，プロセッサからの enqueue 操作の場合と同様である．

3.7 そのほかの機能

3.7.1 メッセージのオーダリング機構

On-Memory FIFO 機構はまた、FMT の SER フィールドを使用して、メッセージの到着順を監視する。センダ側 OMF はメッセージの所定のフィールドに、通し番号として SER フィールドの値を付加しておく。レシーバ側 OMF はそれと自分の SER フィールド の値を比較することにより、ネットワークでメッセージの追い越しがあったかどうか判断する。SER フィールドの更新はすべて On-Memory FIFO 機構が自動的に行なう．各記憶装置には 32B ごとにメッセージの存在を示すビットを用意し、追い越しがあった場合に間を飛ばしてエンキューする。FIFO が溢れると例外を発生してプロセッサに知らせる。(図 8)

SER フィールドに 1B 用意するとすると、各 FIFO ごとに 256 個のメッセージが識別できる。255 番の次は 0 番にラップラウンドする。ネットワーク内であるメッセージが、同一のプロセッサから送信された同一の FIFO に向かう 255 個のメッセージに、追い越されることがなければ良い。この機能により、FIFO に届くメッセージの順序制御を FIFO 内の検索などによりソフトウェアで行う必要がなくなり，FIFO は更に仮想化される．

3.7.2 FIFO の属性

各 FIFO には以下のような属性を設定でき、メッセージの到着時に通常とは異なる処理を行わすことができる。

属性の設定は各 FIFO に設けられた特定のアドレスに対して，属性コードをライトすることにより行う。On-Memory FIFO 機構はそれを認識し、FMT の属性フィールドを変更する。

reactive 属性 この属性を持つ FIFO へのメッセージの到着は、割り込みを発生させる。

緊急を要するメッセージに利用できる。また、リアクティブなリモート・プロシージャ・コールを行うことも考えられる。

interrupt 属性 この属性が ON の FIFO へのメッセージの到着は、割り込みを発生させる。reactive 属性とは異なり、OS のみがこの属性値を変更できる。OS が前述の FIFO のオーバーフロー例外を処理するために使用する。

underflow-exception 属性 リファレンスが行なわれた FIFO のこの属性が ON であると、FIFO アンダーフロー例外を発生する。OFF であると、当該 FIFO にメッセージが到着するまで、キャッシュ・ミスヒットとしてプロセッサを待たせる。OS のみがこの属性値を変更できる。

reject 属性 この属性を持つ FIFO に到着したメッセージは、エンキューされることなく廃棄される。選択的なマルチ・キャストをブロード・キャストの機構を利用して行う時などに使用できる。

ordered 属性 この属性が ON となっている FIFO に届くメッセージは前述のカウンタによる順序制御がなされる。OFF である FIFO に対しては順序制御が行われないので、参照順序がクリティカルでないアプリケーションや、Many-to-one 型の通信などに利用できる。

cacheable 属性 この属性が OFF である FIFO の本体は、エンキューイング時に FIFO キャッシュにキャッシングされない。定義と参照の時間関係がクリティカルでないメッセージに対する FIFO をキャッシング不可とすると、クリティカルなメッセージのリプレースを減少できる。

4 性能予測

本節では On-Memory FIFO 機構の簡単な性能予測を行う．

予測にあたって使用したシステムのパラメータを表 2 に示す．表を基に以下に示す 3 つの場合について，プロセッサ P_S が定義する 1 キャッシュ・ライン分のデータ (32B) をプロセッサ P_D が参照する時，P_S がストアし始めてから，P_D がロードし終るまでの時間を見積もる．

1. write-invalidate 型のキャッシュ・コヒーレンス機構を使用．同期には test-and-set を使用．

表 2: システムのパラメータ

ローカル・バス	バス幅	32 bits
オーディナリ・キャッシュ	サイクル・タイム	40 nsec
FIFO キャッシュ	サイクル・タイム	80 nsec
メイン・メモリ	アクセス・タイム（ワード・リード）	400 nsec
	アクセス・タイム（ワード・ライト）	160 nsec
	アクセス・タイム（ブロック・リード）	640 nsec
	アクセス・タイム（ブロック・ライト）	400 nsec
	サイクル・タイム	60 nsec
ネットワーク	ビット幅	32 bits
	サイクル・タイム	160 nsec

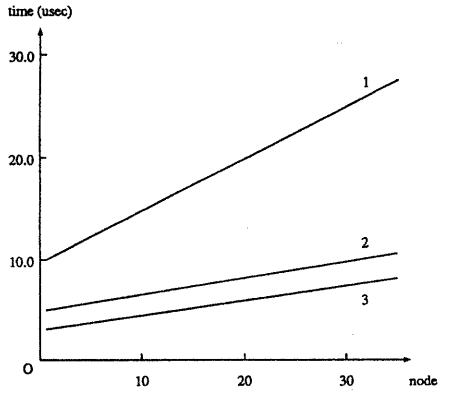

図 9: 性能予測結果

2. On-Memory FIFO 機構を使用．ただし，FIFO キャッシュは使用しない．

3. On-Memory FIFO 機構を使用．

予測結果を図9に示す．横軸は P_S，P_D 間の距離（ホップ数）を，縦軸は所要時間を表す．

On-Memory FIFO 機構は本来プログラムの繰り返し構造に対応するために考案されたものであるが，このような単純な場合でも従来の通信方式より優れていることが分かる．それは主にネットワークに同期とコヒーレンスのためのデータがながれないことに起因する．メッセージ・パッシング型の計算機では，システム・コールやプロシージャ・コールの処理時間を含まないデータの転送時間が 2. の場合と同程度となる．

5 今後の課題

本機構はまだ発展段階であり，以下に示す問題が残っている．

各キャッシュの容量 本機構では，オーディナリ，FIFO，FMLB の 3 種のキャッシュを使用する．機能や目的の異なるユニットを分割することは処理が本来持っている並列性を引き出すが，この分割により柔軟性が低くなってしまう．それが問題とならないよう，これらの容量の比を最適化することが重要であるが，それはアプリケーションに強く依存している．最適比を求めるために，アプリケーションに対する研究を進める必要がある．

FIFO の位置と柔軟な通信形態 2 章の論旨から明らかなように，本機構は「複数のプロセッサが 1 つの識別子に対して定義と参照を繰り返すことによって生じる one-to-one 型のプロセッサ間通信」を高速に行うことを主な目的として考え出された．それゆえ現在の構成では FIFO は参照側のプロセッサに張りついており，リモートな FIFO を使用した参照は行えない．しかし，もしこれが可能なら one-to-any 型，many-many 型の通信などがハードウェア的に実現できる．このような通信をハードウェアでサポートすべきか検証し，もしそうならばそれに対応できるように構造を変更する必要がある．

6 おわりに

本稿では共有アドレス空間上にとられた FIFO によってメッセージ・パッシングを行う On-Memory FIFO 機構機構を提案した．本機構においてはプロセッサ間の通信がキャッシュからキャッシュへの転送として実現されることなどにより，既存のいかなる方式よりも高速であることが予想される．4章では簡単な性能予測を行い，単純な場合でも高速に通信が行われることを示した．しかし，本機構はプログラムの繰り返し部分でこそ本領を発揮するはずである．今後は実プログラムによるシミュレーションを行い，実際に本機構が従来方式に比べてどの程度の高速化が達成されるのか示していきたい．

謝辞

日頃から御指導，御討論頂く京都大学富田研究室の諸氏に感謝の意を表します．

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–,” 並列処理シンポジウム JSPP，pp.195-202, 1992 年 6 月

[2] W.J.Dally et al.,“The J-Machine: A Fine-Grain Concurrent Computer,” in Information Processing 89, Elsevier North Holland, 1989.

[3] 上野智生,“可変構造型並列計算機のメッセージ・コプロセッサ”, 九州大学工学部情報工学科卒業論文,1991 年 2 月