ハイパフォーマンス
コンピューティング 72−20
（1998. 8. 7）

SMPクラスタ上でのMPIの実装と評価

高橋俊行† Francis O'Carroll†† 堀 敦史†
手塚宏史† 住元真司†
原田 浩† 石川 裕†

SMPクラスタ上で稼動するMPI通信ライブラリとしてMPICH実装を基にしたMPICH-PM/CLUMPと呼ぶライブラリを実現した。MPICH-PM/CLUMPでは、SMP間での通信性能を向上するために0コピー通信を、SMP内での通信性能を向上するために1コピー通信を実現している。本論文では、SMP内での1コピー通信を実現するために、通信ドライバ内に他のプロセスのデータ読み込み機能を設計し、本機能を用いたgetプロトコルをMPICHに導入している。

Intel社Pentium II 400 MHz Dual構成のPC4台をMyricom社Myrinetネットワークでつなげたクラスタ上で、MPICH-PM/CLUMPを実装評価している。SMP内の通信性能は既存MPICHの実装に比べ2倍のバンド幅である140 MBytes/sec (200 KBytes転送時)を達成している。

Implementation and Evaluation of MPI on a SMP Cluster

Toshiyuki Takahashi ,† Francis O'Carroll ,†† Atsushi Hori ,†
Hiroshi Tezuka ,† Shinji Sumimoto ,† Hiroshi Harada †
and Yutaka ISHIKAWA†

An MPI library, called MPICH-PM/CLUMP, on a cluster of SMP's has been implemented. MPICH-PM/CLUMP realizes zero copy message passing between nodes while one copy message passing within a node to achieve high performance communication. To realize the one copy message passing mechanism on a SMP, a kernel primitive, which enables a process to read data of another process, has been designed. The *get* protocol using this primitive is introduced to MPICH.

The MPICH-PM/CLUMP has been running on a SMP cluster consisting of four Intel Pentium II 400 MHz dual processors. MPICH-PM/CLUMP is twice faster than the standard MPICH implementation on a SMP, i.e., 140 MBytes/sec for the 200 KBytes message size.

1. はじめに

複数のプロセッサが搭載されたSMP計算機が普及し、市販の高速ネットワークでつなげたSMP計算機によるクラスタシステムが比較的安価に実現できるようになった。このようなSMPをノードとするクラスタ（今後SMPクラスタと呼ぶ）におけるSMP間のデータ授受ではメッセージ通信機構が使われる。このような実行モデルに対して、プログラミングレベルでは次のようなモデルが考えられる。

(1) SMPクラスタの実行モデルをそのまま利用する。すなわち、SMP内では、一つのプロセス内に生成されるマルチスレッドおよび共有メモリによる並列プログラミングを行い、SMP間ではプロセスとメッセージ通信による並列プログラミングを行う。

(2) SMP間で分散共有メモリを実現し、ユーザに対しては共有メモリとマルチスレッドによるプログラミングを提供する。

(3) ノード内の各プロセッサにプロセスを割り当て、プロセスとプロセス間のメッセージ通信による並列プログラミングを提供する。

我々は3番目のプログラミングを支援する目的で、SMPクラスタ上の実行時環境であるSCore3.xならびにMPI通信ライブラリ実現の一つであるMPICHを基にした通信ライブラリMPICH-PM/CLUMP*を開発した。MPICH-PM/CLUMPは、既に我々が開発した単一計算機によるクラスタ環境用の通信ライブラリであるMPICH-PMをSMPクラスタ上で稼動するように

† 新情報処理開発機構
Real World Computing Partnership
†† エム アール アイ システムズ
MRI Systems, Inc.

* CLUMPとはCLUster of SMPの略である。

表1 SMP クラスタのハードウエア仕様

プロセッサ	Intel Pentium II
クロック	400 MHz
チップセット	440BX
メモリ	256 MBytes
プロセッサ数	2
ノード数	4
ネットワーク	Myrinet
ネットワークバンド幅	1.28 Gbits/sec/link

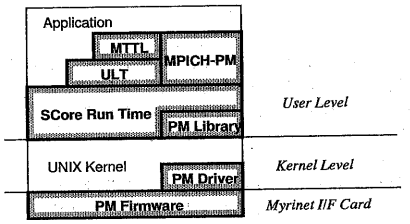

図1 SCore クラスタシステムソフトウエアアーキテクチャ

改良したものである。MPICH-PM/CLUMP により、既存の MPI で記述されているプログラムを改変することなく SMP クラスタ上で稼動することが可能となる。

SMP 上での MPI 通信実現では、通信バッファ領域として共有メモリ領域を確保して、送信側受信側がデータの授受を行う方法が一般的である。この実現ではデータが2回コピーされており、このことが、通信遅延の増大およびバンド幅低下の原因となる。MPICH-PM/CLUMP では、通信ドライバ内にプロセス間でのメモリ読み込み機能を実現し、SMP 内のプロセス間では1回のデータコピーだけで通信が実現できるようにした。

本論文では、第2章において本論文の背景として、我々が使用している SMP クラスタのハードウエア構成ならびにソフトウエア環境である SCore システムについて紹介する。SCore システムにおける SCore2.x ランタイムは単一プロセッサをノードとするクラスタにおける実行時環境を提供している。第3章では、SMP をノードとするクラスタにおける実行時環境である SCore3.x の設計および基本評価について述べる。第4章では、SCore3.x 上に実装した MPI である MPICH-PM/CLUMP の設計および基本評価について述べる。まず、通信ドライバ内に他のプロセスのデータ読み込み機能を設計し、本機能を用いた get プロトコルを MPICH に導入する。第5章では関連研究について述べる。

2. 背　　景

2.1 ハードウエア構成

表1に SMP クラスタのハードウエア仕様を示す。一つのノードに2台の Pentium II 400MHz が接続され、ノード間は Myrinet ネットワークで接続されている。

表2 SCore ランタイム プリミティブの一部

プリミティブ名	機能
_score_get_send_buf	通信バッファの取得
_score_send_message	メッセージ送信
_score_peek_network	メッセージ到着の確認
_score_recv_message	メッセージ受信

2.2 SCore クラスタシステムソフトウエア

我々はカーネルを変更することなく、通信ドライバとユーザレベルプログラムおよびデーモンプロセスにより高性能マルチユーザ環境を実現している[1)]。通信ドライバ PM、大域オペレーティングシステムランタイム SCore、MPI 通信ライブラリ MPICH-PM、ユーザレベルスレッドライブラリ ULT、マルチスレッドテンプレートライブラリ MTTL、プログラミング言語 MPC++ を総称して SCore クラスタシステムソフトウエアと呼んでいる。SMP クラスタ上での SCore システムのソフトウエアアーキテクチャの概要を図1に示す。

SCore が仮定している実行モデルでは、各ノード上に同一のプログラムコードのプロセスが生成され、ネットワークを介して通信によりお互いにデータ授受を行い処理を進めていく。これらプロセス集合を並列プロセスと呼び、個々のプロセスを要素プロセスと呼んでいる。

2.2.1 SCore ランタイム

SCore2.x ランタイムは、並列プロセスの生成消滅および並列プロセスの要素プロセス間通信を支援している。SCore2.x ランタイムでは、オペレーティングシステムおよび通信プリミティブを隠蔽し、システムに依存しないランタイムインターフェイスを提供している。SCore2.x ランタイムの通信に関するプリミティブの一部を表2に示す。通信機能の基本プリミティブ以外に、リモートメモリアクセス機能に対するプリミティブも提供されている。SCore2.x ランタイムは、単一プロセッサをノードとするクラスタ構成の上で、Sun OS4.1.X、NetBSD、Linux オペレーティングシステム上で稼動している。

2.2.2 PM

Myrinet のハードウエア性能を引き出すために、PM[2)~4)] と呼ばれる低レベル通信機能を実現している。PM は信頼性のある非同期メッセージ通信をサポートしている。PM では、ユーザプログラムが Myrinet インタフェースを直接アクセスしてデータ転送を行なうため、システムコールや割り込みなどのオーバヘッドがなく、低い通信レイテンシと高いデータ転送バンド幅を実現している。

さらに、PM は以下のような機能を持っている。

リモートメモリアクセス

送信ノードのユーザアドレス空間のデータを受信ノードのアドレス空間にゼロコピーで直接書き込むリモートメモリライト機能をサポートしている。リモートメモリライト機能では、メッセージ転送で必

要となる主記憶内のデータコピーが不要であるため、高いバンド幅のデータ転送が可能となる[5),6)]。

複数の通信チャネル

複数の独立した通信チャネルを持ち、複数のプロセス / スレッドが同時に Myrinet ネットワークを使用することができる。

ネットワークコンテキストスイッチ

ギャングスケジューリングをサポートするために、各通信チャネルのコンテキストを退避 / 回復する機能をサポートしている。

2.2.3 MPICH-PM

MPICH-PM は、MPICH[7)] を基にして PM が持つリモートメモリアクセス機能を利用した通信ライブラリであり、単一プロセッサ構成のクラスタシステム上で稼動している[8)]。MPICH では、通信のためのプロトコルとして eager プロトコルと rendezvous プロトコルの 2 つを用意している。

eager プロトコルでは、メッセージ送信プリミティブが発行されると直ちにメッセージが受信側に送られるプロトコルである。メッセージが受信側に到着したとき、受信プリミティブが発行されていない場合、メッセージはバッファに格納される。その後、受信プリミティブが発行されたとき、バッファから受信領域へのデータコピーが行われる。

rendezvous プロトコルでは、メッセージ送信プリミティブが発行されると実際のデータ転送は送られず、制御メッセージが受信側に送られる。受信側が受信プリミティブを発行した時に、制御メッセージと同期がとられ、受信側から送信側に対してデータ転送要求のメッセージが送られる。送信側はデータ転送要求のメッセージを受けてリモートメモリライト等の機能を用いデータ転送を行う。

MPICH-PM では rendezvous プロトコルと、PM がもつリモートメモリライト機能を用い、ゼロコピー通信を実現している。

計算機ハードウエア構成に依存するが、データ転送サイズが小さいときには eager プロトコルが有利でありデータ転送サイズが大きいときは rendezvous プロトコルが有利となる場合がある。MPICH-PM の実装では、データ転送サイズによって 2 つのプロトコルを切り替えることができる。すなわち、ユーザがアプリケーション起動時に指定した切り替えデータ転送サイズより、小さいサイズのデータの転送には eager プロトコルを用い、それ以上のものには rendezvous プロトコルを用いることができるようになっている。現在の実装では、このプロトコル切り替えのデータサイズは PM の MTU である 8KBytes 以下である。

3. SCore3.x

3.1 設計・実装

SCore2.x ランタイムは単一プロセッサをノードとするクラスタを仮定し、一つのノードに並列プロセスの要素プロセス一つを割り当てていた。SCore3.x では、SMP をノードとするクラスタ上での実行環境も考慮し、一つのノードに並列プロセスの要素プロセスを複数割り当てられるようにしている。要素プロセス間のメッセージ通信インターフェイスは、SMP 内の要素プロセス間、異なるノード上の要素プロセス間でも同一としている。すなわち、SCore ランタイムにおける送信プリミティブにおいてプロセス番号による送り先を指定するだけで、SCore3.x ランタイムが送り先のノード番号とノード内のプロセスを同定する。SMP 内の要素プロセス間の通信では共有メモリによる通信を実現し、ノード間の通信を必要とする場合には PM による通信を実現している。以下、第 3.1.1 節および第 3.1.2 節において、それぞれ、ノード内通信、ノード間通信について説明する。

3.1.1 ノード内通信

同一ノード内に存在する要素プロセスに対してバッファ領域として共有メモリ領域を確保して、これら要素プロセス間での通信を実現する。送信側が送信データを共有メモリに書き込み、受信側は共有メモリから受信領域にデータをコピーする。本メッセージ通信では、送信側プロセスから受信側プロセスへのデータ転送にはプロセッサによる 2 回のデータコピーが含まれていることに注意して欲しい。

3.1.2 ノード間通信

ノード間通信では、ノード内のある要素プロセスから別のノード内のどの要素プロセスへも自由にデータを運べるようにするため、通信路の多重化が必要である。多重化には PM のチャネルを用いる。PM は、あるノードのチャネルと他のノードの同一チャネル番号を持つチャネル間での独立な通信を可能にしている。そこで SCore3.x ではノード内に生成された各々の要素プロセスに一つの PM チャネルを割り当て、要素プロセスが独立して PM チャネルからデータを受信できるようにしている。送信側ではチャネルが共有資源として扱われ、排他的に使用される。図 2では、ノード 1 上に存在する要素プロセス PN#3 がチャネル 1 番を割り当てられていて、ノード 0 上の要素プロセス PN#0 が PN#3 にデータを送る状況を示している。要素プロセス PN#0 はチャネル 1 番を用いてデータを送るが、送信の際には SCore3.x ランタイムがチャネルの排他制御を行っている。

3.2 評　　価

ノード内通信の性能とノード間通信の性能を図 3に示す。図中、比較のためにメモリコピーのバンド幅も示

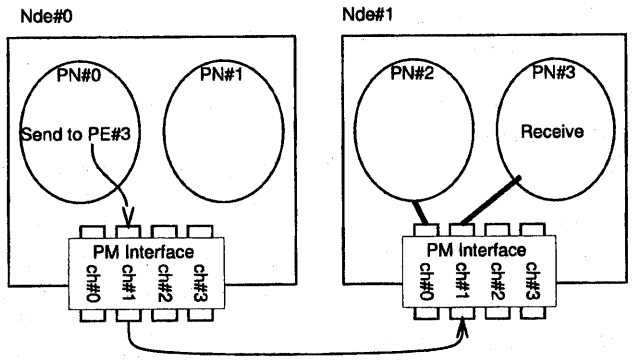

図2 SMP クラスタにおけるノード間通信例

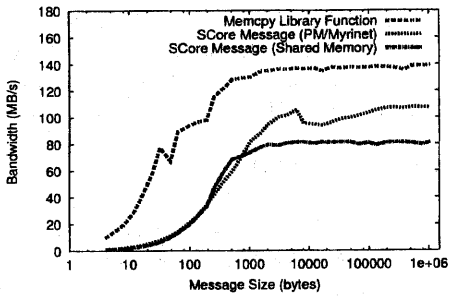

図3 SCore ランタイム /PM の通信性能

す。図からわかるとおり、ノード内通信のバンド幅はメモリコピーのバンド幅に比べて 1/2 しか出ていない。これは、プロセッサによる 2 回のデータコピーに起因する。

4. MPICH-PM/CLUMP

4.1 設計・実装

前章で述べた SCore3.x が提供する通信機能を用いれば MPICH-PM を容易に SMP クラスタ環境上に移植することができる。しかし、第 3.2節で述べたとおり、SCore3.x における共有メモリ上のメッセージ通信機能ではプロセッサによる 2 回のメモリコピーが生じるためにハードウエアの持つメモリバンド幅を生かすことが出来ない。MPICH-PM においてはノード間通信性能を向上するために PM のリモートメモリライト機能を用いたゼロコピー通信を用いた実装を行った。SMP クラスタ環境では、SMP ノード内通信でのメモリコピーをなくすことはできないが 1 回のコピーで済むような機構を設計し実装する。実装された MPI 通信ライブラリは MPICH-PM/CLUMP と呼ぶことにする。

本章では、以降、第 4.1.1節において、PM 通信ドライバ内にプロセス間でのメモリ読み込み機能である_pmVMRead 関数を導入している。第 4.2節において、1 回のデータコピーだけで通信を可能とするために、_pmVMRead 機能を用いた MPICH の get プロトコルについて述べる。

4.1.1 プロセス間でのメモリコピー機能

まず、プロセスが他のプロセスのメモリ空間をアクセスしてデータの授受を行わせる方法として従来から提供されてきているユーザアドレス空間をマッピングする方法（例えば Mach のメモリオブジェクト）あるいは共有メモリ機能（例えば Unix 系の shmem 機能）では、データコピーが 2 回生じることを示す。

アプリケーションプログラムがユーザアドレス空間をマッピングして共有アドレス空間上でデータを処理している限りは、データコピーは生じない。MPI のような通信ライブラリをユーザアドレス空間をマッピングする機構を使って実現することも可能である。しかし、この場合 MPI ユーザは送信あるいは受信時に使用するメモリ領域を仮想アドレスが使用される前にあらかじめマッピングする必要がある。そこで一般的には MPI のような通信ライブラリは共有メモリ上に設けた通信バッファ領域を介してデータの送受を行う。バッファを介する場合は、送信側からバッファへ、バッファから受信側への 2 回のコピーが不可欠である。このように従来のカーネルが提供しているメモリ管理機構では MPI のような通信ライブラリ実現にデータコピーが 2 回生じてしまう。

そこで、従来のアドレス空間のマッピングや共有メモリ機構の代りに、他のプロセスの任意のアドレス空間と自分のアドレス空間とのデータ転送を可能とするカーネルプリミティブを設計し実装する。これにより、MPI ユーザの送信データアドレスから受信側のメモリアドレスに直接データをコピーすることが可能となる。

カーネルプリミティブとして実現するために、安全性、プロテクションの観点から次のような 2 つの設計が考えられる。

(1) サーバ・クライアント関係にあるプロセス間でデータ転送を可能とする。IPC におけるサーバ・

クライアント実現のような well known port あるいはファイル名を用いて、サーバおよびクライアントがアクセス権のネゴシエーションを行い、クライアントプロセスからのメモリアクセスに対する read/write 権限を許可する。権限を許されたプロセスのみが、その権限内での操作を可能とする。クライアントプロセスに write 権限を渡すのは危険であるが、これはサーバプロセス設計者の責任となる。

(2) 親子関係にあるプロセス間のみがデータ授受を可能とする。UNIX のようにプロセスの生成に起因した親子関係を持つプロセス間にのみプロセスのメモリアクセスを許す。アクセス権限を設定する枠組みは別途何らかの枠組みを用意する必要がある。

今回の実装では、MPI 等の通信ライブラリを実現するために必要なプロセス間でのメモリアクセス機構実現が主であるため、上記 (2) を基にして、以下に示す簡素化した機能を実現することにした。

(1) 親子関係にあるプロセス間のみが他のプロセスのメモリ空間に対して読み込みを可能とする。

(2) PM デバイスドライバにプロセス間メモリ読み込み機能を実現する ioctrl を追加し、この ioctrl を発行する PM ライブラリ関数_pmVMRead を提供する。

4.2 get プロトコル

第 2.2.3節において、MPICH では eager プロトコルと rendezvous プロトコルがあることを説明した。MPICH ではさらに get プロトコルが定義されている。前節で導入した_pmVRead 機能を用いた get プロトコルの実現を図 4および以下に示す。

(1) 送信側から受信側に対して receive_request 制御メッセージを送出する。

(2) 送信側は受信側からの receive_done 制御メッセージを待つ。

(3) 受信側は receive_request 制御メッセージを受付けて、かつ MPI 受信プリミティブが呼ばれたら_pmVRead により送信側のデータを受信側にコピーする。

(4) 受信側はコピー完了後、receive_done 制御メッセージを送信側に送出する。

4.3 基本評価

_pmVMRead のバンド幅を図 5に示す。図中、比較のためにメモリコピーのバンド幅も示す。_pmVMRead はデータサイズが小さいときにはメモリコピーに比べバンド幅が低いが、3Kbyte を超えるデータサイズではメモリコピーとほぼ同等のバンド幅になる。これは、_pmVMRead の呼び出し 1 回あたりに 5usec 弱のオーバヘッドがかかっているためである。このオーバヘッドは ioctrl システムコールにおけるカーネルへのコンテキストスイッチに起因するものである。

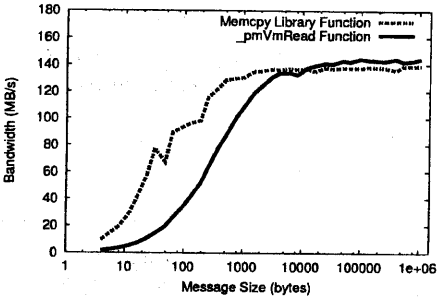

図 5 _pmVMRead のバンド幅

MPICH-PM/CLUMP の一体一通信のバンド幅を図 6に示す。図では、MPICH-PM/CLUMP のノード間通信における eager プロトコル、rendezvous プロトコル、ノード内通信における eager プロトコル、get プロトコルに加え、比較のために、MPICH 1.1 に含まれる lfshmem デバイスの通信バンド幅も示した。lfshmem デバイスは、我々の SMP クラスタ環境において、MPICH 1.1 に含まれるいくつかの SMP に対応したデバイスコードのなかで最大のバンド幅を出すものである。このデバイスコードにはノード間通信のためのコードは含まれていない。

MPICH-PM/CLUMP のノード間通信の結果は、eager プロトコルに有利なものであった。eager プロトコルと rendezvous プロトコルの差は実験に使用するクラスタ環境によって変わる。今回使用したクラスタ環境は、メモリコピーのバンド幅が非常に高いものであることが影響している。評価の際には実験が一対一通信であることと、rendezvous プロトコルのほうがメモリバスへの負荷が低いことにも留意する必要がある。

MPICH-PM/CLUMP のノード内通信の結果は、共有メモリを用いた通信を行う eager プロトコルや lfshmem デバイスにくらべ、_pmVmRead を用いる get プロトコルが、メッセージサイズが 4Kbyte より大きいときに有利であることを示している。そして、メッセージサイズが約 200Kbyte の時には get プロトコルのほうが共有メモリを用いるものよりも、2 倍高いバンド幅である 140Mbyte/s を達成している。

5. 関連研究およびシステム

先に述べた通り、MPICH の SMP 実装では共有メモリを用い 2 回データコピーによる実装が行われている。SMP 上の MPI 実装という観点から TOMPI(Threads Only MPI)[9] は、MPI で記述されたプログラムをコンパイラによって SMP 上のスレッドで稼動するプログラムに変更する。

実現したカーネルプリミティブである_pmVMRead と同様に他のプロセス空間をアクセスする機能として

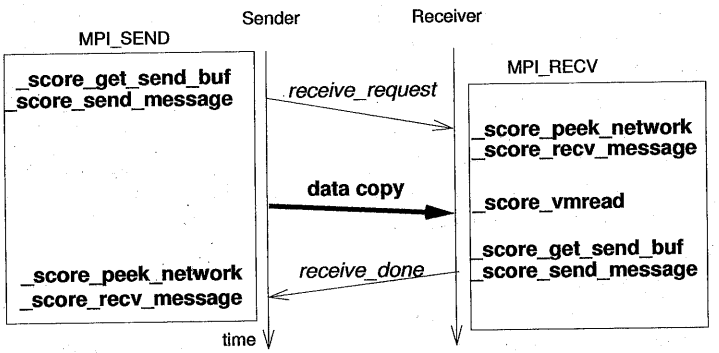

図4 getプロトコル

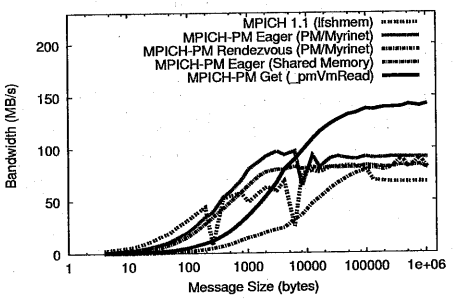

図6 MPICH-PM/CLUMPの一対一通信バンド幅

は、Linuxの/procファイルシステムを利用する方法がある。しかし、Linuxでは、主記憶上に存在しているメモリしかアクセスできないと言う制限があり、今回の実装には使用できない。

カーネルプリミティブの観点からは、Linuxのpreadプリミティブはファイルに対して、任意の領域のデータを読み込むことができる。ユーザプロセスのメモリ領域をファイルとして扱うことができれば、_pmVMReadと同様の機能が実現できることになる。

6. まとめ

本論文では、SMPクラスタ上で稼動するMPI通信ライブラリとしてMPICH実装を基にしたMPICH-PM/CLUMPと呼ぶライブラリの設計、実装、評価について述べた。SMP間での通信性能を向上するために0コピー通信を、SMP内での通信性能を向上するために1コピー通信を実現している。まず、MPICH-PM/CLUMPの実行時環境であるSCore3.xランタイムを設計した。次に、SMP内での1コピー通信を実現するために、通信ドライバ内に他のプロセスのデータ読み込み機能を設計し、本機能を用いたgetプロトコルをMPICHに導入した。

Intel社Pentium II 400 MHz Dual構成のPC4台をMyricom社Myrinetネットワークでつなげたクラスタ上での評価の結果、SMP内の通信性能は既存MPICHの実装に比べ2倍のバンド幅である140MBytes/sec (200KBytes転送時)を達成した。

参考文献

1) 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.
2) 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.
3) Hiroshi Tezuka, Atsushi Hori, Yutaka Ishikawa, and Mitsuhisa Sato. PM: An Operating System Coordinated High Performance Communication Library. In *High-Performance Computing and Networking '97*, 1997.
4) http://www.rwcp.or.jp/lab/pdslab/pm/.
5) 手塚, 堀, O'Carroll, 原田, 石川. ピンダウンキャッシュを用いたユーザレベルゼロコピー通信. 情報処理学会研究報告. 情報処理学会, August 1997.
6) Hiroshi Tezuka, Francis O'Carroll, Atsushi Hori, and Yutaka Ishikawa. Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication. In *IPPS'98*, April 1998.
7) http://www.mcs.anl.gov/home/lusk/mpich/.
8) Francis O'Carroll, Hiroshi Tezuka, Atsushi Hori, and Yutaka Ishikawa. The Design and Implementation of Zero Copy MPI Using Commodity Hardware with a High Performance Network. In *ICS'98*, July 1998. (to appear).
9) Erik D. Demaine. A Threads-Only MPI Implementation for the Development of Parallel Programs. In *11th International Symposium on High Performance Computing Systems (HPCS'97)*, pp. 153–163, July 1997.