JavaによるMPIの実装と評価

日下部 明 *, 廣安 知之 **, 三木 光範 **

* 同志社大学大学院 (現所属 日本オラクル株式会社)　** 同志社大学工学部

本研究では，並列処理におけるメッセージパッシング API の標準である MPI を Java で実装し，PC クラスタ上でその評価を行う．MPI の C/C++/Fortran バインディングではメッセージバッファはプリミティブ型とその派生型の配列に限られていたが，本研究ではシリアライズ可能なオブジェクトをメッセージとして送受信できるように実装した．そのため，ユーザ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することが可能である．また，オブジェクトの送受信に最適化したメソッドを実装した．この機構により並列処理プログラムの記述量が減少し，より自然なオブジェクト指向並列処理プログラミングが行えることを示す．

Implementation of Java-MPI Binding and Its Evaluation

Akira KUSAKABE*, Tomoyuki HIROYASU** and Mitsunori MIKI**

*Doshisha University(Presently Oracle Corporation Japan)　**Doshisha University

In this study, we develop MPI implementation in Java, and evaluate its performance. In C/C++/Fortran binding of MPI, type of message buffer is restricted to primitive array and derived data type. Our implementation of MPI can transport serializable objects as messages. The merit is capability of user defined class transportation. We implement send/receive method optimized object transportation. The method enable to reduce writing of parallel code, and more natural object oriented parallel programming.

1 序論

Java は Sun Microsystems, Inc. によって開発されたオブジェクト指向言語である．Java はインタプリタ型言語，C++に似た構文規則，優れたクラスライブラリといった特徴を持っており，ネットワークアプリケーションの開発を中心に急速に普及した．Java はその実行形態から”write once, run anywhere”であり，優れたポータビリティを備えている．このことが意味するのは，プログラミングにプラットフォーム固有の知識を必要とせず，アプリケーションのアルゴリズムの開発に集中できるということである．そのため，Java によって大規模計算のコードを開発できることは開発者にとって大きな利点をもたらす．しかし，Java は実行速度が遅い，言語仕様が数値計算に向いていないなど，大規模計算をおこなうには不利な欠点を持っている．これらの欠点を補うべく，Just In Time コンパイラの開発 [1, 2, 3] や，数値計算ライブラリの研究 [4, 5] がなされている．

大規模計算を実行させる環境として，並列計算環境は必須である．Java は言語仕様自体にスレッドの使用が考慮されており，ネイティブスレッドが使用可能な Java Virtual Machine とマルチプロセッサ構成のマシンを組み合わせることによって，容易に並列処理コードを記述することができる．しかし，安価に購入できるマルチプロセッサマシンは数プロセッサ構成のものが限界である．そこで，低コストで多数のプロセッサを持つ並列計算環境を構築すべく，クラスタによる並列計算が注目を集めている．

クラスタ環境における通信モデルとして，RMI[6] や CORBA[7] に代表される分散オブジェクトテクノロジと，PVM[8] や MPI[9, 10] に代表されるメッセージパッシングライブラリがあげられる．分散オブジェクトテクノロジはユーザにオブジェクトの位置透過性を提供し，ノード間通信を隠蔽する．この機構はユーザにノード間通信を意識させないプログラミングが可能である．しかし，クラスタ

においてボトルネックとなるのはノード間通信であり，ノード間通信を意識したプログラミングをしないと十分な実行速度が出ない．メッセージパッシングライブラリはノード間通信を明示的にコントロールするため，大規模計算の並列化ライブラリの主流となった．

クラスタ環境で Java による並列処理をおこなうために，PVM や MPI を Java に移植する試みが見られる．JPVM[11] は完全に Java で記述された PVM クローンである．MPIJ[12] は完全に Java で記述された MPI であり，これは Distributed Object Group Metacomputing Architecture（DOGMA）システム [13] を構成するコンポーネントである．MPIJ は MPI の C++バインディングに近いモデルである．MPIJ はデータマーシャリングをネイティブコードで行うことにより，ネイティブコードの MPI と同等の通信性能を出せることを示している．mpiJava[14] はネイティブコードの MPI 実装である MPICH[15] などを Java から呼び出すインターフェースであり，メッセージのデータ型として`MPI.OBJECT`を実装している．Java-to-C Interface generator（JCI）[16] は C によって記述されたネイティブコードの MPI のヘッダファイルから，C のスタブ関数，ネイティブの MPI ライブラリを呼び出す 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．

MPI Forum による仕様では，MPI の Java バインディングは未定義である．そこで，Java Grande Forum（JGF) が MPI の Java バインディング (以下 Java-MPI) を策定中である [17]．JGF による仕様は，MPI Forum による C++バインディングを基に，Java の言語仕様に適するように変更が加えられたものである．我々は JGF による仕様の中で，メッセージのデータ型として`MPI.OBJECT`が追加されたことに着目した．Java ではプリミティブ型以外はすべてオブジェクトであり`java.lang.Object`を継承したサブクラスである．ユーザが定義したクラスのインスタンスも当然オブジェクトである．よって，メッセージのデータ型に`MPI.OBJECT`を指定できることは，ユーザが定義したクラスのインスタンスをメッセージとすることも構文上は可能であることを意味する．

我々は Java Grande Forum による MPI の Java バインディングを基に Java で MPI を実装した．本実装はプリミティブ型の配列だけでなく，シリアライズ可能な任意のオブジェクトを直接メッセージバッファに入れることができる．本稿ではその基本的な性能の評価を行う．また，オブジェクトをメッセージとして送受信できることはコードの記述量を削減することができ，より自然なオブジェクト指向並列処理プログラミングが行えることを示す．

2 Java による MPI の実装

2.1 MPI データ型

MPI 標準はオブジェクトを基本としており，C や Fortran ではオブジェクトハンドルによって内部の不透明オブジェクトを操作していた．MPI-2 標準で追加された C++のバインディングではこれらのオブジェクトを適切なクラス階層に集め，ライブラリ関数をこれらのクラスのメソッドとして定義した．Java-MPI バインディングは MPI Forum による C++バインディングが基本となっている．

MPI の中で最も重要なクラスはコミュニケータをあらわす`Comm`である．すべての通信メソッドは`Comm`もしくはそのサブクラスのメソッドである．`Comm`に定義されている標準送信/受信メソッドのシグネチャを次に示す．

```
void send(    Object buf, int offset,
              int count, Datatype datatype,
              int dest, int tag )

Status recv(  Object buf, int offset,
              int count, Datatype datatype,
              int src, int tag )
```

これらのメソッドでメッセージバッファを表す引数`buf`は，メッセージバッファのデータ型をあらわす引数`datatype`を要素とする配列である．引数`datatype`がプリミティブ型をあらわす場合，メッセージバッファはそのデータ型の１次元配列である．C の場合，多次元配列は１次元配列にマッピングすることができるため，多次元配列の送受信は１次元配列の送受信と同じである．しかし，Java の場合，配列はオブジェクトであり，多次元配列は配列の配列である．よって，多次元配列はオブジェ

クトの１次元配列にマッピングされる．そのため，多次元配列を１度のメソッド呼び出しで送受信するためには，データ型としてオブジェクトが扱えることが必要であった．Java-MPIバインディングではデータ型として`MPI.OBJECT`が定義されている．これにより，多次元配列を１度のメソッド呼び出しで送受信できるだけでなく，そのほかのオブジェクトも送受信できる可能性を持つ構文となった．

異なるデータ型の複合体を送受信する別の方法として，MPI派生データ型を定義する方法がある．Java-MPIバインディングでは派生データ型についても定義されているが，我々は派生データ型を重要視していない．なぜならオブジェクトをメッセージとすることができれば異なるデータ型の複合体を送受信することは可能であり，Javaではそのほうがより自然なプログラミングスタイルだからである．現在の我々の実装では派生データ型は扱っておらず，表１に示すJava-MPIバインディングで定義された基本データ型のみを実装している．

表 1: Basic datatypes in Java-MPI binding

MPI datatype	Java datatype
MPI.BYTE	`byte`
MPI.CHAR	`char`
MPI.SHORT	`short`
MPI.BOOLEAN	`boolean`
MPI.INT	`int`
MPI.LONG	`long`
MPI.FLOAT	`float`
MPI.DOUBLE	`double`
MPI.OBJECT	`Object`

2.2 データマーシャリング

Javaでネットワーク越しにデータを移動させるには`InputStream/OutputStream`およびそのサブクラスを用いる．`InputStream/OutputStream`が直接扱えるのは`byte`の配列だけであるため，メッセージバッファを何らかの方法で`byte`の配列に変換しなければならない．Javaでは配列はオブジェクトであり，その要素数は配列自身が保持しており変更することはできない．つまり，プリミティブ型の配列であっても，キャストによって`byte`の配列とみなすということはできない．

我々の実装が用いたマーシャリング手法は，図１と図２に示すように，メッセージバッファの部分配列をコピーし，これをひとつのオブジェクトとして`ObjectOutputStream`の`writeObject`メソッドで書き出すというものである．受信側では`ObjectInputStream`の`readObject`メソッドによりメッセージオブジェクトを読み出し，受信バッファにコピーする．この手法は部分配列のコピーが必ず発生するという欠点と，オブジェクトヘッダがメッセージに付加されるという欠点を持っているが，`ObjectOutputStream`と`ObjectInputStream`のマーシャリングは一般的にネイティブメソッドであり，高速なマーシャリングが期待できる．また，この実装はプリミティブ型の配列とオブジェクト型の配列を区別しなくてもよいという利点がある．

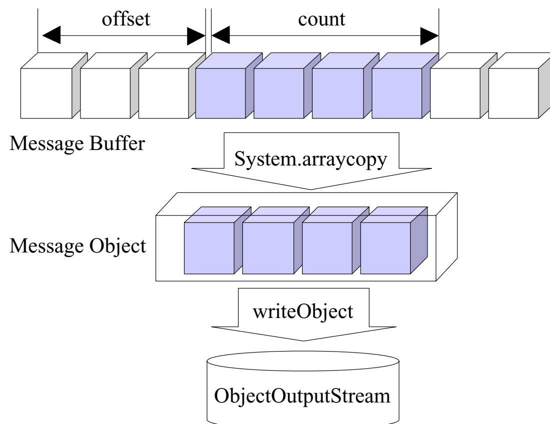

図 1: Marshalling

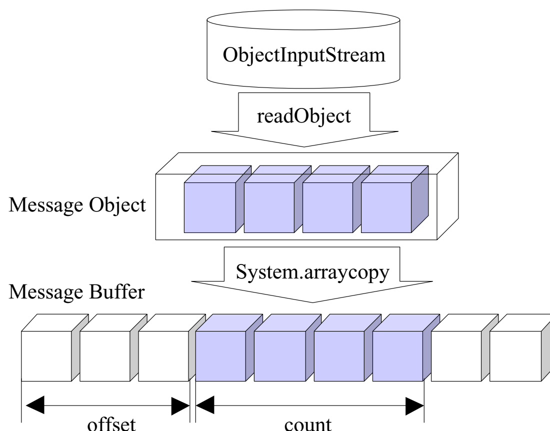

図 2: Unmarshalling

2.3 MPI タスクの起動

クラスタ環境において Java で並列処理を行うためには，リモートマシンに MPI タスクを起動する機構が必要である．MPI のネイティブコードの実装である MPICH[15] や LAM[18] では起動スクリプトを用いて MPI タスクをスタートさせる．MPICH では`mpirun`がリモートマシンに MPI タスクを起動する．LAM では事前に`lamboot`というプログラムでセッションを開始し，`mpirun`で MPI タスクをスタートさせる．mpiJava では`mpirun`へのラッパースクリプト`prunjava`を用いる．MPIJ では事前に DOGMA システムを起動しておく．しかし，Java のコアライブラリにはリモートマシンにプロセスを起動する API は用意されていない．そのため本実装では UNIX 系オペレーティングシステムにほぼ標準的に用意されている`rsh`を利用している．これは本実装における唯一の非 Java 部分である．また，`mpirun`のような起動プログラムはシェルスクリプトなどの Java 以外の言語で記述される．我々は Java 以外の言語の使用を極力避けたかった．それは Java の大きな特徴であるポータビリティを維持するためである．本実装ではクラスライブラリ自身にリモートマシンに MPI タスクを起動する機構を組み込んであるため，MPI タスクの起動に`mpirun`のような起動プログラムを必要としない．Java Development Kit（JDK）における標準的な Java ランタイムの起動方法と，本実装の起動方法を次に示す．

JDK
```
>java class
```

Our MPI Implementation
```
>java -Dnp=number_of_task class
```

このように，JDK における標準的な Java ランタイムの起動方法に，MPI タスクの数をプロパティとして与えるのである．この起動方法を実装するにあたり，MPI の初期化メソッドのシグネチャが Java-MPI バインディングとは異なるものとならざるを得なかった．我々の実装における MPI.init メソッドのシグネチャを次に示す．

Java-MPI Binding
```
void init(  String[] args )
```

Our MPI Implementation
```
void init(  String name_of_class,
            String[] args )
```

Java-MPI バインディングと異なるのは，第一引数に起動するクラス名を与えることである．本実装は MPI.init メソッドの中でリモートマシンに MPI タスクを起動するため，起動するクラスの情報を MPI の初期化メソッドに与える必要があった．C では main 関数に与えられる引数には，起動されたプログラムの名前が含まれるが，Java では main メソッドに与えられる引数には起動されたクラス名が含まれない．そのため，MPI.init メソッドにはクラス名を引数として与えている．

3 オブジェクト送受信

3.1 自然ソルバの並列化

MPI はプリミティブ型の配列を高速に送受信するのに都合がよい仕様となっている．これまで，並列アルゴリズムの研究の主流は有限要素法などの数値ソルバを並列化することであった．数値ソルバの並列化では並列タスク間のメッセージは実数の配列であり，その配列長も問題サイズと並列度で決定され，アプリケーション実行中は変化しない場合が多い．よって，数値ソルバの並列化コードに使用されるライブラリには「プリミティブ型」の「固定長の配列」を高速に送受信できることが求められるのである．MPI はこの目的に最適化された仕様であると考えられる．

しかし，近年は遺伝的アルゴリズムやシミュレーテッドアニーリングなどの自然ソルバを並列化する研究が盛んに行われている [19, 20, 21, 22]．これらの自然ソルバの並列化では，並列タスク間のメッセージは「異なるデータ型の複合体」の「不定長の配列」となる場合が多い．このようなコードを MPI で記述するとメッセージ送受信部分の記述量が膨大になり，バグも混入しやすくなる．異なるデータ型の複合体をメッセージとすることは，データ型として`MPI.OBJECT`を実装し，シリアライズ可能なオブジェクトを送受信できるように実装することで解決することができる．しかし，MPI

ではメッセージの受信側はあらかじめメッセージ長を知っている必要があるため，不定長の配列の送受信はコーディング量が増える．

3.2 オブジェクト送受信への最適化

3.2.1 拡張メソッド

第 2.1 節で述べたように，Java では配列はオブジェクトとして扱われる．配列は自分自身のサイズを知っているので，配列をオブジェクトとして送受信すれば，受信側はあらかじめ配列長を知っておく必要はない．そのため，メッセージ長を気にする必要のないオブジェクトの送受信に最適化したメソッドがあれば，プログラミングが簡単になる．我々はComm クラスに次に示すメソッドを追加した．

```
void sendObject(Object buf, int dest, int tag)
Object recvObject(int src, int tag, Status status)
```

送信メソッドsendObject は引数buf に与えられたシリアライズ可能なオブジェクトをそのままマーシャリングする．標準送信メソッドのように配列である必要はない．また，受信メソッドrecvObject はsendObject によって送信されてきたオブジェクトを受信し，そのリファレンスを戻り値として返す．このメソッドにより，ユーザ定義のクラスのインスタンスの配列であっても，ユーザが配列長を管理する必要なしに送受信することが可能である．

3.2.2 拡張メソッドの実装

拡張メソッドのメッセージバッファの扱いは，第 2.2 節で示したものとは異なっている．標準送信メソッドの場合は図 1 に示したように，メッセージバッファの指定された部分配列をコピーし，その部分配列をマーシャリングする．拡張送信メソッドの実装を図 3 に示す．拡張送信メソッドは与えられたオブジェクトをそのまま送信するため，配列の場合であっても部分配列をコピーするという操作は行わない．そのため，配列の全要素を送信する場合は標準送信メソッドよりも高速である．

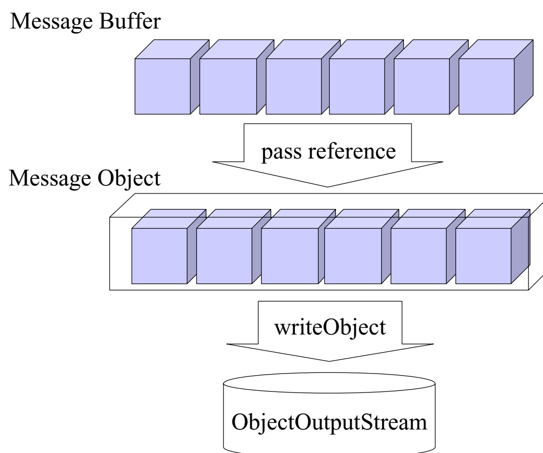

図 3: Marshalling in extended method

拡張受信メソッドの実装を図 4 に示す．標準受信メソッドは図 2 に示したように，受信した部分配列をメッセージバッファにコピーしていたが，拡張受信メソッドは受信したオブジェクトをそのまま戻り値とする．そのため，メッセージが配列であった場合，全要素を受信するなら標準受信メソッドよりも高速である．

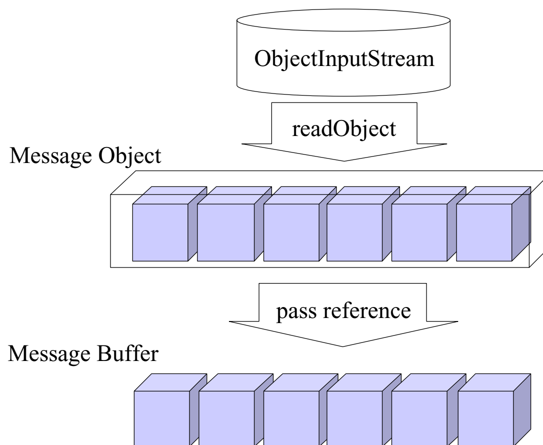

図 4: Unmarshalling in extended method

4 Java による MPI 実装の評価

4.1 実験環境

本節では我々の MPI の実装の通信性能と，コーディング量の評価を行う．評価を行う並列計算環境は表 2 に示すマシンをスイッチングハブで接続したクラスタである．

表 2: Experiment environment

CPU	Pentium3 500MHz (2way SMP)
Memory	128MBytes
Network	Ethernet 100BASE-TX
OS	Linux 2.2.12
JDK	Blackdown JDK1.2.2 RC3

4.2 通信性能

２台のマシンの間を，メッセージサイズを変化させながらその往復時間を測定し，メッセージサイズとバンド幅の関係を測定する．図 5 は標準送受信メソッドにメッセージとして`byte`，`int` そして`double` の配列を与え，その配列のサイズをバイトに換算した場合のメッセージバンド幅である．メッセージの種類にかかわらず，メッセージサイズが 1 KByte 付近のバンド幅が極端に落ち込んでおり，それ以上のメッセージサイズでもバンド幅に大きな変動がある．現在はこの原因は不明であり，調査を行っている．

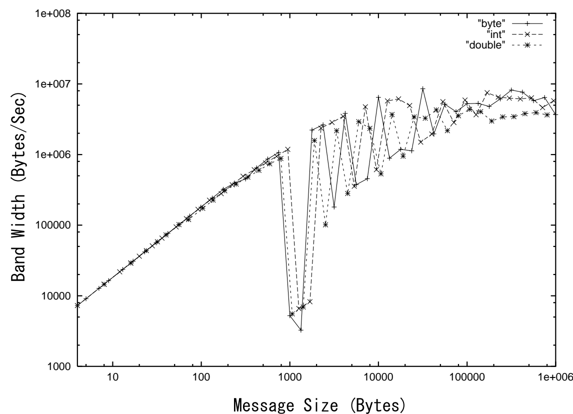

図 5: Band width of byte, int and double

また，図 6 はメッセージを`byte` の配列とし，標準送受信メソッドと，我々が提案する拡張送受信メソッドのバンド幅を比較している．この図から，拡張送受信メソッドのほうがわずかに高いバンド幅を示すことが読み取れる．これは，標準送受信メソッドは必ず配列のコピーが行われているのに対し，拡張送受信メソッドは与えられたオブジェクトのコピーを生成せず，そのままストリームに渡すためであると考えられる．

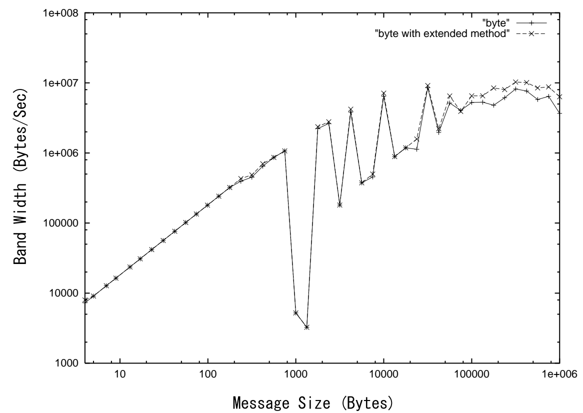

図 6: Band width of byte: standard method and extended method

4.3 コーディング量の削減

本実装はシリアライズ可能なオブジェクトをメッセージとすることが可能である．ここではユーザ定義のクラスのインスタンスをメッセージとするコードを，`MPI.OBJECT` による場合とプリミティブ型のみによる場合，そして第 3.2.1 節で定義した拡張メソッドを用いた場合でコードの記述量の減少を見る．

ここでは，次に示すクラス`Foo` のインスタンスの配列`fooArray` を送受信することを想定する．

```
class Foo implements Serializable {
   private int[] intArray;
          ⋮
}
```

クラス Foo はプライベートメンバ変数として`int` の配列`intArray` を持っている．一般的に，Java ではメンバ変数は`private` 修飾子をつけて，他のクラスからのアクセスを禁止する．`fooArray` の要素数は m，`intArray` の要素数は n とする．`fooArray` をデータ型`MPI.OBJECT` を用いて送信すると次のようになる．

```
COMM_WORLD.send(fooArray, 0, m, MPI.OBJECT, dest, tag);
```

これに対応する受信側のソースコードは次のようになる．

```
COMM_WORLD.recv(fooArray, 0, m, MPI.OBJECT, src, tag);
```

また，拡張送信メソッドを用いて記述すると次のようになる．

```
COMM_WORLD.sendObject(fooArray, dest, tag);
```

これに対応する受信側のソースコードは次のようになる．

```
fooArray = (Foo[])COMM_WORLD.recvObject(src, tag);
```

上記のように，データ型MPI.OBJECTや拡張送受信メソッドを用いると，オブジェクト送受信のコードが極めて簡潔に記述可能となる．これを，データ型MPI.OBJECTや拡張送受信メソッドを用いずに実装した場合を考える．送信側はintArrayの値を取り出し，受信側はintArrayに値を代入しなければならない．intArrayはプライベートメンバであるため，クラスFoo以外からは直接アクセスできない．そのため，次に示すようなintArrayへのアクセスメソッドを用意しなければならない．

```
void set(int[] intArray) {
    this.intArray = intArray;
}

int[] get() {
    return intArray;
}
```

そして，fooArrayの要素ひとつずつにアクセスし，intArrayを取り出して送信しなければならない．そのソースコードを次に示す．

```
for(int i = 0; i < m; i++) {
    COMM_WORLD.send(fooArray[i].get(), 0, n,
            MPI.INT, dest, tag);
}
```

これに対応する受信側のソースコードは次のようになる．

```
for(int i = 0; i < m; i++) {
    int[] intArray = new int[n];
    COMM_WORLD.recv(intArray, 0, n, MPI.INT, dest, tag);
    fooArray[i].set(intArray);
}
```

受信側はfooArray の要素ひとつずつのためにintArrayの受信バッファを毎回確保しなければならない．そして，受信したintArrayをfooArrayの要素にセットしていく必要がある．

以上のように，データ型MPI.OBJECTや拡張送受信メソッドを用いずに，プライベートメンバ変数を含んだユーザ定義のオブジェクトを送受信するにはアクセスメソッドを用意しなければならず，クラス定義の変更が必要となる場合もありうる．また，送受信部分のコード記述も複雑になる傾向がある．

5 結論

本研究はJavaによってMPIを実装した．我々の実装はデータ型MPI.OBJECTにより任意のシリアライズ可能なオブジェクトの配列を送受信できる．また，拡張送受信メソッドを実装した．拡張送受信メソッドは任意のシリアライズ可能なオブジェクトを送受信することができ，そのメッセージサイズはプログラマが管理する必要はない．本研究によって確認されたことを以下に示す．

- 我々の実装は任意のシリアライズ可能なオブジェクトの配列をメッセージとすることができる．よってユーザが定義したクラスのインスタンスをメッセージとすることが可能であり，メッセージ送受信のコード記述量を大きく減らすことができる．

- MPIの送受信メソッドのシグネチャは数値ソルバの並列化コードを記述するには適しているが，自然ソルバの並列化コードを記述するには適していない．我々はオブジェクトの送受信に最適化したシグネチャを持つ送受信メソッドを実装した．このメソッドを用いることでオブジェクトを送受信するコードの記述量を減少させ，より自然なセマンティクスを持つ記述を行うことが可能となった．

- 我々の実装はMPIタスクの起動をMPI.initメソッドの中で行うため，Javaの起動をJDKのスタイルを崩すことなく行うことができる．よってJavaの大きな特徴であるポータビリティを維持している．しかし，この方法を実装するためにはMPI.initメソッドのシグネチャをJava-MPIバインディングとは異なるものにしなければならない．

参考文献

[1] S.Matsuoka, H.Ogawa, K.Shimura, Y.Kimura, K.Hotta, H.Takagi: "OpenJIT —A Reflective Java JIT Compiler", Proceedings of OOPSLA '98 Workshop on Reflective Programming in C++ and Java, pp.16-20, November, 1998.

[2] K.Ishizaki, M.Kawahito, T.Yasue, M.Takeuchi, T.Ogasawara, T.Suganuma, T.Onodera, H.Komatsu and T.Nakatani: "Design, Implementation, and Evaluation of Optimizations in a Just-In-Time Compiler", ACM 1999 Java Grande Conference, 1999.

[3] A.Azevedo, A.Nicolau and J.Hummel: "Java Annotation-aware Just-in-Time (AJIT) Compilation System", ACM 1999 Java Grande Conference, 1999.

[4] R.F.Boisvert, J.J.Dongarra, R.Pozo, K.A.Remington, and G.W.Stewart: "Developing numerical libraries in Java", ACM 1999 Java Grande Conference, 1999.

[5] P.Wu, S.Midkiff, J.Moreira and M.Gupta: "Efficient Support for Complex Numbers in Java", ACM 1999 Java Grande Conference, 1999.

[6] F.Breg, S.Diwan, J.Villacis, J.Balasubramanian, E.Akman and D.Gannon: "RMI Performance and Object Model Interoperability: Experiments with Java/HPC++", Concurrency: Practice and Experience, volume 10, 1998.

[7] A.Gokhale and D.C.Schmidt: "Measuring the Performance of Communication Middleware on High-Speed Networks", SIGCOMM Conference, ACM 1996, 1996.

[8] 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.

[9] Message Passing Interface Forum: "MPI: A message-passing interface standard", International Journal of Supercomputer Applications, 1994.

[10] Message Passing Interface Forum: "MPI-2: Extension to the message passing interface", Technical report, University of Tennessee, July 1997.

[11] A.Ferrari: "JPVM: network parallel computing in Java", ACM 1998 Workshop on Java for High-Performance Network Computing, 1998.

[12] G.Judd, M.Clement, Q.Snell: "Design Issues for Efficient Implementation of MPI in Java", ACM 1999 Java Grande Conference, 1999.

[13] G.Judd, M.Clement, and Q.Snell: "DOGMA: Distributed Object Group Management Architecture", ACM 1998 Workshop on Java for High-Performance Network Computing, 1998.

[14] M.Baker, B.Carpenter, G.Fox, S.H.Ko, and S.Lim: "mpiJava: An Object-Oriented Java interface to MPI", International Workshop on Java for Parallel and Distributed Computing, 1999.

[15] W.D.Gropp, E.Lusk: "User's Guide for mpich, a Portable Implementation of MPI", Mathematics and Computer Science Division, Argonne National Laboratory, ANL-96/6, 1996.

[16] S.Mintchev, V.Getov: "Towards Portable Message Passing in Java: Binding MPI", Proceedings of EuroPVM-MPI, 1997.

[17] 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.

[18] G.D.Burns, R.B.Daoud, J.R.Vaigl: "LAM: An Open Cluster Environment for MPI", Supercomputing Symposium '94, Toronto, Canada, June 1994

[19] R.Tanse："Distributed Genetic Algorithms", Proc.3rd International Conf. Genetic Algorithms. Morgan Kaufmann. P434〜P439,1989.

[20] T.C.Belding："The Distributed Genetic Algorithm Revisited", PROCEEDING OF THE SIXTH INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS P114〜P121, 1995

[21] E.H.L.Arts and J.H.M.Korst: "Simulated Annealing and Boltzmann machines", Wily, Chichester, 1989

[22] K.Holmqvist, A.Migdalas, and P.M.Pardalos, : Parallelized Heuristics for Combinatorial Search, in Parallel Computing in Optimization, A.Migdalas, et al. eds, Kluwar Academic Publishers, p. 269, 1997.