# RHiNETのソフトウェアレイア

宮脇達朗[†]　山本淳二[††]
工藤知宏[††]

我々は，並列処理に適した低レイテンシで広いバンド幅を持つ通信を実現するネットワークインタフェースのコントローラLSIであるMartiniを開発している．Martiniは，送受対象のアドレスがあらかじめ決められている場合のゼロコピー通信（remote DMA），PIOによる通信についてはハードウェアにより高速にサポートするが，より柔軟な通信機能はコアプロセッサにより実現する．そこで本報告では，Martiniコアプロセッサ上のプログラムにより実装する通信機能についてlock, barrier, send/receiveを中心に述べる．

# Software Layer of RHiNET

Tatsuaki Miyawaki ,[†] Junji Yamamoto [††]
and Tomohiro Kudoh [††]

We are developing a network interface controller LSI, called Martini. Martini realizes low-latency high-bandwidth communication suitable for parallel processing. While Martini supports zero-processor-copy (remote DMA) communication or PIO based communication by hard-wired logic, other complicated communication functions such as lock operation, barrier operation or send/receive operations should be supported by the on-chip processor. In this report, design and implementation of such functions are described.

## 1. はじめに

RHiNETでは，ネットワークインタフェースカード（NIC）のコントローラとしてMartini[4]とよぶネットワークプロセッサを用いる．RHiNETは，計算機どうしを接続して効率良い並列処理環境を実現するために，広バンド幅で低レイテンシなユーザレベル通信を提供する．RHiNETのネットワークは物理層で信頼性が保証される．

RHiNETネットワークインタフェースが提供する通信機能には，次のようなものがある．

PUSH: 起動側からターゲット側に対してゼロコピーでデータを転送（DMA）する．送信側と受信側の転送対象領域を指定して転送する．

PULL: ターゲット側から起動側にゼロコピーでデータを転送（DMA）する．PUSH同様，転送対象領域は指定される．

PUT: 起動側からPIO操作（NIC上のメモリ領域への書き込み）によってデータを送信する．受信側ではNIC上のメモリにデータを格納するか，ホストメモリに対してDMAを行なう．

LOCK: 複数のプロセスで競合する資源のへのアクセスの排他制御を行う．

BARRIER: 複数のプロセス間で同期を行う．

SEND/RECEIVE: 送信プロセスは任意の宛先プロセスに対してメッセージをsendする．受信プロセスは，複数のプロセスからのメッセージを同一の受け口で受信する．このとき，同一プロセスからのメッセージは，送信順に受信側に渡される．

Martiniでは，これらの通信機能のうちPUSH, PULL, PUTは完全にハードウェアにより実現される．より複雑な操作を必要とするLOCK, BARRIER, SEND/RECIEVEは，Martiniチップ上のプロセッサ（CorePU）で動作するプログラムにより実現する．

本報告では，PCIバスに装着されるRHiNETネットワークインタフェースについて，ユーザに提供する通信機能を紹介し，Martiniチップ上のプロセッサにより実現される通信機能の実装について述べる．

## 2. Martini

### 2.1 通信の起動

ユーザプロセスからの通信要求はMartini上のwin-

† NEC 情報システムズ (株)
NEC Informatic Systems
†† 技術研究組合 新情報処理開発機構
Real World Computing Partnership

dow と呼ぶ領域への書き込みにより行なわれる．ユーザプロセスは，window に通信に必要な情報を書き，window 上の特定のアドレス（キックアドレス）に書き込む（キックする）ことにより Martini に通信を要求する．window はシステムソフトウェアによって仮想アドレス空間にマップされなければアクセスできないので，他のプロセスのアクセスからは保護されている．window には，その window へアクセスするプロセスが属するプロセスのグループの ID である PGID（Process Group ID）が対応づけされており，同一 GPID を持つプロセス間でのみ通信を行なえる．

### 2.2 Martini コアプロセッサ

Martini コアプロセッサは R3000 をベースとした 5 段パイプラインの 32bit RISC プロセッサである．パイプラインステージは，インストラクションフェッチ（IF），インストラクションデコード（ID），実行（EX），メモリアクセス（MEM），書き戻し（WB）からなる．コアプロセッサには 128Kbyte× 2 の計 256Kbyte のメモリが接続されており，それぞれインストラクション用，データ用として使用される．

Martini 内部は 64bit アクセスが基本であるが，コアプロセッサは 32bit アクセスが最長のため，32bit アクセス - 64bit アクセス変換器を持つ．

Martini 各部で発生する割込は約 20 種類ある．一方，プロセッサ自身には 8 レベルの割込処理能力しかないため，何らか方法によるマッピングが必要である．割込のマッピングおよびマスクを行うのが割込コントローラである．コアプロセッサから割込コントローラ内のレジスタを操作することで，割込のルーティング，マスクを自由に変更できる．

また，タイムアウト処理や各種測定を行うことを考え，64bit の時計とアラームタイマを持つ．時計はリセット解除後からの内部クロック数をカウントしている．一方，アラームタイマは設定された時間後，割込を発生する．アラームタイマの設定はコアプロセッサから自由に変更できる．また，タイマと時計の現在の値も読み出すことができる．

#### 2.2.1 代 行 処 理

基本的なプリミティブである PUSH，PULL はハードワイヤードだけで処理される．しかし，それ以外のプリミティブや未定義のパケットを受け取った場合や TLB のミスヒット時などはハードワイヤード部はコアプロセッサに割り込み，処理を停止する．

コアプロセッサはハードワイヤード部内の各種レジスタや TLB をアクセスすることで必要な情報を収集できる．TLB のリフィルなど必要な処理を行った後，ハードワイヤード部の動作を復帰させる．

## 3. LOCK

負荷の集中を避けるため，分散キューを用いてロックを実装をする [1)]．キューの最後，すなわちロックを最後に要求したプロセスを記録しておくために，manager と呼ばれるプロセスを用意する．

動作を図 1 を例にして説明する．P1, P2, P3 がそれぞれロックに参加するプロセス，manager がこのロックの管理をしているプロセスである．現在 P1 がロックを保持し，P2 が最後にロックの要求をしたノードであるとする．この状態（図 1（a））では，P1 は次にロックを要求しているプロセスを構造体上の変数 `next_proc` に保持している．また，manager は最後にロックを要求したプロセスを同じく構造体上の変数 `last_proc` に保持している．

この状態で P3 がロックの要求をしたとする．P3 は（あらかじめ用意されている）構造体を用いて Martini にロックの要求を開始する．Martini は構造体の指示に従い manager に要求（lock_req）を送信する．（図 1（b））．

manager を持つノードの Martini はロックの要求を受け取ると，変数 `last_proc` を読み取る．さらに `last_proc` に従い，要求（lock_forw）をノード P2 に転送する（図 1（c））．それと同時に `last_proc` を P3 に変更する．

P2 を持つノードの Martini は要求を受け取ると，P2 の構造体の変数 `next_proc` を P3 に変更する（図 1（d））．

図 1（e）では P1 がロックを解除している．ロックの解除を行う場合もロック要求と同じ構造体を使用する．ただし，`type` フィールドを `UNLOCK` に変更する．Martini はロック解除の指示を受け取ると，変数 `next_proc` の値（この場合 P2）に従い， P2 にロックを渡す（lock_reply）．P2 を持つノードの Martini はロックが渡されると，ロックが取得できた事を P2 に通知する．

### 3.1 Martini への実装

Martini の corePU は以下の様にロックを実行する．NIC は次にパケットを送信する相手を決定する情報とキューの状態を示す情報を管理する．この情報はロックを実行するユーザプロセスが確保したホストメモリ上の管理領域におかれる．NIC はこの領域を DMA で読み書きする．

NIC は管理領域に以下の情報を置く．

next_proc: そのノードの次にロックを要求したプロセス（次プロセス）の PID

RID: 次プロセスへの経路情報

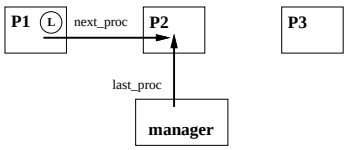
VA: 次プロセス上の管理領域の仮想アドレス

LOCKCONDITION: プロセスの状態

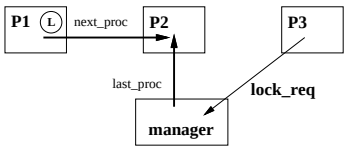
Martini は処理中にいくつかの状態を遷移する．状態遷移図を図 2 に示す．

ここで，Wait, Lock, Queued, Last はそれぞれ以下の意味を持つ．

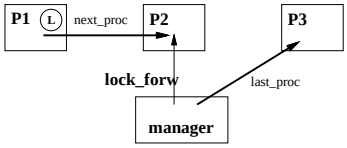
Wait: ロックが獲得出来るのを待っている．



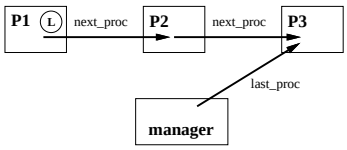

(a) initial



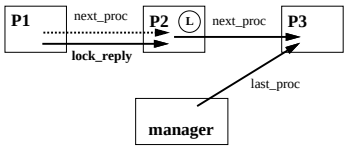

(b) request lock




(c) forward request




(d) expand queue




(e) release lock

**図 1** ロックの動作

Lock: ロック獲得している．
Queued: 自プロセスの次にロック獲得を待っているプロセスがある．
Last: 自プロセスがロックを獲得した最後のプロセスである事を示す．Last 状態の時は既にロックは開放している．

### 3.2 検　　討

分散キュー形式のロックはマネージャへの負荷集中を避ける事が出来るが，マネージャはロック要求をフォワードだけの役割で，どのプロセスがロックを持っているかは一切管理していない為トライロックを実行する事ができない点が課題である．

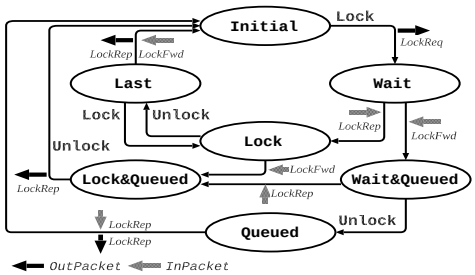



**図 2** ロックの制御の状態遷移

## 4. BARRIER

最も単純なバリアの実現方法は計数バリアである．計数バリアでは管理プロセス（マネージャ）がバリアに参加するプロセスからのバリアリクエストを数え，数が揃うと逆に全ての参加プロセスにバリア完了通知を送る．バリアに参加するプロセス数が多くなるとマネージャの負荷が大きくなり同時にバリア処理のレイテンシも大きくなる．

そこで，マネージャへの負荷の集中を避けるために 2 分木を使ってバリアを実現する．2 分木バリアでは葉→節→根の順にバリア要求を集め，根のプロセスでバリアが成立すると逆方向に根から葉に向かってバリア完了通知を伝達する．

### 4.1 ユーザープロセスからみた BARRIER

バリアに参加する全てのユーザプロセスは，まず全ての参加プロセスの PID を格納した配列を `barrier_init` ライブラリに渡し，2 分木構造を作成する．各プロセスで実行される `barrier_init` は，同一の PID 配列を渡されるから同一の 2 分木を生成し，それぞれのプロセスの 2 分木上の位置にしたがって必要な初期設定を行なう．

ロックと同様に NIC はバリア実行時に NIC が使用する管理領域をホストメモリ上に置くため，プロセスはその領域（BARRIER 構造体）の確保，ピンダウンを行う必要がある．

### 4.2 Martini への実装

Martini の CorePU は以下の様にバリアを実行する．実行時に必要な管理情報（親，子プロセスへのルーティング情報，バリアの実行状態等）はユーザープロセスが確保したホストメモリ上に置かれ，NIC は DMA でアクセスする．

- 2 分木の葉にあたる Martini は，ホストからバリア要求を受けとると，親にバリア要求を送る．
- 2 分木の節にあたる Martini は，全ての子からバ

リア要求を受信し，自らのホストもからも要求を受信すると親にバリア要求を送る．

- 2 分木の根にあたる Martini が全ての子からバリア要求を受信した段階でバリアが成立する．バリアが成立すると逆に 2 分木を下向きに用いてバリア完了通知を送信する．

### 4.3 検　　討

2 分木形式のバリアでは，以下のような検討課題がある．

- 2 分木作成の問題点
  2 分木バリアではバリアを実行する前に 2 分木構造を作成しなければならない．このため，実行中に動的にバリアに参加するプロセスの構成が変化する場合には対応が困難である．
- ネットワークトポロジーとの関連
  2 分木はプロセス ID によって構築されるため，実際のネットワークのトポロジに対して最適な構成で有るとは限らない．

## 5. SEND/RECEIVE

SEND/RECEIVE では，送信側は任意のプロセスに対してメッセージを send できる．受信側は，到着メッセージを受信側プロセスに渡す．このとき，同一プロセスからのメッセージは，送信順に受信側に渡される必要がある．

SEND/RECEIVE では受信側が受信準備ができているとは限らない．効率のために受信側にはバッファを用意するのが普通であるが，受信バッファのサイズは有限であるので，容量が不足してバッファに書き込めない場合にはやはり受信することができない．このためフローコントロールが必要になる．

ここでは，フローコントロールに，Myrinet 上の PM で用いられている Modified ACK/NACK 方式 [2] を用いる．

### 5.1 Modified ACK/NACK 方式

Modified ACK/NACK 方式では，送信側はメッセージに逐次番号（SEQID）を付けて送信し，受信側は受信可能な場合には Acknowledge（ACK）を，バッファが一杯で書き込めない場合には Not-Acknowledge（NACK）を当該受信メッセージの SEQID を付けて返す．NACK を返した場合には，その SEQID を持つメッセージの再送を待つ．同一送信プロセスからのより大きい（後で send された）SEQID を持つメッセージは ACK/NACK を返すことなく破棄する．送信側は，NACK が返された場合，その SEQID 以上の ID を持つメッセージを全て再送する．以後，NACK が返された ID の再送メッセージに対する ACK が返るまで新たなメッセージは送信しない．もちろん，再送されたメッセージを受信した時点でも受信バッファはまだ空いていない場合があり，この場合は再度 NACK を返す．

### 5.2 ユーザプロセスからみた SEND/RECEIVE

SEND/RECEIVE 通信を行なうユーザプロセスは，あらかじめ `open_context` システムコールにより送受信バッファの確保などの必要な設定を行なう．`open_context` システムコールでは，

- コンテキストが用いる論理チャネル番号
- 送信する可能性がある相手プロセスのリスト
- 受信する可能性がある相手プロセスのリスト

を登録する．論理チャネルは，同一のプロセスが複数のコンテキストを持てるようにするもので，同一のチャネル番号を持つコンテキスト間でのみ通信が行なわれる．

送信の際には，`get_send_buf` オペレーション（ユーザレベルで実行）により送信バッファ上の領域を確保し，送信するメッセージをその領域にコピーした上で send を起動する．

受信の際には，受信バッファにメッセージが到着していれば読み出してメッセージが格納されていた領域を解放する．

スイッチの構造 [3] や Martini 内のバッファ量の制限により，Martini で一度に send できるメッセージの大きさ MTU（Maximum Transfer Unit）は，2Kbyte である．MTU を超える大きさのメッセージを送信する場合には，送信プロセスが複数の SEND に分割する必要がある．

### 5.3 Martini への実装

#### 5.3.1 初 期 設 定

`open_context` システムコールにより以下の領域の確保と初期設定を行なう（図 3）．

**送信に用いる領域**

- ホストメモリ上:
  送信バッファ（SB）およびポインタを確保する．SB は送信要求されたメッセージのバッファであり再送にも用いられる．登録されたメッセージは受信確認（ACK）が返された後に削除可能になる．NIC は送信時に SB からデータを DMA により読み出すため，SB の領域はピンダウンされていなくてはならない．効率のため，SB は物理アドレス上で連続した領域に確保される．SB は，リングバッファとして順に用いられる．
- NIC メモリ上:
  コンテキスト情報: 送信の起動は window に対する書き込みによって行なわれるので，window に書き込まれた時にどのコンテキストを起動するのかについての対応を設定するとともに，そのコンテキストの論理チャネル ID を設定する．
  送信側用逐次番号管理テーブル（S_SEQTBL）: メッセージの宛先になる可能性があるプロセスの ID ごとに逐次番号 SEQID を格納する

領域を確保し 0 に初期化する．したがって，システムコールでは宛先のリストを渡す必要がある．実装上は，宛先プロセス ID が昇順に登録されたテーブルと，それぞれに対応した SEQID が登録されたテーブルからなり，利用時にはプロセス ID を binary search により探索し，対応する SEQID を用いる．

送信管理テーブル（SENDTBL）: SENDTBL は，ホストプロセッサを起動することなく NIC のみで再送を行なうために必要な情報を保持するテーブルである．SENDTBL の 1 エントリは SB 上のメッセージの位置及びサイズ，宛先プロセス ID，メッセージの経路情報，メッセージの逐次番号（SEQID），ACK 受領済みかどうかを示すフラグを持つ．SENDTBL の大きさは初期化時に決まるため，acknowledge 待ちの send 要求は一定数に制限される．

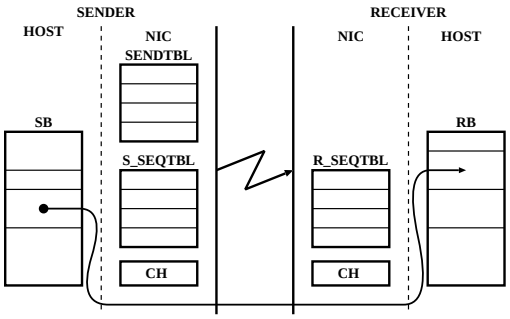



**図 3** SEND/RECEIVE のための領域

**受信に用いる領域**

- ホストメモリ上:
  受信バッファ（RB）およびポインタの領域を確保する．RB は受信メッセージのバッファである．NIC は受信時に RB にデータを DMA により書き込むため，RB はピンダウンした領域に確保されていなくてはならない．効率のため，RB は物理アドレス上で連続した領域に確保される．
- NIC メモリ上:
  コンテキスト情報: 論理チャネル ID と受信側の PID をコンテキストとして登録し，メッセージを受信できるようにする．
  受信側用逐次番号管理テーブル（R_SEQTBL）: メッセージの送り手になる可能性があるプロセスの ID ごとに逐次番号 SEQID を格納する領域．したがって，システムコールでは送り手のリストを渡す必要がある．S_SEQTBL と同様に実装される．

#### 5.3.2 送　　信

通信時には，送信側プロセスは，以下の手順で SEND を起動する．

(1) `get_send_buf` により SB 上に領域を確保する．
(2) SB 上に送信メッセージを（copy するなどして）用意する．
(3) window に送信メッセージの格納場所，サイズ，受信プロセス ID などの必要な情報を書き込み，kick する．（即ち，window 上の特定のアドレス *kick_address* に書き込む）

`get_send_buf` は，NIC 上の SENDTBL に空きがない場合もしくは SB 上に連続した領域が確保できない場合には fail し，プロセスによりリトライされる．

Martini では，window への書きこみにより CorePU に割り込みがかかり処理が起動される．CorePU は，メッセージ逐次番号（宛先 RPID に対応する S_SEQTBL の内容）をメッセージに付加し，window に対応した送信管理テーブル（SENDTBL）に登録する．このとき再送処理中でなければ送信を行なう．また，S_SEQTBL のエントリの内容に 1 を加える．送信パケットには SENDTBL に登録されている内容に加えて，window に対応づけされている PGID，SENDTBL 上の当該エントリの NIC 上アドレス（受信側が ACK/NACK パケットに乗せて返す），当該コンテキストの論理チャネル ID が付加される．

#### 5.3.3 受　　信

受信側の Martini は，まず，チャネル番号，PGID 番号によって受信したパケットが正当なものであるかどうかを調べる．

次に，パケットに付加された逐次番号（SEQID）と R_SEQTBL 上の送信プロセスの逐次番号を比較する．逐次番号が等しくなければ，パケットを破棄する．

逐次番号が等しければ，ホストメモリ上の受信バッファRB に当該メッセージを格納するだけの空きがあれば格納し，ACK パケットを送信側に返し，R_SEQTBL のエントリの内容に 1 を加える．空きがなければ，NACK パケットを送信側に返す．

受信側のユーザプロセスは，受信バッファにメッセージが格納されたことをポーリングにより知り，受信処理を行なう．

#### 5.3.4 再　　送

受信バッファに受信したメッセージを格納できず，受信側が NACK を返した場合，送信側は当該メッセージ以降の SEQID を持つメッセージのパケットを再送する．これは SENDTBL 上の情報のみにより行なえるため，ホストプロセッサの助けを借りることなく，送信側の NIC により行なうことができる．

受信側は，ある SEQID のメッセージを受信できなかった場合，そのメッセージの再送を待つ間，より大きい SEQID を持つメッセージは破棄する．また，再送要求したメッセージを受信した時にもまだ受信バッファが空いていなければ再度再送要求を行なう．

#### 5.3.5 バッファ空き領域の管理

SB 上のメッセージおよび SENDTBL のエントリは，ACK パケットが返された時点で解放することができる．実際には簡単のため SB および SENDTBL はリング状の FIFO として用い，新しいメッセージ/エントリを追加する際に head ポインタを進め，メッセージ/エントリを解放する際には tail ポインタを進める．ACK は必ずしも送信順に返ってくるわけではないためとびとびに ACK が返ってきたメッセージ/エントリがあることになるが，連続して領域を解放できる時にだけ tail ポインタを進める．

### 5.4 検　　討

SEND/RECEIVE の実装では，いくつかの検討すべき選択肢がある．これらについては，今後実機を用いて実装方法を検討していく予定である．

- NIC とホストで共有される情報の管理
  SB, RB の tail ポインタなどは，NIC とホストプロセッサの両方で必要な情報である．SB の tail ポインタおよび SENDTBL の空エントリ数は，NIC により更新されるもので，ホストが `get_send_buf` を実行する際にチェックする必要がある．RB の tail ポインタは，ホストプロセッサにより更新されるもので，NIC が RB の空きをチェックする際に必要な情報である．
  PCI バスに装着されるネットワークインタフェースでは，バスの競合により性能が低下する可能性があるので，バスの使用頻度は小さい方がよい．このため，チェックする側は若干古い可能性があるコピー（stash）を持ち，空きがなくなった時にはじめて最新の情報をアクセスする手法が考えられる．
  例えば SENDTBL の空きエントリ数は，領域を解放した時に増える．ホストはやや古い情報を stash として持ち，空きがゼロになったときのみ NIC から最新の情報を得ることにより，PCI バスの使用頻度を減らすことができる．NIC から最新の情報を得ても，空きがゼロであった場合には，プロセスは空きができるのを待つことになる．この場合，以後 NIC 上の情報をポーリングをするのか，更新時に NIC から stash に書き込むのかも選択肢となる．
- 再送バックオフ時間
  Modified ACK/NACK 方式では，再送をしても受信側の RB に空きがない場合，再送が繰り返されることになり，ネットワークが混雑する．再送するまでのバックオフ時間を制御することにより混雑を緩和することができる可能性がある．
- アクティブ再送要求
  さらにネットワークの混雑を減らすためには，RB に空きがない場合に単に送信を停止する要求を送り，空きができた時点で再送を要求する手法も考えられる．
- 信頼性の低いネットワークへの対応
  受信側に再送中を示すフラグを設ければ，通信路の信頼性が低い場合にパケットの消失を検出することができる．

## 6. おわりに

現在，シミュレータ上での動作の確認と実機への実装作業を進めている．また，実行時間の計測，実アプリケーションで負荷がかかった場合の性能評価と実装の改良を行なう．Martini の CorePU を用いれば，この他にも様々な柔軟な処理の実装が可能である．今後，PUSH/PULL を使って受信側で全ての待ち合わせを行なう isend/ireceive[5] の実装なども進める予定である．

CorePU の性能はホストプロセッサの性能よりかなり低く，また使用できるメモリ量も限られるため，CorePU に行なわせる処理とホストプロセッサで行なう処理のバランスが重要である．

**謝辞**

SEND/RECEIVE 機能の実装検討にあたり，PM での実装法について御教示頂くとともにさまざまな助言を頂いた新情報処理開発機構の住元真司氏，NEC 情報システムズの長谷川篤史氏に感謝します．

## 参 考 文 献

1) T.Kudoh, J.Yamamoto, F.Sudoh, H.Amano, Y.Ishikawa, and M.Sato. *Memory based light weight communication architecture for local area distributed computing*, pp. 133–139. Innovative architecture for future generation high-performance processors and systems. IEEE Computer Society press, 1997.
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. 並列処理シンポジウム JSPP'96, 第 95-6 巻, pp. 41–48, 1996.
3) 西宏章, 多昌廣治, 工藤知宏, 天野英晴. 仮想チャネルキャッシュを持つネットワークルータの構成と性能. 並列処理シンポジウム JSPP'99, 第 99-6 巻, pp. 71–78, 1999.
4) 山本淳二, 田辺昇, 西宏章, 土屋潤一郎, 渡辺幸之介, 今城英樹, 上島利明, 金野英俊, 寺川博昭, 慶光院利映, 工藤知宏, 天野英晴. 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ネットワークインタフェースチップ: Martini. デザインガイア 2000, No. 2000-ARC-140, pp. 19–24, 11 2000.
5) 工藤知宏, 山本淳二, 建部修見, 佐藤三久, 西宏章, 天野英晴, 石川裕. PC 間ネットワークによる共有アドレス空間を持つ並列処理システム. 情報処理学会研究報告 ARC, 第 21- 21 巻, pp. 121–126, 1999.