社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－ARC－162（23）
2005－HPC－101（23）
2005／3／8

DIMMスロット搭載型ネットワークインタフェースDIMMnetにおけるスレッドライブラリ

森 拓郎† 濱田 芳博† 中條 拓伯† 並木 美太郎†

本研究ではクラスタ向けに開発されたNICであるDIMMnetを通信路として用いるスレッドライブラリ「蜂箱」について研究する。DIMMnetはPCクラスタを構築する際のボトルネックを解消し広帯域で低遅延な通信を実現するNICである。まず、クラスタにおいて分散処理を効率よく行うために、システムソフトウェアとして軽量なスレッドライブラリを作成した。また、DIMMnetの性能を活用するために、スレッドライブラリにおけるメッセージ処理の効率化を行った。本研究では、このスレッドライブラリの検証と今後の有効性について述べる。

A Thread Library for the "DIMMnet" NIC mount with the DIMM slot

Takuro Mori† , Yoshihiro Hamada† , Hiromori Nakajo† and Mitaro Namiki†

In this reseach, the thread library "HIVE" is developed using "DIMMnet" NIC for cluster's channel . "DIMMnet" is designed as wideband and low latency NIC resolving bottleneck of communication for PC cluster. The thread library was developed as a lightweight system software for distributed processing efficiently in the cluster. In addition, it makes efficient message processing to take advantage of the performance of "DIMMnet". In this reseach, the verification of this thread library and the effectiveness in the future are described.

1　はじめに

PCクラスタシステムは安価に構築可能なハイパフォーマンスコンピューティング環境として用いられている。近年のPCにおけるCPUの性能の向上はめざましく、また光通信や高速シリアル通信の登場により相互結合網で使用できる帯域幅は拡大してきている。これらを反映させることによってより演算性能の高いクラスタシステムが構築可能であると考えられる。

しかし、PCIバスを用いたネットワークインタフェースがボトルネックとなりこれらの帯域幅を十分に活用できないのが現状である。

DIMMnet[1]はPCIバスを用いたネットワークインタフェースの問題を解消するために開発されたネットワークインタフェースである。DIMMnetはDIMMスロットに搭載され、通信にメモリバスを用いることによってPCIバスで発生していた問題を解決している。

本研究では、DIMMnetを用いたスレッドライブラリ「蜂箱」を開発する。スレッドライブラリとは、プログラムのフローを関数単位でスレッドという形に抽象化して並列実行するものであり、プロセスと比べて切り替えのコストが少ないことが特徴である。「蜂箱」はDIMMnet通信路を用いてスレッド単位での並列分散処理を行う。ユーザはPOSIX準拠のスレッドライブラリ[2]を用いてスレッドプログラムを書き、スレッドライブラリはスレッドを複数のマシンによって並列処理を行い高速な演算を実現する。

また、Linuxが標準で用いているpthreadライブラリ[3]はシステムコールを用いてカーネルレベルで実装されているため、実行形態がユーザから分かりやすくプロセスのブロックにも対応できるが、処理にかかるオーバヘッドが大きい。そのため「蜂箱」はユーザレベルでスレッド処理を行うことによりオーバヘッドを低減させる。

関連した研究としては、DSM(分散共有メモリ)を用いた分散スレッドの研究[4]がある。これはマシン間でのスレッド情報のやり取りにDSMを用いて並列処理を行わせるものである。マシン全体で情報を共有するため、スレッドの管理や負荷分散に効果を発揮する。しかし、スレッドが増大するにつれて共有する情報が増え、DSM処理のオー

† 東京農工大学工学教育部
Graduate School of Technology, Tokyo University of Agriculture and Technology

バヘッドが増加する。

並列環境向けのスレッドライブラリは IBM の NPTL[5] が存在する。これは SMP 向けに作成されたカーネルスレッドを用いるスレッドライブラリである。SMP を効率的に利用できるように設計されている。本研究では SMP を視野には入れないためシングルプロセスの処理環境でより高速に処理を行えるスレッドライブラリを研究する。

また、分散処理用コンパイラである OpenMP と POSIX スレッドプログラムを連携させたものが存在する [6]。これはコンパイル段階でスレッドの動作先を決定するもので、関連性の深いスレッドをあらかじめ同じマシンで動作するように分散させる。コンパイル時点でスレッドの動作するプロセスを決定することにより、プロセス間の通信を最小に抑え、結果として処理時間の短縮に繋げるという利点を持つ。しかし、スレッドの動作プロセスをあらかじめ決定することにより、実際に処理を行った際に動的な負荷分散が行えないという問題点も存在する。

本研究におけるスレッドライブラリは動的なスレッドの分散処理によって負荷分散を行い、扱う情報を必要最低限のものとして通信に費やされるオーバヘッドを少なくとどめることにより処理を効率化させる。

2 DIMMnet

DIMMnet はメモリスロットである DIMM スロットに搭載される NIC である。通信におけるコントローラチップには、RHiNET にも搭載されている Martini[7] が用いられている。

独自の送信方式を持ち、ハードウェア的に送受信を行うことによって低遅延で広帯域な通信を可能としている。また、PCI バスを用いないことにより既存のものではボトルネックとなっていた DMA による衝突等の問題を解決している。

DIMMnet は以下のような 2 つのハードワイヤード化されたワンコピー通信の機構を持つ。

(1) AOTF(Atomic On-The-Fly)

(2) BOTF(Block On-The-Fly)

(1) の AOTF は小さいメッセージの低遅延な通信に適しており、8[bytes] までの転送を行う。4[bytes] の送信で約 3[μs] と、高速な処理が可能である。

(2) の BOTF は 474[bytes] 単位で連続したのメッセージの送信を行う。帯域幅は 190[Mbytes/s] あり、今後の改良によるさらなる広帯域化が期待されている。

2.1 DIMMnet 利用方法

メッセージの送信は、送信用のメモリに対する書き込みを契機として行われる。書き込まれたデータはハードウェアが検知し、通信先の受信用メモリに対して書き込まれる。この送信用メモリに対してオフセットをつけて書き込むことによって、送信用メモリに対して同様のオフセットをつけて書き込むことができる。このように DIMMnet におけるメッセージの送受信方式は一般的な NIC とは違い、DSM に類似した動作を示す。

2.2 DMA 衝突対策

DIMMnet は伝送ケーブルを送信用、受信用と 2 つ持っている。これは、PCI バスなどにおいて発生する DMA 衝突を防ぐためである。DMA 衝突とは、送信と受信が同時に発生した際にホストメモリと NIC の間で DMA の衝突が起き、帯域幅が半減する現象である。伝送ケーブルを用途ごとに 2 種類つ持つことにより、同時送受信における帯域幅を保証する。

2.3 割り込み

DIMMnet は DIMM スロットに搭載されている。このため、メッセージの受信などをハードウェアが検知したとしても OS に対して割り込みを発生させることができない。

3 目標

本研究ではスレッドプログラムによる並列分散処理を利用できるスレッドライブラリ「蜂箱」を作成する。その目標は以下のものである。

(1) スレッドプログラムによる分散並列処理

(2) DIMMnet による広帯域低遅延通信路の利用

「蜂箱」はユーザの書いたスレッドプログラムによって分散並列処理を行う。負荷分散を意識する必要がなく、記述性の良い並列処理を可能とする。

通信路には DIMMnet を用いる。連続した細粒度通信や広帯域通信においても安定した性能が得られるため、これを利用してメッセージ処理を行い効率的にスレッドの処理を行う。

4 設計

4.1 全体構成

「蜂箱」は POSIX 準拠のスレッドライブラリとする。POSIX スレッドはスレッドライブラリの規格として一般的であり様々なプログラムが作成されている。これによりアプリケーションソフトウェアのソースレベルでの互換性を提供する。

カーネルレベルスレッドはスレッド処理時にモードの切り替えを必要としてオーバヘッドが大きくなると考えられるため用いない。そのため処理は

ユーザレベルスレッドモデルで行うものとする。

「蜂箱」は複数のスレッドコンテキストを保持し、スケジューラによって管理され仮想プロセッサに対して割り当てられる。

他のマシンに対してスレッド処理を行う際は通信ライブラリを用いて DIMMnet に通信要求を出し、引数等を送信することによって実現する。

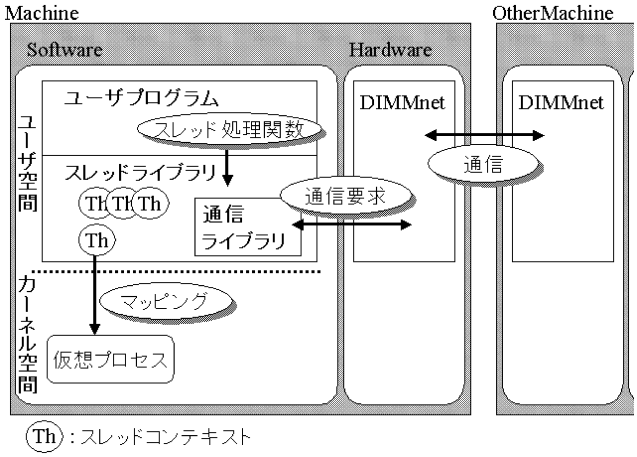

図 1: 全体構成

4.2 設計方針

本研究におけるスレッド処理において重要な点は以下のものである。それぞれについて述べる。

(1) 軽量なスレッド分散処理
(2) 動的なスレッド分配
(3) 分散したスレッドの管理

(1) の処理の軽量さはスレッドの大きな優位性である。蜂箱における処理時間はスレッドのものに通信のオーバヘッドが加えられるため、通信に用いるメッセージのサイズや送信の方式を考慮することにより、通信オーバヘッドを低減させて処理効率の低下を防ぐ。

(2) は分散処理を行う際のユーザプログラムの変化や実行環境の変化に対して、柔軟にマシンの負荷分散を行うために重要な機能である。各マシンの負荷を知るため、マシン間でのスレッド情報のやり取りが必要であるが、(1) でも述べた通り冗長な通信は避けなければならない。そのため通信する情報はマシンの保持している実行可能なスレッドの数のみとする。これによりマシンの負荷状態はある程度予測できる。

(3) において重要となるのはそれぞれのマシンはスレッドの情報をどの程度まで知るかということである。(2) で述べたように負荷分散に必要な情報は別途与えられているので、実行されているスレッドの詳しい情報を知る必要があるマシンは pthread_create が実行されてスレッドの生成を要求したマシンと、その要求を受け取り、スレッドを生成して実行しているマシンのみとする。詳しい情報とは、スレッドの ID や現在の状態、実行されたスレッドルーチンなどである。

4.3 外部設計

「蜂箱」が実装するスレッド処理関数は以下のものに加え、単一プロセス内での排他制御を行う pthread_mutex_lock および unlock である。これは、POSIX スレッド規格全体の約 6.5[%] であるが、ある程度のスレッド処理の実行は可能である。

(1) pthread_create - 新たなスレッドの生成
(2) pthread_exit - スレッドルーチンの終了
(3) pthread_join - スレッドに対する同期

(1) の pthread_create は、新たにスレッドを生成する。引数として与えたスレッドルーチンへのポインタと引数からスレッドを生成し、その ID を得ることができる。生成されたスレッドは DIMMnet で接続されているいずれかのマシンで動作する。

(2) の pthread_exit はスレッドルーチン内で用いられ、そのスレッドルーチンを終了する。この際の引数は、そのスレッドに対して同期を取った際に得ることができる。

(3) の pthread_join は作成されたスレッドに対して同期を取る。引数として pthread_create で得られたスレッド ID を与え、pthread_exit で与えた値を得ることができる。

4.4 内部設計

4.4.1 内部概要

スレッドライブラリ内部は以下の 2 つの機構に分けられる。

(1) スレッド管理ブロック
(2) DIMMnet 通信管理ブロック

(1) はスケジューラによるスレッド管理をはじめとしてスレッドの生成や削除、同期といった処理からスレッドのブロック、排他制御等を行う。スレッド処理は全てユーザレベルで行うためモードやプロセスの切り替えが発生せず、少ないオーバヘッドで実行が可能である。他のマシンに対してスレッドの生成を行う場合は、DIMMnet 通信管理ブロックに対してメッセージ送信の要求を行う。また、DIMMnet 通信管理ブロックがメッセージの受信を通知された場合に呼び出され、メッセージの処理を行う。

(2) は他マシンに対する通信の管理を行う。送信はスレッド管理ブロックからの要求によって行われる。メッセージの受信を検知した場合はスレッ

ド管理ブロックに通知し、処理を行わせる。メッセージの送受信に加え、タイムアウトやパケットロスに対する制御を行う。一連の処理はスレッド管理同様ユーザレベルで行うことによりオーバヘッドの増加を防ぐ。

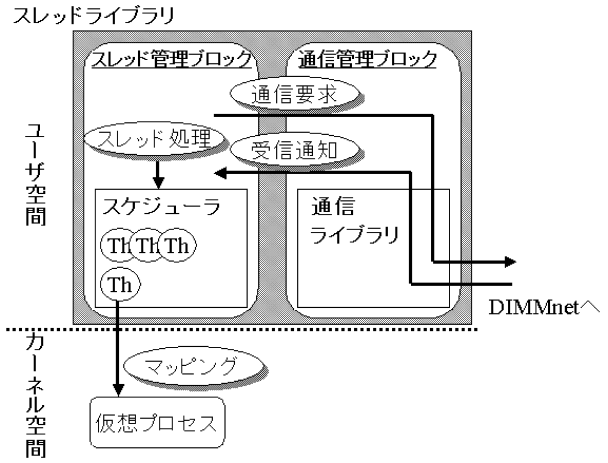

図 2: 内部概要

4.4.2 データサイズと通信機構

蜂箱では、DIMMnet の通信機構は細粒度通信に向いた AOTF を用いる。これは、送信する必要があるデータのサイズが小さいためである。

スレッド生成に必要なデータは、スレッドルーチンへのポインタ (4[bytes]) と引数 (4[bytes]) の 8[bytes] である。生成後に得られるスレッド ID も 4[bytes] と小さい。またスレッドへの同期では、対象となるスレッドの ID(4[bytes]) と返り値 (4[bytes]) がやり取りされる。この程度のサイズのメッセージであれば AOTF が BOTF よりも効率的であることは計測により得られている。

4.4.3 ポーリング

DIMMnet はメッセージ到着の際に割り込みを発生させられないため、タイマ割り込みによるポーリングを行う。ポーリングに費やされる時間はマシン台数に比例するが、参照するメモリがマシン 1 台につき AOTF1 回の送信メッセージのサイズ (1〜8[bytes]) であるため、軽量である。

5 DIMMnet によるスレッドライブラリ構築

5.1 構築における利点

本研究におけるスレッドライブラリにおいて、通信路に DIMMnet を用いることによる利点は以下の 2 つが挙げられる。

(1) 帯域と遅延
(2) DMA 衝突

(1) は分散処理において処理時間に影響を与える重要な値である。したがって、用いられる NIC はより広帯域で低遅延なものが望ましい。DIMMnet は既存の NIC に比べて優秀な帯域と遅延を実現している。

(2) は送受信が頻発する環境において、帯域幅低下を引き起こす重要な問題である。本研究におけるスレッドライブラリでは、スレッドの生成や同期においてマシン間での通信が頻繁に行われる。そのため DMA 衝突の問題が解決されていない場合、頻発する通信による帯域幅の低下は避けることができない。DIMMnet は送信用と受信用にそれぞれケーブル用意しているため、DMA に対する問題は改善されている。そのため安定した帯域幅が得られる。

5.2 構築における問題と解決法

DIMMnet と既存の NIC との違いのため、以下の点を考慮する必要がある。

(1) 割り込み処理
(2) 細粒度な通信

(1) はメッセージの受信に応じて行うのが一般的には効率的であるが、DIMMnet は割り込みを発生させることができないため代替の手段を考慮する必要がある。代替としてはポーリングによるメッセージチェックを用いる。また、ポーリング用いることによるメッセージの処理効率が低下を防がなくてはならない。そのためには、一度のポーリングの間に多くのメッセージを処理できるようにすることである程度解決すると考えられる。

(2) は DIMMnet のメッセージ送信方式 AOTF を用いて実現される。AOTF が通信を行う際のメッセージは 4[bytes] 単位で送信されるため、通信の失敗によるパケットロスもこの単位で発生する。受信側はこのパケットのロスを検知して必要なデータのみに対して再送の要求を出す必要がある。

これは DIMMnet が DSM のようなオフセットを用いた送信を行うことを利用することによって書き決することが可能であると考えられる。受信用バッファに対してデータを連続的に送信させた場合に発生したパケットロスは、受信したバッファの中の一部だけデータの受信が行われていない穴として検知できる。受信側はその部分のデータのみに対して再送要求を出せばよい。

6 評価

評価に用いたマシンは CPU に Pentium III 1266MHz、OS に RedHatLinux6.2 を搭載している。

6.1 単体でのスレッド処理

スレッドライブラリ蜂箱と、LinuxPthread において連続したのスレッド処理を行い、生成および終了と同期の所要時間を測定した。

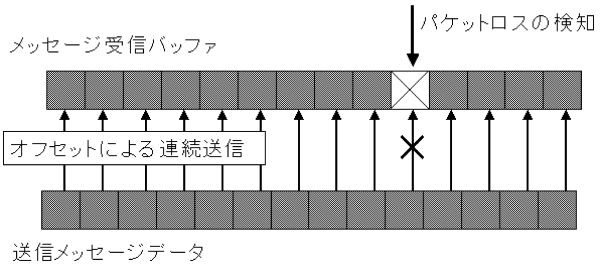

図 3: パケットロスの検知

表 1: スレッド処理時間の比較

スレッド処理	Linux pthread	蜂箱
生成	4572.4[ns]	456.6[ns]
終了と同期	6204.0[ns]	355.8[ns]

結果では、蜂箱が LinuxPthread に比べてはるかに高速に処理を行えている。この要因は、蜂箱がユーザレベルスレッドモデルであるのに対して、LinuxPthread が clone システムコールを用いたカーネルレベルスレッドモデルであるということが考えられる。蜂箱はユーザレベルのみを用いたスレッド処理によるオーバヘッドの低減ができていると言える。

6.2 スレッドの分散処理

実際に蜂箱を用いて別マシンに対してスレッドを分散させて実行し、スレッドの生成および終了と同期の処理に費やされる時間を測定した。比較対象としては 100[Mbps] の Ethrernet 環境を用いた。

表 2: スレッド処理時間

	Ethernet	DIMMnet
生成	95.6[μs]	17.0[μs]
終了と同期	95.7[μs]	14.0[μs]

結果として、Ethernet より約 5〜6 倍高速に処理が行えた。スレッドへの同期は送信するメッセージが生成のものと比べて少ないため、少ない時間で処理が行えていることが分かる。

6.3 タイマ割り込み

メッセージのチェックにポーリングを用いているが、Linux でのタイマ割り込みは標準で 10[ms] とメッセージ受信に対するポーリングとしては非常に低速である。これを改善するために、Linux カーネルを改変してタイマ割り込み時間を変更した。これに伴い、割り込み回数の増加による割り込みオーバヘッドの増加が考えられる。そこでタイマ割り込みの周期ごとのオーバヘッドを測定した。

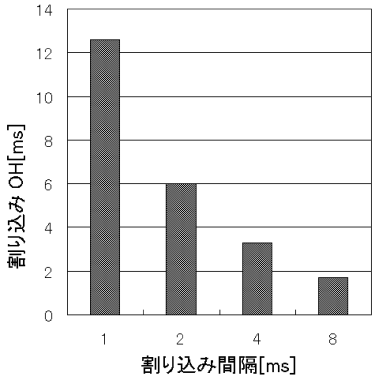

図 4: 1[s] あたりの割り込みオーバヘッド

タイマ間隔が短くなるとポーリングの効率はそのぶん向上するが、一方で割り込みに費やされるオーバヘッドは大きくなる。割り込み間隔を 1[ms] とした場合では、オーバヘッドが全体の処理の 1[%] 以上を占めることとなった。これはタイマ割り込みの際のプロセススイッチのコストの増加が原因である。

また、単純な Linux カーネルの改変では、1[ms] が最小のタイマ割り込み間隔である。しかし、数十マイクロ秒で通信を行う DIMMnet にとってはこの間隔は十分であるとはいえない。このようなタイマ割り込みの方式には限界があるといえる。

6.4 メッセージ受信バッファ

DIMMnet が通信を行うにあたり、通信用のメモリ領域をマシン 1 台に対してどれだけ与えるかということはメッセージ処理効率上重要である。そこで、メッセージ受信バッファの増加に伴う処理効率について測定した。なお、バッファのサイズは 4[Kbytes] の容量をマシン台数で分割するため、128[bytes] を最大とした。

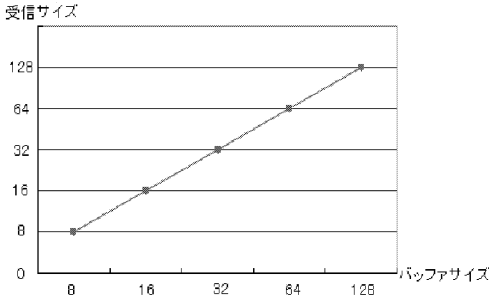

図 5: 受信バッファ利用効率

一度のメッセージ処理でほぼ全てのバッファを利用できていることが分かる。これはメッセージ処理の間隔であるポーリングの間隔よりも、メッセージを送信するのに費やされる時間が非常に短いため、全てのバッファにメッセージを送信できているからである。

これにより、単一のメッセージと同等のサイズである 8[bytes] のバッファを用いた場合と、128[bytes] のバッファを用いた場合では スレッド生成で約 10 倍、スレッドへの同期で約 16 倍の性能向上となった。

7 考察

「蜂箱」の性能において最大のボトルネックとなっているのはタイマ割り込み間隔の長さによるメッセージ処理効率の低下である。これを解決する方法としては以下のものが考えられる。

(1) メッセージの一括送信
(2) 協調型スレッドモデル

(1) は大量のメッセージ送信が発生する場合に有効となる手法である。DIMMnet の通信機構の内で今回用いていたのは細粒度通信 AOTF であった。それは、用いる個々のメッセージサイズの小ささからであったが、各マシンに割り当てることのできるバッファが 128[bytes] と小さいという問題が残った。そこで広帯域通信用の通信機構 BOTF を用いることが考えられる。BOTF は 474[bytes] のメッセージを連続して送信することが可能であり、送信されるメッセージが十分大きい場合には有効に働く。

(2) は、ユーザレベルスレッドモデルとカーネルレベルスレッドモデルを双方利用した方式である。通常ユーザレベルスレッドモデルのみでは、ページフォルトなどによって仮想プロセッサがブロックされると、その上で実行しているユーザスレッドも自動的にブロックされ、仮想プロセッサがブロックされたことがユーザ空間に対して通知されない。そのため、スレッドのブロックがユーザ空間から見えない。また、ブロックされた仮想プロセッサが復帰する際もユーザ空間にされない。

そこで、スレッド処理に関連する事象をカーネルレベルで検知し、カーネルレベルからユーザのハンドラに対して通知を行うことによってユーザレベルに処理を委ねることができる。また、カーネルレベルスレッドの機能を利用してスレッドの実行を別の仮想プロセッサに割り当てることも可能である。これにより、メッセージ到着のポーリングだけではなく、ユーザ空間では対応できない事象に対して柔軟に処理が行える。メッセージ処理のみならずてスレッド処理全般において性能向上が期待できる。

8 おわりに

今回の論文において、DIMMnet とユーザレベルスレッドを用いることによってスレッド単位で処理を並列化することができた。また、メッセージバッファについて改良することによってスレッド生成で約 10 倍、同期で約 16 倍性能が向上した。

今後は今回挙げられた解決法を実施し、さらなる処理の効率化について検討する予定である。

参考文献

[1] 今城 上嶋 濱田 中條 工藤 天野 田邊, 山本. DIMM スロット搭載型ネットワークインタフェース DIMMnet-1 の試作 情報処理学会 HPC 研究会 (SWoPP2001),No.77,pp.99-104.

[2] POSIX スレッドプログラミング David R. Butenhof 著 油井 尊 訳. アスキー, 1998.

[3] The LinuxThreads library http://pauillac.inria.fr/ xleroy/linuxthreads/.

[4] Bernd Dreier, Markus Zahn, and Theo Ungerer. The rthreads distributed shared memory system. In *Proc. of the 3rd Int'l Conference on Massively Parallel Computing Systems (MPCS'98)*, 1998.

[5] The Next Generation POSIX Threading Project http://www-124.ibm.com/ developerworks/oss/pthreads/.

[6] Kazuhiro Kusano, Shigehisa Satoh, and Mitsuhisa Sato. Performance evaluation of the omni OpenMP compiler. *Lecture Notes in Computer Science*, volume 1940.

[7] 山本 淳二 田邊 昇ほか. 高速性と柔軟性を併せ持つネットワークインタフェース用チップ: Martini 情報処理学会研究報告 2000-ARC-140 pp.19-24.