

Virtual Interface Architecture の Internet 拡張方式

小林 伸治 陣崎 明

E-mail: {koba,zinjin}@flab.fujitsu.co.jp

新情報処理開発機構 並列分散システム富士通研究室

並列計算機等で利用する高速なネットワークアダプタのアーキテクチャとして VIA (Virtual Interface Architecture) が提唱されている。しかし、広域分散した並列計算機を Internet 接続して大規模な並列計算を行わせるためには、現在の VIA の規格は不十分である。そこで、我々は Internet での利用に向けた VIA の拡張を提案し、その実装と評価を行っている。本論文では拡張した VIA の方式とその実装について説明し、性能評価の結果を述べる。

An Internet Extension of the Virtual Interface Architecture

The VIA (Virtual Interface Architecture) is proposed as an architecture for the fast network adapters to be used in parallel computers. However, if we are to solve the large-scale parallel computations by connecting the widely distributed parallel computers using the Internet, the specification of the VIA is insufficient. We are proposing an extension of the VIA for use in the Internet, and are doing evaluations using our implementation. In this paper, we explain our extension of the VIA and its implementation, and present the results of the performance evaluation.

Shinji Kobayashi and Akira Jinzaki

Parallel and Distributed Systems Fujitsu Laboratory, RWCP

1. はじめに

高速な処理を必要とする分野において並列分散システムは大きな成功を納めているが、現在の大きな課題として、拡大する高速化への要求に対応するためのシステム大規模化がある。このような課題に対し、従来の専用高速ネットワークで結合した並列分散システムを ATM や WDM などの高速広域ネットワークによって結合することでより一層の大規模並列分散システムを実現しようとするアプローチが現実的になってきた。

並列分散システムを広域分散結合する時、専用ネットワークと専用プロトコルを用いるのは現実的でなく、既存の広域ネットワークを利用しながら高速な通信を実現する必要がある。すなわち広域ネットワークで標準的に利用されている IP (Internet Protocol) を通信プロトコルとして採用することが必須である。ところが、IP は通常の並列計算機のプロセッサ間通信で用いられ

るようなプロトコルと比較して処理量が重いため、この重いプロトコル処理を高速に実行する技術が必要になる。

並列分散システムで IP を利用するには大別して 2 つの方式がある。ひとつは TCP (Transmission Control Protocol) や UDP (User Datagram Protocol) といった代表的なプロトコルを用いて socket API (Application Programming Interface) でプログラミングする方法であり、もうひとつは MPI 等の並列分散向け通信プロトコルを運ぶキャリアとして IP を利用する方法『専用プロトコル over IP 方式』である。性能的には後者の方式が優れていると考えられるが、このような方式を実現するためには従来にない API を開発すると同時に効率的な『専用プロトコル over IP』の実現が大きな鍵となる。

従来我々は高速な通信を実現するため、ネットワークアダプタ上でプロトコル処理を行うアーキテクチャを提唱し、実装および評価を行って

きた¹⁾。この技術によってネットワークアダプタ上で IP 処理を高速に実現すると同時に『専用プロトコル over IP 方式』を効率的に実現するための新しい API として VIA (Virtual Interface Architecture) を拡張した Comet-VIA を検討、試作評価したので報告する。

2. VIA

VIA は Compaq、Intel、Microsoft の 3 社が協同で開発しているネットワークインタフェースのアーキテクチャであり、並列計算等のユーザプロセスから高速にネットワークインタフェースを使用することを目標にしている。仕様書の 1.0 版²⁾が公開されている。

VIA ではネットワークインタフェースカード (NIC)が複数のコンテキストを持ち、カーネル内のドライバと協調してこれらのコンテキストを仮想的なインタフェース(VI)として複数のユーザプロセスに提供する。各ユーザプロセスはそれぞれの VI を専有することができる(図 1)。

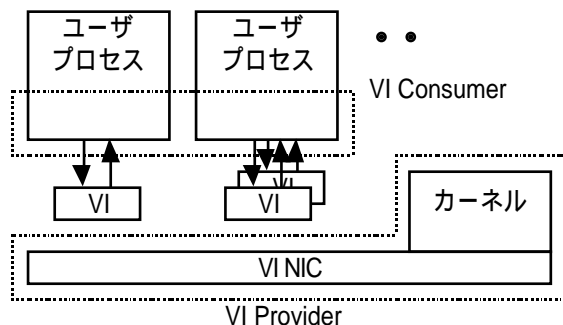


図 1: VIA のモデル

VIA に対応した NIC とカーネル内ドライバとを併せて VI Provider と呼び、VI Provider が提供する VI を利用するユーザプロセス内のライブラリまたはアプリケーションを VI Consumer と呼ぶ。各 VI は送信用、受信用の 2 つのディスクリプタキュー、およびこれらのキューにディスクリプタを追加したことを通知する Doorbell と呼ばれる機構を持つ。ディスクリプタキューはユーザプロセスのメモリ上に構成され、そのアドレスが Doorbell で通知される。VI NIC はディスクリプタを参照しながらユーザプロセスに直接 DMA

することでデータを送受信し、その結果をディスクリプタに反映する。ディスクリプタの記述からデータの送受信までカーネルの介入無しに行えるため低オーバーヘッドの通信を実現できることが特長である(図 2)。

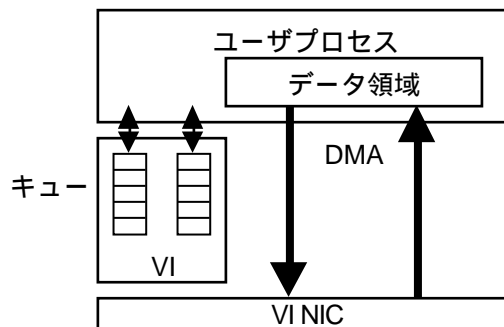


図 2: VIA におけるデータの流れ

VIA における VI はネットワークインタフェースの仮想化であると同時に、通信の終端点 (endpoint)でもある。VIA の仕様は VI 間でコネクションを確立し、これらの中でメッセージをやり取りするという通信プロトコルも含んでいる。

VIA の仕様では VI Provider と VI Consumer との間の API は規定していないが、サンプル API が付録として記述されている。Intel はこれを拡張して VIPL (VI Provider Library) という API を Intel VI Architecture Developer's Guide³⁾ の一部として公開しており、これが事実上の標準 API と言える。

ネットワークインタフェースを仮想化してユーザプロセスに直接提供するというアイデアは VIA に始まったものではない。U-Net⁴⁾や Virtual Network Transport Protocols⁵⁾を挙げることができる。VIA は仕様を定義して公開し、ベンダに広く採用を呼びかけている点がこれらの研究プロジェクトとは異なる。多くのベンダが賛同していることから、VIA は今後のネットワークアーキテクチャの一標準となる可能性を持っている。性能的にも、UCB の試験実装による評価⁶⁾によれば他の研究プロジェクトと比較して遜色がない。

しかし、Internet での通信に VIA を適用しようと

すると、現状の VIA には問題がある。Internet では、IP 上で様々なプロトコルが利用される。TCP や UDP を利用することもあれば、IP 上に他の通信プロトコルやアプリケーション独自のプロトコルを実装する場合もある。また、IP 自身も現在利用されている IPv4 (IP Version 4) には限界が指摘されており、次世代の IPv6 (IP Version 6) の開発が進められているところである。

このように多様な Internet のプロトコルに VIA を適用しようとする、仮想的なネットワークインタフェースである VI には単純な機能しか持たせることができず、IP を始めとする各種プロトコルはユーザ空間で各プロセス毎に処理せざるを得ない(図 3)。現在のようにカーネル内で IP 等のプロトコル処理を行う方式と比較するとすべてユーザ空間で処理できるため高速化の可能性はあるが、プロトコル処理をハードウェアで行うような高機能ネットワークアダプタを活かすことができない。さらに、IP の Routing Table (経路表) をプロセス間でどのように共有するかなど課題も多い。

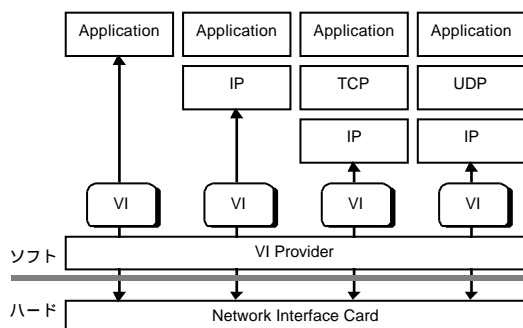


図 3: VIA での Internet プロトコル利用

3. Comet-VIA

前節で述べた VIA の問題点は、1 種類の VI しか提供できないことが根本にある。低レベルなネットワークインタフェースとして見える VI、IP の処理をしてくれる VI、TCP/IP の処理をしてくれる VI というように様々な種類の VI を提供することができれば各アプリケーションは自分が必要とする VI を自由に利用できる。しかし、たとえば TCP/IP の処理を行う VI を用意したとき、実際に TCP/IP の処理はどこが担当するかという

問題が生じる。ユーザ空間のライブラリとして実現したのではプロセス毎に実装する方式と同様に共有データの問題が発生するし、カーネルで実現するとコンテキスト切り替えが必要となり、せつかくの VIA 方式が活かせない。

そこで考えられるのが、プロトコル処理をネットワークアダプタ内で行う方式である。IP 等の複雑なプロトコルを処理できる高機能なネットワークアダプタを用意し、必要なプロトコル処理をすべてネットワークアダプタ内で行うのである。この方式ではユーザ空間から直接ネットワークアダプタを利用する VIA 方式の高速性をそのまま活かせる上、高機能ネットワークアダプタをフルに活用できる。プロトコル処理の一部はハードウェア化可能であり、ハードウェア化によって処理の高速化が期待できる。また、ネットワークアダプタ内で各プロトコルスタックを 1 つずつ用意すればよいのでデータ共有の問題も発生しない。

上記のような考え方にに基づき、VIA を次のように拡張した Comet-VIA を提案する⁷⁾。

- 1 つの VI Provider が複数種類の VI を提供できるようにする。その際、VI を通信の終端点(endpoint)には限定しない。
- 各 VI の実現に必要なプロトコル処理はネットワークアダプタで実装する。

Comet-VIA のモデルを図 4 に示す。

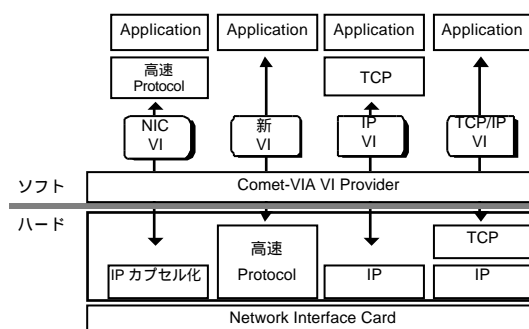


図 4: Comet-VIA のモデル

Comet-VIA ではアプリケーションが直接 Comet ネットワークアダプタを利用するため低遅延、高帯域を必要とする並列計算のようなアプリケーションに対応できる。また、プロトコル処理の一部をハードウェアで実現可能なため IP のよ

うな重いプロトコルでも高速な処理が可能である。こうした特長を利用すると、並列計算向けの高速な専用プロトコルをネットワークアダプタで高速に IP カプセル化してネットワークに送出するという方式も可能になる。すなわち、『専用プロトコル over IP』である。IP 処理のオーバーヘッドが大きければ専用プロトコルを使用する意味が無くなってしまいが、オーバーヘッドが十分小さければ IP を利用することでその専用プロトコルを広域分散環境でも利用できるように拡張したことになる。最初に述べたように、実際の広域ネットワークを利用するには IP が大前提であるため Comet-VIA のような方式で並列計算機向けの専用プロトコルを高速に IP 化できることは非常に重要である。

Comet-VIA における VI はネットワークインタフェースの仮想化であり、必ずしも通信の終端点ではない。つまり、VIA の通信プロトコルとしての側面を満たしているわけではない。しかしながら、VIA の通信プロトコルを満たす VI も Comet-VIA が提供する様々な種類の VI の 1 つとして実現することができる。

4. Comet-VIA の実装

Comet-VIA で『専用プロトコル over IP』を実現できることを検証するために、独自の専用プロトコルである Comet-FM を IP カプセル化して通信する VI を持つ VIA を実装した。この VI はいわば IP というネットワークメディアを持ったネットワークインタフェースとして動作する。

本実装ではネットワークアダプタとして我々が開発した Comet ネットワークアダプタを用いており、ネットワークインタフェース制御、ユーザパケットの IP カプセル化処理をアダプタ内部で処理している。本章では実装の詳細を説明する。

4.1. Comet ネットワークアダプタ

Comet はプロトコル処理の一部をネットワークアダプタ上のハードウェアで行うことにより通信の高速化を目指したシステムである⁸⁾。Comet ネットワークアダプタはプロトコル処理を行うエンジンを持ち、PMC (PCI Mezzanine Card) 規格のネットワークインタフェースを 2 枚搭載で

きる PCI (Peripheral Component Interconnect) 規格のカードである(図 5)。現在はプロトコル処理を Comet ネットワークアダプタ上の汎用プロセッサでソフトウェアエミュレートしているが、今後 FPGA や ASIC にしていく予定である⁹⁾。

汎用プロセッサによるエミュレーションをしている現在の試作でも、1 つのネットワークインタフェースから受信したデータを IP カプセル化して他のネットワークインタフェースに送信する処理を約 40 μ 秒で行える。ASIC 化することでさらに高速化を見込んでいる。

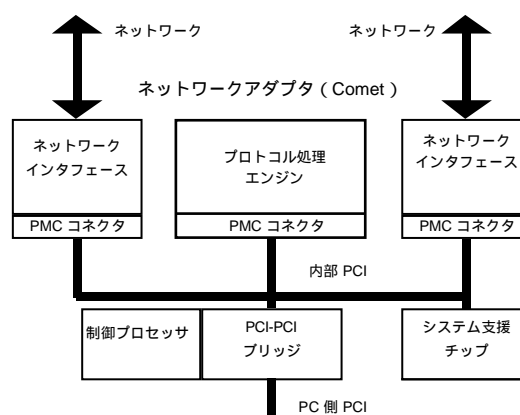


図 5: Comet ネットワークアダプタ構成

4.2. VI Provider の実装

Comet ネットワークアダプタを用いたシステムで Comet-VIA モデルを実現するため、Comet ネットワークアダプタ用の VI Provider を実装した。この VI Provider はライブラリ(VIPL)、ドライバ、Comet ネットワークアダプタ上のファームウェアから成る。OS は BSD/OS 3.1 を使用している。

VIPL は Intel Developer's Guide の API の内、Comet-VIA に必要なものを実装した。VI 間のコネクション管理や Remote DMA といった機能は VIA の VI 間通信プロトコルの一部であり、Comet-FM を IP カプセル化する VI の実現には不要なため実装していない。表 1 に実装した主な VIPL 関数を示す。Vip で始まる関数は Intel Developer's Guide の API にある関数であり、vipl_ で始まる関数は独自に追加した関数である。

表 1: 実装した主な VIPL 関数

関数	機能
VipOpenNIC()	VI Provider のオープン
VipCloseNIC()	VI Provider のクローズ
VipCreateVI()	VI の作成
VipDestroyVI()	VI の開放
VipRegisterMem()	メモリ領域の登録
VipDeregisterMem()	メモリ領域の解放
VipPostSend()	送信ディスクリプタポスト
VipPostRecv()	受信ディスクリプタポスト
VipSendDone()	送信完了チェック
VipRecvDone()	受信完了チェック
VipSendWait()	送信完了待ち
VipRecvWait()	受信完了待ち
vipl_cbuf_setup()	ディスクリプタ設定

VIA では、キューにディスクリプタを追加したことを通知する Doorbell という機構が VI 毎に必要な。本実装では、Comet ハードウェアが備えている 32bit 長の FIFO メモリを Doorbell として利用している。現在の Comet は 8 つの FIFO メモリを備えているが、システムが 2 つ利用し、各 VI 毎に送受信として 2 つずつの Doorbell が必要になるため、本実装が提供できる VI の数は最大 3 である(図 6)。

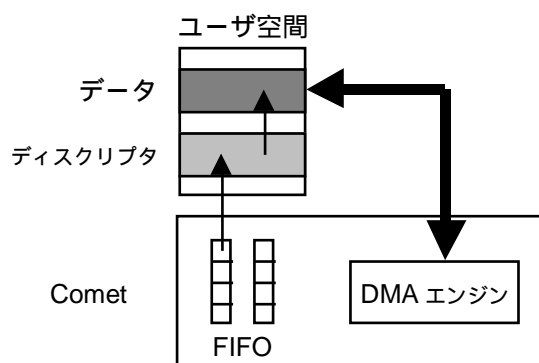


図 6: FIFO を利用した VI の実現

ディスクリプタはメモリ上の構造体として定義しており、FIFO メモリにはこの構造体へのポインタを設定する。Comet はメモリ管理機構を持たないため、このポインタは物理アドレスで与える必要がある。VIPL はディスクリプタの物理アドレスを設定するために仮想アドレスから物理アドレスに変換する必要が生じる。変換にはカーネル内関数をシステムコールで呼び出す必要があるが、変換のたびにシステムコールを呼んでいたのでは性能が大きく低下する。そこで、ライブラリ内で変換結果をキャッシュして利用するようにした。

仮想空間上で連続した領域も物理アドレス空間で連続しているとは限らない。このため、ディスクリプタは複数の領域を指示できるようにしてあり、仮想空間上の連続領域を与えると必要に応じて複数の領域に分割しながらディスクリプタに物理アドレスを設定する関数を用意した。上述のように物理アドレスへの変換はキャッシュしているため比較的高速に行えるが、最大の性能を発揮できるのはデータ領域が連続した物理アドレスに配置されているときである。この制約を無くすためにはネットワークアダプタ上に MMU のようなメモリ管理機構が必要になる。

ディスクリプタや送受信データ領域は Comet ネットワークアダプタがアクセスするため、常時物理メモリ上に存在する必要がある。VIA ではこれらの領域を登録するために VipRegisterMem() という関数を使用する。本実装では VipRegisterMem() が呼ばれたときに対象領域を固定的に物理メモリに割り当て、ページアウトされないようにする。

物理メモリに割り当てた領域は VipDeregisterMem() で解放されるまで割り当てられたままとり、先に説明した仮想アドレスから物理アドレスへの変換キャッシュの対象となる。解放時にはこのキャッシュをフラッシュすることでキャッシュが誤った値を返さないようにしている。

4.3. 専用プロトコルの実装

Comet-VIA VI Provider を利用する専用プロトコル、Comet-FM を実装した。この専用プロトコ

ルは Illinois 大学の FastMessages 1.1¹⁰⁾を基本にして設計したものである。

FastMessages 1.1 において各ノードは一意のノード番号で識別される。送信時には送信先ノード番号をアプリケーションが指定する。送信側は送信データと共に、受信側で起動すべき受信ハンドラを指定する。受信データを直接取り扱えるのは受信ハンドラのみである。

オリジナルの FastMessages 1.1 は送信処理にプログラム I/O を用いているが、Comet-FM では送受信ともに VIA の DMA を用いる。オリジナルの送信プリミティブ FM_send() はプログラム I/O であるため送信完了までブロックする。Comet-FM では DMA の特長を活かすため、ブロックしない FM_send_nw() を追加した。FM_send_nw() したデータがすべて送信されるまで待つ FM_wait() も追加した。アプリケーションプログラムは FM_wait() 等で送信完了したことが明らかになるまで FM_send_nw() で送信したデータ領域を操作してはならない。

送信バッファ領域は Comet の DMA エンジンがアクセスするため、VipRegisterMem() を用いて登録しておく必要がある。このため、送信バッファ領域を登録するためのプリミティブ FM_register()、および解放するプリミティブ FM_deregister() を追加した。アプリケーションは FM_send() や FM_send_nw() を呼び出す前に送信バッファ領域を FM_register() で登録しておく必要がある。

受信に関しては、あらかじめ一定の領域を受信バッファとして確保しておき、VipRegisterMem() および VipPostRecv() 関数で受信ディスクリプタに登録しておく。ネットワークから受信されたデータは Comet が受信バッファに DMA し、受信ディスクリプタに反映する。アプリケーションが FM_extract() を呼び出すことで受信ディスクリプタのチェックが行われ、受信されたデータがあればそのハンドラを起動する。受信ハンドラ完了時に受信データ領域は破棄されるので、必要があれば受信ハンドラは受信データをグローバル領域等にコピーする。

今回実装した Comet-FM の通信プリミティブを

表 2 に示す。

表 2: Comet-FM の通信プリミティブ

プリミティブ	機能
FM_register()	送信バッファ領域登録
FM_deregister()	送信バッファ領域解放
FM_send()	送信(ブロック)
FM_send_nw()	送信(ノンブロック)
FM_wait()	送信完了待ち
FM_extract()	受信ハンドラ起動

Comet-FM はライブラリとして実装しており、下位ライブラリとして前節で説明した Comet-VIA VIPL を利用する。Comet-VIA VIPL は FM パケットを IP カプセル化してネットワークに送出する VI を提供しており、Comet-FM のライブラリはこの VI をオープンして使用するだけで FM over IP の通信が実現できる。Comet-FM のライブラリが行うのは通常の FM ライブラリ同様、宛先ノード ID 等を含んだ FM パケットを作成するところまでであり、IP 化の処理を意識する必要は無い。IP カプセル化、および FastEthernet フレーム化は FM ヘッダに基づいて Comet ネットワークアダプタが行う(図 7)。

アプリケーション ライブラリ Comet

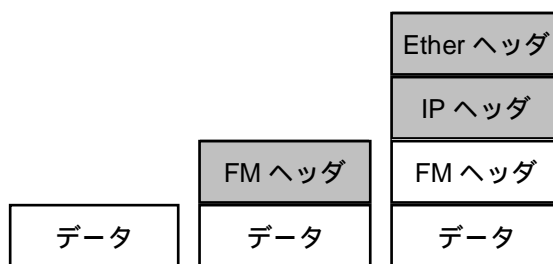


図 7: Comet-FM の処理区分

VIA における VI はコネクション型の通信終端点であり、データの送受信はコネクション確立後に行われる。このため、VIA ではディスクリプ

タやデータパケット中で宛先を指示する必要がない。しかし、Comet-FM 用の VI はそのような終端点ではなく、仮想的なネットワークインタフェースである。1 つの VI から送信されるパケットの宛先は 1 つに限定されない。このため、Comet ネットワークアダプタは FM パケットを解析して宛先 FM ノード番号を調べ、それに基づいて宛先 IP アドレス等を決定する必要がある。これを実現するためには FM ノード番号と IP アドレスとの対応表が必要である。対応表の検索はハードウェア化可能なためオーバーヘッドは小さい。Comet-VIA ではこのような対応表をユーザプログラムから設定、管理するためのインタフェースが必要である。現在の実装では Comet ネットワークアダプタのファームウェア起動時にあらかじめ固定的な対応表を設定している。

5. 評価

前節で説明した、Comet-VIA を利用した Comet-FM の実装を用いて性能測定を行った。測定は PentiumPro 200MHz の PC を 2 台用い、これらに Comet ネットワークアダプタを搭載して行った。Comet ネットワークアダプタには PMC FastEthernet カードを装着し、これらを UTP クロスケーブルで直結した。比較のために、UltraSPARC-II 250MHz のワークステーションで Solaris 2.6 上の UDP/IP の性能を測定した。

並列計算等に利用する通信プロトコルは信頼性を提供する必要がある。メッセージのいくつかは途中で廃棄されるようなことは許されない。メッセージが廃棄される可能性がある場合は確認のメッセージを応答するといった方式で信頼性を保証する必要がある。UDP/IP はメッセージ廃棄の可能性があるプロトコルであり、多くのメッセージを連続して送信するなどしてリソースが不足するとアプリケーション側に通知することなくメッセージを廃棄する。このため、UDP/IP を並列計算機向け通信プロトコルのキャリアとして利用する場合は確認メッセージによる信頼性確保が欠かせない。

一方、Comet-FM では VI に渡されたメッセージは廃棄されることなくネットワークに送出される。このため、アプリケーションは必要なだけ送信バッファやディスクリプタ領域を用意し、

アプリケーションに最適なフロー制御を行うだけでよい。

様々なサイズのメッセージを連続して送信してバンド幅を測定した。測定結果を図 8 に示す。

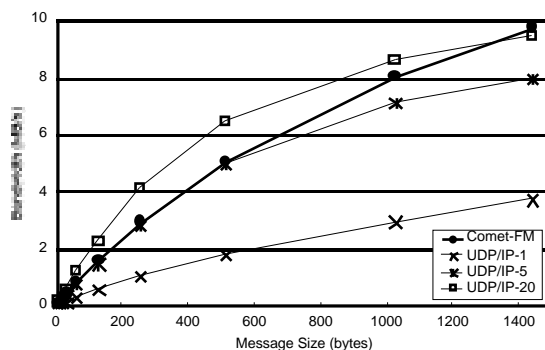


図 8: バンド幅

図 8 の UDP/IP-1、UDP/IP-5、UDP/IP-20 はそれぞれ 1 メッセージ、5 メッセージ、20 メッセージ毎に確認メッセージを送信する場合である。測定した環境ではこれ以上連続してメッセージを送信するとメッセージの廃棄により測定できなかった。20 メッセージ連続の場合もメッセージサイズが 512 バイト以上ではまれにメッセージ廃棄が起こるため、繰り返し数を少なくして測定を行った。

Comet-FM はノンブロック方式の `FM_send_nw()` を用いて送信処理を行う。送信バッファと送信ディスクリプタ領域は 5 メッセージ分用意し、フロー制御のために 5 メッセージ毎に `FM_wait()` で送信処理完了を待っている。本実装ではこれ以上のバッファを用意しても結果は変わらない。

現在の Comet-VIA 実装は汎用の DMA エンジンを使用していることやプロトコル処理をソフトウェアエミュレートしていることなどから遅延が比較的大きい。このため Comet-FM と UDP/IP-20 とを比較すると小さいメッセージサイズにおいて Comet-FM のバンド幅が低くなっている。しかし、メッセージサイズが大きくなると遅延の影響が小さくなり、1440bytes では UDP/IP-20 をわずかながら上回り、約 10MB/s のバンド幅を実現している。今後の Comet ネット

ワークアダプタでプロトコル処理をハードウェア化すれば遅延も小さくなる見込である。

今回の測定により、アプリケーションから与えられた専用プロトコルのパケットを IP 化しながら通信する『専用プロトコル over IP』でも最高約 10MB/s と FastEthernet の帯域に近い値が得られることが確認できた。より高速なネットワークや様々なアプリケーションに対応するためにはオーバーヘッドの詳細な解析が必要である。そのために各段階での処理でどの程度の遅延が生じているかを分析するのが今後の課題である。

6. まとめ

VIA を Internet 環境で利用する際の問題点について述べ、それを解決するための拡張、 Comet-VIA を提案した。Comet-VIA を用いれば様々なアプリケーションの要求に合わせた種類の VI を提供でき、Internet 環境で必要になる様々な種類のプロトコルでオーバーヘッドの少ない高速な通信が実現できる。

Comet-VIA を実証するために、高機能ネットワークアダプタ Comet 上に Comet-VIA を実装し、Comet-VIA の枠組みを用いてネットワークアダプタ内で専用プロトコルを IP 化する『専用プロトコル over IP』方式を実現できることを示した。この専用プロトコル Comet-FM の性能を評価し、FastEthernet の帯域を使い切る程度に十分高速であることを確認した。Comet-VIA のオーバーヘッドの詳細な解析は今後の課題である。

広域分散した並列計算機群を Internet を利用して接続するためには高速な IP 通信が鍵となる。そのためにはネットワークアダプタ上でプロトコル処理の一部をハードウェア化する方式が有望であるが、IP の世界では様々なプロトコルが利用されるため、Comet-VIA のように様々な種類のインタフェースを統一的に扱える枠組みが必要である。Comet-VIA により高速な IP 通信が実現できることを示したことは、広域分散した並列計算機群を相互接続して利用する環境に道が開けたことを意味する。

今後は Comet-FM 以外の VI も実装して様々なアプリケーションに対応できることを実証するとともに、Comet-VIA で必要となる API を整理し

てまとめる予定である。また、Gigabit Ethernet 等、より高速なネットワークにも適用し、Comet-VIA の評価を続けていく。

参考文献

- 1) 古賀久志, 陣崎明: IEEE1394 バスの計算機ネットワークへの適用, JSPP'98 ポスターセッション, pp.158 (1998)
- 2) Virtual Interface Architecture Specification Version 1.0, <http://www.viarch.org/>
- 3) Intel VI Architecture Developer's Guide V1.0, <ftp://download.intel.com/design/servers/vi/intel.pdf>
- 4) Welsh, M., Basu, A. and Eicken, T.: Incorporating Memory Management into User-Level Network Interfaces, *Hot Interconnects V* (1997)
- 5) Chun, B., Mainwaring, A. and Culler, D.: Virtual Network Transport Protocols for Myrinet, *Hot Interconnects V* (1997)
- 6) Buonadonna, P., Geweke, A., Culler, D.: An Implementation and Analysis of the Virtual Interface Architecture, *SC98* (1998)
- 7) 小林伸治, 陣崎明: Comet における VIA 的インタフェース, 信学技報, Vol. 98, No. 234, CPSY98-63, pp.23-28 (1998)
- 8) 陣崎明, 中村修, 村井純: 並列ネットワークサーバ Comet のアーキテクチャとその応用, 信学技報, Vol. 98, No. 234, CPSY98-62, pp.15-22 (1998)
- 9) 陣崎明, 中村修, 村井純: ギガビットルータ Comet のアーキテクチャとその評価, インターネットコンファレンス '98 論文集, pp.89-96(1998)
- 10) Pakin, S., Karamcheti, V. and Chien, A.: Fast Messages (FM): Efficient, Portable Communication for Workstation Clusters and Massively-Parallel Processors, *IEEE Concurrency*, Vol. 5, No. 2, pp. 60-73 (1997)