システムソフトウェアと
オペレーティング・システム 65−10
（1994．7．21）

型情報に基づく遅延隠蔽を行うプロセス間通信機構

宮澤 元　猪原 茂和　上原 敬太郎　益田 隆司

東京大学 大学院 理学系研究科 情報科学専攻
〒113 東京都 文京区 本郷 7-3-1

要旨

分散協調アプリケーションを支援する64bitオペレーティングシステムLucasではメモリマップ技術に基づく仮想記憶管理機構を採用しており、プロセス間通信も分散共有メモリを介して行う。通常の分散共有メモリプロトコルでは、主にメッセージ数を減らして性能向上をはかっているため、ポインタを含む構造を持つデータをストリーム的に通信するとメッセージ受信者のレイテンシが大きくなってしまう。本稿では構造を持つ送信データの型情報を用いてプリフェッチを行いメッセージ受信者のレイテンシを下げるプロトコルについて述べる。コンパイル時の型情報を用いて送信データの型情報を自動生成する方法についても述べる。

Interprocess Communication Mechanism with Latency Hiding Using Type Information

Hajime Miyazawa, Shigekazu Inohara, Keitaro Uehara and Takashi Masuda

Department of Information Science, Graduate School of Science,
University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113

Abstract

The 64-bit operating system Lucas, which supports distributed cooperative applications, adopts the virtual memory management based on the memory-mapping technique. In Lucas, processes communicate via distributed shared memory (DSM). Because most DSM protocols improve their performance by decreasing the number of messages, they make latency of receivers large when they are used for stream-like communication of structured data which contain pointers. This paper describes a mechanism which enables interprocess communications with low latency by prefetching structured data using their type information. It also describes the automatic way of extracting type information of data by utilizing compile-time type information.

1 はじめに

近年、分散環境における協調作業を支援するためのアプリケーションが重要となってきている。我々が64bitアドレス空間を持つプロセッサをターゲットとして開発中のLucasオペレーティングシステム(OS)は、メモリマップ技術に基づく仮想記憶管理機構を採用しており、分散協調アプリケーションで必要なプロセス間でのデータの共有や通信を分散共有メモリを用いて単なるメモリ参照として行なうことができる。そのためポインタを含む構造を持ったデータをプロセス間で共有、通信することも容易である。

しかし、通常の分散共有メモリで用いられる一貫性制御プロトコルは主にキャッシュ間の一貫性を緩めてメッセージ数を減少させることによって性能向上をはかっているため、ストリーム型の通信のように送信者と受信者が固定され、かつ受信者は必ず最新のデータにアクセスしなければならないような通信に用いた場合、メッセージ受信者のレイテンシが大きくなってしまうという欠点がある。これを解決するために、Lucasプロセス間通信(Lucas IPC)では、既にメッセージ送信者の送ったデータを積極的に受信者側に転送するeager sending機構を提案した[7]が、この機構はバイトストリームのように、ポインタを含む構造を持たない通信データに対して用いられるものであった。

本稿では、Lucas IPCの拡張としてメッセージの送信者が通信データと共にその型情報を与えることで、受信者のアクセスパターンに応じたプリフェッチを行なうtyped prefetch機構について述べる。この機構を用いることで、受信者側のレイテンシを下げると共に、不必要なデータ転送を抑制することができる。また、コンパイル時の型情報を利用することでメッセージ送信者が与える通信データの型情報を自動的に生成することにより、アプリケーションプログラマの負担を従来と同等におさえることができる。

2 Lucasプロセス間通信システムの概要

我々が開発中の協調作業アプリケーション向け64bit OS Lucasでは、分散協調作業を分散共有メモリベースで行なうものと捉えており、そのサポートのためにregionという仮想記憶管理機構を採用している。regionとは仮想記憶空間上のある領域を占めるものであると共に二次記憶上の格納単位であり、Machでいうメモリオブジェクト、あるいはMulticsでのセグメントに近い概念である。各プロセスは、regionをアドレス空間にマップすることによってregionに対する処理をおこなう。例えば、複数のプロセスが同一のregionをアドレス空間にマップすることによって、分散共有メモリを構成することができる。

本研究におけるLucas IPCでは、regionが物理的に共有されているような同一ホスト上での通信では、単にメモリに対して読み書きすることで通信が行なわれるので、従来のプロセス間通信で必要だったユーザ空間からカーネル空間への通信データのコピーがなくなり、高速な通信が期待できる。

一方、通信を行なうプロセス同士が互いに異なったホスト上にあるような分散環境の場合、通常の分散共有メモリで採用されているようなプロトコルをそのまま適用するのは不適当な場合がある。例えば、ストリーム通信をinvalidation protocolを用いた分散共有メモリ上で実装することを考える。まず、送信者が分散共有メモリに書き込みを行なうと、受信者のもつキャッシュにinvalidation messageが発行される。ここで、受信者が分散共有メモリから読み出しを行なうと、送信者側のキャッシュに対する遠隔参照を行なってしまう。そして再び送信者が書き込みを行なうと、またinvalidation messageが発行されることになる。分散共有メモリの一貫性制御プロトコルの研究の流れは、キャッシュ間の一貫性を緩め、メッセージ数を減少させることで性能向上をはかるものが主流である[2]が、このようなプロトコルでは、例に挙げたストリームのように送信者と受信者が固定され、なおかつ受信者が送信者の送った最新のデータを受けとらなければならないような通信を行う場合、受信者側のレイテンシが大きくなってしまうという問題がある。以下では、この問題を解決するために既に我々が提案したeager sending機構について述べ、つづいてポインタを含む構造を持ったデータを通信する場合の拡張であるtyped prefetch機構について述べる。

2.1 Eager sending機構

Lucas IPCでは、分散共有メモリ上でストリーム通信を行なう場合の上述の問題点を解決するため、既にeager sending機構を提案している。これは、送信者側IPCサーバが、送られたデータを無条件に受信者側IPCサーバに送ることによって、受信者のレイテンシを下げようというものである。

送信者の通信データの送り出しと受信者の通信データの読み込みは、通信用にマップされているregionの特

```
kern_return_t
lucas_stream_send( mach_port_t  ServerPort,
                   rid_t        RegionID,
                   int          Address,
                   int          Size );
kern_return_t
lucas_stream_recv( mach_port_t  ServerPort,
                   rid_t        RegionID,
                   int          Address,
                   int          Sizes );
```

図 1: send と recv のインターフェース

定のメモリ領域に対して、send 命令、recv 命令を明示的に発行することによって行なわれる。これらの命令を通信同期とみなすことによって、IPC サーバはそのメモリ領域を送信データとして送受信する。実際のインターフェースを図 1に示す。

Eager sending 機構によりプロセス間通信する場合、以下の手順となる (図 2)。

送信側:

- 送信者は、自分のアドレス空間にメモリマップした通信用 region に通信データを書き込み、そのメモリ領域に対して send を発行する。
- 送信者側の IPC サーバは、送信者からの send を受けとると、send されたメモリ領域を受信者側の IPC サーバに無条件に送る。

受信側:

- 受信者は、自分のアドレス空間にメモリマップした通信用 region 上の受け取ろうとするメモリ領域に対して recv を発行する。
- recv を受けとった受信者側の IPC サーバは、既に recv されたメモリ領域を受けとっているかをチェックする。既に受けとっていれば、recv に対して "成功" を返す。まだ受けとっていなければ、recv をブロックして送信者側の IPC サーバから目的のメモリ領域が送られてくるのを待つ。

2.2 Typed prefetch 機構

分散共有メモリを用いたシステムでは、ポインタを含む構造をもったデータをそのまま通信できることが利点のひとつとして挙げられる。しかし、2.1節の eager sending 機構では、バイトストリームのように構造を持たないデータの通信を行なうのには適しているが、ポインタを含む構造を持ったデータを扱うのには不十分である。

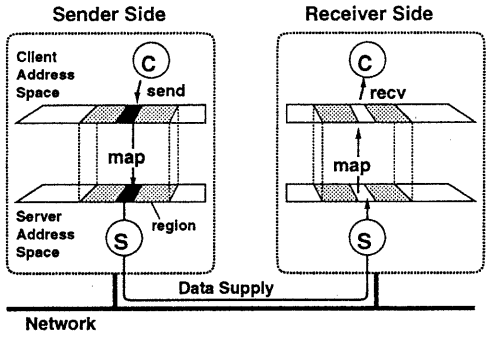

図 2: Eager sending 機構による通信

例えば、二分木のデータ構造を通信することを考える。送信者と受信者が二分木を同じ順番で (例えば両方とも幅優先順序で) 通信する場合は eager sending 機構でも問題はない。しかし、送信者は幅優先順序で送り、受信者は深さ優先順序で受けとるような場合、受信者が受けとろうとしているノードを送信者が送り出すまでにかなりの時間がかかってしまい、受信者のレイテンシが大きくなる。また、受信者がデータ構造全体を必要とするとも限らない。例で挙げた二分木が探索木として使われていた場合、実際に受信者が必要とするノードは、木の全体からみればごくわずかであろう。

このような構造をもったデータに対しても、できる限りレイテンシの小さい通信をサポートするために、Lucas IPC では typed prefetch 機構を導入する。この機構は、ポインタを含む構造を持ったデータの通信に適している。ポインタを含むデータ構造に対する受信者側のアクセスはポインタをたどったものになることが多いと予想される。そこで、送信データの構造情報を利用して近い将来受信者が要求すると思われる部分のみを送ることで、受信者のレイテンシを下げるとともに、不要なネットワークトラフィックを避けるのが本提案の基本原理である。

Typed prefetch 機構においても、eager sending 機構と同様に send と recv によるインターフェースが用いられる。送信者側、受信者側の IPC サーバ間では、受信者側 IPC サーバから送信者側 IPC サーバへメモリ領域を要求する request、受信者側 IPC サーバから送信者側 IPC サーバに受信者の参照位置を伝える notify、送信者側 IPC サーバから受信者側 IPC サーバへメモリ

領域を供給するsupply、送信者側IPCサーバから受信者側IPCサーバへメモリ領域を先送りするprefetchの4種類の通信が行なわれる。Typed prefetch機構を用いた通信例を図3に示す。

- 図3-(1)で、送信者は送り出すメモリ領域に対してsendを発行することにより、そのメモリ領域に含まれるデータ構造の型情報も送信者側IPCサーバに渡す。型情報については、2.3節で詳しく述べる。sendされたメモリ領域は、すぐには受信者側には送られず、sendマークが付けられる。受信者は参照したいメモリ領域に対してrecvを発行する。この領域は、まだ送信者側IPCサーバから受信者側IPCサーバに送られておらずアクセスできないので、受信者はブロックされ、受信者側IPCサーバは送信者側IPCサーバにrequestを出し、recvされたメモリ領域にはrecvマークを付ける。

- 図3-(2)で、送信者は、次のメモリ領域に対してsendを発行している。　送信者側IPCサーバでは、requestされたメモリ領域を受信者側IPCサーバに送る。同時に、図3-(1)のsend時に送信者から渡された型情報を利用して、このメモリ領域からポインタで参照されているメモリ領域についても、受信者側IPCサーバに先送りする。この先送りを、型情報を利用したプリフェッチと呼んでいる。送信者側IPCサーバは受信者側IPCサーバに送出したメモリ領域にsupplyマークをつける。受信者側IPCサーバでは、送られたメモリ領域に対して、accessibleマークをつける。また、このメモリ領域にはrecvマークがついていたので、受信者に"成功"を返す。これによって受信者は図3-(1)でrecvを発行したメモリ領域をアクセスすることができる。受信者は続いて次のメモリ領域にrecvを発行している。

- 図3-(3)では、受信者が図3-(2)でrecvを発行したメモリ領域はプリフェッチにより既に送られているので、受信者側IPCサーバは受信者にすぐ"成功"を返す。受信者はこの時点でそのメモリ領域にアクセス可能となり、自分の処理を続行できる。受信者側IPCサーバは"成功"を返した後、送信者側IPCサーバにnotifyメッセージを送ることにより、受信者がどのメモリ領域をrecvしたかという情報を送信者側IPCサーバに伝える。送信者側IPCサーバではその情報に基づいて次のプリフェッチの対象となるメモリ領域を決定して、そのメモリ領域を受信者側IPCサーバに送る。

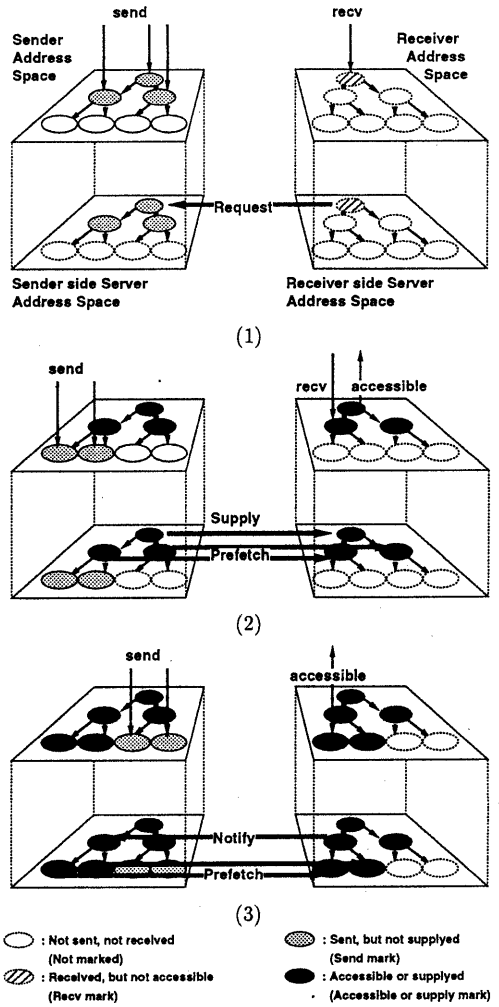

図 3: 型情報を利用したプリフェッチ

2.3　構造情報の自動生成

送信者があるメモリ領域の指定のみをした場合、IPCサーバがその領域に含まれるデータの型情報を知ることは一般には不可能である。従って送信者はsend時にメモリ領域に含まれるデータの型情報を渡す必要がある。ここで型情報とは、指定されたメモリ領域中に含まれるポインタの位置と、そのポインタによって示されるサイ

```
typedef struct type_info {
        int     ptr_addr;
        int     size;
} type_info_t;

typedef type_info_t     *type_info_array_t;
```

図 4: 型情報を示す構造体

```
    struct object {
        struct object   *next;
        int             val;
    } *obj;

    ...
    obj = (struct object *)REGION_ADDRESS;
    obj->next = obj + 1;
    obj->val = 0
    LUCAS_STREAM_SEND_OBJECT( ServerPort,
                              StreamID,
                              *obj );
```

図 5: 構造体単位の send 使用例

ズの組を要素とした配列で示される (図 4)。しかし、この情報をアプリケーションプログラマが用意して渡さなければならないとすると、送信対象のメモリ領域に含まれるデータ構造が複雑なものだった場合、プログラマにかかる負担は非常に大きい。

そこで我々は、コンパイル時の型情報を利用して、送信対象のメモリ領域に対する型情報を自動的に生成するというアプローチを取った。コンパイル時に決定され実行コードに埋め込まれた型情報を実行時に読み出すことにより、ソースコード中に含まれる型の定義を抜き出すことができる。この型定義を利用して図 4に示される型情報を内部で自動生成する send インターフェースを用意した。このインターフェースを用いたプログラム例を図 5に示す。図 5で ServerPort は、Mach のポートで、送信者側 IPC サーバを特定するのに使われている。StreamID は、通信用に使われている region の ID である。struct object へのポインタ obj に、通信用 region のマップされているアドレスを代入し、値をセットしてから送信を行なっている。LUCAS_STREAM_SEND_OBJECT が型情報を自動生成する send インターフェースである。これはは C 言語のプリプロセッサマクロを利用して定義されており、通信するオブジェクトにはユーザ定義の構造体を含む任意の型を取ることができる。マクロ内では、渡された引数の型を、あらかじめ読み出しておいた構造体定義と照合し、プリフェッチに必要な型情報を自動生成する。

3 Lucas IPC システムの実装及び実験結果

現在、Lucas オペレーティングシステムは DECstation 5000 上の Mach を利用してプロトタイプの開発を

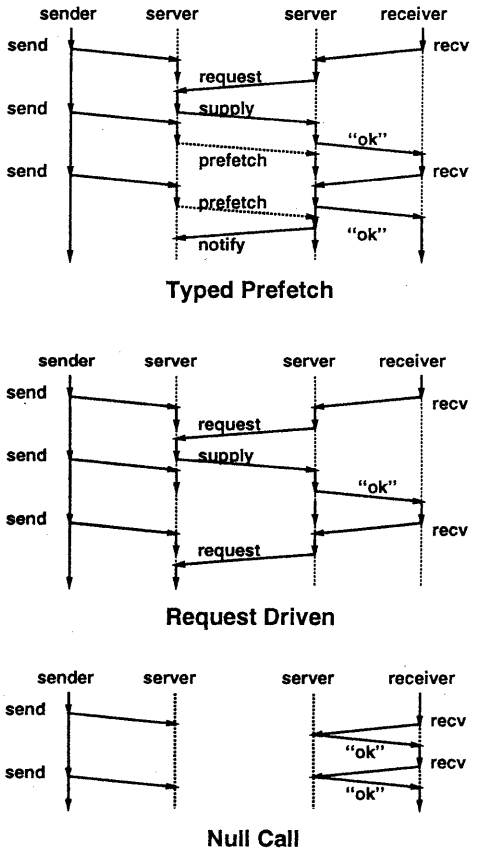

図 6: 各プロトコルの処理の流れ

行なっており、Lucas IPC システムは、Mach のユーザレベルサーバとして実装されている。また、型情報生成システムに用いるコンパイラは、GNU C compiler をベースにして作成した。Lucas IPC システム本体と型情報生成システム部分ともに C 言語で記述されており、8000 行程度のコード量である。

この Lucas IPC システムの性能を測定するために、いくつかの実験を行なった。実験では typed prefetch 機構と request driven 機構と null call の三つをを比較した。Request driven 機構とは、受信者が recv を発行するごとに受信者側 IPC サーバが送信者側サーバに要求を出し、送信者側サーバはその要求に従って必要なメモリ領域を転送する通信機構である。Null call とは recv の代わりに受信者側 IPC サーバに何もしない IPC を発行するものである。図 6に typed prefetch、

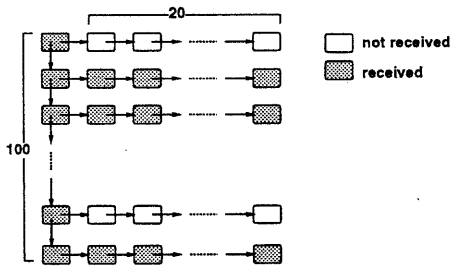

図 7: サンプルデータの構造

request driven、null call の各場合について時系列に従った処理の流れを示す。Typed prefetch、request driven では、IPC サーバで管理用データ構造の操作などに若干時間がかかっているが、null call では何もしないで "成功" を返しているだけである。また、typed prefetch 機構ではプリフェッチが成功した場合と失敗した場合で受信者側 IPC サーバの動作が異なる。

実験用のサンプルデータとして、線形リストのリスト構造 (図 7) を通信し、時間を測定した。各要素リストの長さは 20、要素リストの数は 100 である。受信者はこのデータ構造の全体にアクセスするわけではなく、いくつかの要素リストを選んで recv するものとし、全要素リスト中の recv する要素数の割合を変化させることとする。また、受信者は recv を実行する前に一定時間待つようになっている。これは実際の通信で、送信者から送られたデータに対して、何らかの処理を行なってから次のデータを受けとる状況を想定している。この待ち時間も可変である。

3.1 プリフェッチによる遅延隠蔽

受信者がデータを受けとる割合を 0.5 と一定にして、サンプルのデータ構造の通信に要した時間を測定し、一回の recv あたりの平均待ち時間を調べた。図 8は受信者の待ち時間を 0 秒から 0.5 秒まで変化させたときのグラフである。横軸に待ち時間、縦軸に recv に要した時間から null call に要した時間を引いたものである。

待ち時間 0 秒では、typed prefetch も request driven とほとんど差はない。これは、待ち時間のない状況では、プリフェッチされたメモリ領域が受信者側 IPC サーバに転送される前に、受信者がそのメモリ領域に対して recv を発行することによって、受信者側 IPC サーバが送信者側サーバに request を出してしまい、プリフェッチの効果がほとんど現れないためである。

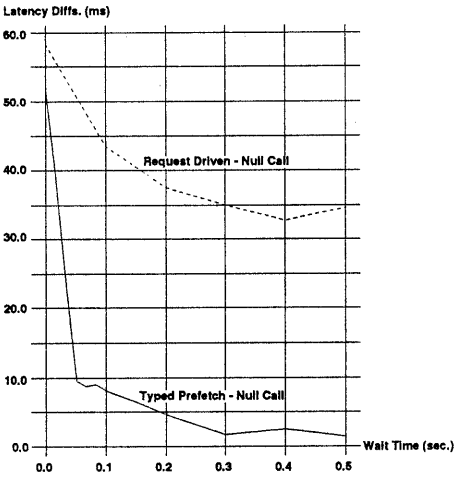

図 8: プリフェッチによる遅延隠蔽効果

受信者クライアントの待ち時間が大きくなり、受信者と受信者側 IPC サーバの並列性が生かされるようになるとプリフェッチは効果的である。これは受信者が既に受けとったメモリ領域に対してなんらかの処理を行なっている間に、IPC サーバはプリフェッチされたメモリ領域を受渡しできるからである。

3.2 読み出し率による遅延の変化

受信者の待ち時間を 0.3 秒に固定してサンプルのデータ構造の通信に要した時間を測定し、一回の recv あたりの平均待ち時間を調べた。図 9は受信者の読み出し率を 0.1 から 0.7 まで変化させたときのグラフである。横軸に読み出し率、縦軸は図 8と同様、recv に要した時間から null call に要した時間を引いたものである。

待ち時間を 0.3 秒と大きめに取っているため、typed prefetch では、遅延が隠蔽され、レイテンシは 0 に近くなるはずだが、読み出し率の高い部分で遅延が生じている。これは request driven についても同じことがいえるが、ページングコストのためだと考えられる。この実験の前後にページングの累積処理回数を調べてみると、request driven による通信で読み出し率 0.1 においては転送中に合計して 500 回程度のページングが発生していた。これが読み出し率 0.5 では、3000 回強となる。従って、ひとつのメモリ領域の転送については、読み出し率 0.5 の場合、読み出し率 0.1 の場合に比べ、平均してページング 1.1 回分のコストが余分にかかることになる。今回は、ページングコストがどの程度なのか測定

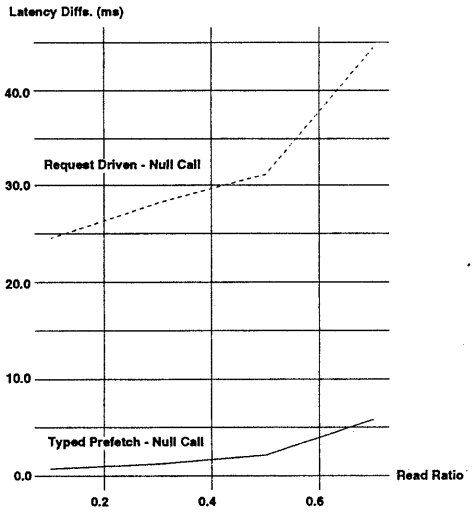

図 9: 読み出し率によるプリフェッチ効果の変化

することはできなかったが、write() システムコールが 10ms 弱かかることを考えると、ページフォルト 1.1 回分でもおよそ 5 ～ 10ms 程度のコストがかかるものと思われる。Request driven では、このコストが、全てグラフに現れる形となるが、typed prefetch では、通信遅延の隠蔽と同様にページングによる遅延もある程度隠蔽できるため、図 9のようなグラフになると解釈できる。

4 今後の拡張および最適化

今後の Lucas IPC システムの拡張として、双方向通信への拡張と受信者の参照情報の送信者への反映の 2 点について検討中である。また、メモリマップ技術を活用した最適化についても設計を進めている

4.1 双方向通信への拡張

現在の Lucas IPC のプロトコルでは、ストリーム通信と同様に、送信者と受信者を固定した通信を行なうため、通信によるデータの流れは一方通行に制限される。これは通信用 region の一貫性を send、recv を用いた同期によって保持しているためである。従って、送信者側は一度 send したメモリ領域に対しては変更を加えず、受信者はまだ recv していないメモリ領域に対して参照、変更を行なわないという制限を守らなければならず、双方向通信は実現できていない。

そこで、この制限を守ったままで疑似的に双方向の通信を行なうことのできるシステムを設計中である。現在のシステムでは受信者が recv したメモリ領域について、は、それに対して受信者側で何らかの処理を行なうというだけだったが、既に recv されたメモリ領域に対しては、recv したクライアントが send を発行すれば通信方向を逆転させることができる。この場合、一つのメモリ領域に注目すると、ある時点で見れば送信者と受信者が固定された単一方向の通信であるが、受信者が受けとったメモリ領域に変更を加えた後、送信者に送り返す、という単位でフェーズを切替えることによって疑似的な双方向通信が可能である。また、単一方向通信の場合と同様な問題で、ストリームを多段にすることも、現在のシステムではできないが、このフェーズ切替えによる双方向通信と同様な方法で実現できる。

4.2 受信者参照情報による送信者スケジューリング

Typed prefetch 機構における受信者側サーバから送信者側サーバへの request 及び notify は、受信者の参照位置の情報を送信者側サーバに伝えるためのものであるが、現在はその情報は送信者から供給される型情報を用いたプリフェッチに利用されるだけであり、送信者に渡されることはない。そのため、送信者自身は受信者がデータ構造のどの部分を参照しているのか知ることはできず、受信者が全く必要としていないデータだけを生成していってしまうようなことも起こり得る。

送信者のデータ生成順にある程度の自由度があり、受信者の要求があればそれに応じたデータを生成することができるような場合には、現在は送信者側サーバのレベルで止まっている受信者の参照位置情報を送信者が利用できるようなインターフェースを追加することにより、この点を改善することができる。さらに、この参照位置情報を送信者自身が供給した通信データの型情報と組合せ、upcall などの手段を用いて送信者に反映することができれば、アプリケーションプログラマの負担を軽減するという意味からも効果的である。また、upcall ではなく、送信者のスレッドのスケジューリングにこの機構を組み込むと、汎用並列計算機をターゲットとした場合のスレッドスケジューリングとメッセージ通信機構の融合手法として有効であると思われる。

4.3 メモリマップを利用した高速化

現在、通信に用いられる region は IPC サーバがクライアントにメモリマップしているので、同一ホスト上の

通信では送信者と受信者は物理的な共有メモリを持つことになる。しかし、Lucas IPCのプロトコルでは通信同期にsendとrecvを使わなければならないため、通信データを共有できても、同期としてのサーバへのIPCが発生してしまい、物理的な共有メモリを持つことによる通信の高速性をある程度犠牲にしてしまっている。

そこで現在IPCサーバが持っている管理用データを、クライアントにもメモリマップし、その管理用データにアクセスするためにIPCサーバの機能の一部をライブラリとしてクライアントにも持たせることにより、ローカルなIPCを単なる手続き呼び出しとすることで、同一ホスト上での通信においては、IPCをなくすことができる。しかし、分散環境においては、送信者側からのデータ転送や、受信者からのrequestやnotifyなど、ホスト間の通信を実際に行なっているのはIPCサーバであるので、やはりIPCサーバに何らかの形でローカルなIPCを発行しなければならないという問題は残る。最終的にはIPCサーバにはページャ及びネームサーバとしての役割を残すだけで、プロセス間通信はユーザクライアントのアドレス空間にメモリマップされたregionに対してアクセスを行なうライブラリレベルで実装する方向で検討中である。

5 関連研究

プロセス間通信機構をユーザレベルで実装することにより高速化しようというアイディアは、[1]で述べられている。[6]ではメモリマップされたファイルを用いたストリームI/Oについて述べられているが、プロセス間通信との関係については直接触れられてはいない。メモリマップを使いカーネル空間とユーザ空間の間でのコピーオーバヘッドをなくす方法については、[3]でコンティニュアスメディア向けのプロセス間通信機構として述べられている。[4, 5]では、ファイルシステムにおけるプリフェッチについて扱っている。特に[4]では、ファイル中にツリー構造を想定した場合、その構造に従ってプリフェッチを行なうことについて触れているが、これは、あらかじめファイルの構造をC++のクラスとして与えておく必要があり、プロセス間通信の送信時にサーバに型情報を動的に与えるLucas IPCのアプローチとは異なる。

6 まとめ

64bitのアドレス空間を仮定したオペレーティングシステム上での型情報を利用した効率的なプロセス間通信について述べた。サーバレベルで、通信データ内のデータ構造の型情報を利用したプリフェッチを行なうことで、通信遅延を隠蔽して、低レイテンシの通信が可能である。また、プリフェッチに用いられる型情報を、コンパイル時の型情報を利用して自動生成することで、アプリケーションプログラマの負担も軽減される。

参考文献

[1] Brian N. Bershad, Thomas E. Anderson, Edward D. Lazowska, and Henry M.Levy. User-level interprocess communication for shared memory multiprocessors. *Transactions on Computer Systems*, Vol. 9, No. 2, pp. 175–198, May 1991.

[2] K. Gharachorloo, A. Gupta, and J. Hennessy. Performance evaluation of memory consistency models for shared-memory multiprocessors. In *Proc. Fourth Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, pp. 245–257, New York, 1991. ACM.

[3] Ramesh Govindan and David P. Anderson. Scheduling and ipc mechanisms for continuous media. *Operating Systems Review*, Vol. 25, No. 5, pp. 68–80, 1991.

[4] Andrew S. Grimshaw and Edmond C. Loyot, Jr. ELFS: object-oriented extensible file systems. Technical Report TR-91-14, Univ. of Virginia Computer Science Department, July 1991.

[5] David Kotz and Carla Schlatter Ellis. Prefetching in file systems for MIMD multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 2, pp. 218–230, April 1990.

[6] Orran Krieger, Michael Stumm, and Ron Unrau. Exploiting the advantage of mapped files for stream I/O. In *Proceedings of the USENIX 1992 Winter Conference*, pp. 27–42. USENIX Association, January 1992.

[7] 宮澤元, 猪原茂和, 上原敬太郎, 益田隆司. Lucasオペレーティングシステムにおけるメモリマップ技術を用いたストリーム型プロセス間通信. 第47回(平成5年後期) 情報処理学会全国大会, 第4巻, pp. 23–24, Oct 1993.