計算機アーキテクチャ 125－10
（1997. 8. 20）

SMPクラスタ向けネットワーク・インタフェース上AM通信

松田元彦† 手塚宏史† 田中良夫†
久保田和人† 安藤誠† 佐藤三久†

SMPノードを用いてクラスタを構成した場合，メッセージ・パッシングによる通信では各ノード上のスレッドに対してバッファ管理に排他制御が必要になりバッファへのコピーが共有資源であるバスに負荷をかけるといった問題がある．これらは通信プリミティブをリモート・メモリ転送にすることで避けることができる．本稿では，この実装にネットワーク・インタフェース (NI) 上のアクティブ・メッセージ (AM) 通信を使用することを提案する．データ転送はNI上のDMAの起動によって実行されるので，NI-NI間で要求をやり取りするNI上のAMにうまく適合する．リモート・メモリ転送ベースの並列計算では明示的な同期が必要になるが，この同期もNI上のAMで実現する．これはホスト-NI間のオーバヘッドを削減するので高速に実装できる．

Network Interface Acitive Messages on SMP Clusters

MOTOHIKO MATSUDA ,† HIROSHI TEZUKA ,† YOSHIO TANAKA ,†
KAZUTO KUBOTA ,† MAKOTO ANDO † and MITSUHISA SATO†

Remote memory operations are considered more suitable than message passing on SMP clusters, because message passing suffers from handling of message buffers that needs mutual exclusions and copying of messages that burdens the limited bus traffic. In this paper, we exploit Active Messages on Network Interfaces (NI) to implement primitives for remote memory operations. Active Messages exchanged between NIs suitably handle invocation of DMA on the remote NIs without involving a host processor. Also, Active Messages on NIs provide synchronization mechanism which is necessary for phasing computations in a parallel computation based on remote memory transfers. Active Messages on NIs enable fast barrier synchronization by reducing the host-NI overhead.

1. はじめに

本稿では，ネットワーク・インタフェース (NI) 上にアクティブ・メッセージ (AM) を実装し，SMPクラスタに適した低オーバヘッドなデータ転送と高速な同期を提供する通信について述べる．

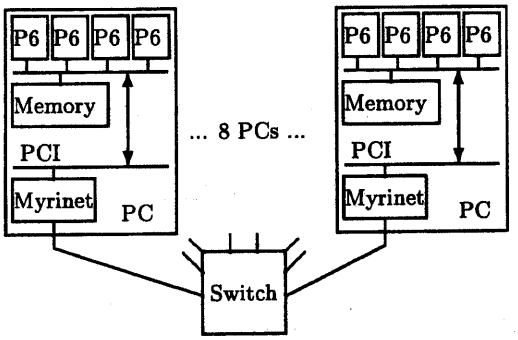
ネットワーク・ハードウェアにはNI上にプロセッサを持つものも少なくない．たとえばMyrinet[1]，Paragon, Meiko CS-2, Flashなどが知られている．NIには専用プロセッサが使用されることが多いが，ホスト(メイン)・プロセッサと同じ汎用プロセッサを使用するものもある．これらは主にデータリンク層相当を実装するのに用いられており，ホスト・プロセッサから通信処理の負担を軽減している．

アクティブ・メッセージ (AM)[2] は，RPCのようにメッセージ中にハンドラ関数が指定され，受信側ではメッセージを受信した時点でそのハンドラを起動する．AMは一般に低レベルの通信機能のみを提供するだけである．つまりAMの実装は受信とAMハンドラの起動のみを行う．メッセージ・パッシングを実装するにはバッファ管理やフロー制御などが必要であるが，AM自体はこれらを提供しない．AMは基本的な通信機能を提供するだけであるので，低オーバヘッドで実現されている．

我々のターゲットとなる計算機は対称型マルチプロセッサ (Symmetric Multi-Processor, SMP) PCをノードするSMPクラスタである．SMP計算機上でマルチ・スレッド計算を行う場合，メッセージ・パッシングによる通信を使用するとバッファ制御などに排他制御が必要になる．また，バッファへのコピーはバスが共有資源であることから単一プロセッサの場合より性能に与える影響は大きい．

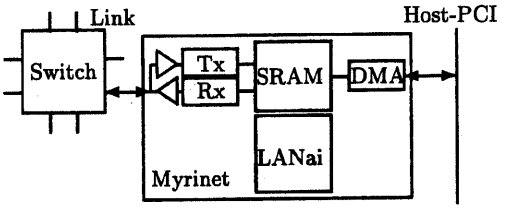
そこでこれらの問題点を避けるため，通信プリミティブとしてリモート・メモリ転送を使う．リモート・メモリ転送により通信に関わる排他制御を最小限にすること，およびバスへの負荷を抑えることができる．リモート・メモリ転送はNI上のDMAエンジンによって行うこ

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



- P6: Pentium Pro 200 MHz
- Switch: 8-port Myrinet スイッチ

図 1 SMP クラスタの構成



- Tx: 送信 DMA エンジン
- Rx: 受信 DMA エンジン
- DMA: 対ホスト・メモリ DMA エンジン
- LANai: 32-bit カスタム CPU コア

図 2 Myrinet の構成

とができるので, ホスト・プロセッサが関与する必要がなくなる.

通信プリミティブをリモート・メモリ転送にすると, メッセージ・パッシングでは暗黙であった同期の問題が表面化する. リモート・メモリ転送を使用する並列計算ではバルク転送に続く同期という形で計算が進められるので, 同期の中でも特にバリア同期が重要になってくる. そこで同期を NI 間の通信により実装することにした. NI 間で同期を実装することによりホスト-NI 間のオーバヘッドをなくし高速に実現するできる. またホストのポーリングの手間を NI が肩代わりするのでホストのオーバヘッドも削減できる.

これらホストの関与を減らし, NI 間でリモート・メモリ転送と高速な同期を実装する枠組みとして AM を使用する. AM では要求は送り手からの一方通行であるが, これは所望のプリミティブを実装するのに適している. NI 上のプログラミングに AM を使うことで見通し良く拡張性のある実装が可能である. NI 上の処理はすべてハンドラを経由するのでプログラムの各部分が独立になり, 他に影響を与えずに新しいハンドラを追加することができる.

この拡張性により同期を含んだ通信プリミティブなどの実装を実験するのが容易になっている. 今回は一例としてリモート・メモリ転送と同期を複合したプリミティブである通知機能を付加したリモート・メモリ転送を実装した.

以下では, 使用したネットワークとその NI 上のプロセッサの基本性能について述べたあと, 実装する NI 上の AM と通信モデルを説明する. 続いて実装方法と基本性能を示す. 本稿では SMP 計算機クラスタ上で有効な通信方式として検討しているが, その評価はプログラミング上の定性的な議論にとどめる. 定量的な評価は今後の課題である. 最後に関連研究とまとめを述べる.

2. クラスタの構成と基本性能

2.1 SMP クラスタの構成

図 1 にターゲットとなる SMP クラスタの構成を示す. 4-way SMP の Pentium Pro PC (200 MHz, 512 KB cache, 128 MB メモリ) を 8 台接続している. ネットワークとしては Myricom 社 Myrinet[1] を使用している. 各 PC に Myrinet PCI ボードを挿し, 1 つの Myrinet 8-port スイッチを介して接続している. PC の OS には Solaris/x86 を使用している.

Myrinet は通信リンク, ホスト・インタフェース, スイッチから構成される. 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. ホスト・インタフェースは PCI である. Myrinet スイッチは 8 × 8 のクロスバで cut-through 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ソース・ルーティングを使う.

2.2 Myrinet ボードの構成

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 32-bit CPU コアを中心に 256 KB の SRAM メモリ, 通信リンク・インタフェースと 3 つの DMA エンジンにより構成される. SRAM 以外は 1 チップ上に実装されている. DMA エンジンの 1 つはホストの主記憶と Myrinet ボード上の SRAM 間で DMA を行う. 他の 2 つの DMA エンジンは Myrinet ボード上の SRAM と通信リンク間で DMA を行うものであり, 送信/受信それぞれ専用である. Myrinet ではすべての通信は Myrinet ボード上の SRAM を経由する. ホスト・プロセッサと Myrinet のやり取りは主記憶-SRAM 間の DMA, あるいはホスト・プロセッサによる SRAM へのアクセスにより行われる. ホストは Myrinet ボード上の SRAM を PCI を通して直接アクセスすることができる. DMA としてはワード単位 (32 bit) の連続転送のみがサポートされている.

表 1 Myrinet の基本性能 (実測値)

Myrinet CPU (LANai)	
LANai MIPS	33 M 命令/sec (MIPS)
送受信エンジン (4 KB まで)	
送信 Gap	0.48μsec + 30.1ns/word (最大 131.3 MB/sec)
送受信 Overhead	約 0.1μsec (DMA 起動時間)
ネットワーク・レイテンシ (1 word ピンポン)	
Latency	片道 1.35μsec
DMA エンジン (4 KB まで)	
ホスト → SRAM Gap	1.32μsec + 31.6ns/word (最大 121 MB/sec)
SRAM → ホスト Gap	0.85μsec + 32.1ns/word (最大 121 MB/sec)
DMA Overhead	約 0.1μsec (DMA 起動時間)
ホストからの Myrinet 上の SRAM アクセス	
PCI を経由する read	0.69 μsec/word
PCI を経由する write	0.51 μsec/word

2.3 Myrinet の特徴

LANai は 32 bit 演算器を持つ RISC ライクな CPU である. 演算は 4 段パイプラインで, 1 サイクルに 1 命令を実行する. プログラムは SRAM に保持されるが, SRAM へのアクセスは 1 サイクルに 2 回行うので命令実行にメモリ参照があったり DMA 中であってもほぼ 1 サイクルに 1 命令を実行できる.

LANai には GNU GCC をポーティングした C コンパイラが提供されており, プログラミングを容易に行うことができる. 通信や DMA エンジンなどはレジスタに書き込むことで起動されるが, これらのレジスタには変数名が割り当てられているため代入文によりアクセスすることができる.

ネットワークの特徴としては以下のものが挙げられる.

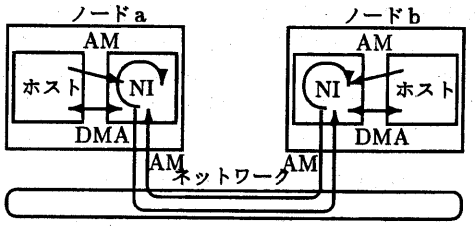
- ネットワークは非常にエラーレートが低い. ネットワークをリアイアブルだとして扱える.
- ハードウェア・フロー制御を持っている. リンク上ではフロー制御を行っているのでデータのロスはない.
- スイッチはクロスバであり, ルーティング経路は静的に指定する.

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る. 現実装では CRC 等のエラー検出は行っているが異常検出時の再転送などの処理は行っていない. 異常の場合アプリケーションを異常終了する. 今回のクラスタ構成ではトポロジはクロスバ 1 段の接続なのでデッドロックについて考慮する必要はない.

2.4 NI の基本性能

表 1 に Myrinet 上のプロセッサである LANai から見た基本性能を列挙する. NI 上で AM の処理を行うためそのプロセッサである LANai から見たオーバヘッドが性能を大きく左右する. そこで LogP モデル[3] に基づいて Latency, Overhead, Gap について実測した. ここで Latency は主にハードウェアによる通信レイテンシ, Overhead は LANai にかかる処理時間, Gap は次にリソースが使えるまでの時間間隔とする. DMA に関しては Latency と Gap は同じ意味である.

DMA 起動にかかる Overhead はレジスタへの書き込み (2–3 レジスタ) のみでありオーバヘッドは少ない. ここで注意が必要なのはホストによる Myrinet SRAM への PCI を経由する読み書きである. ホスト-NI 間のコストが大きいので, 低オーバヘッドな実装を行うのに考慮が必要である.



- ホスト-NI 間では, NI はホストが生成した AM パケットを SRAM 上にコピーし, コピー終了後そのパケット中のハンドラを呼び出す.
- NI-NI 間では, 呼び側 NI は作った AM パケットを Tx-DMA により送りだし, 受け側 NI は Rx-DMA からパケットを受信する. 受け側 NI は受信ししだいそのパケット中のハンドラを呼び出す.

図 3 NI 上の AM

3. NI 上の AM

3.1 NI 上の AM 呼び出し

実装する AM モデルを単純化するため, ホスト-NI 間の呼び出しも AM を使う. 一般に AM ではハンドラの起動はリモートで行われるが, この実装ではホストからの要求はローカル計算機の NI で実行される. 図 3 にホスト-NI 間, NI-NI 間の AM 呼び出しの関係を示す.

ホスト-NI 間の呼び出しでは, まずホストが必要な引数を書き込んだ AM パケットを生成し, 次にそれをコピーによって NI に引き渡す. NI はポーリングによりホストからの要求をチェックしているので, 要求があればそのパケット中のハンドラを呼び出す. 実装するリモート・メモリ転送やバリア同期といったプリミティブの場合ホスト上で行う処理はほとんどないので, ホストの処理は渡された引数をパケットにコピーするだけである.

本実装での AM パケットはタグ, ハンドラ・アドレス, 引数 6 個の固定サイズ (32 byte) である. ハンドラ・アドレスには NI のプログラム・アドレスを使う. タグは実装上は全く不要であるがパケットのチェックに使用している. パケットは転送に無駄があるが実装を簡単にする理由で固定サイズになっている. リモート・メモリ転送に使うパケットではこれに加えデータサイズ分のデータを持つ.

今回の実装ではシステム構成がデッドロック・フリーであるので，リプライなしの AM 実装になっている．この点はハンドラ呼び出し毎にリプライを必要する Generic AM (GAM)[4] とは異なっている．また重要な特徴として，NI 上の AM ハンドラの実行はすべてシリアライズされている．

4. 通信モデル (通信プリミティブ)

通信モデルとしてはリモート・メモリ転送とバリア同期を実装する．リモート・メモリ転送を基本にすることで，下位通信レイヤでのバッファの管理等が不要になりそれに従ってフロー制御も不要になる．

実際には，使用する PC では PCI から主記憶全体が DMA の対象となるのでデータを直接転送することが可能である．よってリモート・メモリ転送にホストが関与する必要がない．DMA のためには物理メモリをあらかじめページングされないようにピンダウンする必要があるが，単一タスクを想定しているので問題ない．

リモート・メモリ転送として，次のようなインタフェースを実装した．

```
xx_bcopy(src_node, src_addr, dst_node,
    src_addr, size)
xx_bcopy_notify(src-node, src-addr,
    dst-node, src-addr, size, flag-addr)
xx_sync()
```

bcopy ではソース・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ソース・ノードへフォワードされる．NI 上の AM を使ったフォワードなのでこのコストは小さい．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るまで待つ．これとバリア同期使うことで転送／同期のフェーズを保証する．

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終了の通知機能を実装する．転送終了後デスティネーション・ノード上で指定されたフラグがセットされる．このインタフェースはネットワークに FIFO 性を仮定しない通信モデルでは重要である (今回ターゲットのネットワークには FIFO 性がある)．

バリア同期には次のようなインタフェースを用意した．

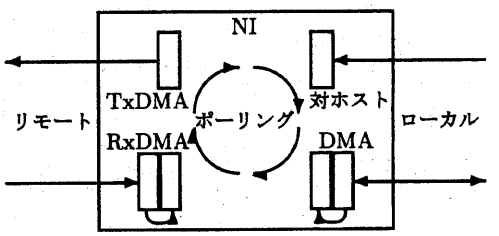
```
xx_barrier()
xx_barrier_post()
xx_barrier_wait()
```

バリア同期は NI 間で同期が取られるので，バリア同期中ホストのプロセッサがポーリングする必要がない．そこでファジィ・バリアも用意した．

5. NI 上の AM 実装

5.1 ポーリングによる AM の実装

図 4 に AM のポーリング・ループの概略を示す．処理ユニットとして NI が持っているのは 3 つの DMA エンジンである．LANai はこれらのリソースにホストからの要求を加えたものを，順次ポーリングして処理の終了や要求の到着を管理する．



- ポーリングのループは各 DMA エンジンの終了とホストからの要求をチェックする．
- ホストからの要求は SRAM 上のフラグよって知らされる．ホストからの要求に対してはパケットをコピーするためにメモリ -DMA を起動する．
- メモリ -DMA と Rx-DMA には待ちキュー (長さ 1) があり，DMA 終了後すぐにサービスできないパケットはそこに置かれる．

図 4 ポーリングによる NI 上の AM 処理

ホスト -NI 間のパケット・コピーは DMA によって行っている．これは，Myrinet の基本性能 (表 1) から分かるようにホストからの PCI を介した書き込みはたいへん遅いためである (AM のパケットのコピーだけで 4μsec 以上かかる)．この値はこのまま CPU のオーバヘッドになるため設計方針に反する．ホストは NI に対して SRAM 上にとったフラグをセットすることでパケットのコピーを要求する．これによってホストから PCI を介した書き込みはこのフラグに対するものだけになっている．

5.2 リモート・メモリ転送の実装

メモリのピンダウン．主記憶への DMA は物理アドレスに対して行われるため，リモート・メモリ転送を実現するため物理ページをページングされないようにピンダウンしている．これにはメモリ転送が行われる領域をアプリケーション開始時点であらかじめピンダウンしている．

使用している OS である Solaris は mlock() 等のシステムコールを持っているのでこれを使用してページをピンダウンする．ただし，これらのシステムコールはスーパユーザのみに許されるので一般アプリケーションからは利用できない．そのためシステムコール呼び出しをデバイス・ドライバに組み込んでいる．また，ピンダウンしたページの物理アドレスを取得する処理もデバイス・ドライバに組み込んでいる．

物理アドレス・テーブル．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 にはローカルなメモリの物理アドレスをページテーブルとして持っている．ネットワーク経由でやり取りされるアドレスはすべて仮想アドレスである．

現在のクラスタでは，各 PC が持っている物理メモリは 128 MB であるので，物理アドレスを 1 ワードで記

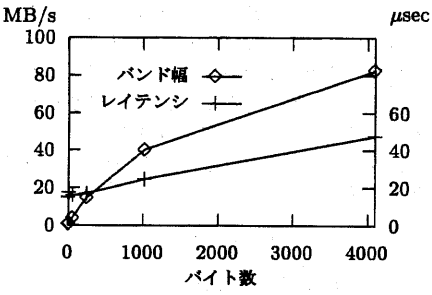


図 5 リモート・メモリ転送の性能

憶すると 128 KB 必要になる (1 ページは 4 KB). これは NI 上の SRAM 256 KB に入る値である. ⋆

メモリ転送の実装. メモリ転送には, まずソース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デスティネーションに送信する. デスティネーション側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.

アドレス例外のチェック. 仮想アドレスから物理アドレスの変換ではテーブルのレンジとエントリをチェックしているので, 違法なアドレスは検出される.

5.3 バリア同期の実装

バリア同期の実装にはバタフライ・パターンで通信する方法をとった. これはノード数 P に対して $\log(P)$ ステップで終了する. 各ノードはそれぞれ $\log(P)$ ステップの 1 対 1 の同期操作を実行する. ノード n は i-th ステップでノード $n \oplus 2^i$ と同期をとる. $\log(P)$ のベースを 2 より大きくとる場合もあるが, ノード数が 8 と少ないので 2 に固定した.

各ステップの 1 対 1 の同期操作は SRAM 上にフラグを取り AM 呼び出しでそのフラグをセットしている. フラグは各ステップごとに個別に用意している. この実装では複数のバリア呼び出しによるフラグへの書き込みが混同される可能性があるので, フラグに書き込む値は単調に増加するカウンタの値を使っている.

6. 性　　能

6.1 リモート・メモリ転送

図 5 に 4 KB までのリモート・メモリへの転送速度とレイテンシを示す. リモート転送はページ境界で区切られるので, 最大で 4 KB である. これはピンダウンしたユーザ空間へ直接 DMA している. 小さいサイズでは約 16 μsec 程のオーバヘッドが見られる. 転送速度は最大で 82.5 MB/s である.

あとで示すように, 小さい転送量の場合のコストはほとんどがホスト-NI 間の AM 呼び出し時間である. デスティネーション NI から転送終了のアクノレッジを返しているがそのコストは観測できない程度の時間である.

6.2 バリア同期

以下にバリア同期にかかる時間を示す.

nPE	NI 上の AM	メッセージパッシング
2	23.79 μsec	13.465 μsec
4	31.45 μsec	30.369 μsec
8	38.67 μsec	47.424 μsec

対比のためプラットフォームが異なるが同じ Myrinet を使った性能の良い通信ライブラリである PM[5] を使った場合の性能をあげる. この値は Sun SPARC Station 20 を使用するワークステーション・クラスタでの計測結果である[6]. バリアはメッセージ・パッシングを使ってホスト上で実装される. バリア自体はバタフライ・パターンで通信する同様の実装である.

NI 上の AM の実装ではホスト-NI 間のコストが大きいが NI 間で処理が行われる各ステップに必要なコストは約 7.5 μsec と, ホスト上でバリアを実装する場合より小さくなっている.

6.3 ホスト-NI 間の呼び出しコスト

上の結果よりホスト-NI 間の呼び出しコストが大きいことが予想される. 実際にそのコストを計測したものを以下に示す.

AM 呼び出し Overhead	約 2 μsec
空の AM 呼び出し Gap	7.33 μsec
リプライのみの AM 呼び出し	17.54 μsec

AM 呼び出し Overhead はホスト・プロセッサが関与するオーバヘッドの時間である. AM 呼び出し Overhead は主記憶間での AM パケットのコピーと 1 ワードの NI 上の SRAM 書き込みである. 実装では AM パケットがピンダウンされた領域にあるとは限らないので 1 度コピーしている. また 1 ワードの書き込みは, AM 要求を NI に知らせるために必要である. 主記憶上でのパケット・コピーのコストは DMA によってキャッシュラインが shared になっているので意外に大きい.

空の AM 呼び出しは, ハンドラはなにもせず AM 要求を消費するだけである. これは次の要求が受け付けられるようになるまでの Gap 時間を計測している.

リプライのみの AM 呼び出しでは, AM ハンドラはホストに対してリプライを返すだけの処理を行う. この呼び出しではホストが結果を取るまでのレイテンシを計測している. NI のポーリング・ループは処理がない場合 1 μsec 程度で終わるので, このほとんどがホスト-NI のオーバヘッドである. ホストはバス・スヌープで DMA による書き込みを知ることができるので, 結果の通知を主記憶上のフラグを待つことで実現している.

⋆ 現在は暫定的に SRAM 量 128 KB のボードを使用しているので, ピンダウンできる物理メモリ量が制約されている.

7. 関連研究

7) では Split-C プリミティブを実装する場合について NI 上での AM の性能評価を行っている. 中には CS-2, Paragon, Berkeley NOW が含まれている. Paragon と 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実装できることを示している. NOW は本稿で扱った Myrinet を使ったシステムであるが, DMA の領域が主記憶全体でないため Split-C プリミティブの実装は実験に終わっている.

本研究では NI 上の AM をリモート・メモリ操作にとどめずバリア等の同期プリミティブの実装に使用した. そして NI 間で AM 呼び出しを行うことでコストのかかるホスト-NI 間の呼び出しを減らせることを示した.

FM Myrinet[8], LAM[9] は Myrinet を使った AM であるがハンドラはホスト・プロセッサで実行される. これらはプラットフォームに独立な AM インタフェースを実現している. 本研究では, 汎用の AM のインタフェースを実装することは行っていない.

クラスタ計算機の 1 つの形態として, 最下位の通信としてリモート・メモリ転送と同期プリミティブを提供しているものがある. 例えば T3E[10], Memory Channel[11] や Dolphin SCI board[12] などでは, ハードウェアでリモート・メモリ転送と同期プリミティブを実現するための機能を実装している. 同期プリミティブは test-and-set の変形などのアトミックなオペレーションを基本にしている. これらは低オーバヘッドな通信の実現と高速な同期を可能にしている. 同期プリミティブとしては分散キューを管理するためのアトミックなスワップなどが考えられているが, これらは NI 上の AM で実装可能であるので並列プログラミングに有効であるかどうかの検証使うことができる.

8. おわりに

ネットワーク・インタフェース (NI) 上の AM を使って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装した. 低オーバヘッドな処理は SMP 上のマルチスレッド・プログラムからの要請である. このためにリモート・メモリ転送をプリミティブにした通信を実現した.

同期に関しては, NI-NI 間で同期を実行をすることにより高速化を図った. これはホスト-NI 間のオーバヘッドを減らすことができる. 同期プリミティブに関してはその処理量は少ないので 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である.

これらのプリミティブは AM という枠組みにより実装されているので, 新たな機能を実現することも容易である. その例として同期のための通知機構を含んだリモート転送を実現した. ただし, リダクション等の計算を含む通信のクラスはこのような AM による実装に適していると考えられるが, NI 上のプロセッサが浮動小数点ユニットを持っていないのが残念である.

実装の第一段としてはある程度の性能を達したが実装にまだまだ改善の余地がある. 今後はその性能を上げるとともに, NI 上で各種の通信／同期プリミティブを実装し SMP クラスタでの有効性をアプリケーションを通じて検証する予定である.

参考文献

1) http://www.myri.com.

2) T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active Messages: a Mechanism for Integrated Communication and Computation, In *Proceedings of the 19th Int'l Symp. on Computer Architecture*, May 1992.

3) D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation, In *Proc. of the 4th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, May 1993

4) D. Culler, et al. The Generic Active Message Interface Specification, 1994. http://now.cs.berkeley.edu/AM/active_messages.html.

5) 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, June 1996.

6) 田中良夫, 久保田和人, 佐藤三久, 関口智嗣. 並列アルゴリズムにおける Collective 通信の性能比較, 情報処理学会研究会報告 96-HPC-62, August 1996.

7) A. Krishnamurthy, K. E. Schauser, C. J. Scheiman, R. Y. Wang, D. E. Culler, and K. Yelick. Evaluation of Architectural Support for Global Address-Based Communication in Large-Scale Parallel Machines, In *Architectural Support for Programming Languages and Operating Systems* (ASPLOS-VII), October 1996.

8) http://www-csag.cs.uiuc.edu/projects/comm/fm.html.

9) http://now.cs.berkeley.edu/AM/active_messages.html.

10) S. L. Scott. Synchronization and Communication in the T3E Multiprocessor, In *Architectural Support for Programming Languages and Operating Systems* (ASPLOS-VII), October 1996.

11) M. Fillo, and R. B. Gillett. Architecture and Implementation of MEMORY CHANNEL 2, *Digital Technical Journal*, Vol. 9, No. 2, April 1997.

12) http://www.dolphinics.com.