社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2003－OS－93 (12)
2003／5／8

Javaによるユーザレベルトランスポート層の実現と評価

山城 潤[†] 河野 真治[☆,††]

PC clusterによる通信では、フロー制御をユーザが行なうことが必要になる場合がある。しかし、現在広く使われているTCPでは、フロー制御がカーネルレベルで行なわれているために、ユーザがフロー制御を行なうことはできない。そこで、UDPにフロー制御APIと信頼性を付加することによってユーザによるフロー制御を可能にするPC cluster向け通信ライブラリをJavaで実装し、その性能を検証する。

Realize and evaluation of user level transport layer in Java

JUN YAMASHIRO[†] and SHINJI KONO[☆,††]

In the communication for PC cluster, sometimes need to flow control by user. TCP widely used in PC cluster. But, TCP cannot flow control by user. We implemented and evaluated communication library in Java. This library added flow control API and reliability in UDP.

1. はじめに

近年、PC clusterなどが安価に構築できるようになり、並列分散プログラムが実用的に使われるようになってきた。並列分散アルゴリズムで使われるデータ構造も単純な配列から複雑なデータ構造、例えば、スパース行列、決定二進木、ダブルリンクトリスト、さらに、相互参照されたオブジェクトなどが使われるようになってきている。

本研究室で研究している並列検証系でも決定二進木とPC cluster全体でユニークなIDを持つ項などを使用している。これらのデータ構造を持つ並列分散アルゴリズムをCやC++で記述することは不可能ではないが、かなり難しい。ここでは、Javaで並列分散アルゴリズムを記述するのに適した並列分散アルゴリズム用の通信ライブラリを提案する。

Java用の通信ライブラリは、Java 1.4に含まれており、低いレベルのTCPやUDPを使用することができる。上位レベルのライブラリとしては、Voyager[3]、HORB[4]、Aglets[5]などが知られている。しかし、これらは、インターネット、あるいは、サーバクライアントでの使用を前提としており、PC Cluster向きの通信ライブラリとは言えない。

† 琉球大学理工学研究科情報工学専攻
Information Engineering Course , Graduate School of Engineering and Science, University of the Ryukyus.
☆ 琉球大学工学部情報工学科
Information Engineering, University of the Ryukyus.
†† 科学技術振興事業団さきがけ研究21(機能と構成)
PRESTO, Japan Science and Technology Corporation.

PC cluster上での通信はインターネットと異なり、通信自体は高速なEthernetスイッチや専用のネットワークを使うことが多い。インターネットでは、特定の通信によりネットワークの帯域が占有されてしまうことは全世界のインターネットが止まることを意味するために禁止されていることであるが、PC cluster上では必要であれば帯域を占有して構わない。また、並列分散アルゴリズムでは、フロー制御や信頼性の保証などもアルゴリズムの一部であり、OSやネットワークの下位の層で自動的に行われる制御では不十分な場合がある。また、TCPは連続的なストリーム通信を前提としているので、多数のノードに多様なメッセージを送る並列分散アルゴリズムの実装には向いていない。実際、TCPを用いた通信ライブラリMPICH[6]などでは、特定の通信量でスループットが低下するなどの異常が起きることが知られている。

そこで、トランスポート層をユーザ空間に持って来ることにより、フロー制御や信頼性保証(Aknowledgeやシーケンス番号)、また、ブロッキング・デブロッキングを、並列分散アルゴリズムから直接に制御できるライブラリが望ましい。本研究室では、UDPを直接使用する通信ライブラリSuci(Simple User level Communication Interface)[2]をPrologとC言語で実装し評価してきた。

本論文では、Suciライブラリを Javaで使用する方法を提案し実装したので、それについて報告する。

2. 何故Javaなのか?

これまでSuciライブラリはPrologとCに対する

ものを実装してきた。もともと、並列検証系がPrologで記述されていたために、Prologの Stream I/O に合わせて設計されている。

しかし、Prologのデータ構造には並列分散アルゴリズムを記述する上で若干のデメリットがある。

- Prologの項は共有を直接に表現できないので決定二進木の記述に向かない。
- Prologのプログラムと Suciとのデータのやりとりには文字列を経由した変換が必要になる。
- Prologは習得が難しい言語なので、広く使ってもらいにくい。
- Prologのデータ構造は基本的に変更不可なので繰り返し使うデータを使用する時にはコピーが必要となる。

これらの Prologの特徴は、共有メモリマシン上での並列実行には適していたが、PCクラスタのような直接の共有の無い並列計算では欠点となることが多い。そこで、アルゴリズム自体をオブジェクト指向言語で記述することにした。

広く使われているオブジェクト指向言語としてはC++と Javaがあるが、複雑なデータ構造を持つアルゴリズムの場合はGCがある方が望ましい。また、プログラム記述の見通しの良さなどから Javaを選択した。もちろん、C++でも記述することは可能であり、この場合は、Cで記述されている Suciライブラリとの接続は容易である。ただし、オブジェクトを転送する際の表現の変換は自前で行う必要がある。Javaの場合は serializeというネットワーク転送用の methodが既に用意されている。

Suci ライブラリの Javaへの実装は以下の二つの方法が考えられる。

- JNIなどを用いてCで記述した Suciライブラリを呼び出す
- Suci ライブラリ自体を Pure Javaで記述する

前者は Suci ライブラリがCで記述されているために実装は容易であるが、Suci ライブラリはプラットホーム毎に用意する必要がある。後者は、ポータビィリティに優れている。しかし、Suci ライブラリを Javaで書き換える必要がある。

Suci ライブラリ自体がダブルリンクトリストなど用いた比較的複雑なプログラムとなっているので、これ自体にオブジェクト指向技術を用いることにより、ライブラリ自体の質が向上する可能性がある。また、Javaの持つメモリ管理機構、特に、GCを含む機構を使用することができるので、単純な文字列ではなく、オブジェクト自体を通信する場合に利点が出る可能性がある。

ただし、Java から UDPをアクセスする時のオーバヘッドはCよりは大きいと考えられる。また、Suci ライブラリはポーリングベースなので、パケットが来ているかどうかを常にチェックする必要がある。これもオーバヘッドとなる。Unix上での実装では、これは基本的には select システムコールに落ちるはずである。そこで、本研究では、試験的な Javaによる Suciの実装を行い、これらののオーバヘッドについて調べることにした。

3. Suci について

ここでは、Suci の特徴について説明する。

3.1 UDP による通信

Suci は PCクラスタの通信に使用することをを前提にして開発されたライブラリである。トランスポート層プロトコルに UDPを使用することで、通信の軽量化をはかり、UDPに欠けている信頼性を付加することができるように、再送処理を行えるようにする APIを提供している。

Suci では、メッセージ受信ループを前提としており、並列分散アルゴリズムの一部として、常に、パケットが自ノードに到着したかどうかを調べる必要がある。これにより、send/recv などで待ち合わせが生じることがなく、その待ち時間の間に自ノードでの計算を行うことができる。Suci は、ポーリングベースの通信ライブラリということができる。マルチスレッドで Suci ライブラリを使用しても構わないが、Dual CPU などでない限り、ほとんど利点がない。この論文ではマルチスレッドに関する実装や評価は行っていない。

3.2 ユーザレベルトランスポート層の実現

UDPでは送信したパケットのすべてが順序よく到達することは全く保証されない。そのために、送信したデータの内、いくつかのパケットが到着しなかったり、到着したパケットの順序がバラバラになることがある。

そのために、パケットが到達しなかった場合には、再送信のための処理を行う必要が生じる場合がある。また、到達したパケットの順序を整列しなおす必要がある。

そこで、Suci では UDPに信頼性を付加するために、送信するデータにシーケンス番号や送信先の/送信元のアドレスなどを記録する Suci ヘッダーを追加してから送信する。受信側では Suci ヘッダーの情報を元にして Acknowledge の送信や分割されたパケットの再構築を行う。

3.3 Suci の API

Suci の API は、基本的なデータ送受信のためのメソッドとフロー制御のためのメソッドから構成されている。API の解説を表 1 に示す。

Suci では、IP アドレス/ポート番号の組合せ対応した Host ID を利用して、通信先を識別する。また、データを送信するときには、送信先を事前に指定して行う (図 2)。

メソッド名	役割
datagram_read()	データの読み込み
datagram_write(byte[] sendBuf)	指定したノードへのデータ送信
datagram_ping(int dest_id, int mtimeout)	指定したノードに対して Ping を打つ
datagram_destinaton(in new_dest_id)	送信先ノードの Host ID を指定する
datagram_ready(int timeout)	到着したパケットの読み込み
datagram_retransmisson(int dest_id)	指定したノードへの再送処理の要求
ack_sync(int id)	指定した ID に対して、Acknoledge を送る

表 1 Suci の主な API

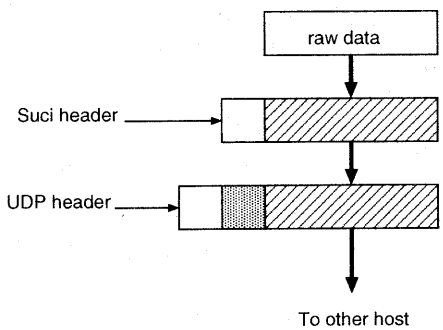

図 1 Suci と UDP スタックによるデータのカプセル化

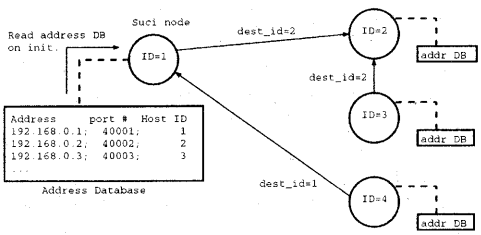

図 2 Suci による通信

4. Suci for Java の実装

Suci for Java は Java2 SDK 1.4 で実装された nio(New I/O)[7] の機能を利用している。そのため、Suci for Java の実行には nio の機能が実装されている Java VM が必要になる。

4.1 Suci for Java のクラス構成

Suci for Java では、API を提供するクラス**Suci**、パケットを表現するクラス**SuciPacket**、ノードのアドレスを保持するクラス**AddressDatabase**、個々のノードの IP アドレス、ポート番号を保持するクラス**SuciAddress** から構成されており、その関係は 3 で表される。

4.2 Suci のデータ送受信

Suci for Java はデータの送受信のためにバッファーの役割をするキューを持っており、再送される可能性のあるデータを Acknowledge が返って来るまで保持

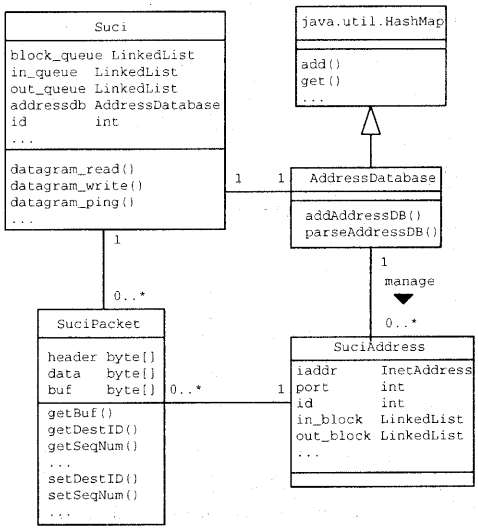

図 3 Suci for Java のクラス図

している。(図 4)

送信時には、datagram_write() でアプリケーションから渡されたデータを MTU ごとのパケットに分割し、ヘッダーを付加する。パケットは送信先アドレスごとに用意されている送信用キューに渡されたのちに送信される。

一度送信したパケットも再送しなければならない可能性もあるので、送信された後もパケットはキューに残っており、送信先から Acknowledge が来て再送する必要がなくなったことが確認されてからパケットを削除する。

送信元から受け取られたパケットは、送信元アドレスごとに用意されている入力キューに追加される。データを構成するために必要なパケットがたまると、完成したデータを (パケットの形で) 格納するためのブロックキューにパケットを移す。アプリケーションが datagram_read() を呼び出すと、Suci for Java はブロックキューにあるパケットを一つのデータとしてつなぎ直してからアプリケーションに返す。

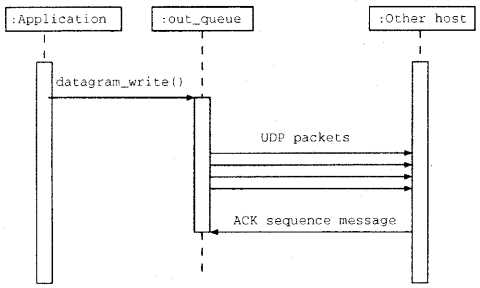

図 4　パケット送信時の動作を表すシーケンス図

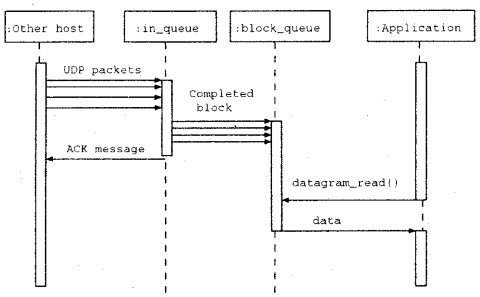

図 5　パケット受信時の動作を表すシーケンス図

4.3　パケットの組み立て

受信されたパケットはまず送信元のアドレスごとに用意されている SuciAddress.in_block(LinkedList クラス) に格納される。

その後、Suci.deblocking() メソッドでは、データを構成するパケットが全て SuciAddress.in_block に格納されると、Suci.block_queue(LinkedList クラス) にパケットをまとめて移動させる。

```
/* パケットを受け取り、そのパケットを
 * ヘッダー中のフラグを参考にして分類する。*/
private int deblocking(SuciPacket spack,
                       SuciAddress from_addr) {

  // ブロックの最初のパケットならば...
  if (spack.flagInType(this.BOB)) {
    /* 入力キューを初期化する */
    from_addr.in_block.clear();

    // ブロックの最後のパケットかどうか...
    if (spack.flagInType(this.EOB)) {
      /* ブロックを構成するパケットが
       * 一つだけなので、ブロックが
       * 完成されたものとみなして
       * block_queue にパケットを突っ込む。*/
      block_queue.addLast(spack);
      return 1;

    } else {
      /* まだ分割されたパケットが
       * 残っているので、
       * 入力キューにパケットを追加する。*/
      this.in_queue.addLast(spack);
      from_addr.in_block.add(spack);
      return 0;

  } else {
  // ブロックの最初のパケットではない。

    /* 最後のパケットならばブロックを構成する
     * パケットを block_queue に移動する。*/
    if (spack.flagInType(this.EOB) {
      this.in_queue.removeAll(from_addr.in_block);
      from_addr.in_block.add(spack);
      this.block_queue.addAll(from_addr.in_block);
      from_addr.in_block.clear();
      return 1;

    /* まだパケットが残っているときには
     * 入力キューにパケットを追加する。*/
    } else {
      from_addr.in_block.add(spack);
      return 1;
    }
  }
}
```

4.4　データの読み込み

受信されたパケットの内、全てのパケットが到着したことが確認されて、元の順序どおりに並び変えられたパケットが Suci.block_queue に格納される。

Suci.block_queue に格納されたパケットの列は、datagram_read() で byte[] に変換されてアプリケーションに渡される。

```
public byte[] datagram_read() {
  SuciPacket spack;
  ByteBuffer bbuf = ByteBuffer.allocate(this.BUFSIZ);

  for (;;) {
    /* de-blocking されたパケットが
     * 存在するならば、End of Block に
     * 達するまでパケット中のデータを
     * バッファーにコピーする */
    if (block_queue.size() > 0) {
      spack = (SuciPacket)block_queue.removeFirst();
      bbuf.put(spack.getData());
      /* 読み込んだパケットが End of Block の
       * 場合には、パケットの読み込みを終了し、
       * byte[] にして返す。*/
      if (spack.flagInType(this.EOB)) {
        int pos = bbuf.position();
        byte[] newbbuf = new byte[pos];
        bbuf.flip();
        System.arraycopy(bbuf.array(), 0,
                         newbbuf, 0,
                         newbbuf.length);
        return newbbuf;
      }
    }
    /* データを復元できるだけのパケットが
     * まだ到着していない場合には、
     * データが到着するまで待ち続ける。 */
```

```
    while (block_queue.size() == 0) {
      if (datagram_select() == 0) {
        return null;
      }
        receive_packet();
      }
    }
}
```

5. Suci for Java によるプログラミング

Suci for Java でのプログラミングスタイルは`while`ループによってイベントを待つポーリングベースである。

データを送信する API は datagram_write() であるが、この API の役割は UDP でデータを送りつけるだけなので、そのままデータを送っても相手は確実にデータを受信しているとは限らない。

受信側はデータを受信したことを確認すると、API ack_sync() で送信側に ACK を送信するので、送信側は ACK のパケットを受信するために datagram_ready() でパケットを読み込む。このとき、ACK が受信された場合には、送信が確認されたパケットが送信キューから取り除かれる。その後、datagram_retransmission() で到達したかどうか確認されていないパケットを再送する。

```
/*==== 送信側 ====*/
/* 初期化処理
 * アドレスデータベースを読み込み、
 * UDP による通信を受け付ける。
 */
Suci suci = new Suci("addressdb.txt", 1);
int target_id = 2;  // 送信先 ID の初期値
byte[] sendBuf = new byte[2048];

/* データ送信 */
suci.datagram_write(sendBuf);

/* 送信キューにデータがたまっている間は
 * メッセージを再送信する。 */
while (suci.datagram_queue_length(target_id) > 0) {
  // ACK が来ていたら読み込む。
  suci.datagram_ready(timeout);
  // メッセージの再送信
  suci.datagram_retransmission(server_id);
}

/*==== 受信側 ====*/
Suci suci = new Suci("addressdb.txt", 2);
byte[] recvBuf = new byte[2048];
int recvSize;

/* データ受信 */
while ((recvSize = suci.datagram_read(recvBuf)) > 0) {
  // Acknowledge 送信はプログラマーが行う。
  suci.ack_sync(sender_id);

  hoge(recvBuf);  // なんらかの処理

  // 次のパケットが来るまで待機する。
  suci.datagram_ready(timeout);
}
```

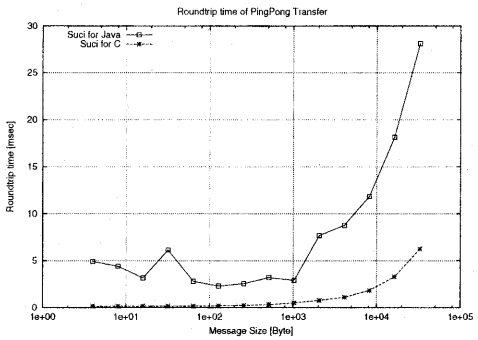

図 6 Suci for Java と C 言語版 Suci との比較 (ラウンドトリップタイム)

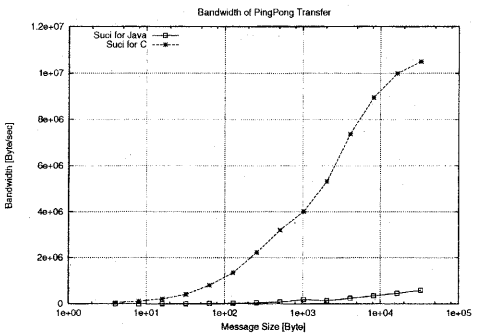

図 7 Suci for Java と C 言語版 Suci との比較 (帯域幅)

6. 他の通信ライブラリとの性能比較

6.1 ベンチマーク

Suci for Java の性能を確かめるために、2 つのノード間でパケットをピンポン転送させて、パケットの往復にかかる時間と、消費する帯域幅について測定するプログラムを Suci for Java、C 版の Suci ライブラリ、Java 上の TCP で実装し、それぞれの実行結果を比較した。

C 版 Suci ライブラリとの比較では、ラウンドトリップタイム (図 6) は Suci for Java が C 版 Suci ライブラリの約 5〜40 倍の時間がかかっており、帯域幅 (図 7) も Suco for Java と C 版 Suci ライブラリでは約 20〜160 倍の開きがある。

TCP との比較では、ラウンドトリップタイム (図 8) は Suci for Java が TCP の約 7〜30 倍の時間がか

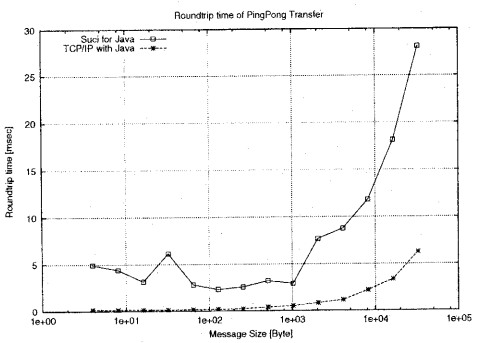

図 8 Suci for Java と TCP との比較 (ラウンドトリップタイム)

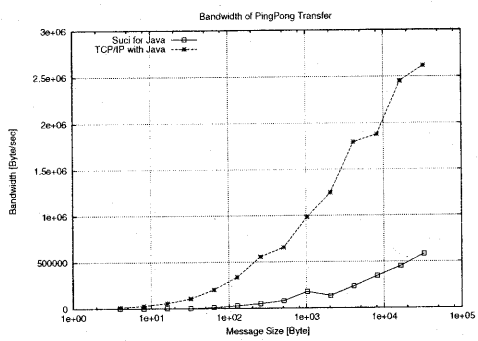

図 9 Suci for Java と TCP との比較 (帯域幅)

かっており、帯域幅 (図 9) も Suci for Java と TCP では約 9〜35 倍の開きがある。

6.2 Jcluster

Jcluster[8] は Zhang Baoyin が開発した Java 向けの並列環境である。Jcluster は Suci for Java と同様に、UDP に信頼性を付加することによって通信の高速化をはかっている。

最新のリリースである Jcluster toolkit V 0.9 では、MPI like な API も提供しており、マルチスレッドによる通信とタスクスケジューリングの最適化が行われている。

Jcluster の配布セットに含まれているサンプルコードを使い、Jcluster によるプログラミングを説明する。

```
/* jcluster/userApps/test/PingPang.java */
/**
Author: Zhang BaoYin
**/

package test;

import jcluster.JTasklet ;
import jcluster.JException ;
import jcluster.JEnvironment ;
import jcluster.JMessage ;

import java.io.IOException ;
import java.util.Enumeration ;

/* Jcluster のプログラムはクラス JTasklet を
 * 継承し、プログラムの中身は
 * work() メソッドとして実装する。
 */
public class PingPang extends JTasklet{

  public PingPang() {
  }

  public void work(){
    /*
     * env.pvm_parent() の返り値によって
     * そのプログラムが「親」として動作するのか、
     * 「子」として動作するのかが決まる。
     * (Jcluster 環境での環境変数となる env
     * (クラス JEnvironment) は JTasklet のメンバーである)
     */
    if (env.pvm_parent () == -1){
      // プログラムは「親」として動作する。
      System.out.println ("begin spawn child...");

      /*
       * 子プロセスの実行を開始する。ここでは、
       * クラス test.PingPang のプロセスを 1 個実行する。
       */
      env.pvm_spawn ("test.PingPang",1);

      try{
      // 子プロセスが実行されるまで待機。
      Thread.sleep(500);
      }catch(InterruptedException e){
      }

      try{
        JMessage m = new JMessage(env,8);
        // 子プロセスからデータを受信する。
        JMessage mm = env.pvm_recv ();

        /* 子プロセスから送られてきたデータを
         * JMessage.unpack() メソッドで取り出して表示する。
         * (この例では"hello parent!"という文字列が
         * 表示される) */
        System.out.println(mm.unpack ());

        // 子プロセスの task ID を取得
        int child = mm.getSource ();

        // prepare
        for (int j=0;j<3000;j++){
          // 子とのピンポン転送を行う (前準備)

          /*  mm.getSource() で取得した
           * task ID のプロセスにメッセージを送る。*/
          env.pvm_ssend (m,child,0);
          env.pvm_recv ();
        }
```

```
    long s, t, p;

    p = 0;

    /* パケット 3000 往復*5 回のピンポン転送を行い、
     * 実行にかかる時間を測定する */
    for (int i=0;i<5;i++){
      s = System.currentTimeMillis ();
      for (int j=0;j<3000;j++){
        env.pvm_ssend (m,child,0);
        env.pvm_recv ();
      }
      t = System.currentTimeMillis ();
      p = p + (t-s);
  }

  // 結果を表示して親プロセスは終了
  System.out.println("The latency is " +
                     p*1000/(5*3000*2) +
                     " microseconds.");
}catch(JException e){
  System.out.println(e.getMessage ());
}

System.out.println ("parent over.");

} else {
  // プログラムは「子」として動作する。
  System.out.println ("Child begin...");

  JMessage m = new JMessage(env);

  try{
    /* JMessage.pack() メソッドを使い、
     * メッセージにデータを詰め込む。
     */
    m.pack ("hello parent!");
    env.pvm_send(m,0);

    m = new JMessage(env,8);

    // prepare
    for(int i=0;i<3000;i++){
      // 親とのピンポン転送を行う (前準備)
      env.pvm_recv ();
      env.pvm_ssend(m,env.pvm_parent ());
    }

    // ピンポン転送の本番
    for(int j=0;j<5;j++){
      for(int i=0;i<3000;i++){
        env.pvm_recv ();
        env.pvm_ssend(m,env.pvm_parent ());
      }
    }
  }catch(JException e){
    System.out.println (e.getMessage ());
  }
  System.out.println("Child over.");
}
```

```
  }

} // the end of class
```

Suci for Java と、Jcluster との主な違いは以下の通りである。

- Suci for Java のプログラミングスタイルがポーリングベースなのに対し、Jcluster のプログラミングスタイルはスレッドベースのプログラムである。
- Suci for Java はフロー制御のための低レベル API を用意しているのに対し、Jcluster では PVM/MPI like な API が用意されているが、これは read/write/multicast/spawn などの基本的な操作に限定されている。

7. まとめと今後の課題

測定の結果、Suci for Java の現在の実装では C 版 Suci や TCP と比較して良いパフォーマンスを出していないことがわかった。

現在、パフォーマンス低下の理由について調査しており、その結果を元に Suci for Java のパフォーマンスをを改善し、Jcluster のような他の Java 用並列ライブラリとの差別化をはかる予定である。

また、Suci for Java を使わずに JNI 経由で C 版 Suci ライブラリを利用するライブラリを開発し、Suci for Java を利用する場合や、Jcluster などを利用する場合と比較してどちらのパフォーマンスが優れているのかを検討する。

参 考 文 献

1) 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
2) 屋比久 友秀, 河野 真治. 並列分散ライブラリ Suci の実装と評価. システムソフトウェアとオペレーティングシステム予稿集, 2002
3) Voyager http://www.recursionsw.com/products/voyager/voyager.asp
4) HORB http://horb.a02.aist.go.jp/horb/
5) Aglets http://aglets.sourceforge.net/
6) MPICH http://www-unix.mcs.anl.gov/mpi/mpich/
7) New I/O APIs http://java.sun.com/j2se/1.4.1/docs/guide/nio/
8) Jcluster http://vip.6to23.com/jcluster/