SMP クラスタ向けネットワーク・インタフェース AM 通信

松田元彦† 田中良夫†
久保田和人† 佐藤三久†

NICAM は，通信オーバヘッドと同期レイテンシの低減を目的とする SMP PC クラスタのために設計した Myrinet 上の通信であり，ネットワーク・インタフェース（NI）上のマイクロ・プロセッサを活用することを特徴としている．通信処理を NI 上で直接行うことで，ポーリング等の負担からホスト・プロセッサを解放しオーバヘッドを低減する．また，同期プリミティブを NI 間で処理することで，ホスト-NI 間のインタラクションを不要にし，オーバヘッドと同時にレイテンシを低減する．NICAM では，NI 上のプログラムに拡張性を持たせるため，アクティブ・メッセージ（AM）の枠組みを利用している．AM を使うことにより，データ転送と同期を組み合わせる等の新しい通信プリミティブの定義が容易になる．これは NI 上の比較的低速なプロセッサによる実行を補うために，処理回数を減らすのに効果がある．NICAM を用いたデータ並列計算を行う実験では，低オーバヘッドな通信が計算/通信のオーバラップに有効であることを示す．また，NI 上で処理するバリア同期の応用として，データ並列ライブラリでの同期コストを完全に隠蔽可能であることを示す．

Network Interface Acitive Messages on SMP Clusters

Motohiko Matsuda,† Yoshio Tanaka,† Kazuto Kubota† and Mitsuhisa Sato†

NICAM is a communication layer for SMP PC clusters connected via Myrinet, designed to reduce overhead and latency by directly utilizing a micro-processor equipped on the network interface. It adopts remote memory operations to reduce much of the overhead found in message passing. NICAM employs an Active Messages framework for flexibility in programming on the network interface, and this flexibility will compensate for the large latency resulting from the relatively slow micro-processor. Running message handlers directly on the network interface reduces the overhead by freeing the main processors from the work of polling incoming messages. It also makes synchronizations faster by avoiding the costly interactions between the main processors and the network interface. In addition, this implementation can completely hide latency of barriers in data-parallel programs, because handlers running in the background of the main processors allow reposition of barriers to any place where the latency is not critical.

1. はじめに

PC クラスタもノード計算機が複数のプロセッサを持つ SMP（Symmetric Multi-Processor）クラスタがプラットフォームとして重要になっている．一般に計算性能がバス能力に制約されることも多いが，SMP 計算機では複数のプロセッサがバスを共有しているため特にバス・トラヒックに注意を払う必要がある．しかし，PC クラスタでは通信がバスを経由する I/O によって行われるので，通信によってもトラヒックが発生する．その低減には真のデータ転送以外のトラヒックの発生を減らす意味で，通信へのプロセッサの関与を低減する必要がある．

また，計算と通信のオーバラップは通信待ちによる無駄を低減する手段として知られているが，オーバラップが有効に機能するためにも，通信へのプロセッサの関与を低減するのが望ましい．特に，SMP クラスタでは通信レイテンシは変わらないままノードあたりの計算能力が数倍に大きくなっていることから，オーバラップの効果は大きいと考えられる．

すなわち，SMP クラスタ上の通信としては，通信処理にかかわるホスト・プロセッサ上の処理を低減すること，つまり，LogP モデル[1] の意味で通信オーバヘッドを低減することが望ましい．特に，データ並列計算においては，データ転送のレイテンシは問題とな

† 新情報処理開発機構
Real World Computing Partnership

らないのに対して，オーバヘッドは性能に大きく影響を与えることが知られている[2]．これはオーバヘッドがプロセッサの利用効率に直接かかわるためである．

そこで，通信オーバヘッドを極力低減する通信として NICAM を設計した．NICAM では，通信ハードウェアとして Myrinet[3] を使用するが，そのネットワーク・インタフェース（NI）上に備わっているマイクロ・プロセッサを活用することで通信オーバヘッドを低減する．NI 上のマイクロ・プロセッサを活用することにより，次のような低オーバヘッドの通信を提供する．

- リモート・メモリ転送ベースの低コストな通信
- ホスト・プロセッサ間の排他処理の削除
- ポーリングからのホスト・プロセッサの解放
- 主記憶上のフラグを用いる低コストな同期通知
- 同期プリミティブのバック・グラウンド処理

さらに NICAM は，同期プリミティブを NI 上で処理することにより，レイテンシを低減する．データ転送ではレイテンシが性能に与える影響は小さいが，同期に関してはレイテンシが重要である．NI 上で同期プリミティブを処理することにより，不必要にホスト・プロセッサに処理を戻す必要がなくなる．これにより，ホスト-NI 間の起動/待合せが不要になり，オーバヘッドとともにレイテンシが低減される．分散メモリ MPP 計算機ではリモート・メモリ転送やバリア同期等をハードウェアで提供していることが多い．一方，NICAM ではこれらの機能をコモディティ・ハードウェアの NI を使って提供する．ただし，同期に関してはハードウェアによる実装に比べるとレイテンシが大きくなることを考慮して，レイテンシ・トレラントな同期[4] を提供する．

NICAM の有効性を示す実験として，データ並列計算への応用を示す．低オーバヘッドな通信についてはその効果を直接示すことは困難であるので，かわりに計算と通信のオーバラップでの有効性を示す．そこでは，スレッド数が増えて通信の粒度が小さくなってもオーバラップに効果がみられる．NI 上で実装される同期については，同期レイテンシを完全に隠蔽できることを示す．データ並列計算ではバリア同期が多用されるが，通信処理とメモリ・アロケータと協調することで，バリア同期をレイテンシに寛容な場所に移動することができる．ただしこの移動に効果があるためには，同期がホスト・プロセッサに対してバック・グラウンドで処理される必要がある．

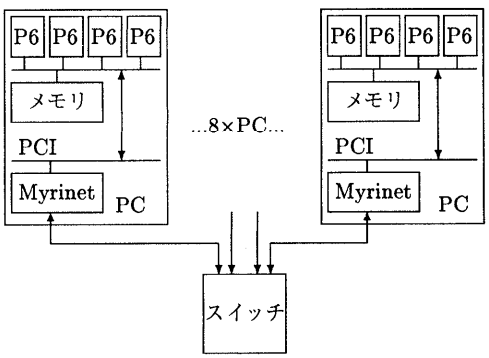
本論文の構成は次のとおりである．まず，2 章で，使用するクラスタ・プラットフォームとそのネットワークである Myrinet について述べる．続いて 3 章で，通信モデルを説明し，その実装方法と基本性能を示す．次に 4 章で，データ並列ライブラリへの応用としてバリア同期レイテンシの隠蔽方法を示す．その後，5 章で，計算/通信オーバラップとレイテンシ隠蔽の実験結果を示し，最後に関連研究とまとめを述べる．

2. プラットフォーム SMP クラスタ

2.1 SMP PC クラスタ COMPaS

実験プラットフォームである COMPaS[5] は，Pentium Pro PC をノードとする SMP PC クラスタである．**図 1** に COMPaS の構成を示す．各ノードは 4-way SMP の PC（Pentium Pro 200 MHz，L2 キャッシュ 512 KB，450GX チップセット，メモリ 128 MB）であり，これを 8 台接続している．ネットワークには Myricom 社 Myrinet[3] を使用している．各 PC に Myrinet PCI ボードを挿し，1 つの 8 ポート・スイッチを介して接続している．ノードの OS には Solaris 2.5 を使用している．以下では，通信レイヤ設計のベース情報となる SMP PC 単体の性能特性をあげる．

表 1 に単一ノード上で複数スレッドを同時実行させた場合のメモリ・バンド幅（リード，ライト，コピー）を示す．数値は全体での総バンド幅，すなわち各プロセッサで観測される値の合計値である．この測定はキャッシュ・サイズに比べて十分大きなメモリ領域に対して連続して処理を行った．表からバンド幅の合計はプロセッサ数によらずほぼ一定であることが読み取れる．これは単純なメモリ操作では単一のプロセッサがバス・バンド幅全体を使い尽くしていることを意味する．すなわち，ノード内のプロセッサの有効利用にはバス・トラヒックを抑えることが不可欠である．また，後に示すようにノード間の通信バンド幅は約 105 MB/s



- P6：Pentium Pro 200 MHz
- スイッチ：8 ポート Myrinet スイッチ

図 1 SMP PC クラスタ COMPaS の構成
Fig. 1 COMPaS, a Pentium Pro SMP PC cluster.

表 1 総メモリバス・バンド幅の変化
Table 1 Aggregate bandwidth of the memory bus.

プロセッサ数	リード	ライト	コピー
1	360.0	91.1	85.6
2	258.2	106.6	88.9
3	251.7	106.6	87.7
4	250.3	99.7	88.3

（単位は MB/s）

表 2 単一ノード内でのバリア同期時間の比較
Table 2 Barrier synchronization time of a single node.

プロセッサ数	PPro SMP	Sun SMP
1	0.11	0.05
2	1.09	1.84
3	1.78	2.23
4	1.83	2.70

（単位は μsec）

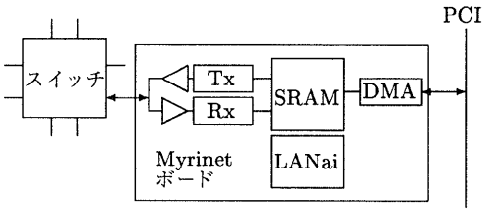
であるが，これはライトのバンド幅にほぼ等しい．

バス・バンド幅の結果に対し，キャッシュ・コヒーレンシ操作は全プロセッサが 1 つのバスにつながっていることから性能は悪くない．**表 2** にノード内のスレッド間でバリアをとった場合の性能比較を示す．PPro SMP が COMPaS で使用するノードでの値である．比較として，バス性能が比較的良いといわれる Sun SMP（Sun Enterprise 4000）の値を示した．使用したバリアはアトミックな命令（テスト・アンド・セット等）は使用せず，すべてキャッシュ・コヒーレンシ操作で実行される．この結果はコヒーレンシ操作の性能を反映していると考えられる．このバリアは，プロセッサ台数分のフラグを利用するもので，各プロセッサは自分に割り当てられたフラグをセットする．すべてのフラグがセットされた時点でバリアが成立するが，この条件はプロセッサ 0 番がチェックし他のプロセッサに通知する．

2.2 Myrinet ネットワーク・ハードウェア

Myrinet[3] 通信ハードウェアは通信リンク，スイッチ，ホスト・インタフェースから構成される．通信リンクは片方向 160 MB/s の速度を持つ 8 bit 並列，双方向のデータパスである．Myrinet スイッチは 8×8 のクロスバで構成され，カットスルー・ルーティングを行う．このスイッチは任意段数を任意のトポロジに接続可能である．ルーティングは，メッセージの先頭に付けられた情報による静的なソース・ルーティングを使う．ホスト・インタフェースは I/O アダプタ（PCI ボード）として実装される．

図 2 に Myrinet ボードの構成を示す．33 MHz で動作する LANai 4.x と呼ばれるカスタム CPU コアを中心に，SRAM メモリ，通信リンク・インタフェース，そして 3 つの DMA エンジンにより構成される．DMA エンジンの 1 つはホストの主記憶とボード上の SRAM 間で DMA を行う．他の 2 つは送信/受信用であり，SRAM と通信リンク間で DMA を行う．Myrinet ではすべての通信はボード上の SRAM を経由して行われる．



- Tx：送信用 DMA エンジン
- Rx：受信用 DMA エンジン
- DMA：主記憶 DMA エンジン
- LANai：カスタム RISC プロセッサ

図 2 Myrinet NI ボードの主要構成
Fig. 2 Essential elements of the Myrinet board.

Myrinet の特徴としては以下のものがあげられる．

NI 上のマイクロ・プロセッサ LANai は 32 bit 演算器を持つ RISC マイクロ・プロセッサである．

高信頼なネットワーク エラーレートが非常に低くリライアブルなネットワークとして扱える．また，ハードウェアでフロー制御を行っており，通信リンク上でのデータ・ロスはない．

全物理メモリに対して DMA 可能 DMA 機能に制約がなく，ホスト上の物理メモリ全体に対して DMA 可能である．これにより，ユーザ・プログラムのバッファ領域に直接データを転送すること（いわゆるゼロコピー通信）が可能である．

大容量の SRAM SRAM は NI 上のマイクロ・プロセッサのプログラムと通信データのバッファ領域に使われるが，128 KB から 1 MB と比較的大容量である．

これらの特徴により通信レイヤの実装が非常に簡単になっている．NICAM の実装ではネットワーク物理層をリアイアブルだと見なし，異常検出時の再転送等は行っていない．通信データの CRC 等のチェックは行うが，エラーの場合には単純にアプリケーションを異常終了する．

3. NICAM 通信プリミティブ

3.1 通信レイヤのデザイン

通信プリミティブとしては主に，データ転送，ブロード・キャスト，バリア同期を提供する．データ転送はリモート・メモリ転送をベースにしている．リモート・

メモリ転送はメッセージ通信に比べてオーバヘッド削減の点から有利だと考えられるためである．たとえば，メッセージ通信ではフロー制御やバッファ管理等が必要になる．SMP 計算機ではこれらに加えてホスト・プロセッサ間の排他制御が必要である．さらに，メッセージ通信ではバッファへのコピーが必要になる場合がある．バッファへのコピーはバスが共有されていることから単一プロセッサの場合よりも性能に与える影響は深刻である．また，リモート・メモリ転送は NI 上の DMA によって直接実行されるのでホスト・プロセッサの関与は不要である．

ホスト・プロセッサへの同期成立の通知には，主記憶上のフラグを用いる．SMP PC のハードウェアはキャッシュ・コヒーレンシ操作を実装しているので，フラグを用いる通知は高速である．また，主記憶上のフラグはキャッシュされるので，ホストがビジー・ウェイトしてもバス・トラヒックは発生しない．

NICAM では，NI 上のプログラミングにアクティブ・メッセージ（AM）[6] の枠組みを用いる．AM ではデータ・パケット中にハンドラ関数が指定され，受信側はパケットを受信した時点でそのハンドラを起動する．ハンドラはパケットからバッファへのデータのコピー等の処理を行う．NI 上のマイクロ・プロセッサのプログラミングに AM を使うことで拡張性のある実装が可能である．AM ではすべての処理がハンドラ呼び出しを経由するので，プログラムの各部分が独立であり，拡張は新しいハンドラを追加することで行うことができる．

一般的な AM の実装ではハンドラの処理をホスト・プロセッサで行うが，そのときホストにはポーリングや割込み等の処理コストがかかる．通信処理を NI で行うことでこのポーリングの負荷もホストから削減できる．この場合，通信オーバヘッドは NI 上のプロセッサの起動だけになる．

NICAM は単一ユーザ，単一ジョブのみをサポートし，ジョブの間でのプロテクションは提供しない．通信にかかわる資源は独占的に利用できるものと仮定している．通信プリミティブはスレッド・セーフであり，NI 上で要求を逐次化しているので SMP ノード上のプロセッサ間での排他制御は必要ない．ただしスレッド数はプロセッサ数（COMPaS の場合は 4）を上限としている．

3.2 リモート・メモリ転送

図 3 に NICAM のプリミティブをあげる．リモート・メモリ転送としては，ローカルな `bcopy` とインタフェースを合わせた `nicam_bcopy` を提供する．これには送り元と送り先のノード番号を指定する．送り元ノードと転送要求を発行したノードが一致しない場合，要求は NI 上で AM を使って送り元ノードへフォワードされる．`nicam_sync` はローカルなノードが発行したすべての転送が処理された時点でフラグをセットする．

`nicam_bcopy_notify` はリモート・メモリ転送のバリエーションであり転送完了の通知機能を実装する．転送終了後，送り先ノード上で指定されたフラグが `onoff` の値に従ってセットされる．この同期を含む通信はデータ並列計算で重要である．リモート転送ベースの通信によって同期を削減する最適化が知られているが，そのプリミティブにはリモート・ノードへの通知を仮定している[7]．同様に，`nicam_bcopy_countup` では指定したカウンタと受信回数が一致したところでフラグがセットされる．このメッセージ数のカウントを使うと，通信フェーズの終了検出に明示的な同期を不要にできることが知られている[8]．

リモート転送に使用する領域は，あらかじめページングされないように物理メモリにピンダウンする必

初期化

```
nicam_init()
nicam_lock_memory(addr, range)
```

リモート・データ転送（転送のみ）

```
nicam_bcopy(src_node, src_addr,
        dst_node, dst_addr, size)
nicam_sync(flag_addr)
nicam_write1(dst_node, flag_addr, val)
```

リモート・データ転送（到着通知あり）

```
nicam_bcopy_notify(src_node, src_addr,
        dst_node, dst_addr, size,
        flag_addr, onoff)
nicam_bcopy_countup(src_node, src_addr,
        dst_node, dst_addr, size)
nicam_set_counter(flag_addr, count)
```

ブロードキャスト

```
nicam_bcast(src_node, src_addr, dst_addr,
        size, flag_addr, onoff)
nicam_bcast_discard(onoff)
```

バリア同期

```
nicam_barrier(flag_addr)
```

メッセージ通信サポート

```
nicam_bcopy_src(key, dst_node, src_addr,
        size, flag_addr, onoff)
nicam_bcopy_dst(key, src_node, dst_addr,
        size, flag_addr, onoff)
```

図 3 NICAM の主な通信プリミティブ
Fig. 3 Operations of NICAM.

要がある．これは nicam_lock_memory によって行う．NICAM では実行環境として単一タスクを想定しているので，物理メモリの領域を自由にピンダウンしている．

NICAM ではメッセージ通信は直接サポートしないが，プログラムの単純な書き換えでリモート・メモリ転送に置き換え可能なことも多い．送信側が受信側のバッファ・アドレスを知っている場合，リモート転送を直接使用することが可能である．このとき同期にはフラグ書き込みが使用できる．しかし，送信側が受信側のバッファ・アドレスを知らない場合，単純な書き換えは難しい．この状況をサポートするために，プリミティブとして nicam_bcopy_src, nicam_bcopy_dst を用意した．これは送信側にキューを持ち，key を使った簡単なマッチングによる送信・受信のランデブを行う．双方がそろった時点でリモート転送が起動される．

3.3 バリア同期

NICAM のバリアはその成立を主記憶中のフラグを使ってホスト・プロセッサに通知する．フラグはバリア起動時にリセットされ，バリアが成立した時点で再びセットされる．このバリアはバリア成立待ちを含まないので，ファジィ・バリア[4),9)] になっている．また，バリアのインスタンスごとに異なるフラグを使用するので，バリア区間のオーバラップを許すバリアにもなっている[10)]．

このバリアはノード間のバリアを提供するもので，SMP ノード内のプロセッサ間でのバリアは含意しない．ノード内の同期は共有メモリを使用するので高速であり，自由な形態が考えられるので NICAM では規定していない．しかし，NICAM ではバリアの成立をフラグにより通知するので，各プロセッサが独立にバリア成立をチェックすることが可能である．これによりバリアのファジィ性をプロセッサ間で直接利用できる．

さらに，このバリアは複数インスタンスを許す一般化された実装を用いている[4)]．つまり，起動したバリアの成立以前に次のバリアの起動を開始してもよい．ただし，NICAM ではマルチ・ステージのアルゴリズムを使用するため文献 4) と異なり，ステージごとのキューを使う実装になっている．バリアのインスタンスは FIFO 順に成立するので，実装はキューで十分である．通信レイテンシが大きい場合やクラスタのスケールが大きい場合に，複数バリアの同時実行に意味がある．

図 4 にバリアの実行ステップを示す．バリアの実装は，バタフライ・パターンで通信するマルチ・ステージ（ノード数 P に対して $\log(P)$ ステップ）のアルゴリズムを使用する．それぞれのステージでは，一対（2-way）のジョインが行われる．ノード n は i 番目のステージでノード $n \oplus 2^i$ と同期をとる．このジョイン部分には小さなサイズのキューが置かれ，複数のバリアが同時実行可能なようになっている．この実装ではノード数は 2 の巾に制約される．

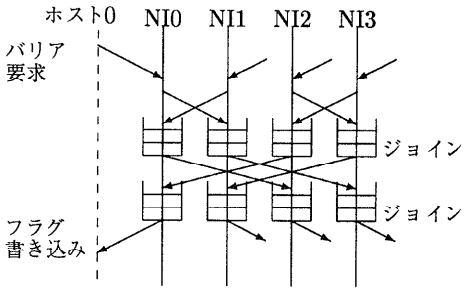


図 4 マルチ・ステージのバリアの実行ステップ
Fig. 4 Steps of the multi-stage barrier.

3.4 実　　装

NI によるポーリングの実装　NI は AM 要求ソースに対してポーリングでチェックを行う．要求ソースには，ホストからの要求，受信 DMA エンジン，主記憶 DMA エンジンの 3 つがある．NI はそれぞれの要求ソースにパケットがあればそのパケット中のハンドラを起動する．ホスト–NI 間では，ホストからの要求があるか，NI により要求が受け付けられたかどうかを互いに知らせるためフラグ対を使用したハンド・シェイクを行う．ホストから NI への要求の通知は，NI 上の SRAM にとったフラグをセットする．逆に NI からホストへの受け付けアクノリッジは，主記憶中にとったフラグをセットする．マルチ・スレッド対応には，プロセッサ数のフラグ対を用意している．それぞれ独立のフラグ対を使用するのでホスト・プロセッサ間での排他制御は不要である．

物理アドレス変換　仮想アドレスから物理アドレスへの変換テーブルを NI 上の SRAM に持っている．SRAM には自ノードの物理メモリ・アドレスをページ・テーブルとして持っている．つまりネットワーク経由でのやりとりにはすべて仮想アドレスを用いる．現在，各ノードが持っている物理メモリは 128 MB であるが，これはページごとの物理アドレスを 1 ワードで記憶する場合 NI 上の SRAM に十分に入る量である．仮想アドレスから物理アドレスへの変換ではテーブルのレンジとエントリをチェックするので，違法なアドレスは検出される．

リモート・メモリ転送の実装　メモリ転送には，ま

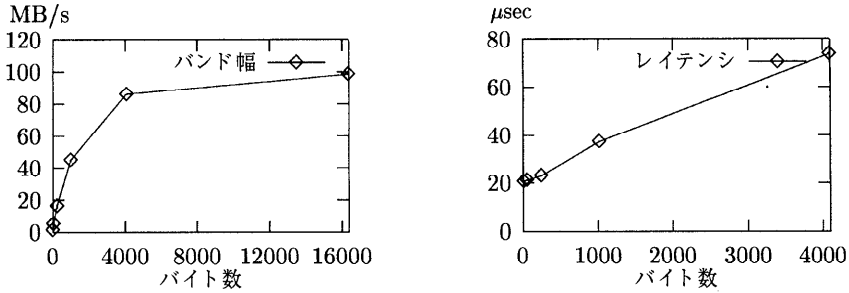


図 5 NICAM の基本性能．バンド幅（スループット）とレイテンシ（片道）
Fig. 5 Basic performance of NICAM, latency and bandwidth.

ず送り元側では与えられた仮想アドレスをテーブルを引いて物理アドレスに変換後，DMA を起動する．DMA が終了すれば，そのデータに AM 用のヘッダと引数を付けて送り先に送信する．送り先では与えられた仮想アドレスを物理アドレスに変換して DMA を起動し，リモート・メモリ転送を完了する．

3.5 基本性能

リモート・メモリ転送 図 5 に 16 KB までのリモート・メモリへの転送速度と 4 KB までのレイテンシを示す．リモート転送ではユーザ空間へ直接 DMA を行っている．小さいサイズでは片道で約 20 μsec のレイテンシが見られる．転送速度は最大で約 105 MB/s (64 KB 時) である．転送の立ち上がりを示すパラメータである $N_{\frac{1}{2}}$（最大転送速度の半分の速度が得られるデータ・サイズ）は，約 2 KB である．

バリア同期 図 6 にバリア同期にかかる時間を示す．対比のため同じ Myrinet を使った通信ライブラリである PM[11] を使った場合の性能をあげる．PM でのバリアにはバタフライ・パターンで通信する NICAM と同様の実装を使った．これはメッセージ通信を使ってホスト上で実装されている．

2 ノードでは PM に比べて NICAM が遅い．ホスト–NI 間のハンドシェイクのコストが大きいためである．しかし，バリアの各ステップの実行は NI で直接行われるためホスト–NI 間のコストはかからない．よってコストは小さく，ノード数が増えるに従い NICAM の方が速くなる．ノード間にかかるコストはステージあたり約 7 μsec である．

呼び出しオーバヘッド 通信プリミティブの呼び出しにかかるオーバヘッドであるが，実装では `nicam_bcopy` に約 5 μsec かかる．`nicam_bcopy` の呼び出しは以下の 3 つの処理を行う：(1) 引数のチェック，(2) AM パケットの NI の SRAM へのコピー（ハンドラ・アドレスと引数 6 ワード分），(3) SRAM 上のフラグへの書き込み．ここで，ステップ (2) と (3) の後に逐次化命令（`CPUID` 命令）を必要とするが，このコストが大きい．Pentium Pro 系のプロセッサでは書き込み順が保証されないので，ストア・バッファをフラッシュするために逐次化命令が必要である．

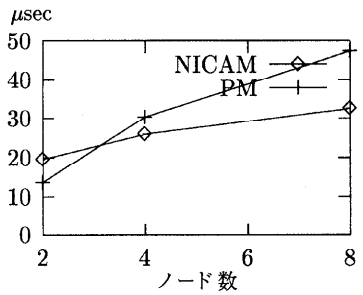


図 6 NICAM と通信ライブラリ PM のバリア同期時間の比較
Fig. 6 Barrier synchronization time between nodes using NICAM and a message passing library PM.

実装のトレードオフ ホスト–NI 間のハンドシェイクや同期成立の通知に主記憶のフラグを使用することはバス・トラヒックとオーバヘッドを減らすために必要である．しかし，これはレイテンシが犠牲になっている．ホスト・プロセッサに比べて格段に低速な NI のプロセッサでは，この主記憶のフラグをセットする処理のコストが大きい．別のトレードオフとしては，NI 上の SRAM にフラグを置きホスト・プロセッサが PCI バスを介してチェックする方法がある．低レイテンシに主眼を置く Myrinet を用いる通信レイヤでは，この実装を選択していることが多い[11)~13)]．ただしその場合，ホストによるポーリングによってバス・トラヒックが発生することになる．

4. バリアのデータ並列計算への応用

4.1 SPMD データ並列ライブラリ

ホスト・プロセッサの介入なしに実現されるバリア同期の応用としてデータ並列ライブラリをあげる．この応用では，バリアをバック・グラウンドとして処理

することでデータ並列計算で現れるバリアのコストを隠蔽することができる．

NICAM の重要なターゲットの 1 つとして，C++ によるデータ並列ライブラリへの適用がある[14]．これはベクタ計算ライブラリであり Fortran 90 ライクなオペレーションを提供する．提供するオペレーションはリモート・メモリ転送を用いて実装することができる．ライブラリは SPMD で実行され，基本的には BSP（Bulk Synchronous Program）モデルに従って実行されると考えられる．SPMD 動作ではノードを基本的に単一プロセッサとして扱い，配列へのオペレーションの適用時点でノード内の複数プロセッサを利用するものとする．つまり配列の割当てや解放に関しては各ノードごとに行われ単一プロセッサとして動作を行う．ライブラリはベクタ計算を基本としており，すべてのデータ並列動作は並列オペレーションの適用である．

BSP モデルでは，通信と計算がバリア同期で足並みを揃える「スーパー・ステップ」を区切りにして進むと考える．ライブラリでは各オペレーションごとに同期するためバリア同期が頻繁に必要になるので，バリアのコストを削減することが重要である．

ライブラリでの通信にリモート転送を使用する場合，リモート・ノードでのデータ・アドレスを知る必要が生じる．このため，メモリ・アロケータは全ノードで同じアドレスにデータを割り当てるようにしている．ベクタ計算ライブラリは SPMD で実行され全ノードでいっせいにデータの割当てが起こるので，同一アドレスへ割り当てることが可能である．

4.2 バリア同期のコストの削除

データ並列計算では，リモート・メモリ転送を使用することによりバリア同期を削除する方法が知られている[7]．このとき，リモート転送では使用中の領域を上書きする可能性があるが，通信とメモリ・アロケータを協調することで上書きを避けることができる．これには，通信をつねに新たに割り当てられたメモリ領域に対してのみ行うようにする．これによって上書きの可能性はなくなる．

ただし，これをライブラリに適用するにはさらに条件が必要である．ライブラリではメモリ領域の使用に関する詳しい情報が得られないため，メモリ領域へのすべてのノードから参照がなくなってから解放する必要がある．全ノードの解放を待たずにローカルに解放した場合，再割当てされた領域にリモートから書き込まれる可能性が生じるためである．この条件は，ローカルな解放に対してバリアをとることで保証できる．すべてのノードにおいて解放の時点に到達したことは，バリアにより確認することができる．

このような目的にバリアを使用する場合，バリアが通信や計算と独立した時点に起動されるのでホスト・プロセッサ介入の必要がないことが望まれる．メモリ領域の解放ごとにホスト・プロセッサのポーリングが必要になると逆に性能が悪くなることも考えられる．また，このバリアはデータ領域解放時に起動されるのでノード間のばらつきが大きいとも考えれる．複数バリアの同時進行を許容していることも望まれる性質である．

5. 実験結果

5.1 計算/通信オーバラップでの効果

COMPaS 上では，分散メモリ計算とマルチ・スレッド計算の両方を使うハイブリッド計算の研究が進められている．ハイブリッド計算では，プロセッサを有効利用するため計算/通信のオーバラップを行う．ここでの例題には，ヤコビ法によるラプラス解法を使用する．それぞれの繰返しで新しい配列 u' の値を古い配列 u の値を使って計算する：

$$u'_{i,j} = (u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1})/4$$

配列の分散は，同サイズの帯に分割し各ノードに分散する．各ノードではそれをさらに分割してそれぞのプロセッサに割り当てる．境界には隣接するセルの値を保持するゴースト・セルを置く．繰返し間でゴースト・セルの値を更新する．境界内部の要素にはノード間の依存がないので，通信を待たずに計算を開始することが可能である．

図 7 にノード数を 8 で固定，各ノードのスレッド数を 1，2，4 と変えた場合の実行結果を示す．結果は単一プロセッサで実行した場合との比で示す．配列のサイズは 1024×1024 である．この問題ではバス・トラヒックが大きいため，スレッド数を増やしたときのス

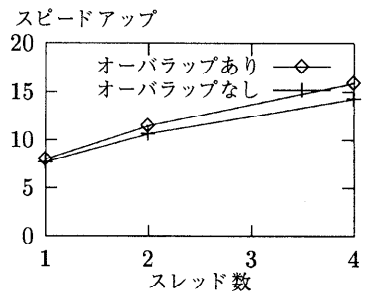


図 7 計算/通信のオーバラップによるスピードアップの比較
Fig. 7 Speed-up difference of Laplace solver between overlapping and non-overlapping versions.

```
_cshift(caddr_t dst, caddr_t src, int dir,
    int off, int siz)
{
    /*self はノード番号*/
    /*nodes は全ノード数*/
# if BSP
    nicam_barrier(barr_flag);
    while ( *barr_flag != 1 );
# endif
    shift_sense = !shift_sense;
    nicam_bcopy_notify(self, (src + off),
        (self+nodes+dir)%nodes, dst, siz,
        shift_flag, shift_sense);
    while ( *shift_flag != shift_sense );
}

_reclaim(caddr_t storage)
{
# if BSP
    free(storage);
# else
    insert_to_pending_list(storage);
    nicam_barrier(storage);
    /*解放できるものがあるかチェック*/
# endif
}
```

図 8 C++ データ並列ライブラリでの cshift のコード概要
Fig. 8 A skeleton of the relaxed cshift code.

ピードアップが悪い．この状況でもオーバラップにより約 10%のゲイン（32 プロセッサ時）がある．スレッド数が多くなるに従って通信回数は増えるが，オーバラップによるゲインが大きくなっていることから，通信オーバヘッドは十分小さいと考えられる．

5.2 データ並列ライブラリでのバリア削除

レイテンシ隠蔽の効果を示すため，2 つのバージョンの cshift（cyclic shift）オペレーションを比較する．cshift は配列を指定された量だけ環状にシフトする処理である．**図 8** にバリアを削除した版とバリアで同期した版を示す．BSP のプリプロセッサ行を持つものがバリアで同期した版である．バリアで同期した版ではライブラリ・ルーチンの入口にバリア同期が必要である．一方のバリアを削除した版ではライブラリ・ルーチンの入口から領域解放時にバリアを移動している．この場合，領域の解放はバリアが成立するまで遅延される．

図 9 に cshift の実行時間を示す．8 ノードを使用し各ノードではプロセッサ 1 つで実行している．実験はサイズ $N \times N$ の 2 次元アレイに対して cshift を行った．シフト量は 1 である．グラフでは N を横軸にとっている．結果は，バリアのコスト分の速度向上が得られている．さらに同期条件が緩くなっているのでわずかではあるがバリアのコスト以上に速度が向上

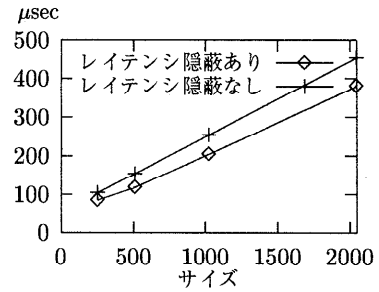


図 9 バリア・レイテンシ隠蔽の効果
Fig. 9 Relaxation effect in cshift operation.

している．

6. 関連研究

NI 上の AM 文献 15), 16) では NI 上での AM の性能評価を行っている．そして Paragon や Berkeley NOW 上では NI 上の AM がホスト上の AM に比べて低レイテンシであることを実験により示している．NICAM では，SMP クラスタ上で望まれる低オーバヘッド通信の実現に NI 上のプロセッサを利用している．さらに，NI 上の処理をリモート・メモリ転送にとどめず，バリア等の同期プリミティブの実装にも使用し有効性を示した．

Cray T3E T3E[17] では，リモート・メモリ転送，DMA エンジン，バリア同期等リモート転送ベースの通信に必要な機能の多くをハードウェアで提供している．リモート・メモリ操作には，アトミックなインクリメントやデクリメントを含んでいる．NICAM はこれらプリミティブをコモディティ・ハードウェア上で提供しようとするものである．しかし，ハードウェアによる実装と異なりレイテンシが大きいため，レイテンシ・トレラントな実装を採用している．また，NICAM はソフトウェアによる実現であるのでメッセージ通信のための若干複雑なプリミティブ等も提供可能である．

Myrinet を使用する通信 Myrinet を使用する他の BIP[12], FM[13], PM[11] といった通信に対して，NICAM では通信オーバヘッドの低減に重点がある．この点ではレイテンシやバンド幅で劣っている部分がある．通信レイテンシについては，NICAM の約 20 μsec に比べて，他は 7 μsec から 10 μsec 程度と報告されている．FM 等の報告から，ホスト・プロセッサが直接 NI 上の SRAM にアクセスするのがレイテンシ削除に効果があることが知られている．また，ホストに比較して遅い NI での処理が大きいのもレイテンシの点では劣る原因になっている．ただし，バンド幅に関しては NICAM も，直接 DMA を使用する通

信（いわゆるゼロコピー通信）によりほぼハードウェアの限界に近い性能を実現している．

7. おわりに

NICAM では NI 上のプロセッサを有効活用し，低オーバヘッドなデータ転送と高速な同期プリミティブを実現した．NI 上のプログラミングを柔軟に行うために AM の枠組みを利用した．低オーバヘッドな通信の効果を，計算/通信オーバラップの実験で示した．また，NI による同期の実装が高速であること，同期レイテンシを隠蔽するのに有効であることを示した．

これらプリミティブは SMP PC クラスタ上の通信に対する要請であったが，SMP 上に限らず単一プロセッサでも有効なものである．デザイン・トレードオフとしては，レイテンシやバンド幅よりも通信オーバヘッドの低減を重視した．これはオーバヘッドが直接プロセッサの利用にかかわっており，その低減により計算資源の有効利用が可能になると考えられるためである．一方で同期に関してはレイテンシが重要であるが，これについては同期プリミティブをデータ転送から組み立てるのではなく直接 NI 上に処理を実装した．これにより不必要にホスト上の処理が介在することのない低レイテンシな実装が可能になった．

NICAM ではリモート・メモリ転送をベースにしたが，データ並列計算に関する限りこれに問題はない．データ並列計算に現れる通信では送信側でデータの送り先アドレスが分かっていることが多いので，一方向通信 (one-sided) が適している．他方，メッセージ通信では，送信/受信をペアにするためにノード間で実行順序を入れ換えたり，不必要に同期がとられる等の無駄がある．

通信プリミティブを NI 上に実現するうえで，バリア同期に関してはその処理量が少ないので NI 上の比較的性能の低いプロセッサによる処理でも有効性が示せた．ただし，リダクション等の計算を含む通信のクラスは NI による処理が望まれる部分であるが，Myrinet の NI は浮動小数点ユニットを持っていないので有用なオペレーションを実装できない点が残念である．

参考文献

1) Culler, D.E., Karp, R.M., Patterson, D.A., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R. and von Eicken, T.: LogP: Towards a Realistic Model of Parallel Computation, *Proc. 4th Symp. on Principles and Practice of Parallel Programming*, pp.1–12 (1993).
2) Martin, R.P., Vahdat, A.M., Culler, D.E. and Anderson, T.E.: Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture, *Int'l Symp. on Computer Architecture (ISCA'97)* (1997).
3) Boden, N.J., Cohen, D., Felderman, R.E., Kulawik, A.E., Seitz, C.L., Seizovic, J.N. and Wen-King, S.: Myrinet – A Gigabit-per-Second Local-Area Network, *IEEE MICRO*, Vol.15, No.1, pp.29–36 (1996).
4) 松本 尚：細粒度並列実行支援マルチプロセッサの検討，情報処理学会論文誌，Vol.31, No.12, pp.1840–1851 (1990).
5) Tanaka, Y., Matsuda, M., Ando, M., Kubota, K. and Sato, M.: COMPaS: A Pentium Pro PC-based SMP Cluster and Its Experience, *IPPS'98 Workshop on Personal Computer Based Networks of Workstations*, Chiola, G. and Conte, G. (Eds.), Lecture Notes in Computer Science, Vol.1388, pp.486–497, Springer-Verlag (1998).
6) Eicken, T., Culler, D.E., Goldstein, S.C. and Schauser, K.E.: Active Messages: a Mechanism for Integrated Communication and Computation, *Proc. 19th Int'l Symp. on Computer Architecture*, pp.256–266 (1992).
7) Gupta, M. and Schonberg, E.: Static Analysis to Reduce Synchronization Costs in Data-Parallel Porgrams, *Symp. on Principles of Programing Languages 1996*, pp.322–332 (1996).
8) Kim, J.S., Ha, S. and Jhon, C.S.: Efficient Barrier Synchronization Mechanism for the BSP Model on Message-Passing Architecutres, *Proc. 1st Merged International Parallel Processing Symposium and Symposiumon on Parallel and Distributed Processing (IPPS/SPDP'98)* (1998).
9) Gupta, R.: The Fuzzy Barrier: A Mechanism for High Speed Synchronization of Processors, *3rd Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-III)*, pp.54–63 (1989).
10) 高木浩光，有田隆也，曽和将容：細粒度並列実行を支援する種々の静的順序制御方式の定量的評価，並列処理シンポジウム JSPP'91, pp.269–276 (1996).
11) Tezuka, H., Hori, A., Ishikawa, Y. and Sato, M.: PM: An Operating System Coordinated High Performance Communication Library, *High-Performance Computing and Networking*, Lecture Notes in Computer Science, Vol.1225, pp.708–717, Springer-Verlag (1997).
12) Prylli, L. and Tourancheau, B.: BIP: A new protocol designed for high performance net-

working on Myrinet, *IPPS Workshop on Personal Computer based Networks of Workstations (PC-NOW)* (1998).
13) Pakin, S., Lauria, M. and Chien, A.: High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet, *Supercomputing'95* (1995).
14) Matsuda, M., Sato, M. and Ishikawa, Y.: OBP Lib: An Object-Oriented Parallel Library and its Preliminary Performance, RWCP Technical Report, TR-97009 (1998).
15) Schauser, K.E., Scheiman, C.J., Ferguson, J.M. and Kolano, P. Z.: Exploiting the Capability of Communications Co-processor, *10th International Parallel Processing Symposium* (1996).
16) Krishnamurthy, A., Schauser, K.E., Scheiman, C.J., Wang, R.Y., Culler, D.E. and Yelick, K.: Evaluation of Architectural Support for Global Address-Based Communication in Large-Scale Parallel Machines, *Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, pp.37–48 (1996).
17) Scott, S.L.: Synchronization and Communication in the T3E Multiprocessor, *7th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, pp.26–36 (1996).

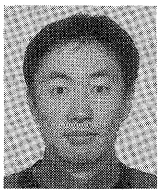
(平成10年9月2日受付)
(平成11年2月8日採録)

松田 元彦（正会員）

昭和38年生．昭和63年京都大学理学部卒業．同年住友金属工業（株）入社．平成7年より技術研究組合新情報処理開発機構に出向．オブジェクト指向言語によるデータ並列計算の研究に従事．

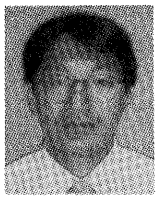
田中 良夫（正会員）

昭和40年生．昭和62年慶応義塾大学理工学部数理科学科卒業．平成7年同大学大学院理工学研究科数理科学専攻博士課程単位取得退学．平成8年より技術研究組合新情報処理開発機構に勤務．現在同組合並列分散システムパフォーマンスつくば研究室主任研究員．工学博士．並列システムの性能評価およびクラスタシステムでの高性能計算に関する研究に従事．ACM会員．



久保田和人（正会員）

昭和39年生．昭和63年早稲田大学理工学部電子通信学科卒業．平成5年同大学大学院理工学研究科博士課程修了．同年（株）東芝入社．平成7年10月より平成10年9月まで技術研究組合新情報処理開発機構に出向．工学博士．並列計算機のプログラミング環境，性能評価環境，計算機クラスタシステムに興味を持つ．



佐藤 三久（正会員）

昭和34年生．昭和57年東京大学理学部情報科学科卒業．昭和61年同大学大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成3年，通産省電子技術総合研究所入所．平成8年より，技術研究組合新情報処理開発機構つくば研究センターに出向．現在，同機構並列分散システムパフォーマンス研究室室長．理学博士．並列処理アーキテクチャ，言語およびコンパイラ，計算機性能評価技術等の研究に従事．日本応用数理学会会員．