プログラミング－言語・基礎・実践－研究報告　9－3
（1992．　10．　30）

並列計算機上のオブジェクト間の間接的通信の実現について

小中 裕喜　横田 隆史　瀬尾 和男

三菱電機 (株) 中央研究所

大域的かつ構造的に参照可能なオブジェクトを導入することにより、オブジェクト間の間接的な通信を可能とする通信モデルを提案する。本モデルは通信の抽象化、分散化、局所化、階層化を容易にし、オブジェクト間の動的な通信経路の設定をサポートしながら、疎結合並列計算機においても効率の良い実現を可能とするものである。本稿では本通信モデルと、その応用について述べる。また現在、本通信モデルを採用したプロトタイプ言語を Intel 社の iPSC/860 の上に試験的に実装しているが、その概要についても言及する。

Indirect Communication between Objects on a Parallel Machine

Hiroki Konaka, Takashi Yokota, Kazuo Seo

Central Research Lab., Mitsubishi Electric Corp.

1-1, Tsukaguchi-honmachi 8-chome, Amagasaki, Hyogo, 661, Japan

This paper describes a communication model that enables indirect communication between objects, by means of global objects accessed in a structured way. This model not only makes it easy to abstract, distribute, localize and stratify dynamic communication, but can be realized efficiently on a loosely-coupled multi-computer system. We are implementing a prototype language featuring this communication model on Intel iPSC/860. This paper shows the model and its applications, as well as the implementation.

1 はじめに

計算機に対する高性能化への要求は増大の一途をたどっている。VLSI技術の発達にともなうプロセッサ単体の高速化は、命令パイプラインの進化やキャッシュなどさまざまな技術の進展にも支えられて、前進を続けてきた。プロセッサの単体性能の向上と並行して、複数のプロセッサをネットワークで結合して同時に処理を行なわせる並列処理技術は、高性能化へのもうひとつの自然なアプローチである[7]。

並列処理マシンは制御の点からSIMDとMIMD、メモリの点から共有メモリ型と分散メモリ型に分類される。SIMDマシンは構成が比較的容易なため、CM-2[3]など商用に供されているものも多い。処理の単位が均一で規則的なデータ構造、通信を扱う場合には適しているが、逆にそのような種類の問題に適用分野が限られてくる。共有メモリ型マルチプロセッサはメモリへのアクセスがボトルネックとなって、スケーラビリティを持ち合わせない。従って処理の柔軟性、スケーラビリティという点で今後の大規模並列処理マシンの中で分散メモリ型マルチコンピュータの占める位置は大きなものとなるであろう。

分散メモリ型マルチコンピュータ上の並列処理は個々の要素プロセッサ（PE）上の計算とPE間の通信から成り立つが、このときプログラマにどの程度並列性と通信を意識させるかが、マルチコンピュータにおけるプログラミングモデル構築上の分岐点となる。

例えば、従来の逐次型言語によって記述されたプログラムから並列性を抽出してプログラムを分割し、各PEにマッピングして実行させるアプローチもある[6]。あるいはコヒーレントキャッシュを用いた仮想共有メモリを見せることもある。しかしながら、これらはプログラミングが比較的容易である一方、並列性が不必要に失われやすい。また実行効率を高めるためには負荷分散、データ分散、通信分散を効率的に行なわなければならない。このためには静的な解析やプロファイルなどで得られた情報をもとに最適化する技術が必要となるが、これらは一般的に困難である。

メッセージパッシングメカニズムをもったマルチスレッドプログラミングモデルもよく用いられる。特に、A-Net[1]やABCL/1[5]などの並列オブジェクト指向のアプローチは問題の対象領域を自然にマッピングできることを大きな特徴としている。

並列オブジェクト指向では、一般に複数のオブジェクトが互いにメッセージを交換しながら協調的に処理を進めていく。このとき、静的に通信相手が定まる場合には相手のアドレスを保持しておけばよいが、動的に変化する何らかの性質を共通にもつ相手と通信を行ないたい場合には、その通信経路の動的な設定が問題となる。

システムが放送手段を提供している場合は放送を通じて通信相手を動的に探索することも考えられるが、これは一般にコストが高い。さらにシステムが放送手段を提供していない場合には、通信する可能性のあるすべてのオブジェクトのアドレスを各オブジェクトがそれぞれ保持しなければならず、メモリの浪費を招く。

また動的な通信経路の設定を可能とするために、場とよばれるようなオブジェクトの環境を設定し、それを介した間接的な通信を行なうものもある。そのような例としてLinda[2]がある。Lindaでは Tuple Spaceという大域的な場におけるタプル/テンプレートマッチングによって通信を行なう。しかし、マルチコンピュータへの実装を考えた場合、Tuple Spaceをそのまま物理的に実現すれば、そのアクセスの集中が問題となる。

本研究では並列オブジェクト指向をベースとして、その上に大域的かつ構造化して参照可能な通信用オブジェクトを導入することにより、オブジェクト間の動的な通信経路の設定、柔軟な通信を可能とするプログラミングモデルを提案する。本プログラミングモデルは分散メモリ型マルチコンピュータを主なターゲットとし、プログラマビリティをできる限り損なわず、効率的な実行を実現することを目指している。しかしながら、他のタイプの並列処理マシンや、分散コンピューティング環境などへの適用も可能と考えている。

本稿ではまず2章で通信用オブジェクトとそれを用いた通信モデルについて説明する。3章ではIntel社のiPSC/860^{TM}[4]へのプロトタイプ言語の実装について言及する。4章では本プログラミングモデルの応用についてその概要を示す。

2 通信用オブジェクト

2.1 通信用オブジェクトとは

通信用オブジェクトは通信に関する柔軟な記述や通信路の動的な設定を可能とするために導入されたものであり、通常のオブジェクトとの相違点は次の通りである:

- 他のオブジェクト（他の通信用オブジェクトも含む）から大域的に参照される。
- 通信用オブジェクトに対する構造化参照が可能である。

各通信用オブジェクトはインデックスの組によって一意に指定される。そしてインデックスの次元によって定められた空間の中で構造化された集合体を構成する。

オブジェクト間では通常のメッセージパッシングの他に、通信用オブジェクトを介した通信が可能である。送信オブジェクトは通信用オブジェクトに対しメッセージを送出する。一方、受信オブジェクトは自分の受け取るべきメッセージを通信用オブジェクトからフェッチする。通信用オブジェクトは該当するメッセージがあれば、受信オブジェクトにそのメッセージを送出する。すなわち各通信用オブジェクトはLindaのTuple Spaceのようにふるまうよう記述される。ただし、パターンマッチング機能は必ずしも必要としない。また、送信用オブジェクト、受信用オブジェクトがどの通信用オブジェクトにアクセスするかは、オブジェクトの何らかの静的もしくは動的な性質から決定される。

以上が通信用オブジェクトを介した通信の基本的な形態である。しかしながら、通信用オブジェクトに単にメッセージのプールとしての機能をもたせるだけではなく、メッセージを他の通信用オブジェクトに転送したり、メッセージに局所的な統計処理などのデータ加工を施したりする機能を付加することによって、種々の形態・機能をもつ通信を実現できる汎用的な枠組を提供することが可能である。また、通信用オブジェクトの集合体を構造化することにより、場を表現することも可能である。通信用オブジェクトを介したさまざまな通信の形態を図1に示す。

通信用オブジェクトを介した通信は以下のような特徴を持つ：

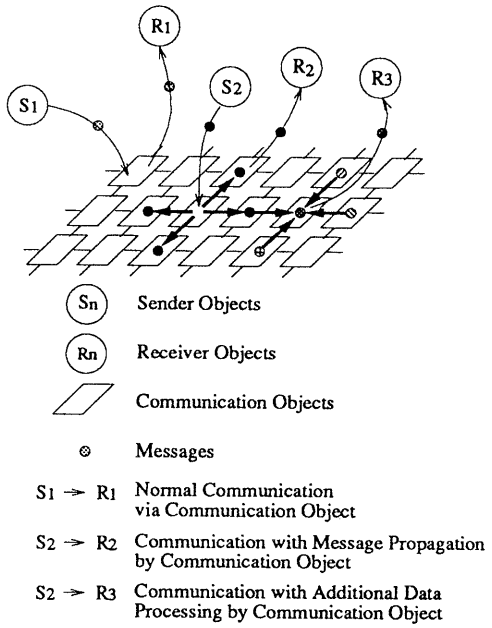

図 1: 通信用オブジェクトを介した通信

間接性 送信オブジェクトが受信オブジェクトを陽に知る必要はない。従ってオブジェクトの可動性が増す。オブジェクトがある通信用オブジェクトにアクセスすることが、その通信用オブジェクトとの直接的な関係を示し、複数のオブジェクトが同一の通信用オブジェクトを参照することが、それらの間の関連を示す。

非同期性 送信オブジェクトと受信オブジェクトの間で同期をとる必要がない。

階層性 通信用オブジェクト（またはその集合体）が、さらに他の（種類の）通信用オブジェクトを介して通信を行なうことが可能である。従って、複数種類の通信用オブジェクトの集合体を利用した階層化通信が可能である。

分散性 通信用オブジェクトの集合体で場を形成した場合、場へのアクセスが個々の通信用オブジェクトに分散される。

局所性 ある通信用オブジェクトを介した通信は他の

通信用オブジェクトを介した通信と独立であり、その処理は局所性をもつ。また、ある通信用オブジェクトを媒介としたオブジェクトのグループ化が可能であり、グループ内の通信が局所化される。

抽象化 通信用オブジェクトによるメッセージの伝達は通信に新たな性質（近傍性、拡散性、指向性など）を付与することになる。このような通信の性質はオブジェクトの動作の記述とは独立し、通信用オブジェクトの記述にカプセル化することが可能である。

またこれは必須ではないが、通信用オブジェクト間で一斉に同期をとったり、あるいは reduction を行なうなどといった、global operations をサポートすることは有用であると考えられる。

2.2 並列計算機上での実行

2.2.1 並列計算機へのマッピング

マルチコンピュータ上での実行を考える場合、通常のオブジェクトと通信用オブジェクトをそれぞれどのPEに割り当てるか、が大きな問題となる。通信用オブジェクトはその性質上、静的に生成・配置が可能な場合が多い。通信用オブジェクトの動的な生成・消滅・移動・分割・結合は別稿で論じるとして、ここでは通信の分散とコストの観点から、このような通信用オブジェクトの静的な配置を一般的に扱ってみることにする。

同種の複数の通信用オブジェクトの間には近傍性が定義される。例えば通信用オブジェクトが2次元のインデックス (X,Y) で指定されるとする。このとき例えば、2つの通信用オブジェクト $(X_1,Y_1),(X_2,Y_2)$ の距離を $|X_1-X_2|+|Y_1-Y_2|$ と定義してやることにより、近傍性を論じることが可能となる。同様にしてマルチコンピュータシステムにおいても、ネットワークの距離や最小遅延時間などによってPE間の近傍性を論じることが可能である。

このとき、近い通信用オブジェクトを近いPEに連続性をもってマッピングするのか、それとも逆に離れたPEに不連続にマッピングするのか、がプログラムの通信に関する効率に大きく影響する。

このようなマッピングの決定については考慮すべき要素が数多くあるが、ここでは通信用オブジェクトの参照特性に注目する。すなわち、各オブジェクトの通信用オブジェクトに対するアクセスパターン、オブジェクト全体としての通信用オブジェクトに対するアクセスパターン、通信用オブジェクト同士の通信パターンからマッピング方式をおおづかみに定める。

まず個々のオブジェクトの通信用オブジェクトに対するアクセスが局所性をもっている場合を考える。このとき、オブジェクト全体としても同様に局所的な通信用オブジェクトにアクセスが集中しやすい場合には、不連続なマッピングが望ましい。これによってアクセスが特定のPEに集中することを緩和できるからである。一方、オブジェクト全体としては通信用オブジェクト全体に対して比較的均質にアクセスが行なわれる場合には、連続的なマッピングが望ましい。これは、オブジェクトが頻繁にアクセスする通信用オブジェクト群の存在するPEの近傍にそのオブジェクトを配置しておくことによって、通信の分散/局所化を図ることが可能となるからである。

各オブジェクトの通信用オブジェクトに対するアクセスが局所性をもっていない場合でも、オブジェクト全体としての通信用オブジェクトに対するアクセスがいくつかの特定の範囲に局在化する場合には、不連続なマッピングが望ましい。

同種の通信用オブジェクト間の通信について考えると、近傍の通信用オブジェクト間でメッセージの伝搬や局所的な統計処理などを行なう場合は、連続的なマッピングが望ましい。

より一般的には連続的なものと不連続なものとの中間的なマッピングが考えられる。例えばある領域の通信用オブジェクトにオブジェクト全体のアクセスが集中しやすく、なおかつ近接した通信用オブジェクト間での通信も頻繁に行なわれる場合である。この場合は部分的な連続性を保ちつつ全体としては不連続にマッピングすることなどが考えられる。このようなマッピングの例を図2に示す。

またあるオブジェクトが特定の通信用オブジェクトに頻繁にアクセスすることがわかっている場合、その通信用オブジェクトの存在するPEの近傍にそのオブジェクトを移動するなど、オブジェクトの配置を通信用オブ

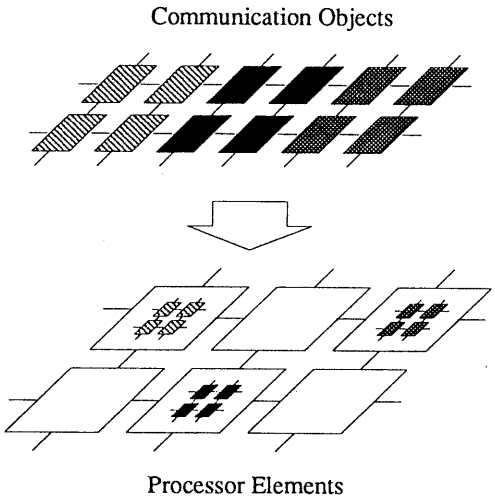

図 2: 通信用オブジェクトのマッピング例

ジェクトから逆に定めることも有用である。

2.2.2 通信の簡略化

オブジェクトがあまり頻繁に移動 / 消滅しない場合、通信用オブジェクトを介して設定された通信経路を用いて送信オブジェクトのアドレスを送り、以降の通信を通常のメッセージパッシングで行なうことが可能である。

また、オブジェクトが静的に生成されて移動しない場合には、コンパイル時に通信用オブジェクトを介した通信を通常のメッセージパッシングに置き換えればよい。このときは、通信用オブジェクトは単にプログラムの記述性の向上のために利用されるだけである。

3 プロトタイプ言語

以上に述べてきたモデルに基づき、C++ のサブセットをベースとしたプロトタイプ言語の処理系の作成を開始している。これは本通信モデルの有効性の確認、有用な通信・同期メカニズムの実現、最適化実行用戦略の抽出などのための実験環境の構築を目的としている。そのため、オブジェクトの PE へのマッピングなど、低レベルな記述も明示的に行なうようにしている。

ターゲットマシンは iPSC/860 であり、Perl による C++ へのトランスレータと、マルチスレッドライブラリ、および GNU C++ 処理系の組合せで実現する。以下にプロトタイプ言語の概要を示す。

3.1 オブジェクトと通信用オブジェクト

オブジェクトは動的に生成・消滅が可能であるが、通信用オブジェクト[1]はプログラム実行開始時に生成され、消滅しない。通常のオブジェクトクラスは `class` によって宣言され、通信用オブジェクトのクラスは `comm` によって宣言される。メソッドは C++ のメンバ関数と同様に定義する。

3.2 プログラムの流れ

プログラムはクラス宣言とメソッド定義、PE 内大域変数の宣言、通常の関数定義、およびタイプやマクロの定義などからなる。このうち、関数 `main()` と通信用オブジェクトクラスのメソッド `init()` だけはプログラムの実行において特別な意味をもつ。

プログラムの実行は通信用オブジェクトの生成から始まる。すなわち各 PE において、宣言されている通信用オブジェクトクラスのメソッド `init()` を呼び出す。各通信用オブジェクトクラスのメソッド `init()` には、その PE の上に存在すべき通信用オブジェクトを生成する手続きを記述しておく必要がある。

続いて全 PE 上の通信用オブジェクトの生成を待って、PE0 の上で `main()` を実行する。ここでは、必要に応じて通常のオブジェクトの生成や、適当なオブジェクトへの計算開始メッセージの送信などを行なう。

3.3 Gloval Pointer (GP)

グローバルポインタは全アドレス空間でオブジェクトを一意に指定するためのもので、メッセージの送信先などに用いられる。グローバルポインタであることを示す *classname*`GP` というタイプは自動的に定義される。

オブジェクトの GP(GPO)

PE 番号と PE 内ローカルアドレスの組

通信用オブジェクトの GP(GPC)

PE 番号と PE 内ローカルインデックスの組

[1]英語では Communication Object と仮に呼ぶことにする

3.4 Built-ins

3.4.1 オブジェクトの生成

通常のオブジェクトの生成は次のように行なう。

```
GPO create( Constructor, int pe );
```

一方通信用オブジェクトはメソッド init() の中で、次のように生成される。

```
GPC create( Constructor );
```

コンストラクタには通信用オブジェクトを指定するインデックスを引数として与える。

3.4.2 オブジェクトの消滅

```
void die(GPO gpo);
```

3.4.3 通信用オブジェクトの GP の獲得

各通信用オブジェクトクラスには at() というメソッドを用意して、他のオブジェクトがその通信用オブジェクトの GPC を獲得できるようにしておく必要がある。

```
GPC commclass::at(int index0,...);
```

at() を特に定義しなければデフォルトで次の定義が用いられる。

```
GPC commclass::at(int index0,...) {
      return (GPC){
            nidof(index0,...),
            oidof(index0,...)
      };
}
```

従ってプログラマは通常、与えられたインデックスの組をもつ通信用オブジェクトの存在する PE 番号を返す nidof() と、その通信用オブジェクトの PE 内ローカルインデックスを返す oidof() という 2 種類のメンバ関数を用意することによって、通信用オブジェクトの集合体の構造を規定する。

3.4.4 メッセージ送信

非同期通信

```
GP gp <- selector(parameters)
```

これは gp の指すオブジェクトに非同期にメッセージを送信する。

同期通信

```
[replies] = GP gp <- selector(parameters)
[[replies]] = GP gp <- selector(parameters)
```

これらは送り先のメソッド処理が終るまでブロックされ、その後戻り値を用いた処理を継続する同期通信である。前者は、処理をブロックされている間にも、そのオブジェクトに到達するメッセージのうち、その処理に同期通信を必要としないものについてはメソッドを起動するが、同期通信を必要とするものはメッセージキューにいれる。後者ではブロック中はどのメッセージもキューに入れる。メソッドの処理は同期通信以外でブロックされることはない。

戻り値は次のように返す。

```
reply([replies]);
```

メソッドの処理中は、自身とメッセージ送信者のグローバルポインタをそれぞれ self,sender で参照することが可能である。delegation で必要ならば sender を引き渡し、委託先で sender <- reply([...]) と宛先を明示して、返答を行なうことも可能である。

4 応用

4.1 並列問題解決

問題解決の過程を並列化するときには負荷分散と効率的な枝刈りが重要である。本通信モデルを利用してそれらをいかに実現するか、その概要を示す。

例として巡回セールスマン問題をとりあげる。これは N 都市をセールスマンが重複せず 1 つずつ巡回する時の最短経路を求める問題である。この問題を解くのに、中間問題を部分的に解いていくオブジェクトとその中間問題を収容する通信用オブジェクトを用意する。

中間問題は $n < N-1$ 個の都市をまわった後、$N-n$ 個の都市を巡回する時の最短経路を求めることである。とりあえず各都市を重複することなく一巡するものを原問題の可能解と呼ぶ。可能解のうち最適な経路長を与えるものを暫定解とし、中間問題の下界値が暫定解の経路

長（暫定値）以上であれば枝刈りを行なうことが可能である（ただし、都市間の距離は非負であるとする）。

通信用オブジェクトは中間問題と、暫定値（およびそれを与える暫定解）を保持する。暫定値は最初 inf であり、以後よりよい暫定値が見つかり次第その値に更新してその暫定解を保持するとともに、他の通信用オブジェクトに通知する。これには放送や、隣接した通信用オブジェクトへの伝搬などを用いる。

各オブジェクトは特定の通信用オブジェクトのみに中間問題をアクセスする。オブジェクト（あるいは隣接した通信用オブジェクト）からの中間問題の問い合わせに対し、中間問題をもたない通信用オブジェクトは、隣接した通信用オブジェクトに中間問題の有無を問い合わせ、あればそれを受け取ってオブジェクトに与える。

各オブジェクトは中間問題を通信用オブジェクトから１つとってくる。そして部分的に解いて新たに中間問題を生成し、それぞれの中間の経路長を調べる。新たな中間問題のうち枝刈りの対象となるものを除いた残りを通信用オブジェクトに格納する。また、それが可能解であれば、全経路長を計算し、暫定値と比較する。暫定値の方が良ければその可能解を捨てるが、可能解の方がよければそれを新たな暫定解として、上記のように暫定値を更新、伝搬する。

どの通信用オブジェクトにも解くべき中間問題がなくなったとき、残った暫定解を問題の解として与える。解くべき中間問題がなくなったことを検出するのは一般的に分散計算停止判定アルゴリズムを用いればよい。

各通信用オブジェクトは、対応するオブジェクトと協力して局所的に中間問題を解いていくわけだが、暫定値は（準）大域的なデータを用いることが可能となり、枝刈りの効率が向上する。また、中間問題の伝搬により、負荷分散が行なわれる。3 章で示した言語による記述例を図 3に示す。

4.2 情報検索

連続した特徴量を各次元に持つある特徴量空間内の複数の特徴ベクトルのうち、指定した特徴ベクトルに近いものを検索する場合を考える。本モデルでは、特徴量空間を適当に分割してその部分空間をそれぞれ通信用オブジェクトに割り当てる。個々の特徴ベクトルはオブ

```
main() {      // executed only in PE#0
  for (int node = 0; node < NUMNODES; node++) {
    create(Worker(node), node);
    // as many as desired
  }
}

class Worker { PoolGP MyPool; }

Worker::Worker(int node) {
  MyPool = Pool::at(node);
  Main();
}

void Worker::Main() {
  Problem pbm, npbm;
  int tvalue;
  [pbm, tvalue] = MyPool <- GetProblem();
  Problems npbms = solve(pbm); // omitted for brevity
  for (; npbms; npbms = npbms->next) {
    npbm = npbms->pbm;
    if (possibleSolution(npbm)) {     // omitted
      int len = lengthof(npbm);       // omitted
      if (len < tvalue) {
        MyPool <- PutBetter(npbm->path,len);
      }
    } else if (!prune(npbm, tvalue)) {
      MyPool <- PutProblem(npbm);
    }
  }
  self <- Main(); // cont. after switching to other Workers
}

comm Pool {
  Problems pbms;
  int tvalue;                 // temporarily best value
  Path tpath = NULL;          // temporarily best path
  int node;
}

void Pool::Pool (int n) {
  node = n;
  pbms = initialProblems();           // omitted
  tvalue = INFINITE;
}

int Pool::nidof(int n) { return n % NUMNODES; }
int Pool::oidof(int n) { return 0; }     // one Pool per node
void Pool::init() { create(Pool(mynode())); }

void Pool::PutBetter(Path path, int better) {
  if (better < tvalue) {
    tpath = path;
    tvalue = better;
    Pool::at(node+1) <- PutBetter(path, better); // verbose
  }
}

void Pool::PutProblem(Problem npbm) {
  pbms.put(npbm);
}

void Pool::GetProblem() {
    // detection of completion is omitted
  Problem pbm;
  int ntvalue;
  if (pbms.empty()) {
    [pbm, ntvalue] = Pool::at(node+1) <- GetProblem();
    if (tvalue > ntvalue) tvalue = ntvalue;
  } else pbm = pbms.pop();
  reply([pbm, tvalue]);
    // abbrev. for "sender<-reply([pbm, tvalue]);"
}
```

図 3: プログラム例

ジェクトとして実現し、その特徴量から定まる通信用オブジェクトに格納する。

検索に関しては、指定した特徴ベクトルに対応した通信用オブジェクトに問い合わせを行なう。検索要求をうけとった通信用オブジェクトに該当するオブジェクトが存在すればそのオブジェクトを検索結果として返す。なければ隣接する通信用オブジェクトに要求を転送することによって検索の範囲を拡大する。

情報を分類しながら分散させて格納していくとともに、隣接した通信用オブジェクト間のメッセージ伝搬によって検索条件の緩和を実現できる点が特徴である。

4.3 シミュレーション

交通シミュレーションや生態系のシミュレーション等は近接したオブジェクト間において動的に通信が発生する典型例である。前者においては例えば、移動する車同士、車と信号などにおいて動的かつ局所的な通信が発生する。後者においては、捕食や交配などにおいて同様の通信が生じる。これらのシミュレーションにおいては、通信用オブジェクトの集合体は環境として振舞う。

ここでは、シミュレーションにおける同期の問題について考える。

シミュレーションにおいてはオブジェクト間の同期をどこまで細かく厳密にとるか、言い換えればオブジェクトの動作速度にどの程度の偏差が許されるかが問題となる。局所的な同期さえとれていればよいもの、あるいは大域的に同期をとる必要のあるものなど、アプリケーションによってさまざまである。これはシミュレーションによって求める結果がミクロな個体オブジェクトに存在するか、それともマクロな現象なのか、ということにも依存する。

例えば環境をもつ遺伝的アルゴリズムなどにおいては、必要なのは高い適応度をもつ個体の遺伝子であり、大域的な同期の必要は少ない。しかしながら、回路シミュレーションなどは厳密に同期をとることが要求される。

本モデルにおいては、オブジェクトが通信用オブジェクトと局所的な同期を行ない、通信用オブジェクト間で大域的な同期を行なうといった階層的な同期の記述が可能である。通信用オブジェクトにおける同期の粒度と、オブジェクトと通信用オブジェクトの間の同期の粒度を変化させることにより、必要とされる粒度の同期を柔軟に効率よく実現することが可能であると考えられる。

5 おわりに

大域的かつ構造的に参照可能な通信用オブジェクトを導入することにより、オブジェクト間の間接的な通信を可能とする通信モデルを提案した。本モデルは通信の抽象化、分散化、局所化、階層化を容易にし、オブジェクト間の動的な通信経路の設定をサポートしながら、疎結合並列計算機においても効率の良い実現を可能とするものである。

今後は、本モデルに基づくプロトタイプ言語処理系のiPSC/860への実装を進めていくとともに、本モデルの有効性を確認するためのアプリケーションの記述を行ないながら、有用な通信・同期メカニズムの実現、最適化実行用戦略の抽出などを行なう。その上で言語仕様を明確化し、大規模並列計算機上に開発・実行環境を構築する予定である。

参考文献

[1] T. Baba et al. A parallel object-oriented total architecture: A-net. In *Proc. Supercomputing '90*, pages 276–285, 1990.

[2] D. Gelernter. Generative communication in linda. *ACM Trans. on Programming Languages and Systems*, 7(1):80–112, 1985.

[3] W.D. Hillis. *The Connection Machine*. The MIT Press, 1985.

[4] Intel Corporation. *iPSC System Manuals*.

[5] A. Yonezawa. *ABCL – An Object-Oriented Concurrent System –*. The MIT Press, 1990.

[6] 笠原博徳. 最適化コンパイラ技術の現状. 信学会誌, 73(3):258–266, 1990.

[7] 馬場敬信. 超並列マシンへの道. 情報処理, 32(4):348–364, 1991.