高機能ネットワークを構築するギガビットチャネルの性能評価

國澤亮太[†] 松本尚[†] 平木敬[†]

ワークステーションクラスタ上に実現されたマルチユーザ / マルチジョブ環境においては、ユーザプロセス間に高速の通信や同期を用意することが必要である。我々は汎用超並列オペレーティングシステムと協調動作することを目的とした高速かつ高機能なギガビットスイッチングネットワークを開発中であり、その最も基本となるユーザプロセス間の通信を提供する手段としてメモリベース通信を使用する。

作成したネットワークインタフェイスカードの機能を検証するために既存のオペレーティングシステム上でメモリベース通信を実装したが、その評価をおこない、高速なメモリベース通信を実現する上でオペレーティングシステムに必要な機能を述べる。

Performance Evaluation of Gigabit Channel for High Functional Network System

Ryota Kunisawa,[†] Takashi Matsumoto[†] and Kei Hiraki[†]

On multi-user, multi-job parellel enviromnent build upon workstation clusters, fast user level communication and synchronization method is needed. We are developping a high speed, enhanced gigabit switching network system which cooperates our general purpose massively parallel operating system. Memory based communication is the basic communication method for user level communication.

We have implemented memory based communication on existing operating system for testing our network interface card, and evaluate the performance of it. We also describe the mechanism required by operating system for realizing fast memory based communication.

1. はじめに

ワークステーション単体の性能が著しく向上し、スループットの高い通信経路が利用可能になった今日、ワークステーションクラスタ環境における並列処理が現実となりつつある。並列アプリケーションの効率良い実行のためにはユーザ間の高速な通信同期が必要であり、我々が開発中の超並列オペレーティングシステム *SSS-CORE* では、既存のネットワークインターフェィスカードを使用しメモリベース通信 / 同期機構を可能な限り低オーバヘッドなソフトウェアで実装している[1)]。また、通信時のオーバーヘッドをさらに削減し、ネットワーク全体の通信量を削減するためのハードウェアを持つスイッチングネットワークを開発中である。

本稿では、作成したネットワークインターフェイスカードを使用して既存のオペレーティングシステム上でのメモリベース通信を実装し、その性能評価をおこなう。

† 東京大学大学院理学系研究科情報科学専攻
Department of Information Science, Faculty of Science, University of Tokyo

2. メモリベース通信

分散共有メモリ型並列計算機においてマルチユーザ / マルチジョブ環境を提供するためのハードウェアとして MBP(Memory-Based Processor)[2)] が考案された。ユーザプロセスは共有空間をページ単位で自分のアドレス空間にマップし、MBP にはその対応関係を知らされている。MBP はプロセッサのデータアクセスを監視し、共有メモリ空間へのアクセスでかつ自分がローカルに持っていない領域ならば、アクセス先のアドレスをネットワークアドレスに変換してリモートメモリアクセス通信を発行する。

しかし、プログラム中のどの変数を共有空間に割り当てるかはすでに分かっているので、共有空間へのアクセスならばそれを検知してリモートメモリアクセスのためのコードを生成することは可能である。その考えに基づき、MBP のようなメモリアクセスを監視するハードウェアでなく、リモートメモリアクセスの場合にはユーザプログラムで通信を起動する分散共有メモリシステム

が提案された[1]。そのためのメモリベース通信はMBPをソフトウェアで実装したものであり、ユーザ間の通信、同期を低コストで提供する。メモリベース通信の特徴は以下の通りであるが、MBPの以下の機能をソフトウェアで実装したものである。

- 通信相手先プロセスの仮想アドレスを指定したリモートメモリアクセス

ユーザのバッファを直接指定することでコピーの回数が削減できる。受信ルーチンはハードウェアで実装されるかカーネルの内部にあり、Active Message[6]等と異なりユーザの受信ルーチンを呼ばない。このため、受信ノードでのスケジューリングの公平さをある程度保証できる。

- アクセス保護と一貫性管理のプロトコルはページ単位で実装する

MBPの機能である仮想化されたリモートメモリアクセスを実現するには以下の機構が必要である。

(1) 相手先のノードIDから通信路を決定する
(2) 与えられたネットワークアドレスから物理ページを決定する
(3) アクセスに関するプロテクションを調べる

3. 関連研究

U-Net/MM[5]はネットワークインターフェイスカードにアドレス変換機能を持たせ、ユーザの通信バッファが物理メモリにない場合はカーネルにページインを要求する。ユーザのバッファが常に物理メモリに存在しなければならないという問題を解決する。メモリベース通信と異なり通信メッセージ中に相手先の仮想アドレスは入っておらず、受信側はユーザの空バッファの仮想アドレスのキューを持っており、受信時に自分が持っている仮想アドレスを物理アドレスに変換して受信メッセージを書き込む。仮想アドレスをメッセージに入れない点として保護と信頼性を挙げているが、メモリベース通信ではアドレス変換時に保護と信頼性を実現する。

4. SunOSにおけるメモリベース通信の実装

Sunワークステーションにおいて、物理アドレス空間は36ビットであり、これをSPARCプロセッサのMMUが32bitの仮想空間にマップして使用する。SBusも独自の仮想アドレス空間(最大31bit、DVMA空間と呼ぶ)を持ち、IOMMUを用いて物理アドレス空間からマップすることが可能である☆。SPARCのMMUとは異なり、IOMMUはコンテクスト番号を持たない。

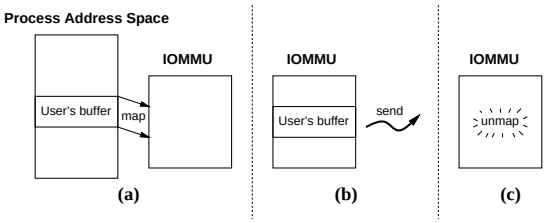

図1 ライトリクエストの送信

以下では、NICを使用して通信するSunOSのデバイスドライバを通信デバイスと呼ぶ。通信デバイスはNICを複数のプロセスで共有して使用するために排他制御を行なうもので、NICに対して複数の通信デバイスが存在し、SunOSでは`mknod`システムコールで作成する。

メモリベース通信は相手先のプロセス番号を必要とするが、今回の実装ではプロセス番号ではなくデバイスのマイナー番号を通信相手のIDとして使用する。メモリベース通信を始める以前に、通信するプロセス同士はお互いが使用している通信デバイスのIDを知っているものとする。

4.1 リモートメモリライトの実装

メモリベース通信は通信相手先のユーザプロセスの仮想アドレスを参照するので、リモートメモリアクセスの受信側ではアドレスの変換が必要である。

4.1.1 送　　信

SunOSにおいて、ユーザのバッファをカーネルがアクセスできるのは、`read`または`write`システムコールを発行した後に、バッファをカーネルのアドレス空間にマップした時点からである。リモートメモリライトを送信する場合は`write`システムコールを発行した時点でデータが用意されているので、SunOSの作法に従えば、`write`システムコール中でユーザバッファをマップし、送信が終ればアンマップすれば良い。

`write`システムコールでは送信側の仮想アドレスとデータの大きさしか指定できないので、`write`に先だって`ioctl`システムコールを用いて通信デバイスに相手先プロセスとその仮想アドレスを知らせる☆☆。実装の都合上(後述)、`write`で一度に送信できるデータがページ境界に収まっていない場合はエラーを返すようにしている。

なお、75MHzのSuperSPARC IIを搭載した

☆ DVMAアドレスは32ビット。最上位ビットは1なのでマスクする必要がある。IOMMUを用いないバイパスモードの時は、最上位ビットが1ならばアドレス変換されるが、0の場合は36ビットの物理アドレスのうち上位6ビットをSBusのスロット毎のコンフィグレーションレジスタから取得し、DVMAアドレスの下位30ビットをオフセットとするセグメントアクセスとなる。

☆☆ `ioctl`と`write`という2つのシステムコールを呼ぶことは二度手間だと思われるかもしれないが、システムコールを実行中のコンテクストは異なっているので、`ioctl`システムコールで送信バッファの仮想アドレスを渡しても送信バッファが割り当てられている物理メモリを引いてマップすることはできない。また、`write`と異なり`ioctl`システムコールの引数として渡すバッファの大きさは決まっているので、`ioctl`を`write`の代替とすることはできない。

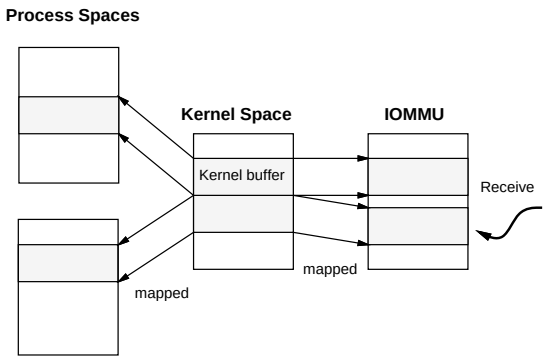

図2　ライトリクエストの受信

SS20(Mbus の周波数は 50MHz、SBus の周波数は 25MHz) において、SunOS 4.1.3 の`ioctl` システムコールと`write` システムコールのオーバーヘッドはそれぞれ 7μS と 17.5μS である。(以下、特に表記しない限り上記のマシンで性能測定を行なっている。)

4.1.2　受　　信

リモートメモリライトを受信した側では、通常の場合ユーザのバッファはマップされていない。ユーザの受信バッファをマップできるのは`read` システムコールであるが、このセマンティクスはメモリベース通信のリモートメモリライトとは相容れない。なぜなら、`read` はすでに存在しているデータを読み込むためのシステムコールであり、`read` を発行した時点でカーネル内のバッファにあるデータをユーザのバッファにコピーするからである。

それゆえ、SunOS での実装においては通信デバイス毎に受信バッファをあらかじめカーネル内に用意して、常に物理メモリに存在させている。ユーザプロセスは通信デバイスに対して`mmap` システムコールを用いて受信バッファをプロセスのアドレス空間に張り付けることができる。ユーザプロセスは、他人からリモートメモリライトでアクセスされる変数はその範囲のアドレスから割り付ける必要がある。

受信バッファが常に物理メモリに存在することは、資源の仮想化の観点からみると重大な欠点である。

5. 物　理　層

我々は高速ネットワークを構築する物理層として光シリアル通信に注目し、Fibre Channel を用いたスイッチングネットワークを構築中である。我々が使用している 1 ギガビットの Fibre Channel トランシーバは送信ポートと受信ポートを持ち、それぞれ 850 メガビット毎秒でデータを転送できる。ネットワークインターフェイスカードは SBus で実装し (図 3)、Sun の SPARC Station 20 上で使用している。

メモリベース通信は受信データを直接ユーザのバッファに書き込むものであるが、ユーザのバッファはメインメモリ上ではなく NIC 上に用意してもよい。その場合、リンクを通した送受信のスループットは向上するが、一般に I/O バスのアクセス速度はメインメモリのアクセス速度より遅いので、プロセッサからアクセスするコストが増加する可能性がある。また送受信バッファの仮想化の観点から NIC のメモリを OS のページング機構に加える必要があり、現実的な実装方法ではない。

上記の理由から NIC は SS20 のメインメモリを送受信バッファとして使用しており、実際のデータのスループットは SBus の制限を受ける。NIC がマスターとなり SBus にバーストトランザクションをかけた場合のスループットを計測したところ、SBus を最大限使用した場合リードは約 26 メガバイト毎秒、ライトは約 48 メガバイト毎秒であり、ライトの方が圧倒的に高速である。その理由としては、リードトランザクションは発行したリクエストが返ってくることを待たなくてはならないこと、また SparcStation20 の Mbus↔SBus ブリッジコントローラはライトバッファを持っているので、ライトトランザクションはターゲットデバイスからのアクノリッジを待つ必要がないためである。そのため、NIC がデータを必要とする場合は、NIC からバーストリードをおこなわず Mbus のブロック転送機能を用いて NIC に向かってバーストライトをおこなうようにした。NIC にバーストライトを行なった場合は最大約 50 メガバイト毎秒のスループットを観測した。

5.1　ハードウェア構成

NIC のハードウェアは、送受信回路、各 64 バイトの符号 / 復号化バッファ、64 バイトの受信ヘッダバッファ、SBus の DMA 回路から構成される。符号 / 復号化バッファはリンク上に流すデータをリアルタイムでエンコード / デコードするために必要である。符号 / 復号化バッファはデュアルポートメモリで実装しており、リンクへの送受信とワークステーションからのアクセスが同時に可能である。NIC の内部バスは 3 ステートバッファを介して SBus と直につながっており、符号 / 復号化バッファは内部バス以外にそれぞれ別ポートでモジュールの入出力とつながっている。受信ヘッダバッファは内部バスにつながっている。

光トランシーバ間のリンクは常に同期が取れている必要があり、データが転送されていない場合はコントロールコードを転送している cite8b10b。今回の実装では、コントロールコードの一部にデータ送信開始コードとアクノレッジコードを割り当てた。ヘッダは 64 バイト、データはすべて 32 バイトを単位として送信する。

データを送信する前にヘッダを作成する。ヘッダのサイズは 64 バイトであるが、現在は通信デバイスの ID と相手先のアドレスのみが入っている。ヘッダを作成後は通信開始コードを送信し、その後はデータをバーストで NIC の送信バッファに書き込めばデータ転送が起動される。NIC はヘッダの送信とデータの送信は特に区

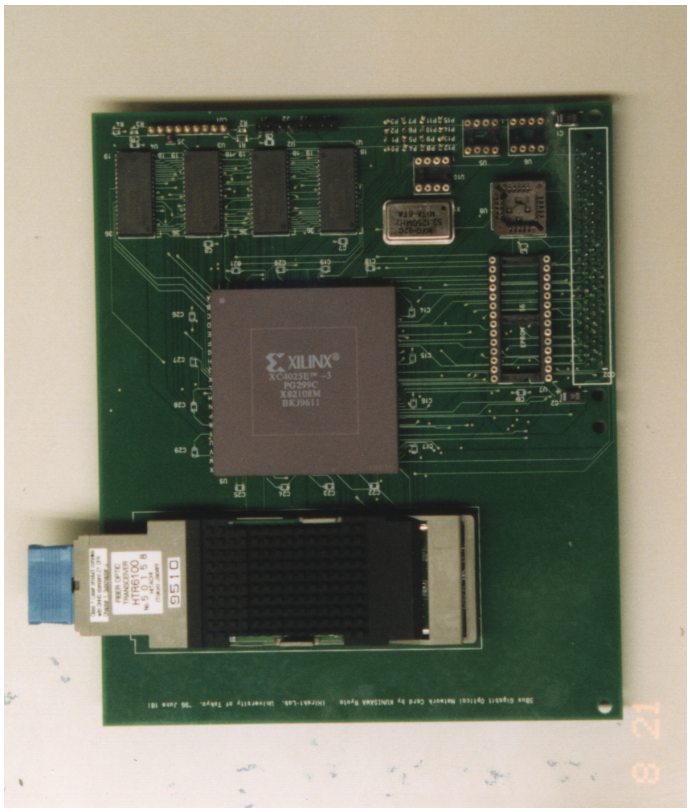

図3　ネットワークインターフェイスカード

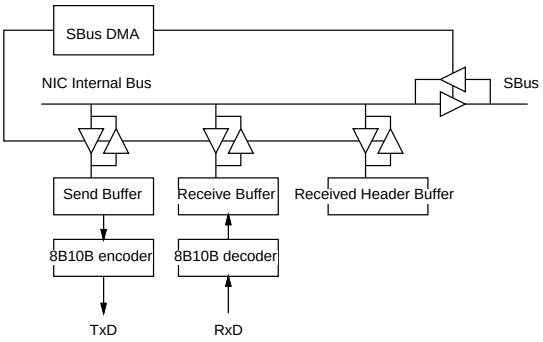

図4　ブロック図

別はしない。

受信側では、データ送信開始コードを受け取ったあとの 64 バイトはヘッダと解釈して NIC コントローラ内のヘッダバッファにコピーする。ヘッダ内の仮想アドレスのページ番号の部分はこの時点でアドレス変換をする (現在はまだ未実装)。それ以後はデータであり、ヘッダバッファ内のアドレスを参照し SBus を介して DMA でメインメモリに書き込む。DMA 転送の後はアドレスのオフセット部分のみをインクリメントして次の DMA アドレスを用意する。そのため一度に転送できるデータはページ境界に収まっている必要があるが、保護と仮想化の観点からも一度のメモリベース通信中に異なるページをアクセスすることは好ましくないので現実的な制約といえる。

基本的にヘッダやデータの 1 単位を受信する毎にアクノレッジを返送するが、復号化バッファが一杯になった場合は余裕ができるまでアクノレッジの返答を遅らせる。

6. 評　　価

現在はまだ NIC 内のアドレス変換部分を実装していないので通信相手先のバッファの IOMMU のアドレスを直接リモートメモリライトのパケットに格納しているが、NIC の TLB にヒットした場合のアドレス変換のコストは以下で示される SunOS でのオーバヘッドに比べると非常に低いので、性能を調べる観点からすれば NIC でアドレス変換が常に成功した場合のデータと同等と考えられる。

図 5は横軸にメモリベース通信で転送するデータのサイズ、縦軸にスループットをとっており (現在の実装では一度に送信するデータの単位は最大 4096 バイトなので、横軸は 4096 バイトまでである)、1024 回測定した結果の平均を表示している。送信、受信側とも Sun の SS20(50MHz の Mbus、25MHz の SBus) であるが、

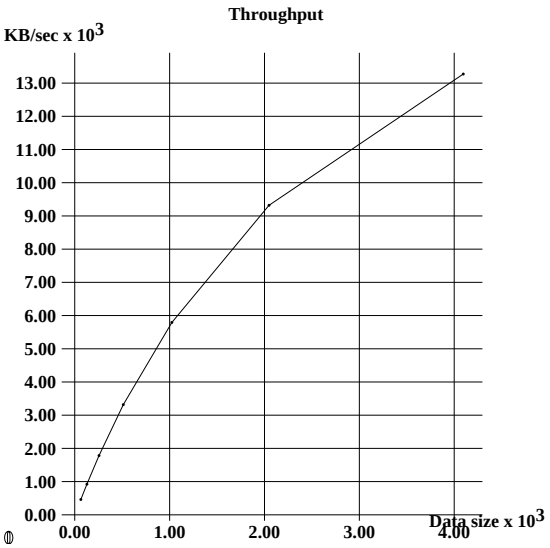

図5　データサイズとスループット

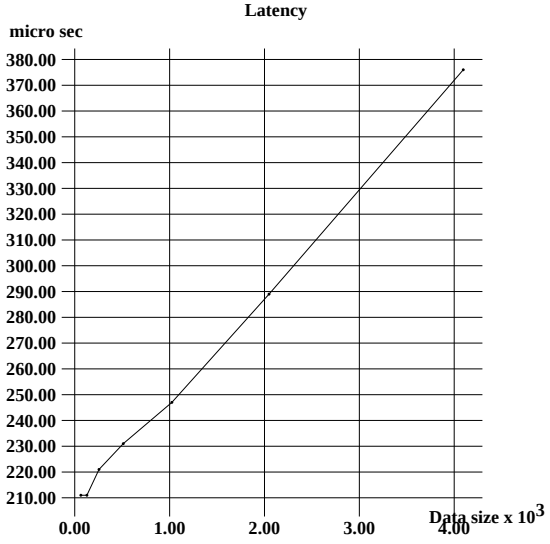

図6　データサイズとレイテンシ

送信側は 75MHz、受信側は 50MHz の CPU モジュールを搭載している。

送信時にシステムコールのオーバヘッドがあるので、送信データのサイズが小さいほどスループットは低くなる。図 6は送信時にシステムコールを発行する直前からデータの転送終了までの時間をロジックアナライザで計測した結果である。データサイズが 64、 128 バイトの場合は 10 回の平均を取っているが、測定結果のばらつきがほとんどなかったのでその他の場合は 3 回の平均を取っている。

モジュールの実際のスループットは図 6の傾きとして表現されている値であり、約 24 メガバイト毎秒である。スループットは SBus で決定し、約 50 メガバイト毎秒まで出せるはずであるが、現在の実装では NIC の符号化 / 復号化のバッファを 32 バイト (SS20 の SBus で 1 回のバーストで転送可能な最大サイズ) しか使用しておらず、受信されたデータの処理が終了するまで送信を止めている。そのため SBus を最大限使用できていない。 NIC コントローラ内のバッファは 64 バイト用意しているので、ダフルバッファリングをおこなってスループットを向上させる予定である。

7. 考　　察

ユーザレベルの高速なメモリベース通信を実装するためには、 SunOS の次の 2 点を改善する必要がある。

- 受信バッファがカーネルに存在する必要があり、資源が仮想化されているとはいえない。
  SunOS のデバイスはユーザのリクエストを受け付けるルーチンとハードウェアからのリクエストを受け付けるルーチンは分かれており、インタラプトルーチンは後者に属する。ユーザプロセスに関する構造体は後者からはアクセスできないようになっており、インタラプトでユーザのバッファを用意すること不可能である。
  そのため、リモートメモリアクセス要求を受信した時点でバッファがなかった場合には用意することができない。それゆえ、受信バッファは常に存在している状態でなければならない。
- オーバヘッドが高い。
  前述の通り、`ioctl` と`write` のオーバヘッドを加算しても高々 25μS である。観測によればオーバヘッドはコンスタントに約 210μS であり、システムコール以上に SunOS のデバイスドライバの実現方法が高いオーバヘッドを生じている。 SunOS では`write` の中で直接データを転送するわけではなく、`physio()` カーネル内ルーチンによってデータ転送要求を登録することになっている☆。転送を行なうルーチン中でユーザのバッファが物理メモリにページインされていることを保証するのが`physio()` である。`physio()` から呼ばれたルーチンユーザのバッファが物理メモリにページインされていることが保証されている。逆にいえば、(`physio()` から呼ばれない限りは)`write` システムコール中であってもユーザのバッファが物理メモリに存在することは保証されない構造となっている。

8. 終　り　に

ネットワークが十分に速くなった現在、ユーザレベルでの高速な通信を実現するには OS のオーバヘッドもデータ転送にかかる時間に比べて無視できるほど小さいことが必要であり、またユーザのバッファをデバイスド

☆ physio() ルーチンの引数として一度に転送するデータサイズを決定するルーチンと実際に転送をおこなうルーチンを指定する。

ライバからアクセスできるようにする機能が必要である。

我々が現在開発中の汎用超並列オペレーティングシステム *SSS–CORE* ではこの両方の機能を持っていて軽いメモリベース通信を実現している。送信時のオーバヘッドは 100baseTX を使用した場合で 5μS 以下であり[4)]、高速なネットワークインターフェィスカードの性能を十分に引き出すことが可能である。今後は *SSS–CORE* でのメモリベース通信の実装をおこなう。

謝　　辞

本研究は情報処理振興事業会 (IPA) が実施している独創的情報技術育成事業の一環として行なわれた。

また、ハードウェアの製作にあたり、メンターグラフィックス社とシノプシス社の University Program を用いた。両社に深く感謝します。

参　考　文　献

1) 松本 尚, 平木 敬, “汎用超並列オペレーティングシステム SSS–CORE のメモリベース通信機能”, 情報処理学会 第 53 回全国大会 (September 1996)

2) 松本 尚, 平木 敬, “超並列計算機上の共有メモリアーキテクチャ”, 信技報, CPSY92-96, pp.47–55 (August 1992)

3) 松本 尚: “マルチプロセッサ上の同期機構とプロセッサスケジューリングに関する考察”, 計算機アーキテクチャ研究会報告 No.79-1, 情報処理学会, pp.1–8 (November 1989)

4) 松本 尚, 平木 敬: “100Base-T によるメモリベース通信の性能評価”, SWoPP 阿蘇'97 コンピュータシステム研究会 (CPSY-4) で発表予定 (August 1997)

5) Anindya Basu, Matt Welsh, Thorsten von Eicken: “Incorporating Memory Management into User-Level Network Interfaces”, Department of Computer Science, Cornell University, Technical Report TR97-1620, February 13th, 1997,
`http://128.84.154.16/U-Net/papers/unetmm.pdf`

6) Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser: “Active Message: a Mechanism for Integrated Communication and Computation”, In *Proceedings of the 19th International Symposium on Computer Architecture*, ACM, pp.256–266 (May 1992),

7) Susan A. Mason: “SBus Handbook”, *Sun Microsystems, Inc.* (1994)

8) A. X. Widmer, P.A. Franaszek: “A DC Balanced, Partitioned Block, 8B10B Transmission Code”, *IBM Journal of Research and Development*, Vol.27, No.5 (September 1983)