プログラミング 2−11
（1995. 8. 24）

C++ をベースとしたオブジェクト並列プログラミング言語 MAPPLE

小野 剛　今崎 憲児　福見 幸一　天野 浩文　牧之内 顕文
(九州大学 工学部 情報工学科)

概要

近年，プログラミングやデバッグが容易である等の理由から，様々なデータ並列言語が提案されている．データ並列言語はデータ構造として配列を利用しているため，数値計算などの応用分野では有効である．しかし，データベースやCADなどのデータインテンシブな応用分野では，データの静的配置などの様々な問題が存在する．本論文では，これらの問題点を指摘し，現在設計・開発している超オブジェクト並列プログラミング言語MAPPLEの設計思想，言語機能および実装方法について述べる．

MAPPLE:
an Object Parallel Programming Language based on C++

Tsuyoshi Ono , Kenji Imasaki , Koichi Fukumi ,
Hirofumi Amano and Akifumi Makinouchi

Abstract

Data-parallel languages are effective for numerical computation since they use a matrix as a basic unit of parallelism. However there are some problems for data-intensive applications such as database systems or CAD. In this paper, we present shortcomings of data-paralle languages such as static data allocation, and give the design concept, language facility and implementation of a new massively object-parallel programming language, called MAPPLE now under development.

1 はじめに

本論文ではC++をベースとしたオブジェクト並列プログラミング言語 MAPPLE (MAssively object-Parallel Programming LanguagE, 旧称INADA/MPP) の設計思想，言語機能および実装方法について述べる．

近年，プログラミングやデバッグが容易であるなどの理由から，データ並列に基づいた様々な言語 (Data-parallel-C, C*, NCX, HPF など) が提案されている．データ並列言語はデータ構造として配列を利用しているため，配列の添字を使った参照が利用できるなど，数値計算の分野では有効であると思われる．

しかし，データベース，CAE，CAD，CAM などのデータインテンシブな応用分野では複雑な構造のデータを大量にリアルタイムで処理するため，データ並列言語の次のような点が問題となる．

- 配列の要素の型が数値や文字列などの基本的なデータ型に限定される．
- コンパイル時に並列に処理するデータの個数が確定していなければならない．
- 実プロセッサへのマッピングがコンパイル時に決定している．

例えば，オブジェクト指向データベースではリストなどの集合型を扱うため，基本的なデータ型しか用いることができないデータ並列言語ではデータの表現が難しい．また，時間によって処理すべきデータの個数が変動する場合，データの個数や実プロセッサへのマッピングが静的に決定されるような言語では対処しづらいと思われる．

現在我々が設計開発しているオブジェクト並列プログラミング言語MAPPLEはこれらの問題を解決し，数値計算だけでなくデータベースやCADなどの分野でも利用できる言語を目指している．

従って，MAPPLEには，

- 並列処理の対象として，数値や文字列などの基本的なデータ型以外に，データの集まり (collection) としての集合型も扱うことができる．
- 実行時にオブジェクトを動的にプロセッサに割り当てることが可能である．
- オブジェクトの個数の変動に対処できる．

などの特徴がある．

本論文の構成は次の通りである．まず，2節でMAPPLEの設計思想を述べる．次に3節で言語機能を例題を用いて説明する．4節でそれらの機能がどのように実現されているかを述べ，5節で関連研究について話す．最後に6節でまとめと今後の課題を述べる．

2 設計思想

MAPPLEはC++を最小限拡張したオブジェクト並列プログラミング言語である．ベースとなる言語にC++を選んだ理由は，C++がオブジェクト指向に対応しており，しかも最近ではかなり普及しているという点からである．また，オブジェクト並列を採用したのは，並列に処理すべきデータをオブジェクトと考えた方がデータベースやCADなどの応用分野で都合がいいと判断したからである．

オブジェクト並列はデータ並列を拡張した概念である．オブジェクト並列ではオブジェクト (データ構造とそれを操作するための関数 (メソッド) を合わせ持ったもの) を並列計算機の各PE (Processing Element) に分散させ，メソッドを1つずつ同期を取りながら並列に実行する．

また，最近の並列計算機ではブロードキャスト専用の高速ネットワークが整備されているため，MAPPLEにおけるプロセッサ間の通信ではブロードキャストを多用している．従って，MAPPLEではオブジェクトのメソッドを並列に実行させる場合でもブロードキャストメッセージを用いる (図1参照)．

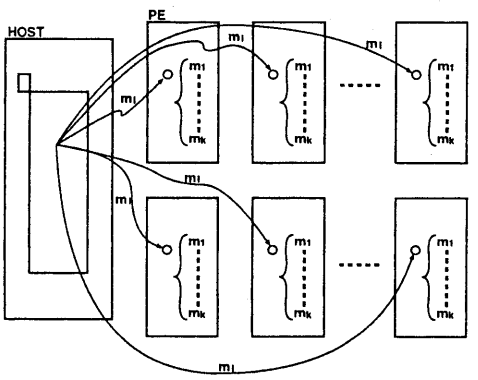

図 1: MAPPLEにおけるメソッドの並列実行

MAPPLEではオブジェクトを集合オブジェクト (Set Object) とそうでない素オブジェクト (Atomic Object) に分ける．集合オブジェクトはデータの集まり (collection) であり，データ構造として“集合”のインターフェイスを備えていなければならない．集合オブジェクトの実装はリスト，配列，ハッシュなどユーザが必要に応じて定義する．集合オブジェクトの要素を要素オブジェクト (Element Object) と呼ぶ．要素オブジェクトは素オブジェクトでも集合オブジェクトでもよい．

集合オブジェクトや素オブジェクトはある1つのPE上に生成されるため，複数のPEをまたがることはできない．しかし，PE上のオブジェクトのメソッドを

並列に実行するためには，各 PE 上のオブジェクトをまとめて 1 つの集合と見る必要がある．MAPPLE ではこの問題に対し，“和集合”という概念を導入することで対処している．和集合オブジェクト (Union Object) は各 PE 上の同一の型の集合オブジェクトまたは素オブジェクトを管理するオブジェクトである．従って，和集合オブジェクトは集合オブジェクトの特別なものと考えることができる．

これらのオブジェクトを模式的に表すと図 2のようになる．また，上記のオブジェクト以外にシステム定義のためのオブジェクト (VHMO) がある．

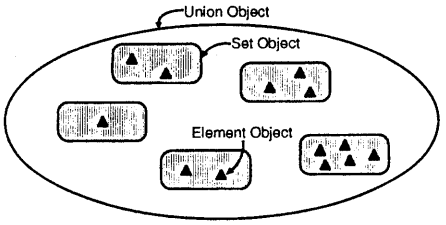

図 2: MAPPLE におけるオブジェクト

3 MAPPLE の言語機能

3.1 揮発ヒープ管理オブジェクト

揮発ヒープ管理オブジェクト (Volatile Heap Management Object,VHMO) とは，PE 上の揮発ヒープ領域およびその上に生成されたオブジェクトを管理するためのシステムオブジェクトである．VHMO は各 PE 上に 1 つずつ存在する (図 3参照)．

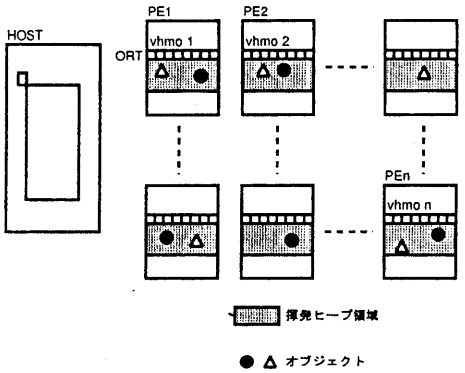

図 3: 揮発ヒープ管理オブジェクト

VMHO が行っている仕事は次の 3 つである．

- 自分が管理している揮発ヒープ領域にオブジェクトを生成する．
- 揮発ヒープ領域上のオブジェクトを消去する．
- 自分が管理しているオブジェクト宛に送られてきたメッセージを受け取り，メッセージの内容を解釈した後，オブジェクトにメソッドを実行させる．

MAPPLE はターゲットマシンとして 2 次元メッシュ型並列計算機を考えており，ユーザがメインプログラムの先頭で，

```
VHMO Vhmo(8,8);
```

と記述すると，8 × 8 の全 PE 上に VHMO が生成され，ホスト上の Vhmo に OID が返される．また，Vhmo[5] と書くと，PE5 上の VHMO のリファレンス (OID) を得ることができる．

3.2 集合オブジェクト・要素オブジェクト

集合オブジェクトは PE 上の同一の型のオブジェクトを管理するオブジェクトである (図 4参照)．

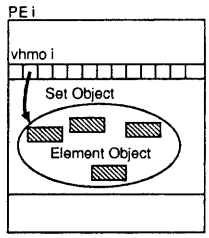

図 4: 集合オブジェクトおよび要素オブジェクト

従って，

```
SET<ELEM>* Set;
Set = new(Vhmo[5])SET<ELEM>();
```

と記述すると，ELEM 型の要素オブジェクトを持つ集合オブジェクトが PE5 上に生成され，ホスト上の変数 Set に OID が返される．要素オブジェクトは，

```
ELEM* Elem;
Elem = new(Set)ELEM(10,71);
```

と書くことで生成することができる．この場合，集合オブジェクト Set 内に要素オブジェクトが生成され，ホスト上の変数 Elem に OID が返される．また，要素オブジェクトのリファレンスが不要な場合は，

```
new(Set)ELEM(10,71);
```

と記述する．この場合，ホストにOIDが返されないため，メッセージの本数が生成する要素オブジェクトの数だけ減少する．

上記の例から分かるように，オブジェクトは実行時に動的に生成することが可能である．

3.3 和集合オブジェクト

1つの和集合オブジェクトはPEに分散された同一の型の素オブジェクトまたは集合オブジェクトを1種類だけ管理する．和集合オブジェクトを1つ生成すると，全PE上に集合オブジェクトが1つずつ生成され，概念的にPE上のオブジェクトを要素に持つ1つの巨大な集合オブジェクトとみなすことができる(図5参照)．例えば，メインプログラム中で，

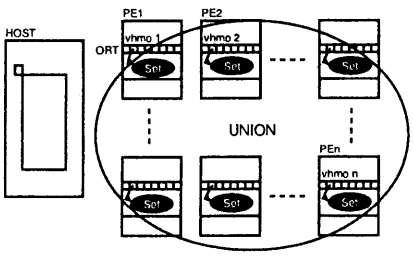

図 5: 和集合オブジェクトの概念図

```
UNION<SET<ELEM>>* Union;
Union = new UNION<SET<ELEM>>();
```

と記述すると，要素オブジェクトの型がELEMである集合オブジェクトを全PE上に生成し，ホスト上の和集合オブジェクトUnionに各集合オブジェクトのリファレンス(OID)を返す．

また，和集合オブジェクトが管理しているオブジェクトのリファレンスを得るため，次のメソッドが用意されている．

- `ref(int i)` … PEi上のオブジェクトのリファレンスを得る

従って，要素オブジェクトを100個round-robin方式で生成するには，メインプログラム中で，

```
for (i = 0; i < 100; i++)
  new(Union->ref(i%64))ELEM(i,10);
```

と書けばよい(但し，PEの個数は64とする)．

3.4 メソッドの並列実行

MAPPLEでは階層的な並列実行が可能である．具体的には，和集合オブジェクトの要素オブジェクトおよび，それが集合オブジェクトならばその集合オブジェクトの要素オブジェクトのメソッドを並列に実行することができる．和集合オブジェクトは並列に処理すべきオブジェクトが入った巨大な容器とみなすことができ，データ並列における配列全体のような役割をしている．

並列実行には`for all`文を用いる．構文は

```
for all [ type temp-val ]+ in union do {
    [ temp-val.method();]+
}
```

となる．但し，

- *type:* 並列に実行するオブジェクトの型
- *temp-val:* オブジェクトを表すテンポラリ変数
- *union:* 和集合オブジェクトへのポインタ
- *method:* 並列に実行するメソッド名

を表す．また，`for all`文ではメソッドを1つ実行する毎に全PEで同期が取られる．

素オブジェクトまたは集合オブジェクトのメソッドを並列に実行する場合，各オブジェクトは図6のように見える．この場合，和集合オブジェクトUnion内の

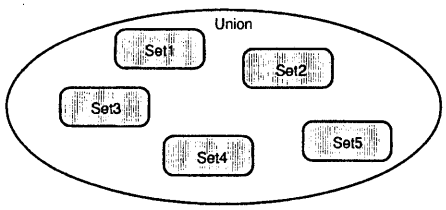

図 6: ユーザの立場から見た集合オブジェクト

集合オブジェクトSet1,Set2,...,Set5はそれぞれメソッドを並列に実行するため，自分が管理している要素オブジェクトは見えるが，他の集合オブジェクトが管理している要素オブジェクトは見えなくなる．

従って，例えば集合オブジェクトのメソッドを並列に実行する場合は，

```
for all SET<ELEM> s in Union do{
    s.method1();
}
```

と記述する．素オブジェクトの場合も同様である．

要素オブジェクトのメソッドを並列に実行する場合，各オブジェクトは図7のように見える．この場合，和集合オブジェクトUnion内の全要素オブジェクトe1,e2,...,e12のメソッドが並列に実行されるので，各々を管理している集合オブジェクトは意識しなくて

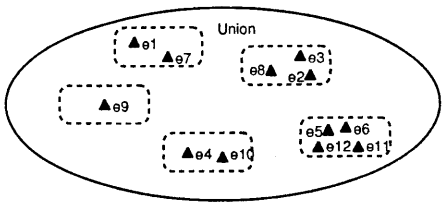

図 7: ユーザの立場から見た要素オブジェクト

よい．この要素オブジェクトによるメソッドの並列実行はデータ並列による演算の並列実行という形態を処理すべきデータのトポロジーや個数にとらわれなくてもいいように拡張したものと言うことができる．

従って，

```
for all ELEM e in Union do{
    e.methodA();
}
```

と記述すると，和集合オブジェクト Union 内の要素オブジェクトのメソッド methodA が並列に実行される．

並列に実行するメソッドに戻り値があると，その戻り値を何処に集めるかによって記述の仕方が変わる．

• 素オブジェクトの場合

各 PE 上の全ての素オブジェクトがメソッドを実行し戻り値を返すと，呼び出し側では戻り値の集合ができる．従って，ユーザはあらかじめ戻り値を入れるための集合オブジェクトを生成し，for all 文内で戻り値のオブジェクトを集合オブジェクト内に生成しなければならない．

例えば，メソッド methodA() が int 型の戻り値を返すとすると，

```
SET<int>* Rset;
Rset = new SET<int>();
for all ELEM e in Union do{
    new(Rset)int(e.methodA());
}
```

となる．

• 集合オブジェクトの場合

ある条件を満たす要素オブジェクトを検索してくるような集合オブジェクトのメソッドを並列に実行させると，集合オブジェクトが検索結果として複数の値を返す可能性がある．

この場合，検索結果をメソッド内部で 1 つの大きな入れ物に詰め込んで呼び出し側に返すという方法が考えられるが，この方法はあまり実用的ではない．というのは，通常検索して得られた結果に対して別のメソッドを並列に実行するということが考えられるからである．

MAPPLE ではこの問題に対し，ユーザがあらかじめ和集合オブジェクトを生成しておくことで対処している．具体的には，ユーザが集合オブジェクトを要素に持つ和集合オブジェクトを生成し，メソッド内でその和集合オブジェクトが管理している同一 PE 上の集合オブジェクトに検索して得られたオブジェクトを挿入する．

例えば，年令が 40 歳の従業員を探し出し，その人達の給料を 10 万アップさせるには，

```
(メインプログラム)
UNION<SET<EMP>>* UnionA;// 従業員のデータベース
UNION<SET<EMP>>* UnionB;// 検索結果を入れる和
                           集合オブジェクト
UnionB = new UNION<EMP>();
for all SET<EMP> s in UnionA do{
    s.search(UnionB,40);
}
for all EMP e in UnionB do{
    e.up_salary(100000);
}
```

とプログラミングすればよい．

3.5 リダクション

要素オブジェクトのメンバ変数 a の総和を求めるなどのリダクション処理を行うにはリダクション関数を用いればよい．リダクション関数には，

- void sum_reduction(T* sum, T a)
  … a の総和を求めて， sum に代入する．
- void max_reduction(T* max, T a)
  … a の最大値を求めて， max に代入する．
- void min_reduction(T* min, T a)
  … a の最小値を求めて， min に代入する．

  但し， T は変数の型とする．

の 3 つを用意している．また，必要に応じて別のリダクション関数をユーザが追加できる．

3.6 オブジェクト間の通信

これまで，メソッドが単に並列に実行する場合について述べてきたが，応用によってはオブジェクト間で通信を行う必要があるものも考えられる．

オブジェクトが他のオブジェクトと通信をするためには並列に実行しているメソッドの内部で for all 文を実行すればよい．通信を行うオブジェクトと通信相手のオブジェクトの組合せは素オブジェクト，集合オブジェクト，要素オブジェクトのどれでもよい．ま

た，異なる和集合オブジェクトが管理しているオブジェクトとも通信できる．

例えば，和集合オブジェクトUnion1の要素オブジェクトが和集合オブジェクトUnion2の要素オブジェクトと通信を行う場合，

```
(メインプログラム)
for all ELEM e1 in Union1 do{
    e1.method1(Union2);
}

(要素オブジェクトELEMの定義)
void ELEM::method1(UNION<SET<ELEM>>* Union2){
  SET<int>* Set1 = new SET<int>();
  for all ELEM e2 in Union2 do{
      new(Set1)int(e2.method2());
  }
}
```

というように，メインプログラム内でメソッドの引数に通信相手の和集合オブジェクトへのポインタUnion2を渡せばよい．但し，Union2の要素オブジェクトが戻り値を返す場合，戻り値を入れるための集合はあらかじめ生成しておく必要がある．

4 MAPPLEの実装

この節では前節で述べた機能の実装について述べる．

4.1 VHMOの実装

VHMOはオブジェクト参照テーブル(Object Reference Table,ORT)を使って揮発ヒープ領域上のオブジェクトを管理している(図8参照)．また，VHMOに管理されているオブジェクトは“PE番号＋ORTエントリ番号”から成るOIDを持っている．

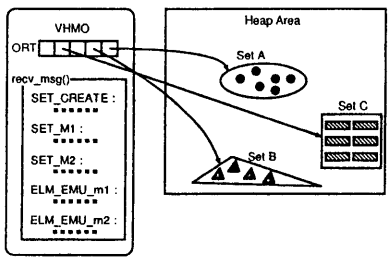

図 8: VHMOの実装

VHMO内には管理しているオブジェクトの型とメソッド名が登録されており，ホストや他のPEからメッセージが届くと以下の仕事を行う(recv_msg())．

1. VHMOがメッセージを受け取る．
2. メッセージから並列に実行させるオブジェクトの型とメソッド名を判断する．
3. メッセージ中のORTエントリ番号からオブジェクトのアドレスを求める．
4. メッセージからメソッドの引数を取り出す．
5. 3.で求めたオブジェクトに引数を渡し，メソッドを実行させる．

ホスト上には各PE上のVHMOのOID(実質的にはPE番号である)を管理するマスタヒープオブジェクト(Master Heap Object,MHO)が存在する．MHOはクラス定義内で[,]を多重定義しており，前述のようにVhmo[5]と書くことでPE5上のVHMOへのリファレンス(OID)を返すことができる．

4.2 集合オブジェクトの実装

MAPPLEでは集合オブジェクトのプロトタイプを提供している．また，ユーザが効率などを考慮し，集合オブジェクトを定義し直すことも可能である．

プロトタイプでは集合オブジェクトが管理するオブジェクトの型に関する制限をなくすため，C++の機能であるtemplateを利用している．

また，要素オブジェクトのメソッドの並列実行を実現するために，あたかも要素オブジェクトのメソッド(例えばm1())が並列に実行されているかのようにエミュレートするメソッド(EMU_m1())をコンパイラがコンパイル時に集合オブジェクトのクラス定義に追加する．

4.3 和集合オブジェクトの実装

和集合オブジェクトは以下の手順で生成される．

1. 和集合オブジェクトがホスト上に生成されると同時に，全PEに和集合オブジェクトが管理するオブジェクト(素オブジェクトまたは集合オブジェクト)を生成するようにメッセージをブロードキャストする(図9参照)．
2. 各PEでオブジェクトを生成した後，そのオブジェクトのOIDをホストに送る(図10参照)．
3. ホストでは各PEから送り返されてきたオブジェクトのOIDを参照テーブルに入れて管理する(図11参照)．

従って，和集合オブジェクトのメソッドrefは参照テーブルを検索し，捜し出したオブジェクトのOIDを返すことで実現されている．また，for all文の際にも，参照テーブルからオブジェクトのOIDが検索される．

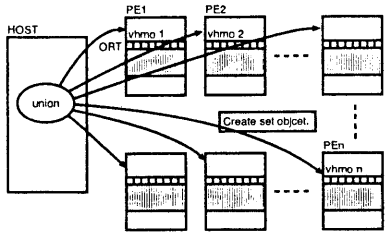

図 9: 和集合オブジェクトの生成 -1

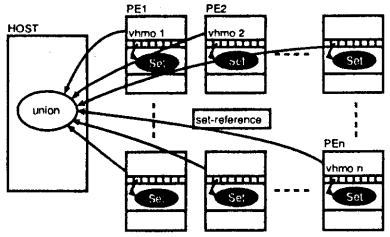

図 10: 和集合オブジェクトの生成 -2

4.4 for all 文の実現

要素オブジェクトの並列実行でメソッドに戻り値がある場合を説明する (図12参照).

1. ホストが和集合オブジェクトから集合オブジェクトの OID を検索し，各 PE に集合オブジェクトの OID，要素オブジェクトの型，メソッド名，引数をブロードキャストする．ホストは各 PE でのメソッドの実行が終了するまで待ち状態になる．

2. 各 PE 上で，VHMO の `recv_msg()`(3.1 参照) が起動し，集合オブジェクトが要素オブジェクトのメソッドのエミュレーション (3.2 参照) を実行する．エミュレーション終了後，要素オブジェクトのメソッドの戻り値をパックしてホストに送る．

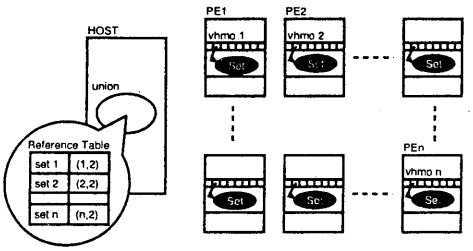

図 11: 和集合オブジェクトの生成 -3

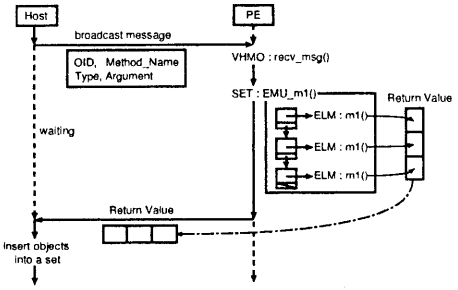

図 12: 並列実行の様子

3. 全 PE 上でのメソッドのエミュレーションが終了すると，ホストに制御が戻る．メソッドに戻り値がある場合，セルから送られてきたメッセージをアンパックして集合に挿入する．

4.5 リダクションの実現

リダクション関数は基本的に各 PE 内でローカルなリダクション結果を求め，PE 間で全体のリダクション結果を求める方法を取っているが，全体でリダクション結果を集める部分が，メソッドを並列に実行している場合と，オブジェクトが通信を行っている (メソッド内 `for all` 文) 場合で多少異なる．

メソッドが並列に実行されている時にリダクション関数が用いられた場合は，ターゲットマシンが提供しているリダクション関数を利用する方が効率が良いので，マシン固有のリダクション関数を用いて実装している．

メソッド内 `for all` 文の場合は，通信元に PE 毎に求めたリダクション結果を送り返し，通信元で全体のリダクション結果を求める．

また，リダクション結果が常に正しい値になることを保証するために，リダクションを行う前に必ず全 PE で同期が取られる．

4.6 オブジェクト間通信の実現

集合オブジェクトが通信を行う場合，`for all` 文が実行される時にホストが行っているように通信相手の OID，オブジェクトの型，メソッド名，引数をブロードキャストする．後の処理もホストが行っている部分が PE に代わるだけで同じである．

要素オブジェクトが通信を行う場合，集合オブジェクトが一旦メソッドのエミュレートを停止し，通信を行った後，再びエミュレートを再会する部分が異なるだけで，基本的にホストから `for all` 文が実行される時と変わらない．

4.7 メッセージのまとめ送り

並列処理では，如何にしてPE間の通信を減らし並列に処理する部分を増やすかが効率を上げるポイントである．これに対し，MAPPLEはホスト-PE間，PE-PE間のメッセージをまとめることで効率化を計っている．

ホスト-PE間,PE-PE間の通信には，メソッドの並列実行か，オブジェクト間の通信の際に，

1. 並列に実行するメソッドを起動させるとき
2. メソッドの戻り値を呼び出し側のホストもしくはPEに返すとき

の2種類がある．1のメッセージの内容は，

- オブジェクトのOID
- 並列に実行するオブジェクトの型とメソッド名
- メソッドの引数

である．メソッドを並列に実行する場合(ホストからPEへの通信)，メソッドの引数は，

- 和集合オブジェクトへのポインタ(オブジェクト間の通信がある場合)
- ホスト上の集合オブジェクトへのポインタ(メソッドに戻り値がある場合)
- モノ変数へのポインタ(リダクションの場合)
- モノ変数，定数

のどれかで，必ず全てのオブジェクトに同じ値が送られる．よって，ホストから全PEへのブロードキャストメッセージとすることでメッセージの本数を減らすことができる．

オブジェクト間で通信を行う場合も`for all`文が起動されるが，この場合は先ほどのように全てのオブジェクトでメソッドの引数が同じということはない．従って，通信をするオブジェクト間で1対1通信を行うことになり，要素オブジェクト間の通信では要素オブジェクトの個数の2乗に比例してメッセージが増加する．そこでMAPPLEでは各PEでメソッドの引数をまとめ，それにどのオブジェクトの引数かを表す情報を付加したメッセージをブロードキャストすることで対処している．また，メソッドの引数がない場合は各オブジェクトから同じメッセージが送られるので，各PEから1本ずつPEの個数分のメッセージがブロードキャストされる．これもメッセージのまとめ送りの1つである．

2の場合もメソッドの戻り値を各PEで1つにまとめ，呼び出し側のホストもしくはPEに送り返すことでメッセージの本数を減らしている．

5 おわりに

本論文ではデータ並列プログラミング言語をデータインテンシブな応用分野で用いる際に生じる問題点を指摘し，それらを解決するために現在設計・開発しているMAPPLEの設計思想，言語機能および実装方法について述べた．我々は既に様々な例題に対する実装の予備的評価を終えており[AI94]，現在はコンパイラを作成中である．

ターゲットマシンには富士通社製並列計算機AP1000を考えており，コンパイラもAP1000の並列処理用のC言語のライブラリ関数を含んだGNU C++のコードを出力する．コンパイラの内部では，集合オブジェクトが要素オブジェクトのメソッドをエミュレートするためのクラス定義の書き換えやメッセージのまとめ送りによる最適化などを行う予定である．

また，ヒープ領域を2次記憶上のファイルにマップし，それを管理する機能をVHMOに追加することで，INADA[AA93]と同様にMAPPLEでも永続オブジェクトが扱えるようになると思われる．従って，今後は永続データも扱えるように拡張していく予定である．

参考文献

[AA93] M. Aritsugi and H. Amano : *View in an Object-Oriented Persistent Programming Language*, Proc. of the International Symposium on Next Generation Database Systems and Their Applications, pp.18–25 (Sep. 1993).

[AI94] 天野, 今崎, 福見, 牧之内：オブジェクト並列プログラミング言語INADA/MPPの設計方針と予備的評価について，文部省重点領域研究第4回シンポジウム予稿集, 2-214–2-220 (Mar.1994).

[GL91] Gannon, D. and Lee, J. K.: *Object Oriented Parallism:pC++ Ideas and Experiments*, Proceedings of 1991 Japan Society for Parallel Processing, pp.13–23 (1991).

[KT94] 日下部，高橋，谷口，雨宮：超並列V言語とその実行方式，並列シンポジウムJSPP'94, pp.41–48 (May. 1994).

[YB91] T. Yoshinaga and T. Baba : *A Parallel Object-Oriented Language A-NETL and Its Programming Environment*, COMPSAC, pp.459–464 (1991).