Java におけるオブジェクト指向メッセージパッシングライブラリの開発

Development of Object Oriented Message Passing Library in Java

日下部 明（知的システムデザイン研究室）
Akira KUSAKABE (Intelligent Systems Design Laboratory)

Abstract Recently PC cluster is become major parallel computing environment. In PC cluster, message passing paradigm is suitable. PVM and MPICH are major parallel message passing library in C and Fortran. There are some PVM and MPI inplementation ported to Java. These library are ported as similar to its original.These library are concentrated to messaging array of primitive type. Java is sophisticated object oriented language. This paper suggest a message passing class library for Java. Its character is capability to send-receive inherited class of `java.lang.Object` and no necessity of care for message size by programer. In this paper, I implement the class library and evaluate its basic perfomance.

1　はじめに

近年のコンピュータ技術の急速な進歩により，低価格で高性能なハードウェアを入手できるようになった．単体でも動作するコンピュータをネットワークで接続し，並列計算機として用いることはハイパフォーマンスコンピューティングの分野ではもはや一般的になりつつある．このような PC クラスタ型の並列計算機環境においては，独立した個々のコンピュータを協調動作させるためのミドルウェアの存在が重要である．C や Fortran での開発を前提とすれば，PVM[1] や MPI[2] の使用が一般的である．

近年，Sun MicroSystems,Inc. によって開発された Javaが急速に普及しており，Javaを用いてハイパフォーマンスコンピューティングを行おうとする試みが見られる [3]．しかし，クラスタ環境において Javaで並列プログラムを開発するためのミドルウェアで，決定的といえるような実装はまだないというのが現状である．

Java などのオブジェクト指向言語の利点の一つに，ユーザがデータ型を定義できるということがある．そのため，ユーザが定義したクラスのインスタンスを送受信できる API は，並列処理プログラミングにおいて非常に大きな利点がある．本研究は，オブジェクトの送受信に適した並列処理用通信 API を実装し，その評価を目的とする．

2　関連研究

Javaにおけるメッセージパッシングクラスライブラリは PVM や MPI を移植したものが多い．オリジナルの MPI 仕様では Fortran/C/C++での仕様しか定義されていないが，Java Grande Forum のワーキンググループが MPI の Java バインディングを策定している [4]．JPVM[5] は Pure Java による PVM の実装である．既存の PVM との相互運用はできない．jPVM[6] は JNI を利用したネイティブコードの PVM に対するラッパーである．MPIJ[7] は Pure Java による MPI の実装である．これは MPI2-C++のインターフェースに近い．mpiJava[8] は JNI を用いたネイティブコードの MPI に対するラッパーである．これは前述の MPI の Java バインディングにあわせている．JCI(Java-to-C Interface generator)[9] は C のネイティブライブラリのヘッダファイルから，C のスタブ関数，ネイティブメソッドを呼び出す Java のソースファイル，およびこれらをコンパイルするためのシェルスクリプトを生成する．

これらを実装方法により大別すると，JNI により既存のネイティブメッセージパッシングライブラリを呼び出すものと，新たに Java で実装したものとに分類できる．既存の PVM や MPI を呼び出すタイプのものは高い通信性能を示すが [8]，Java の特徴のひとつであるポータビリティに欠ける．逆に，Java のみで実装したものは若干通信性能で劣る [5, 7]．これは，Java ではプリミティブなデータタイプが厳密に区別されており，マーシャリングの際のバイト列への変換に時間がかかるためである [3]．マーシャリングの部分をネイティブコードに置き換えるとネイティブのメッセージパッシングライブラリと同等の通信性能を出せることがしめされている [7]．

3　オブジェクト メッセージパッシング

PVM や MPI は C と Fortran での使用を前提として開発されたため，整数や実数などのプリミティブなデー

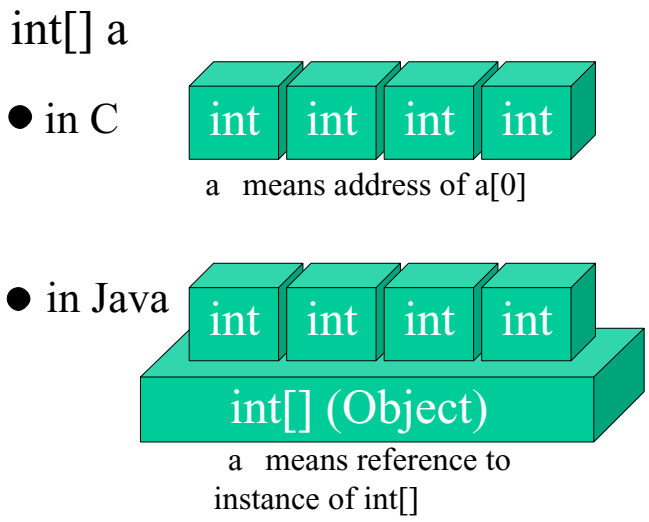

図 1: difference of array between C and Java

タタイプの配列を高速に送受信することに主眼が置かれている．これらは送受信バッファに対するさまざまな操作や送受信モードなどのプリミティブな操作を用意することで，プログラマがメッセージパッシングの部分をチューニングすることを可能にしている．その反面，メッセージバッファのサイズやメッセージのサイズをプログラマが管理する必要があるなど，プログラマへの負担が大きい．また，構造体などの複合的なデータもプログラマの明示的なパック/アンパックによって送受信可能であるが，これもプログラマの負担が大きい．Java Grande Forum によって策定された MPI の Java バインディングでは，データタイプとして`MPI.OBJECT`型を新たに追加した．これにより，シリアライズ可能な`java.lang.Object`の派生型をメッセージとするコードを書くことが可能となった．

MPI の Java バインディングにより，C や Fortran で得た MPI のノウハウを生かしつつ，Java での並列プログラムの開発が可能となった．しかし，著者は Java の特徴をさらに生かすため，オブジェクトをメッセージとすることが可能で，プログラマがメッセージサイズの管理をする必要のないメソッドシグネチャを持ったメッセージパッシングクラスライブラリを提案する．ライブラリの中の，オブジェクトをメッセージとする送受信メソッドのシグネチャを次に示す．

```
void sendObject(Object message, int dest, int tag)
Object recvObject(int source, int tag)
```

C において配列は単にメモリ上に連続して値が配置されているだけであり，そのサイズはプログラマが管理する必要があった．しかし，Java の配列はそれ自体がオブジェクトであり，配列は自分のサイズを知っている（図 1）．送受信するメッセージは配列であっても単一のオブジェクトであるとみなすことで，プログラマがメッセージサイズを管理することから開放している．つまり，受信側は「送信されてきたメッセージはすべて受信する」という方針にすることで，メッセージサイズをメソッドの引数から排除している．この方法は受信のたびに受信バッファの確保を行うため，メモリ効率と速度の点で MPI に劣るが，プログラマの負担を軽減する．プログラマが定義したクラスのインスタンスをメッセージとして送受信したい場合に必要なのは，`java.io.Serializable`インターフェースをクラス定義に追加するだけである．特別なメソッドをプログラマが用意したりする必要はない．

4 起動方法

クラスタを並列計算機として使用する場合，リモートマシンに並列計算タスクを起動する機能は並列処理用ミドルウェアに必須である．PVM ではユーザプログラムの実行前に，PVM タスクを管理する PVM デーモンを起動しておく必要がある．MPI の実装である MPICH[10] や LAM[11] では専用の起動プログラムでリモートマシンにタスクを起動する．提案するライブラリでは，ユーザプログラムが並列ライブラリの初期化メソッドを呼び出したときにリモートマシンにタスクを起動する．つまり，ユーザは JDK の標準的なスタイルそのままでプログラムを実行することが可能である．もっとも単純な起動方法は次のようになる．

```
java -Ddmp.n=number_of_task class
```

このように，起動させる並列タスクの数を Java のプロパティとして与えることで，main メソッドへの引数を一切変更することなく起動させることが可能である．

5 内部実装

本節では提案するクラスライブラリの実装について説明する．本ライブラリは TCP ソケットによる通信に基づいており，並列タスクはすべての並列タスクとコネクションを持つ．それぞれのコネクションは`DMPTaskConnection`クラスのインスタンスによって管理されている（図 2）．本ライブラリは非同期-ブロッキングの通信モデルを提供する．送信側は受信側で対応する受信メソッドが発行されているかどうかにかかわらず，送信メソッドを完了することができる．受信側はメッセージ受信専用のスレッドでメッセージの到着を待機している．受信スレッドはメッセージを受信すると，メッセージキューに入れる．受信メソッドはメッセージキューにメッセージが到着するまでブロックする．

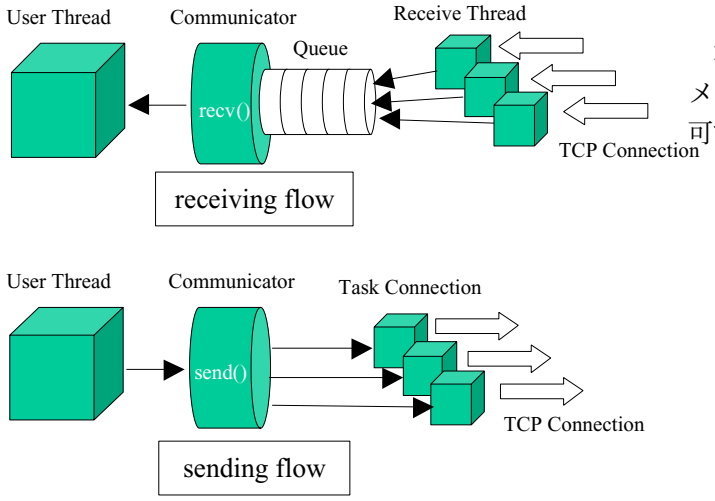

図 2: implementation

6 適用例

本節では提案するクラスライブラリを用いたオブジェクト送受信の例を示す．図 3 のような`TimeStamp` クラスと`Paper` クラスを定義する．`TimeStamp` クラスはプライベート メンバとして`String` クラスのホスト名と`Date` クラスの時刻を持っている．`TimeStamp` クラスはコンストラクタが呼び出されたとき，自分のホスト名と時刻を記録する．`Paper` クラスは`TimeStamp` クラスを格納するための`Vector` クラスのオブジェクトをプライベート メンバとして持っている．

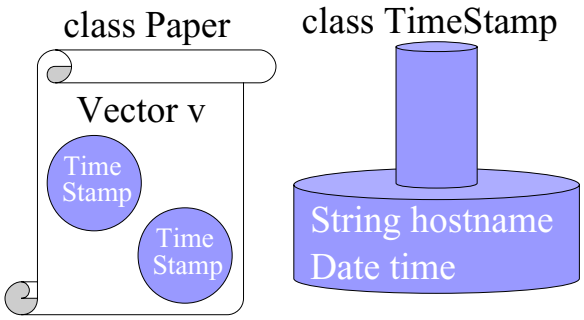

図 3: class Paper and class TimeStamp

並列タスクは`Paper` クラスのインスタンスを受信し，`TimeStamp` クラスのインスタンスを生成して`Paper` クラスのインスタンスに格納する，そして次の並列タスクへ`Paper` クラスのインスタンスを送信する．この概念を図 4 に示す．この操作は Java のソースコード上では次に示す 3 行であらわすことが可能である．

```
Paper paper = (Paper)comm.recvObject(source, tag);
paper.stamp(new TimeStamp());
comm.sendObject(paper, dest, tag);
```

本ライブラリを用いることによって，オブジェクトをメッセージとするコードを極めて自然に表現することが可能である．

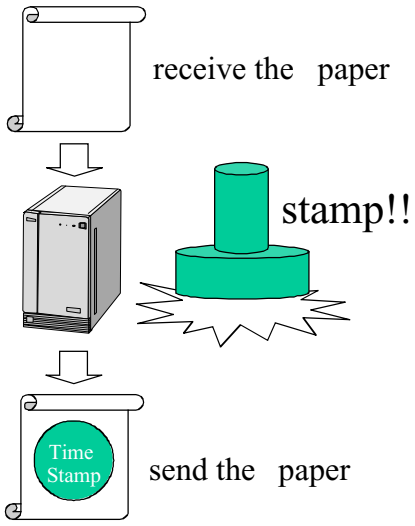

図 4: object message passing

7 通信性能

本節では提案するライブラリの通信性能を評価する．測定した環境は Pentium II 400MHz，メモリ 128MByte のマシンのクラスタで，100BASE-TX のイーサネットとスイッチングハブで構成されている．OS は Linux2.2.12，JDK は IBM が開発した JDK1.1.8 である．図 5 は MPICH，Java の`ObjectInputStream` と`ObjectOutputStream`，提案するライブラリのバンド幅である．MPICH ではあらかじめバッファを確保し，バンド幅の算出に考慮した時間はバッファの内容の送受信にかかった時間だけである．これに対し，`ObjectInputStream/ObjectOutputStream` と提案するライブラリでは，送受信のたびにバッファを確保しなおしており，バンド幅の算出にはバッファ確保の時間も入っている．MPICH でのバンド幅は，この環境においてユーザが並列処理 API を用いて利用できるほぼ最大のバンド幅と考えられる．これらを比較すると，`ObjectInputStream/ObjectOutputStream` は MPICH に近い値が得られているが，`ObjectInputStream/ObjectOutputStream` 上に実装している提案されたライブラリはこれらを大きく下回っている．これは内部実装に大きな改善の余地があることを示している．推測される問題点は受信スレッドによるオーバーヘッド，メッセージキューの検索，受信バッファの確保などが考えられる．

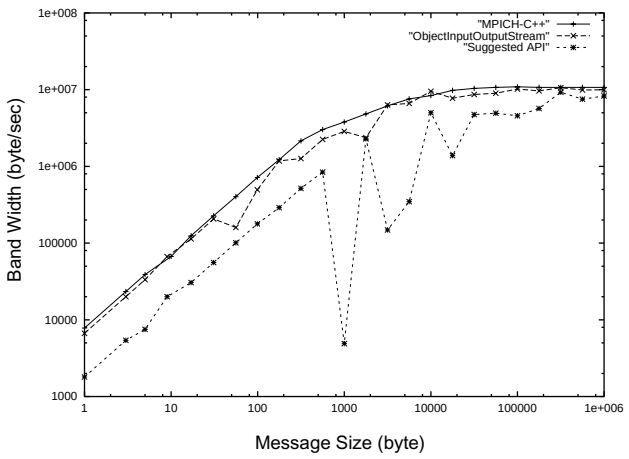

図 5: Band Width

8 実装課題

本ライブラリはひとつの並列タスクがひとつの JVM（Java Virtual Machine）に割り当てられる．シングルプロセッサのマシンを用いてクラスタを構成する場合はこの実装が適している．しかし，マルチプロセッサのマシンを用いてクラスタを構成する場合，同一マシン上に複数の JVM が起動され，これらがネットワークソケットを介して通信を行う．しかし，Java は単一の JVM 上に複数のアプリケーションを起動することが可能であり，この場合，それぞれのアプリケーションはスレッドとして扱われる．図 6 はマルチプロセッサマシンにおけるノード内の並列タスク間通信を表している．図 6a) は１つのプロセッサに１つの JVM を割り当てた場合であり，タスク間通信はネットワークソケットを介して行われる．図 6b) は複数のプロセッサに１つの JVM を割り当てた場合であり，タスク間通信は共有メモリを介して高速に行われる．

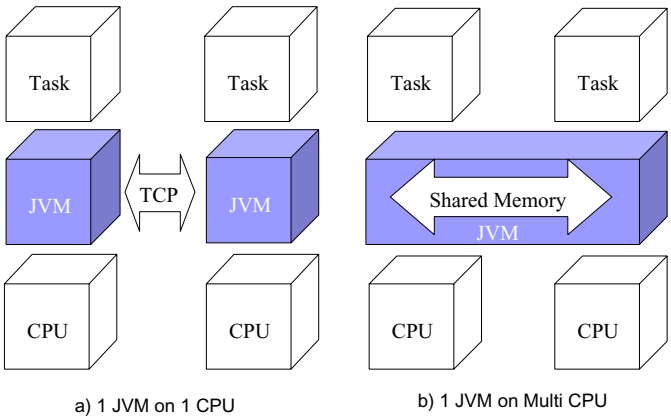

図 6: Intra Node Communication

また，図 7 は２プロセッサのマシン２台におけるノード間の並列タスク通信を表している．図 7a) は１つのプロセッサに１つの JVM を割り当てた場合であり，すべてのタスク間通信はネットワークソケットを介して行われる．この場合，ノード内で２コネクション，ノード間で４コネクションの合計６つの TCP コネクションが生成される．これに対し，図 7a) は２つのプロセッサに１つの JVM を割り当てた場合である．この場合，ノード内のタスク間通信は共有メモリを介して行われ，ノード間のタスク間通信のために１つの TCP コネクションが生成されるだけである．このため，ノード間通信の性能も向上することが見込まれる．

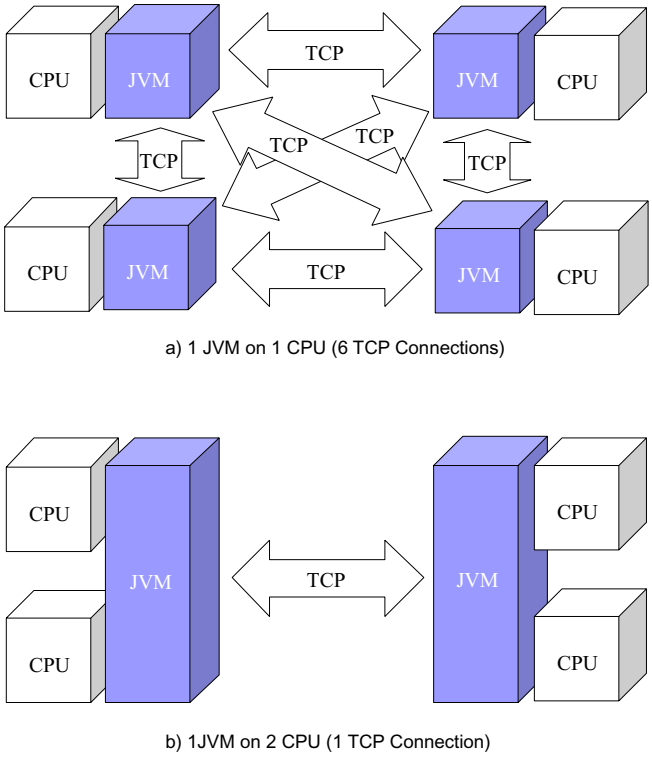

図 7: Inter Node Communication

9 結論

本研究はクラスタでの使用を想定したオブジェクト指向メッセージパッシングライブラリを Java で実装し，その評価を行った．得られた結論は以下のとおりである．

1. メッセージとして`java.lang.Object`の派生クラスを送受信できるようにすることで，より自然なオブジェクト指向プログラミングが可能となる．
2. メッセージが送信されてくるたびに受信バッファを確保することで，プログラマが受信メッセージのサイズを管理しなくてもよい受信メソッドにすることが可能である．

参考文献

[1] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam『PVM 3 user's guide and reference manual』(Oak Ridge National Laboratory,1994)

[2] MPI Forum『MPI: A message-passing interface standard』(International Journal of Supercomputer Applications, 1994)

[3] Java Grande Forum『JGF-TR-1: Making Java Work for High-End Computing』(Java Grande Forum, 1998)

[4] B. Carpenter, V. Getov, G. Judd, T. Skjellum, G. Fox『JGF-TR-3: MPI for Java: Position Document and Draft API Specification』(Java Grande Forum, 1998)

[5] A. Ferrari『JPVM: network parallel computing in Java』(Java Grande Forum, 1998)

[6]『jPVM』http://www.chmsr.gatech.edu/jPVM/

[7] G. Judd, M. Clement, Q. Snell『Design Issues for Efficient Implementation of MPI in Java』(ACM Workshop on Java for High-Performance Network Computing, 1999)

[8] Mark Baker, Bryan Carpenter, Geoffrey Fox, Sung Hoon Ko, and Sang Lim『mpiJava: An Object-Oriented Java interface to MPI』(International Workshop on Java for Parallel and Distributed Computing, 1999)

[9] S. Mintchev, V. Getov『Towards Portable Message Passing in Java: Binding MPI』(Proceedings of EuroPVM-MPI, 1997)

[10] William D. Gropp, Ewing Lusk『User's Guide for mpich, a Portable Implementation of MPI』(Mathematics and Computer Science Division, Argonne National Laboratory, ANL-96/6, 1996)

[11]『LAM』http://www.mpi.nd.edu/lam/