ハイパフォーマンス
コンピューティング 72−19
（1998. 8. 7）

Gigabit Ethernetを用いた高速通信ライブラリの設計

住元真司† 石川裕† 堀敦史†
手塚宏史† 原田浩† 高橋俊行†

我々は、Gigabit Ethernetを用いたクラスタシステム上で並列アプリケーションを効率良く稼動するための高速通信ライブラリGigaE PMを設計開発している。GigaE PMでは、並列処理に必要な信頼性のある低遅延かつ高バンド幅な通信機能を提供すると共に、従来の通信プロトコルであるTCP/IPを支援する。

GigaE PMの設計では、Gigabit Ethernetカード側にプロセッサを持ち、ファームウエアを変更できることを前提としている。ファームウエア側に並列処理用通信に特別化した信頼性のある通信機能を実現し、従来の通信プロトコル処理はホスト側のカーネルに任せる。

初期実装実験として、Essential Communications社のGigabit Ethernetカード上で、信頼性を保証しないユーザレベル通信を実現しファームウエアの性能を評価した。その結果、ユーザレベルで72.1 MB/sのバンド幅（1.5KBytes時）、ラウンドトリップ時間（ユーザデータは0バイト）で35.4 usecの遅延を実現している。

Design of High Performance Comminication Library on Gigabit Ethernet

SHINJI SUMIMOTO,† YUTAKA ISHIKAWA,† ATSUSHI HORI,†
HIROSHI TEZUKA,† HIROSHI HARADA† and TOSHIYUKI TAKAHASHI †

A high performance communication library, called *GigaE PM*, has been being designed and implemented for parallel applications on clusters of computers using Gigabit ethernet. The GigaE PM provides not only a reliable high bandwidth and low latency communication function but also supports an existing network protocols such as TCP/IP.

It is assumed that a gigabit ethernet card has a dedicated processor and its firmware can be modified. A reliable communication mechanism for a parallel application is implemented on the firmware while existing network protocols are handled by an operating system kernel.

A prototype system has been implemented using an Essential Communications gigabit ethernet card. The performance results show that 35.4 micro seconds round trip time for a zero byte user message and 72.1 MBytes/sec bandwidth for a 1.5 Kbytes message are achieved.

1. はじめに

我々は、PC又はUnixワークステーションをギガビット級ネットワークの一つであるMyrinet[1]ネットワークで多数接続したPCあるいはワークステーションから構成されるクラスタ[2]上に、SCoreクラスタシステムソフトウエアと呼ぶ高性能かつマルチユーザ並列プログラミング環境を構築してきている[3),4]。従来市販されてきたギガビット級ネットワークでは数百メートル以上の距離でかつ百台規模の計算機が繋がるようなものはなかった。従って、物理的にも一つの筐体として組み上げる、あるいはラックの中に計算機を並べると言ったシステム構築方法しか取ることが出来なかった。

近年、Gigabit Ethernetの出現により、次世代標準LANを目指し多くのベンダが市販し始めている。今後、量産により価格は下がることが予想される。Gigabit Ethernetを用いてコストパフォーマンスの高いクラスタシステム構築が可能となるだろう。さらに安価なクラスタシステムとしてだけでなく、Gigabit Ethernetを用いたLAN環境において従来のクラスタシステム上のソフトウエアが同様の性能で利用可能になると言う利点も出てくる。

しかし、現状のGigabit Ethernetを用いた通信は伝統的なOSによるTCP/IPプロトコル処理であるため、その通信性能はネットワークの物理性能を引き出せない。例えば、DEC社製Alphaプロセッサ

† 技術研究組合 新情報処理開発機構 つくば研究センタ
並列分散システムソフトウェアつくば研究室
Parallel & Distributed System Software Laboratory
TRC, Real World Computing Partnership
http://www.rwcp.or.jp

(533MHz) の計算機 (OS は Windows NT) と Packet Engine 社製の Gigabit Ethernet を用いた評価結果では、Gigabit Ethernet の物理バンド幅 125 MB/s に対して 29.6 MB/s のバンド幅しか出ないと報告されている[5]。

我々は、Gigabit Ethernet のハードウエア性能を最大限に引き出し、並列処理に必要な信頼性のある低遅延かつ高バンド幅な通信機能を提供すると共に、従来の通信プロトコルである TCP/IP を支援する通信ライブラリ GigaE PM を設計開発している。GigaE PM の設計では、Gigabit Ethernet カード側にプロセッサを持ち、ファームウエアを変更できることを前提としている。Gigabit Ethernet の通信フレームの配送は保証されていないので、ファームウエア側に並列処理用通信に特別化した信頼性のある通信機能を実現する方針をとっている。また、従来の通信プロトコル処理はホスト側のカーネルに任せる。

初期実装実験として、Essential Communications 社の Gigabit Ethernet カード上で、信頼性を保証しないユーザレベル通信を実現しファームウエアの基本性能を評価した。その結果、ユーザレベルで 72.1 MB/s のバンド幅（1.5KBytes 時）、ラウンドトリップ時間（ユーザデータは 0 バイト）で 35.4 usec の遅延が実現できることを確認している。

本稿では、まず、開発環境の概要について述べた後、GigaE PM 高速通信ライブラリで採用予定のネットワークプロトコルおよびホストとネットワークインターフェイス（以降 NIC と呼ぶ）との処理分担について述べる。最後に Essential Communications 社の Gigabit Ethernet カード上での初期実装実験結果を示す。

2. 開発環境の概要

今回の高速通信ライブラリ開発に使用している環境を表 1に示す。使用した Essential Communications 社 PCI Gigabit Ethernet NIC は、HIPPI 用のチップセットである JackRabbit を周辺回路により Gigabit Ethernet の MAC に接続した構成となっている。

表 1　開発環境

ハードウェア	Pentium 150MHz, 430FX チップセット, 64MB ファーストページメモリ
NIC	Essential 社 PCI Gigabit Ethernet NIC model EC-440-SF (33 MHz clock, DMA for PCI and MAC SEEQ 8100 MAC)
スイッチ	Extreme 社 Summit 2
ホスト OS	Redhat 5.0[6] Linux[7] (カーネルは 2.0.32)

表 2　ホスト・NIC 間メモリ転送およびシステムコールコスト

転送方法	N ワード 転送コスト
Host → NIC(HOST)	0.1 x N usec
NIC → Host(HOST)	7.9 x N usec
Host → NIC(NIC)	2.3 + 0.4 x N usec
NIC → Host(NIC)	3.0 + 0.4 x N usec
システムコール	**コスト**
ioctl	1.9 usec

注：(HOST) とはホスト側がプロセッサによりデータ転送をした場合であり、(NIC) とは NIC 側が NIC の持つ DMA 機能によりデータ転送をした場合である。

3. 高速通信ライブラリ GigaE PM の設計

我々は、プロセッサが搭載されていてかつプログラマブルな NIC を想定して高速通信ライブラリ GigaE PM を設計している。現状の NIC 上のプロセッサは最近のホストプロセッサの 1/10 以下程度の能力しかなく、メモリサイズも 1〜2MBytes と非常に限られたものである。また、ホストと NIC 間でのデータ転送遅延とシステムコールのコストの関係は計算機ハードウエアおよびオペレーティングシステムによって様々である。表 1に示した開発環境におけるこれらのコストは表 2 の通りである。

このような制限の下、数百台規模の計算機上のプロセスとの間で信頼性があり低遅延かつ高バンド幅を達成するためには、NIC 上でのプロトコル処理とホスト側での処理分担およびデータ構造の置き場所が重要となる。本節では、以下、Gigabit Ethernet 上で信頼性のある通信を実現する方式について述べた後、ホストメモリと NIC メモリの間でのデータ構造の分担方式について述べる。

3.1 信頼性のある通信の実現

我々が開発している高速通信ライブラリでは、遅延を抑えながら機能を詰め込む必要があるため、信頼性を実現する方式はできるだけシンプルで NIC の CPU に多くの処理をさせない方式を採用する必要がある。

信頼性のある通信を実現するためには、メッセージの到着保証と順序性を保証することが必須である。メッセージの到着保証を妨げる要因には、Gigabit Ethernet でメッセージ伝送中に発生するメッセージの消失と受信側のバッファ不足がある。このためメッセージの消失検出、再送の機構とフロー制御機構は必須となる。メッセージの順序性を妨げられる場合はネットワークの接続で一つのノードへのパスが複数存在する場合、及びメッセージの消失によりあとからメッセージが再送された場合である。前者の問題は適切なルート設定により回避すべきであるため、考慮しない。

我々は、メッセージの到着保証と順序性を保証するための方式として、スライディングウィンドウ方式に **STOP and GO** 方式によるフロー制御を付加した方式を採用した。

3.1.1 スライディングウィンドウ方式

スライディングウィンドウ方式は、メッセージにメッセージ番号をつけてACKを待たずに一度にいくつかのメッセージを送信して、ACKを待つ方式である。この方式はHDLC、LAPBなどの多くのプロトコルで実際に用いられている。

スライディングウィンドウ方式では、現在送信済みのメッセージ番号をiとするとi＋N（Nは定数）までのメッセージ番号を持つメッセージはACKを待つことなく送信する。受信側ではメッセージが到着するたびに受信したメッセージ番号を送信側にACKで返す。送信側は新しいACKが届くたびにiの値をi'に更新し、i'+Nまでのメッセージを新たに送信することができる。

本方式は一度に送るメッセージ数を増やすことにより、伝送路の使用効率を上げることができるが、送信に必要なバッファがメッセージ数に比例して必要になる。

ACKメッセージが到着せずにタイムアウトが発生する場合は、送信側のメッセージが消失した場合、受信側のACKメッセージが消失した場合、及びACKメッセージが送信側のACK受信が他のメッセージの受信のために遅延した場合である。タイムアウトが発生した場合のメッセージの再送は、基本的にACKを受信していない番号から後のメッセージ番号を持つすべてのメッセージを送信することによって行なわれる。なお、受信側は受信が完了していないメッセージ番号より大きいメッセージ番号を持つメッセージはすべて破棄する方式を採用する。

タイムアウト時間の設定はネットワークの構成に依存する。また、ACK送信の方式としては、1つのメッセージに対して1つの独立したACKメッセージが原則であるが、複数受信メッセージ分を一つのACKメッセージにまとめて送信する、ACKを送る相手先への他の送信メッセージにACKの情報を載せて送ることによりメッセージ数を減らすことができる。

本方式では、受信側のNIC上のバッファは受信が完了したメッセージ分の容量があれば良く、受信が完了したメッセージはホストの受信バッファへのDMA転送が終了すれば解放できるため、TCP/IPで実現されているような再送機構に比べ、NIC上の限られたメモリを用いて多数のプロセス通信を同時に扱うことが可能となる。

3.2 STOP and GO方式

STOP and GO方式は、受信バッファが不足した場合には、STOPメッセージを送信側に送信して、送信側の送信をストップさせ、バッファに空きが出た場合に送信側にGOメッセージを送信し、メッセージの送信を再開させる。メッセージの送信を止める場合と再開する場合の決定にはHigh Watermark、Low Watermarkによるものが一般的である。Gigabit Ethernetではメッセージの消失が発生する可能性があるため、STOPメッセージ及びGOメッセージはACKなど他のメッセージと合わせて再度送信する。なお、STOPメッセージを出した後に届いたメッセージはすべて破棄される。

3.3 採用方式の処理手順

採用したスライディングウィンドウ方式に**STOP and GO**方式によるフロー制御を追加した方式について述べる。各メッセージはMsg(送信ノード番号、受信ノード番号、メッセージ番号)で表現されるものとし、一度に送信可能なメッセージ数をN、タイムアウト時間をTとする。

送信ノードの処理

S1 送信ノードsはメッセージMsg(s,r,i)を受信ノードrに送信する。Msg(s,r,i)の送信バッファSBuf(r,i)は送信が終了しても解放しない。また、メッセージ送信時にはメッセージを送信した時間STime(r,i)を記憶しておき、送信済みメッセージ変数MsgIdSent(r)をiにする。

S2 送信ノードsは受信ノードrからのACKメッセージを待つことなく、送信するメッセージがある場合にはメッセージMsg(s,r,j)(MsgIdSent(r) ＜ j ＜ i+N)を受信ノードrに送信する。MsgIdSent(r)=jとする。

S3 消失メッセージLOST(r,k)(i ≦ k ＜ i+N)を受信したら、SBuf(r,j)(j ＜ k)を解放し、i=kとする。S1に戻る。

S4 STOPメッセージSTOP(r,k)(i ≦ k ＜ i+N)を受信したら、SBuf(r,j)(j ＜ k)を解放、i=kとして、以後の送信を停止する。

S4 GOメッセージGO(r,k)(i ≦ k ＜ i+N)を受信したら、S1に戻る。

S4 ACKメッセージACK(r,k)(i ≦ k ＜ i+N)を受信したら、i=kとして、S2に戻る。

S5 現在時刻とSTime(s,r,i)の時間の差がTを越えた場合には、S1に戻る。

S6 S3に戻る

受信ノードの処理

R0 受信済みメッセージ変数MsgIdRecv(s)は未代入の値で初期化され、最初のメッセージ到着時に代入されるものとする。

R1 受信ノードrはメッセージMsg(s,r,i)を受信後、受信済みメッセージ変数MsgIdRecv(s)+1==iの場合、ホストメモリ上のバッファ容量がある場合には、ホストメモリへの転送を開始して送信ノードにACK(r,i)を返送する。この時の受信バッファRBuf(s,i)はホストメモリへの転送が完了するまで解

放されない。受信済みメッセージ変数 MsgIdRecv(s)=i とする。

R2 MsgIdRecv(s)+1==i の場合でホストメモリ上のバッファ容量がない場合は STOP(r,i) を s に返す。それ以降のメッセージは廃棄する。

R3 MsgIdRecv(s)+1 < i の場合はメッセージ消失があったものとして LOST(r,MsgIdRecv(s)) を s に返す。

R4 ホスト側の受信バッファに空きが出た場合には、GO(r,MsgIdRecv(s)+1) を s に返す。それ以降のメッセージは受信する。

3.4 ホストと NIC の処理分担

我々が開発している高速通信ライブラリでは、高バンド幅、かつ低遅延通信の実現が重要となる。従って、ホストと NIC の処理分担は NIC の提供する機能に最適化すべきであるが、実際の処理分担は NIC の提供する機能により大きく変わる。ここでは、メッセージの送受信処理とどのような処理分担があるかについて整理した後、NIC の提供する機能を幾つか選択し、処理分担をどう考えるべきかについて述べる。最後に Essential 社 PCI Gigabit Ethernet NIC での処理分担について述べる。

3.4.1 メッセージ送受信処理

ここでは、ホストと NIC によるメッセージ送受信処理はメッセージ送信用ディスクリプタ、及びメッセージ受信用ディスクリプタと呼ぶバッファの情報と送受信に必要な情報を格納した領域への参照、更新により行なわれるものとして、メッセージの送受信処理について説明する（図 1参照）。

- メッセージ送信の基本処理：
  1) ホストは送信メッセージに関する情報（バッファアドレス、送信メッセージサイズなど）を送信用ディスクリプタに書き込む。
  2) NIC は送信用ディスクリプタが変更されたことを認識し、送信用ディスクリプタの情報を元にメッセージをネットワークに転送する。
  3) 転送完了後、NIC はホストに転送の完了を通知する。ホストは送信メッセージを解放する。
- メッセージ受信の基本処理：
  1) ホストは予め受信メッセージ用のバッファを獲得し、受信メッセージ用バッファに関する情報（バッファアドレス、バッファサイズなど）を受信用ディスクリプタに書き込む。
  2) NIC はネットワークからメッセージの受信があると受信用ディスクリプタから受信バッファの情報を元にメッセージを受信バッファに転送する。
  3) 転送完了後、NIC はホストに転送の完了を通知する。
  4) ホストは受信メッセージを処理した後に受信バッファを解放する。

3.4.2 NIC の持つ機能

ここでは NIC の持つ機能の中でホストとのインターフェイスに関連するものについて、主として PCI バス用の NIC についてまとめる。

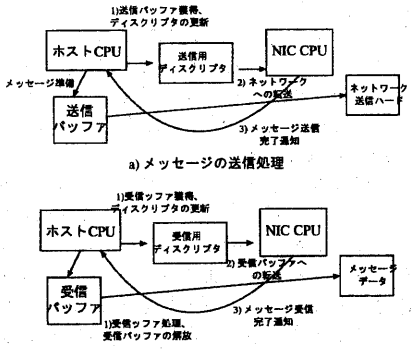

図 1 メッセージ送受信の基本処理

DMA 転送機能 PCI バス用の NIC ではほとんどの場合持っている。

ホスト CPU による NIC 上メモリの直接メモリアクセス NIC 上メモリが PCI バス上に直接マップすることが可能な場合は、ホスト CPU はホストのメモリ空間にマップすることにより直接参照、更新することが可能

NIC 上 CPU によるホストメモリアクセス NIC 上 CPU がホストメモリを直接参照、更新が可能

ホスト CPU による NIC 上メモリの直接メモリアクセスと NIC 上 CPU によるホストメモリアクセスの機能はプロテクション機能の設計に大きく影響する。この 2 つの機能をどちらも持たない（つまり、NIC のレジスタしか見えない）場合にはユーザプロセスが NIC のレジスタの参照、更新可能になるため通信の機能が停止する場合があるため危険である。

3.4.3 カーネル、ユーザと NIC での処理分担

上で述べた送受信処理においてホストと NIC の処理分担を考える上で重要な点を以下にまとめる。

(1) メッセージ送受信用ディスクリプタ：

ホストでの参照更新の主体： カーネル、或はユーザプロセスの選択肢があるが、管理者の項と同じく NIC がホスト CPU による NIC 上メモリの直接メモリアクセスと NIC 上 CPU によるホストメモリアクセスの機能をどちらも持たないばあいはユーザプロセスによる管理はすべきではない。

置き場所： ホスト上メモリ（カーネル、ユーザメモリ）、或は NIC 上メモリ上に置くかの選択になる。NIC 上の CPU が直接ホスト上メモリを参照、更新できる場合はホスト上メモリでも性能上問題になることは少ないが、それ以外の場合は DMA による転送が必要になるため NIC 上メモリに置くべきである。

管理者： ホスト CPU（カーネル、ユーザプロセス）、或は NIC 上 CPU という選択があるが、効率を考えるとホスト CPU、或は NIC 上 CPU のどちらかしかアクセスできないのは問題である。データの性質により考えるべきである。ホスト CPU 内でカーネルかユーザプロセスについては、NIC の持つ機能により選択すべきである。

NICがホストCPUによるNIC上メモリの直接メモリアクセスとNIC上CPUによるホストメモリアクセスの機能をどちらも持たないばあいはユーザプロセスによる管理はすべきではない。

(2) ホストとNICのインタラクション：実際にかかるコストにより選択すべきである。

ホスト→NIC： ホストが直接NICに通知する、或はNIC上CPUがポーリングする。

NIC→ホスト： NIC上CPUがホスト上メモリに書き込む、或はNIC上CPUがNIC上メモリを更新する。

3.4.4 Essential 社 PCI Gigabit Ethernet NIC での処理分担

設計例として Essential 社 PCI Gigabit Ethernet NIC でのホストと NIC の処理分担について検討する。この NIC は、DMA 転送機能、ホスト CPU への割り込み機能は持っているが、NIC 上 CPU によるホストメモリアクセス機能は持っていない。ホスト CPU による NIC 上メモリの直接メモリアクセスは持っているがアクセス方法が 2KB のウインドウを通してしかできないため。ユーザプロセス単位でページ単位で独立にマップするのは不可能である。

これまで検討した内容について、Essential 社 PCI Gigabit Ethernet NIC を用いた開発環境においては以下のようになる。

(1) メッセージ送受信用ディスクリプタ：

ホストでの参照更新の主体： カーネルが行なう。

置き場所： NIC 上メモリに置く。

管理者： ホスト側はカーネルと NIC 上 CPU の間でデータの性質により決定する。

(2) ホストと NIC のインタラクション：

ホスト→NIC： 表2よりホストが直接NICに通知する方式を採用する。

NIC→ホスト： 表2より、NICの DMA によりホスト上メモリに書き込む方式を採用する。

4. ファームウェア試作

性能評価のため, Essential 社 PCI Gigabit Ethernet NIC 上にファームウェアを試作した. 現状は, 最小限の 送受信機能のみを実現しており, メッセージ保証, メッセージ順序保証などの機能は実装していない.

4.1 試作ファームウェア概要

試作したファームウェアはカーネルレベル通信用とユーザレベル通信用の 2 組の送受信ポートを持つ. 各ポートの インターフェイスは NIC 上メモリに置いた RING 方式のディスクリプタにより行なう. カーネルレベル通信とユーザレベル通信の切りわけは, Ethernet フレームのタイプフィールドの値で切り分けｔている. ☆

☆ 現在は, 値 1500 以下の場合： ユーザレベル通信, それ以外： カーネルレベル通信, となっている

5. 性能評価

試作したファームウェアを評価するため、先に述べた開発環境下で性能測定を行なった。測定項目は、ハードウェア性能、実現した試作ファームウェア性能、及び、機能追加の際の参考にするために ping-pong プログラムにおけるファームウェアの処理時間である。

5.1 ユーザレベル通信バンド幅

ユーザプロセスにおけるメッセージ送信バンド幅の結果を図 2に示す。結果より、1500 バイトメッセージ (Ethernet フレーム長は 1514 バイト) 時の送信バンド幅は、72.1 MB/s であった。

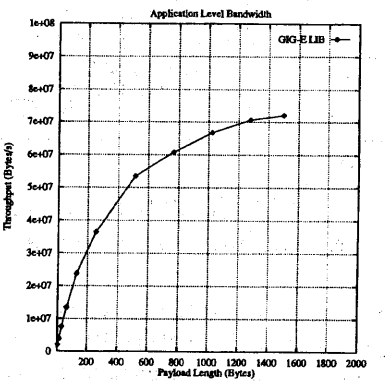

図 2 ユーザレベル通信スループット

5.2 遅　　延

遅延を測定するために、ユーザプロセス 2 ノードで ping-pong プログラムを実行しラウンドトリップタイムを測定した。スイッチの遅延を見るためにスイッチを接続した場合と接続せずに 2 つの計算機をつなげた場合の 2 通りで測定した。測定結果を 図 3に示す。

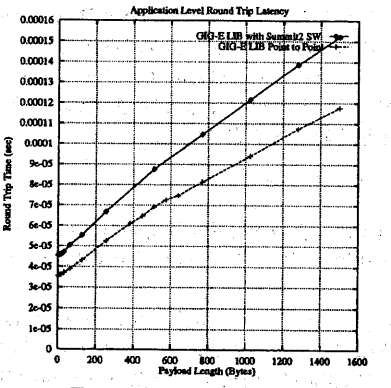

図 3 ラウンドトリップタイム

測定結果より、0バイトユーザメッセージ (Ethernet ヘッダのみ) の時に 1/2 ラウンドトリップタイムで 17.7 usec の遅延、Summit 2 スイッチ接続にすると、22.7 usec である。スイッチの遅延は 5 usec(1500 バイトメッセージ時 17.5 usec) と大きい。

5.3 ping pong プログラムでの NIC での処理時間

図 4に ping pong プログラムにおけるファームウェアの処理時間を示す。

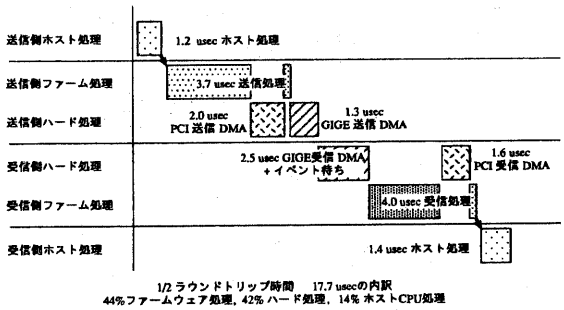

図 4　1/2 ラウンドトリップにおける処理

ファームウェアには 最適化の余地があるが、全体の時間の内、ファームウェア処理の占める割合は 44%、ハードウェアによるデータ転送時間が 42% であった。

6. 関連研究

ユーザレベル通信を行なっている例としては、UNET[9),10)]があるが、NIC のファームウェアを変更するアプローチではなくホスト上のプロセッサで処理を行なう点と UNET 自体では、フロー制御、メッセージ到着保証のための機構を持っていない点が異なる。

VIA[11)]は、Compaq、Intel、Microsoft が中心になって標準化を進めているインターフェイスを共通化するアーキテクチャであり、ユーザレベル通信、Zero-Copy 通信の枠組を持っている。我々は性能優先のアプローチを取っているため、独自にインターフェイスを最適化している. 今後出てくる VIA 搭載のシステムと性能を比較検討する予定である。

7. まとめ

本稿では、現在開発中の Gigabit Ethernet を用いた高速通信ライブラリ GigaE PM の設計について述べた。

NIC 上のプロセッサ能力、ホストと NIC 間でのデータ転送コストおよびシステムコールのコストを考慮し、Gigabit Ethernet 上で数百台規模の計算機上のプロセスとの間で信頼性があり低遅延かつ高バンド幅の通信を実現するために、NIC 上でのプロトコルの設計およびデータ構造の配置を決めた。初期実装実験として、Essential Communications 社の Gigabit Ethernet カード上で、信頼性を保証しないユーザレベル通信を実現しファームウエアの基本性能を評価した。その結果、ユーザレベルで 72.1 MB/s のバンド幅 (1.5KBytes 時)、ラウンドトリップ時間 (ユーザデータは 0 バイト) で 35.4 usec の遅延が実現できることを確認した。

初期実装実験に利用した Essential 社 PCI Gigabit Ethernet NIC は Gigabit Ethernet 初期の製品ということもあり、Gigabit Ethernet の性能を フルに引き出せるだけのハードウェアにはなっていない。また Switch の遅延が大きいのも問題である。今後、設計したプロトコルを実装し、実際のアプリケーション上で評価を行なう予定である。

参考文献

1) http://www.myri.com.
2) 手塚宏史, 堀敦史, 石川裕, 曽田哲之, 原田浩, 古田敦, 山田努. PC とギガビット LAN による PC クラスタの構築. 計算機アーキテクチャ研究会資料, 96-ARC-119, pp. 37–42. 情報処理学会, August 1996.
3) 堀敦史, 手塚宏史, 石川裕. ギャングスケジューリングの高速化技法の提案. 並列処理シンポジウム JSPP'98, pp. 207–214. 情報処理学会, June 1998.
4) Hiroshi Tezuka, Francis O'Carroll, Atsushi Hori, and Yutaka Ishikawa. Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication. In *IPPS/SPDP'98*, pp. 308–314. IEEE, April 1998.
5) http://www.packetengines.com/products/performance/gnicntperf.htm.
6) http://www.redhat.com.
7) http://www.linux.org.
8) 手塚宏史, 堀敦史, 石川裕. ワークステーションクラスタ用通信ライブラリ PM の設計と実装. 並列処理シンポジウム JSPP'96. 情報処理学会, June 1996.
9) Matt Welsh, Anindya Basu, and Thorsten von Eicken. ATM and Ethernet Network Interfaces for User-level Communication. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP)*, December 1995.
10) Anindya Basu, Vineet Buch, Werner Vogels, and Thorsten von Eicken. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *Proceedings of the Third International Symposium on High Performance Computer Architecture (HPCA)*, February 1997.
11) http://www.viarch.org/.