システムソフトウェアと
オペレーティング・システム 90−16
（２００２．６．２８）

並列分散ライブラリ Suci の実装と評価

屋比久 友秀 琉球大学理工学研究科総合知能工学専攻
河野 真治 琉球大学, 科学技術振興事業団さきがけ研究 21

概要

並列分散環境において MPI や PVM 等の TCP を使った通信ライブラリが実装され、利用されている。TCP が抱えるフロー制御の問題を解決する為に、我々は UDP ベースの通信ライブラリ Suci を実装し、基本性能を TCP ベースの MPI と比較した。

Impelmentation and Evaluation of Parallel Distributed Communication Library “Suci”

Tomohide Yabiku Faculty of Information Engineering, University of the Ryukyus.
Shinji Kono Information Engineering, University of the Ryukyus, Japan Science and Technology Corporation

Abstract

In parallel distributed environment, many communication libraies are implemented and used, like MPI and PVM. These communication libraries are based on TCP. TCP has some flow-control problems, so we implemented the UDP-based communication library “Suci” to solve those problems. We evaluated the communication performance of Suci to compare with TCP-based MPI.

1 はじめに

近年、PC クラスタなどの普及により、50 台以上の分散計算環境も比較的容易に利用する事ができるようなってきている。また、社会のすみずみに浸透しているコンピュータパワーとネットワークを利用し、きめ細かなサービスを提共する超分散的なコンピューティングも現実のものとなっている。[1]。

PC クラスタのような比較的密結合な分散環境では、MPI や PVM 等の並列通信ライブラリが利用されている。これらの多くは TCP ベースの通信を採用している。TCP を使って、多数のプロセスと通信を行う場合は、ひとつのプロセスで多数のソケットを開くことになる。このため、多数のノードが存在する分散計算環境では、カーネルの資源を消費し、結果的に計算効率の低下を招く。我々は、ひとつのソケットで複数のプロセスに接続できる UDP ベースの通信ライブラリ Suci(Simple User level UDP Communication Interface)[3] を実装した。このライブラリの特徴として以下の点が挙げられる。

1. ユーザーレベルでフロー制御、輻輳制御が可能。
2. カーネルの資源消費を最小限におさえる。

3. Myrinet[2] や Gigabit Ethernet などの特種なデバイスに依存しない。

このライブラリの設計、実装と TCP ベースの通信ライブラリと比較し、性能評価を行った。

通信レイヤに UDP を用いた例としては、新情報通信処理開発機構が開発した SCore 上に実装されている PM/UDP[4] や Ohio Supercomputer Center の開発した LAM(Local Area Multicomputer)[5] がある。PM/UDP は、独自の PmUDP と呼ばれる通信ライブラリと pmudpd と呼ばれるデーモンプロセスを使って信頼性のあるメッセージ配送を実現している。また、LAM は UDP 上の独自のフロー制御プロトコルを実装することで信頼性を確保している。それぞれ通信レイヤの上には、MPI や PVM などの通信ライブラリを実装している。本稿では、第 2 節で Suci が必要になった背景を述べ、第 3 節で Suci の設計、第 4 節で Suci の実装について述べる。次に第 5 節で Suci の性能評価について述べ、第 6 節でまとめと今後の課題について述べる。

2 ユーザレベルフロー制御の必要性

TCP のフロー制御は、システムレベルに隠蔽されていてユーザレベルから操作する事ができない。これは、TCP が再送のバッファをカーネル内に持ち、カーネルのタイマーを用いて再送処理を行う為である。システムレベルの自動的なフロー制御には、ネットワークの性質を熟知しなければ、行えないフロー制御をアプリケーションから隠すことで、技術者の負担を軽減するといったメリットもあるが、以下のような問題点も抱えている。

1. フロー制御がアプリケーションプログラムから完全に分離されている為、制御できない。
2. 多数の TCP ソケットを開くときや、遅延の大きいネットワークにおける TCP ソケットでは、ユーザが操作でない巨大な送受信バッファがカーネル内部に生じメモリ空間を圧迫する。

1 の問題は、TCP では各ノード間で帯域を一様に分けようとするアルゴリズムが採用されている為である。帯域を公平に分け合う通信品質を必要とするアプリケーションには効果的であるが、通信に優先度が存在するアプリケーションでは、フロー制御が隠蔽されているため、ユーザレベルでフロー制御を行う事ができない。

2 の問題は、大規模な並列分散環境における完全結合型の通信が必要となる場合や通信衛星などの非常に遅延の大きいネットワークを用いた場合に起きる。通常の TCP の最大ウィンドウサイズは TCP ヘッダのウィンドウサイズフィールドが 16bit であるため、2^16=64KB である。1000 台の完全結合型通信を TCP で実現しようとした場合、$64K * 1000 * 2$ ≒128MB もの送受信バッファがカーネル内部に生じる。このような環境では、結局 TCP を使っても、ネットワークやアプリケーションの利用形態を熟知した技術者がウィンドウサイズや MTU 等のパラメータを適切に調節する必要がある。

1、2 の問題が起きるようなアプリケーションには以下のようなものがある。

1. 通信毎に優先度があるような並列分散アプリケーション
2. 多くのノード間でランダムな通信が行われる、大規模な並列分散アプリケーション
3. スループットと実時間性が必要なデータ通信を伴うアプリケーション

我々は、上記のようなアプリケーションで効率の良い通信を行うには、ユーザレベルのフロー制御が必須であると考えた。ユーザレベルにフロー制御を持つことで、ユーザプログラムが操作可能なメモリ空間に送受信バッファを持つため、カーネルによるメモリの圧迫がなくなり、より効率良くメモリが使用でき、結果的にスループットを高める可能性がある。

3 Suci の設計

3.1 フロー制御

フロー制御は、受信側の処理能力やネットワークの状態にあわせて送信を行うことである。TCP では、ソケットバッファの空きサイズを Acknowledge に乗せて送信側に通知し、受信側の処理能力を超える送信をしないようにしている。また、応答確認

の遅延から自動的にウィンドウサイズを調節する。フロー制御をユーザーレベルで全て行うには、応答確認の処理などの細かい処理を全てユーザー空間で行う必要がある。また分散計算環境においてアプリケーションがフロー制御を行うには、任意のノードのアプリケーションが他のノードのフロー情報 (遅延やウィンドウサイズ) を得る必要がある。これを実現する為にはユーザー空間に他のノードのフロー情報を持つ必要がある。Suci はこれらを実現する為に、ユーザー空間に再送用のキューを持ち、アプリケーションからアクセス可能なユーザー空間に相手先毎のフロー情報を持つことにした。以下のように Suci は Ack/Nack とメッセージのシーケンス番号を用いて UDP パケットの欠落や順序の入れ換わりが起きてもメッセージの配送を保証している。

1. 送信ノード

   S1 送信ノードは各メッセージに受信ノード毎に連続したシーケンス番号を付けて送信する。送信したメッセージは再送のために対応する Ack を受信するまでキューに保存する。

   S2 Ack を受信したら対応するキューの領域を解放する。

   S3 Nack を受信したら対応するキューのメッセージを再送する。

2. 受信ノード

   R1 シーケンス番号が連続したメッセージはキューに格納して Ack を返す。

   R2 キューがオーバーフローした場合は受信したメッセージを破棄し Nack を返す。

   R3 古いシーケンス番号のメッセージを受信した場合は Ack を返す。

   R4 シーケンス番号の抜けがあった場合は、そのメッセージを破棄して、正しいシーケンス番号のメッセージに対する Nack を返す。

   R4 最初に Nack を返したメッセージが正常に受信されるまで、受信したメッセージは破棄する。

3.2 輻輳制御

パケットの配送が遅延させられたり、パケットが破棄される状況を輻輳と呼び、輻輳の発生を回避する技術や、ネットワークの状態を輻輳状態から速やかに回復させる技術のことを輻輳制御と呼ぶ。TCP の場合、輻輳制御を実現する為に、輻輳ウィンドウの値を増減させることで、転送速度を調節する。このような輻輳制御は 1 対 1 通信の場合はうまく動くが、第 2 節で述べたように、多対多通信の場合は一様なスループットの低下の原因となり、分散計算環境には適さない場合がある。そこで我々は、図 1、図 2 に示す、多対多の場合の輻輳制御を Suci に採り入れた。以下にその流れを説明する。

1. 輻輳が生じた場合、Ack が取れなかったパケットを再送キューに格納する。

2. 送信側は再送キューの数により輻輳状態を検出する。

3. 送信側は受信側に再送が必要なキューのサイズと一緒に再送許可の要求を送る。

4. 受信側は送信された再送キューのサイズと最大スループットから再送を許可するかどうかを判断し送信側に再送の可否を送る。

5. 送信側は許されたメッセージサイズを送信する。

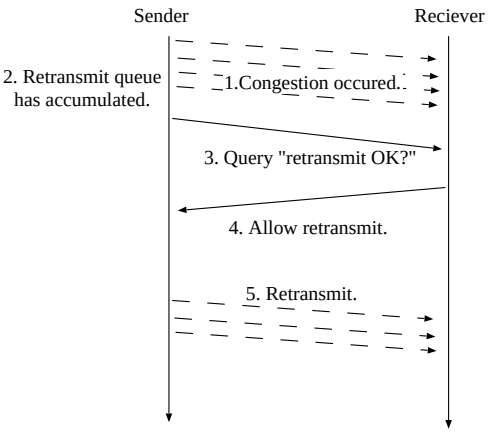

図 1: 輻輳制御

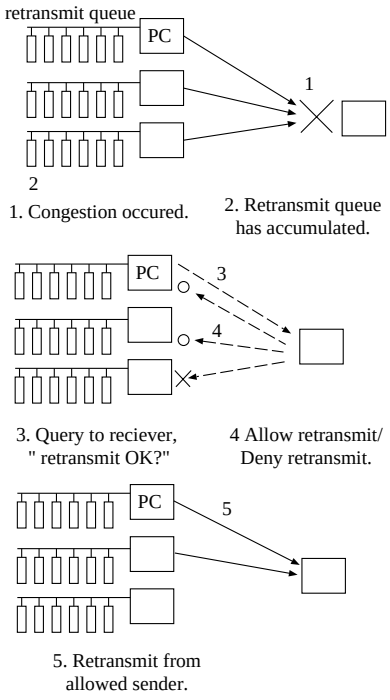

図 2: 輻輳制御の例

4 実装

4.1 Suci の信頼性

Suci は、UDP に信頼性を付加する機構をユーザー空間に持ち、到着順序を保証するためにパケットに送信先毎のシーケンス番号を付加する。このシーケンス番号フィールドはそのパケットに固有のシーケンス番号が入る。また、Suci は、ブロッキング/デブロッキングを行っており、MTU(Maximum Transmission Unit) より大きなブロックが送信された場合、MTU にあわせて送信ブロックを分割しパケットとして送信する図 3。受信側で送信ブロックを組み立てるとき、UDP パケットがどの送信ブロックの一部であるかを区別する必要がある。ブロックの先頭フィールドは、このパケットがどの送信ブロックの一部であるかを区別するためのフィールドである。送信ブロックが分割されたとき、先頭パケットのシーケンス番号をこの送信ブロックの識別番号として扱う。送信ブロックを構成する全てのパケットのブロックの先頭フィールドには、パケットが所属する送信ブロックの先頭パケットのシーケンス番号が入る。図 4 に Suci の概要を示す。まず、通信先毎に割り振られた IP アドレスと ID を Address Database に格納する、ブロックを送信する場合は、`datagram()` でソケットをオープンして`datagram_destination()` で送信先を指定する。次に`datagram_write()` を呼び出し送信ブロックをアウトプットキューに書き込み、送信先に送信する。送信後の手順は第 3 節で詳しく述べた。受信する場合は、インプットキューに送信されたブロックが格納される。ユーザーが送信されたブロックを読み込むには、`datagram_read()` を呼んで送信されたブロックを読み出す。実際には、ブロッキングが完成したら、コンプリートブロックとして、インプットキューから退避される。

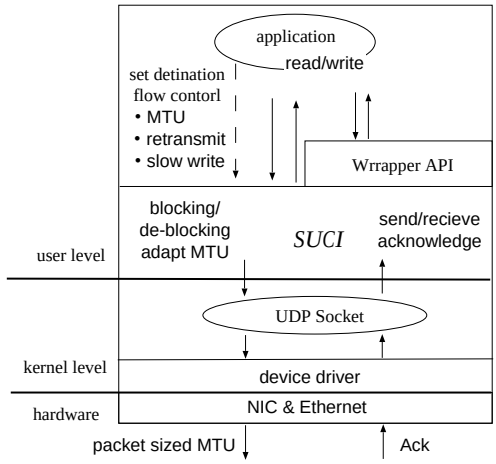

図 3: Suci のアーキテクチャ

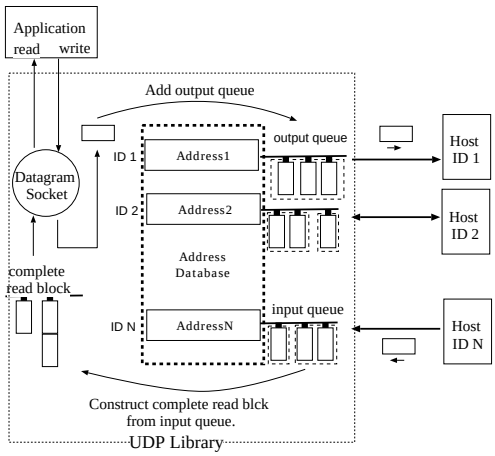

図 4: Suci の概要

4.2 ウィンドウサイズの通知

Suci はユーザーレベルでフロー制御を行う為に TCP と同様のウィンドウサイズの通知機構を持っている。ウィンドウサイズの割り当ては、UDP ソケットバッファの値をデフォルトで使用するが、ユーザプログラムが受信データの処理能力にあわせて変更することもできる。Ack 毎に、現在のウィンドウサイズからインプットキューの合計サイズを引いた受信可能なサイズを送信側に通知する。Suci パケットのセグメントヘッダを図 5 に示す。Suci は、データグラムに到着順序の保証と信頼性の付加するため、データグラムパケットにシーケンス番号を付加し、ブロッキング/デブロッキングのためにデータグラムパケットがどのブロックのパケットであるかを、そのブロックの先頭パケットのシーケンス番号によって区別する。ブロックの先頭フィールドはそのパケットが属するブロックの先頭パケットのシーケンス番号が入る。

Destination ID と From ID は、アドレスデータ構造に登録されているアドレスとセットになった ID が入る。送信先／送信元は ID で指定される。TCP パケットと違い、Acknowledge 番号のフィールドを独立にはもたない。Flag に含まれる Ack が有効な場合、このパケットは Acknowledge として扱われデータは無視される。このとき、シーケンス番号が Acknowledge 番号となる。

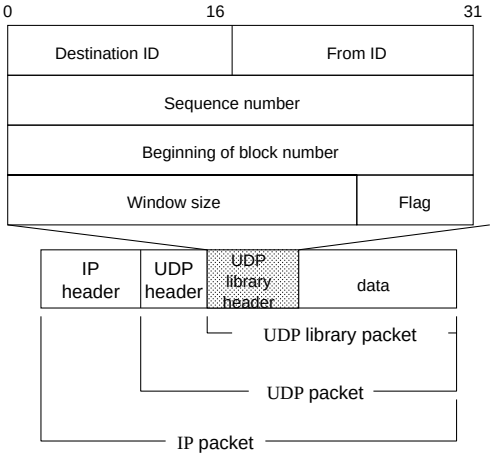

図 5: Suci のセグメントヘッダ

4.3 フロー制御用 API

フロー情報の交換や blocking write における wait はライブラリが自動で行う。プログラムは non-blocking read/write を行うタイミングや wait を指定することができる。また、アプリケーションはフロー情報をライブラリ中のアドレスデータ構造から得ることができる。以下に Suci のフロー制御用 API の概要を示す。

1. `int datagram_ready(sock,sec,msec)`
   `datagram_socket *sock;`
   `int sec,msec;`

   read 可能な block が組み立てられているかを check する。組み立てれられていれば、即座にその block のサイズを返す。組み立てられていなければ、sec+usec の時間 select し、パケットを組み立てようとする。パケットの組み立てに失敗すれば 0 を返す。select 中に予期しないエラーが帰れば-1 を返す。エラーの処理はアプリケーションに任せる。non-blocking read の前に使用し、read の wait 時間を他の処理に使用するために使用する。

2. `int datagram_wait(sock,sec,usec,size,id)`
   `datagram_socket *sock;`
   `int sec, msec,size,id;`

   id 宛に size 分だけ non-blocking write 可能になるまで最大 sec+usec 秒待つ。送信 queue がたまっている場合や window size が空いていない場合、これ以上の non-blocking write は輻輳や受信側のバッファオーバフローの原因になる。アプリケーションプログラムは datagram_wait() を使い任意の地点でこの wait を行える。

3. `int datagram_queue_length(sock,id)`
   `datagram_socket *sock;`
   `int id;`

   ID 宛の Ack の取れていない再送キューの長さをチェックする。返し値は、再送 queue にたまっている write block の数。ネットワーク上に送出する 1 パケットの長さは Library によって MTU にあわせられるが non-blocking write をおこなった場合は、それより小さなパケットが送出される場合もある。その場合には queue にたまっている write ブロックの合計サイズがいくつかを推定するのは難しいが、パケットの処理は、1 パケットのサイズよりも

パケット数のほうがはるかにコストにかかる度合が大きいため、結局は write ブロックの数が問題となる。

以上のようなフロー制御 API を用いた並列分散プログラムの典型は、次に示すようになる。

```
while(1){
    {
      /* 処理 */
    }
    if(datagram_ready(sock,SEC,USEC)){
      /* read ブロックが完成していたら */
      datagram_read0(sock,buf,size);
      {
        /* read したデータの処理 */
      }
    } else {
    /* read ブロックなかったときの処理 */
    /* 送信先の指定 */
    datagram_destination(sock,id1);
    if(datagram_wait(sock,SEC,USEC,SIZE,id1)){
       /* SIZE の分だけ送信 */
        datagram_write0(sock,buf,size);
       {
         /*  write 後の処理 */
       }
    } else {
      /* write できなかった場合の処理 */
    }

    if(datagram_queue_length(sock,id2)){
      /* 再送 queue を check */
      datagram_retransmission(sock,id2);
      /*必要なら再送を行う*/
    }
}
```

上記以外の Suci の主な API を表 1 に示す。

5 評価

Suci の評価には、当研究室のクラスタシステムを用いた。表 2 にその評価環境を示す。比較対象として TCP ベースの通信ライブラリ MPICH+SCORE と MPICH を用いて実験を行った。

5.1 基本性能

最初に、Suci の基本的な性能を評価する為に行った 2 ノード間のピンポン転送によるスループットの測定結果を図 6 に示す。ピンポン転送とは、送信ノードから受信ノードにメッセージを送信し、そのメッセージを受信ノードが受信後、同じサイズのメッセージを送信ノードに送信し、そのラウンドトリップ時間を計測する転送方法である。

表 2: 評価環境

ノード数	44
CPU	Pentium3 800MHz
メモリ	512Mbyte/node
ディスク	10GB/node
SCore	4.2.1
マザーボードベースクロック	133Mhz
NIC	EtherExpressPro 100
スイッチ	Catalyst C2980-GA
OS	Linux 2.4.10

この実検ではメッセージサイズを 4 バイトから 500K バイトまで二乗づつ変えながら複数回のピンポン転送を行い、経過時間からスループットを求めた。Suci は 4 バイトから 30K バイトまでは他の通信ライブラリと比較して良い結果が得られた。特に、1024 バイト以下では、MPICH と比較して約 2 倍程度、MPICH+SCORE と比較しても 1.5 倍程度のスループットが得られた。しかしながら、メッセージサイズが大きくなるにつれて 65536 バイトをピークに序々にスループットが落ちてきている。これは、Suci 内部の MTU よりも大きなメッセージを送信しようとすると Suci の中でデブロッキングが発生し、受信側では、ブロッキング処理が発生する。メッセージサイズが特に大きな場合は、このブロッキング/デブロッキングのオーバーヘッドが大きいという事と、送信ブロックの数が増える事で再送処理が数回発生していたことによると考えられる。

次に、バースト転送によるデータ転送スループットの測定結果を図 7 に示す。ピンポン転送のスループット測定と同様にメッセージサイズを 4 バイトから 500K バイトまで二乗づつ変化させながら実検を行い、複数回の一方向バースト転送を行い。MPICH は約 7.5MByte/s 付近でスループットが上がらなくなっているが、Suci と MPICH+SCORE は約 10MByte/s のスループットを計測した。MPICH+SCORE で 16K バイト時に著しくスループットが落ちているがこれは、TCP を使った通信では良く見られる現象である。Suci ではピンポン転送でスループットの低下が見られた同じメッセージサイズ付近でスループットが減

表 1: Suci の主な API

Stream Open	`datagram(addrdb,myaddr,myport,myid)`
Choice of transmitting point	`datagram_destination(distid,sock)`
Transmission	`datagram_write(sock,buf,len)`
Reception	`datagram_read(sock,buf,len)`
Check of the packet	`datagram_ready(sock,sec,msec)`
Check of the transmission queue	`datagram_queue_length(sock,id)`

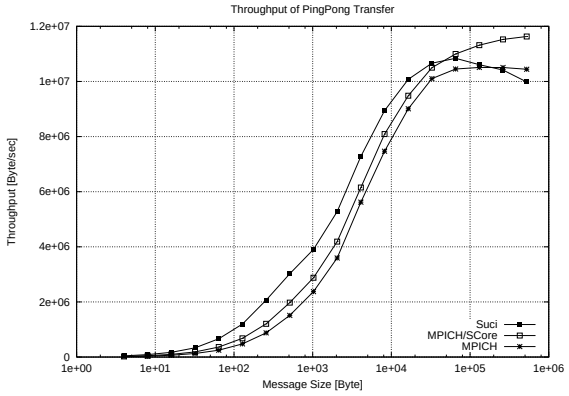

図 6: ピンポン転送のスループット

少しているのが見られる。これは、ピンポン転送の時と同じ原因と考えられる。

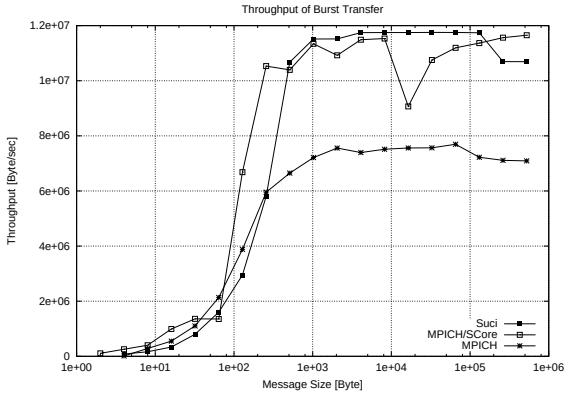

図 7: バースト転送のスループット

5.2 collective 通信性能

これまでは、2 ノード間で実験を行ったが、以下では、1 対 N の場合の Susi の基本性能を MPICH を対象に比較する。

まず、複数のノードから 1 台に対してメッセージを送信するベンチマークを行った。測定は、各ノードから受信ノードに対してメッセージサイズを 4 バイトから 65536 バイトまで変化させ、各ノードから全てのメッセージを受信するのに要する時間からノード 1 台あたりのスループットを求めた。図 8 にその測定結果を示す。Suci は、MPICH と比較して立ち上がりが遅いが、これは、ユーザーアプリケーション側で再送を制御する為に Slow Write を行っているからである。Slow Write を行わないと、1 つのノードにパケットが集中する為、UDP のバッファからオーバーフローする現象が見られたからである。同じ理由でメッセージの変化させる範囲を上記の Broadcast 実験の時よりも少なくした。あまり、送信メッセージのサイズが大きいと UDP のバッファからあふれてパケットロスが起きてしまう。その為、 Suci はメッセージの再送処理を行う。これは、非常にコストの高い処理である。これはアプリケーションの Slow Write のアルゴリズムによって防ぐことができる。

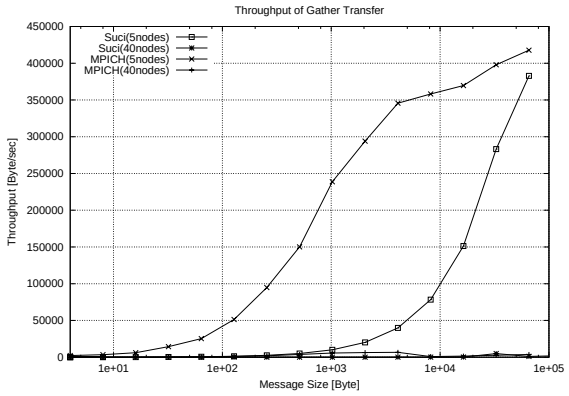

図 8: Gather のスループット

次に行った実験は、１ノードから他の全ノードに対してメッセージを送信する場合のスループットを計測した。Susi には、ブロードキャスト用の API が用意していない為、Tree 構造で、ブロードキャストするようにベンチマークプログラムを作った。５ノードと 40 ノードを対象に実検を行った。測定方法は、メッセージサイズを 4 バイトから 524288 バイトまで変化させて、受信ノードが全メッセージを受信するまでの時間から１ノードあたりのスループットを求めた。図 9 にその結果を示す。Suci のピーク性能は、メッセージサイズが 16384 バイトの時にスループットは約 5MByte/Sec を示した。16384 バイト以降は、５ノード、40 ノードのいずれの場合にも、Suci はスループットが低下している。これは、ブロッキング/デブロッキングに時間がかかっていると考えられる。一方,TCP をベースにした MPICH は、ノード数 5 の時点で 780KByte 付近で頭打ちになっている。ノード数が多くなると TCP はそれだけ、通信効率が悪くなっている事が分かる。これは、TCP ソケットがカーネル資源を浪費していることが原因として推定される。

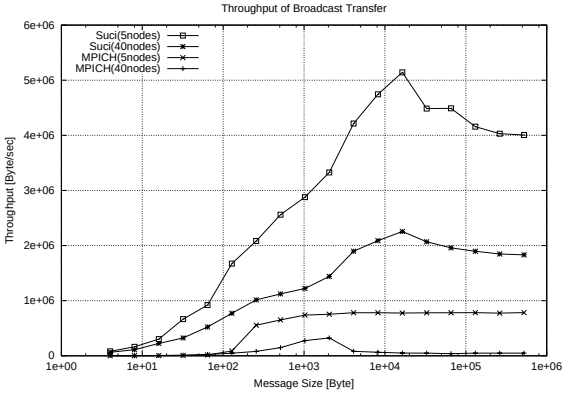

図 9: Broadcast のスループット

6　まとめ

本稿ではユーザーレベルでフロー制御を行うことのできる UDP ベースの通信ライブラリ Suci の性能評価を行った。基本性能に関しては、メッセージサイズが 65536 バイト付近でスループットが低下するものの、ピンポン転送の実検では TCP ベースの MPI と比較して 1000 バイト以下では 1.5 倍から 2 倍の性能が得られた。

また、バースト転送では、スループットの立ち上がりが若干遅いものの、MPICH/SCore では、メッセージサイズが 16384 バイト付近で、極端にスループットが低下してしているが、Suci では、それが見られない。比較的、高いスループットで安定している。1 対他の通信では、ブロードキャストに関しては比較的、通信効率は良いが、1 台にメッセージを集中させるような送信では、UDP のパケットロスが発生し、再送処理がおこることで通信効率は悪くなっている。

DVTS でのフロー制御では、フロー制御 API を使うことによって通信効率が向上し、フロー制御 API の有効性を示した。

今後の課題としては、より実用に近いかたちのベンチマークプログラムを Suci に移植し性能比較を行いたい。また、我々は Susi はあくまで、低レベルの通信ライブラリと考えているので、その上のレベルの通信ライブラリを被せて、ユーザーアプリケーションから扱い易い上位層の通信ライブラリを提共する事を考えている。

参考文献

[1] 塚本, 平野, 超分散情報社会システムへの招待, 情報処理第 36 巻 9 号 (1995)

[2] http://www.miricom.com/

[3] 河野真治, 神里健司. UDP を使った分散環境とその応用. 日本ソフトウェア科学会第 16 回大会論文集, 1999

[4] 増田哲之, 手塚宏史, 住元真司, 堀敦史, 石川裕. 通信ライブラリ PM の UDP 上への移植と評価, HOKKE'99, 情報処理学会,pp.127-132(1999).

[5] Gregory D. Burns, Raja B. Daoud and James,R. Vaigl. "LAM: An Open Cluster Environment for MPI". In Supercomputing Symposium '94, June 1994. Toronto, Canada.

[6] 尾屋祐二, 後藤滋樹, 西尾章治朗, 宮原秀夫, 村井純編, トランスポートプロトコル, 岩波書店, 2001.

[7] 小川晃通：DV(Digital Video) over IP, http://www.sfc.wide.ad.jp/DVTS/

[8] 玉城圭健, 神里健司, 天野嘉登, 河野真治データグラムを用いたマルチポイント DV ビデオ配信, JSSST2001, 日本ソフトウェア科学会第 18 回大会.