超並列オブジェクトベース言語 OCore の並列計算機上での実装

小中裕喜[†] 石川裕[†] 前田宗則[†]
友清孝志[†] 堀敦史[†]

OCore はマルチコンピュータ上で効率的に実行されるよう設計された，並列オブジェクトベース言語である．オブジェクトの集合の構造化や通信の分散，大域的操作などを可能とする「共同体」という概念の導入により，データ並列計算やマルチアクセスデータの記述を容易にする．本論文では OCore の概要と並列計算機 Paragon XP/S を対象とした言語処理系の実装方式，およびその予備性能評価について述べる．また共同体を利用した応用例として分子動力学シミュレーションプログラムをとりあげる．

An Implementation of the Massively Parallel Object-based Language OCore for Multi Computers

HIROKI KONAKA,[†] YUTAKA ISHIKAWA,[†] MUNENORI MAEDA,[†]
TAKASHI TOMOKIYO [†] and ATSUSHI HORI [†]

OCore is a massively parallel object-based language, designed to generate efficient code for multi-computers. In addition to the fundamentals of parallel object-oriented languages, OCore introduces the notion of "community" that enables a structured set of objects to perform distributed communication and global operations. Communities support data parallel computations as well as multi-access data abstractions. In this paper, we present an overview of OCore, an implementation of its language processing system for the Paragon XP/S multi-computer, and the preliminary evaluation of the system performance. We also take a molecular dynamics simulation program as a real application using a community.

1. はじめに

近年マルチコンピュータの研究開発が数多く行われるとともに，その上で実行されるアプリケーションも，定型的な並列計算を行うものだけではなく，複雑な構造をもった科学技術計算，知識処理，自己組織化，社会シミュレーションなど，非定型な並列計算を必要とするものが多くなってきている．

一方，並列計算機に用いられるプログラミング言語にも数多くのものが提案されている．Fortran など従来の逐次型言語で記述されたプログラムから並列性を抽出するようなアプローチも存在するが，多くの言語では並列性は明示的に記述され，それらは着目する並列性によって大きく，データ並列かコントロール並列かに分類される．

データ並列言語は定型的な科学技術計算の記述などに適している．コントロールの面で逐次型言語との連続性をもっており，処理の非決定性が少ないため，良好なプログラマビリティを有する反面，非定型な処理の効率化が困難である．一方，コントロール並列言語は非定型計算に適している．特に，問題領域をプログラムにマッピングしやすい並列オブジェクト指向言語は，これまでに数多く提案されており，中には並列性を抽象化した概念をもつものもある．

例えば CA (Concurrent Aggregates)[1]では通常のオブジェクトのほかに representative と呼ばれる同種のオブジェクトの集合 aggregate を扱う．aggregate には通常のオブジェクトと全く同様にメッセージを送ることが可能であり，それはランタイムによって適当な representative に送られた後，必要ならばその representative 自身がさらに適切な representative へとメッセージをフォワードする．従って，ボトルネックとなるオブジェクトをこのようなマルチアクセス

† 新情報処理開発機構
Real World Computing Partnership

のデータ抽象である aggregate に置換していくことにより，通信の分散と処理の効率化を図るようなプログラム手法が可能である．しかしながら，受信 representative が特定される場合においてもメッセージをフォワードするオーバヘッドを伴う．また，データ並列計算は言語レベルではサポートされていない．

pC++[2]は C++に collection というクラスを導入し，collection の中のオブジェクトに対するデータ並列操作を可能としている．オブジェクトの実プロセッサへのマッピングの記述には HPF のような template，alignment の指定方法を導入し，継承をも利用した柔軟な構造化，マッピングの指定を可能としているが，コントロール並列性はサポートしていない．

このような従来のアプローチに対し，我々が提案する超並列オブジェクトベース言語 OCore では，構造化されたオブジェクトの集合である「共同体」という概念を導入することにより，データ並列性とコントロール並列性を言語レベルでサポートしている．共同体は，プログラミングにおいては抽象化した並列性を与え，マルチコンピュータ上の実行においては，通信の分散と定型的な並列処理の効率化を可能とする．

以下，本論文では 2 章で超並列オブジェクトベース言語 OCore の概要を述べる．3 章では，並列計算機 Intel Paragon XP/S をターゲットとした言語処理系プロトタイプの実装方式を述べ，4 章でその予備性能評価について述べる．5 章では共同体を利用した応用例として分子動力学シミュレーションプログラムをとりあげる．

2. OCore

OCore は以下のような特徴を有する超並列オブジェクトベース言語である：

- オブジェクトの集合の構造化と通信の分散，効率的な実装などを目的として「共同体」という概念を導入している．
- アルゴリズムの記述と，資源管理，実行最適化，デバッグ，例外処理などに関する記述の分離を可能とするメタレベルアーキテクチャを有する．
- 同期構造体を用いた柔軟な同期，通信を可能とする．
- 静的な型づけと簡素な言語セマンティクスによりプログラムの安全性と実行効率の向上を図る．
- グローバル GC を提供する．

以下，OCore におけるオブジェクトと共同体の概要を述べる．

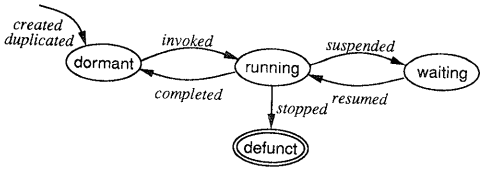

図 1 OCore オブジェクトの状態遷移図
Fig. 1 State transition diagram of an OCore object.

2.1 オブジェクト

OCore におけるオブジェクトは，メッセージパッシングなどを行いながら処理を進める計算主体である．オブジェクトのふるまいはクラスによって記述される．クラスにはスロット，メソッド，ブロードキャストハンドラ，ローカル関数，メタ変数，イベントハンドラ，例外ハンドラなどが定義される．メソッドやローカル関数，および 2.2 節で述べるブロードキャストハンドラの処理はベースレベルで行われる．一方，イベントハンドラは，オブジェクトの生成やメッセージ処理の終了などのシステム定義のイベント，あるいはユーザ定義のイベントの発生に応じて起動されるものであり，また例外ハンドラは例外事象の発生によって起動されるものである．イベントハンドラと例外ハンドラの実行はメタレベルで行われ，そこではベースレベルでは利用できない特別な擬変数，メタ変数，オペレータの参照/実行が可能となっている．メタレベルアーキテクチャの詳細に関しては文献 8) を参照されたい．

OCore オブジェクトの状態遷移図を図 1 に示す．オブジェクトは生成時には *dormant* 状態にある．そこにメッセージが到達すると *running* 状態となり，メッセージに対応したメソッドの処理が開始される．メソッドには同期，非同期の 2 種類があり，オブジェクト間のメッセージパッシングも，対応するメソッドによって同期か非同期かが決定される．非同期送信の場合には送信オブジェクトの処理はそのまま継続されるが，同期通信の場合には送信オブジェクトは *waiting* 状態となる．そして相手オブジェクトからの返答があると再び *running* 状態に遷移する．*running* 状態や *waiting* 状態の時に到達したメッセージはオブジェクトのメッセージキューに格納される．1 つのメッセージの処理が終った時点でキューイングされたメッセージが存在すれば，引き続いてその処理を行い，なければ *dormant* 状態に戻る．このように OCore ではオブジェクト内におけるメッセージ処理は逐次的である．

また，同期・非同期のメッセージパッシングに加え，

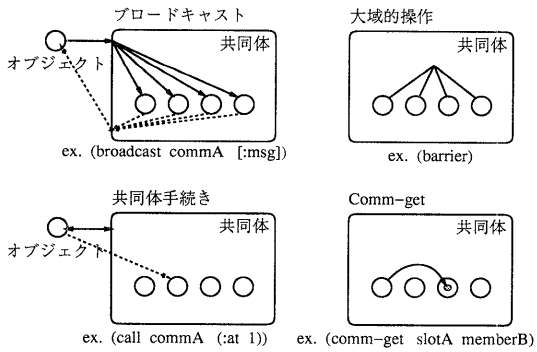

図2 共同体に関する操作
Fig. 2 Operation images with respect to communities.

より柔軟な同期・通信を可能とするために，OCoreでは基本オブジェクトとしてI-structure[5]，Q-structure[6]などの同期構造体を提供している．これらは，メッセージデータを遅延送信したり，メッセージの返答を他のオブジェクトに直接送るといった記述を容易にする．

2.2 共 同 体

共同体はオブジェクトの集合を，インデックスによってはられた多次元空間の中で構造化するものであり，データ並列計算やマルチアクセスデータの記述とその並列計算機上での効率的な実行を容易にする．共同体のインスタンスに属するオブジェクトはメンバオブジェクトと呼ばれ，そのクラス定義では通常のクラス定義に加えてメンバオブジェクト特有の操作を利用できる．**図2**に共同体に関するいくつかの操作の概念図と記述例を示す．以下に，共同体に関する記述と各操作の概要を説明する．

2.2.1 共同体テンプレート

共同体の振舞いはメンバオブジェクトのクラスと，共同体テンプレートと呼ばれるものによって記述される．共同体テンプレートには，メンバオブジェクトのクラス，メンバオブジェクトが構成する論理空間の次元数，および後述する共同体手続きがベースレベルで記述される．またメタレベルの記述として，メンバオブジェクトのインデックスからそのメンバオブジェクトの存在するノード番号とそのノード内のローカルなインデックスを与えるマッピング関数，共同体インスタンス生成時の初期化のためのイベントハンドラなどがある．マッピング関数を省略した場合はデフォルトのマッピングが用いられる．

メンバオブジェクトのクラスはパラメトリックに定義することも可能である．共同体テンプレートの記述は共同体の実装に大きく依存するが，多くの場合プログラマはライブラリ化されたパラメトリックな共同体テンプレートを利用することになる．

2.2.2 共同体に対する操作

共同体は，共同体手続きと呼ばれるインタフェースを提供する．これはメンバオブジェクトはもとより，共同体インスタンスを知る任意のオブジェクトが利用可能な手続きであり，共同体テンプレートで定義される．共同体手続きの例としては，マッピング関数を用いてインデックスからメンバオブジェクトの参照を求めて返すものなどがある．例えば，共同体インスタンスの各メンバオブジェクトに対して動的に相手を変えながら通信を行うオブジェクトは，メンバオブジェクトのテーブルを用意することなく，単にこの手続きを利用するだけでマッピング情報を用いた効率的な通信を行うことが可能となる．さらに，メンバオブジェクト間の論理構造を反映した共同体手続きを用意することにより，その論理構造を利用した通信の記述も容易となる．

また，共同体インスタンスの全メンバオブジェクトに対して，メッセージをブロードキャスト（broadcast）することも可能である．

このように共同体に属さないオブジェクトからも，共同体手続きやブロードキャストを利用することにより，共同体の個々のメンバオブジェクトあるいは全メンバオブジェクトに対してメッセージを送信することが可能である．

2.2.3 メンバオブジェクトにおける操作

各メンバオブジェクトは自分の属する共同体インスタンスや，その次元数およびサイズ，共同体内における自分のインデックスを知ることが可能である．

また，同一の共同体インスタンスに属するメンバオブジェクトの間ではバリア同期やリダクションなどの大域的操作が可能である．リダクションでは各メンバオブジェクトからの値の総和や最大値，論理和などを求めることが可能である．さらに，あるメンバオブジェクトが，明示的なメッセージパッシングを行うことなく他のメンバオブジェクトのスロットに効率的にアクセスできる，comm-getという操作も利用可能である．

共同体手続きを利用した，メンバオブジェクト単体へのメッセージパッシングは，通常のオブジェクト間のメッセージパッシングと同様，対応するメソッドによって同期，あるいは非同期に行われる．一方，ブロードキャストされたメッセージは各メンバオブジェクトにおいて，メソッドではなくブロードキャストハン

ドラによって処理される．ブロードキャストハンドラにも同期，非同期の2種類があり，その処理はメソッドの処理と類似している．しかしながら，同期通信におけるリプライの処理は大きく異なる．同期型のブロードキャストハンドラでは，リプライを行う前に全メンバオブジェクトの間でバリア同期が暗黙的に行われ，その後ある1つのメンバオブジェクトだけがリプライを行う．また，バリア同期の代わりにリダクションを行い，その結果をリプライすることも可能である．

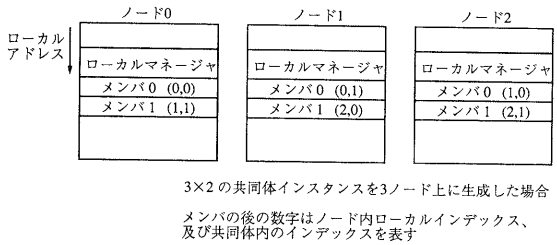

図3 共同体におけるメモリアロケーション
Fig. 3 Memory allocation in a community.

3. Paragon XP/S 上での実装

3.1 Paragon XP/S

Intel Paragon XP/S[4]は i 860 XP を要素プロセッサとするマルチコンピュータである．各ノードには計算用と通信用の計2つの i 860 XP が搭載されている．クロックは 50 MHz で，1ノードあたり 75 MFLOPS のピーク性能をもつ．OS は OSF/1 R 1.2 である．メモリ空間はノードごとに独立している．システムは動的に空間分割され，同時に複数のプロセスが同一ノードを利用することはない．

ノード間の通信は NX メッセージパッシングライブラリによって記述される．NX は csend (), crecv () などの同期通信，isend (), irecv () などの非同期通信，giadd () などの大域的操作を提供する．

3.2 実装の概要

プロトタイプシステムは OCore から C++へのトランスレータ，Paragon 用に移植した GNU C++，OCore ランタイムシステムなどからなる．OCore ランタイムシステムは Paragon 上に構築したマルチスレッドライブラリ上で実現されている．ノード間のシステムメッセージは NX を用いて送受信し，受信したメッセージは Active Message[9]と同様に処理される．本ライブラリではスレッドのローカル/リモート生成などのマルチスレッドのサポートに加え，リモートメモリアクセスやロック，同期構造体，ノード単位/スレッド単位の大域的操作などをサポートしている．ここで，ノード単位/スレッド単位の大域的操作とは各ノードで1つ/複数のスレッドが参加するものである．大域的操作に参加していない他のスレッドの活動が妨げられない点で，NX が提供する大域的操作とは本質的に異なる．

オブジェクトは *running* 状態になったとき実行フレームを与えられ，*dormant* 状態にもどったときにその実行フレームを解放する．実行フレームはメソッド処理に必要な一時変数の格納などに用いられる領域であり，フリーリスト管理されている．メソッド内での関数呼び出しや共同体手続き呼び出しにおいても同様の実行フレームが動的に割り当てられる．キューイングするメッセージを格納する領域もフリーリスト管理されている．

共同体の実装は，メンバオブジェクトの集合が共同体生成時に決定されるか，その論理構造が動的に変化するか，などで大きく異なる．現在は最も単純なもの，すなわちメンバオブジェクトの集合が共同体生成時に決定・生成されて永久に存続し，メンバオブジェクト間の論理構造が変化しないものだけを扱っている．メンバオブジェクトの各ノードへのマッピングは，メンバオブジェクトのインデックス値から定まる一次元の ID をもとに，各ノードにインタリーブするというデフォルトのものを用いる．

共同体インスタンスの生成は以下の手順で行う．まずノードごとのメンバオブジェクトの生成や大域的操作のサポートを行う，ローカルマネージャというものを各ノードに生成する．各ローカルマネージャはそのノードにマッピングされるメンバオブジェクトの生成・初期化を行う．このとき各メンバオブジェクトにはインデックスとローカルマネージャのアドレスが渡される．最後に大域的操作のために必要な初期化を行って各ノードの処理を終了する．

共同体における大域的操作や comm-get などを効率的に処理するため，ローカルマネージャおよび同じローカルインデックスをもつメンバオブジェクトは，**図3**に示すように，各ノードにおいて同じローカルアドレスをもつようにメモリをアロケートしている．このようなアロケーションを実現するため，各ノード上のメモリ空間の一部はグローバルに管理されている．

共同体における大域的操作はライブラリの提供するスレッド単位の大域的操作を用いている．まずノードごとにローカルな処理を行った後，ノード単位の大域的操作を行い，必要ならば結果を各スレッドに返す．ノード単位の大域的操作は NX のメッセージパッシ

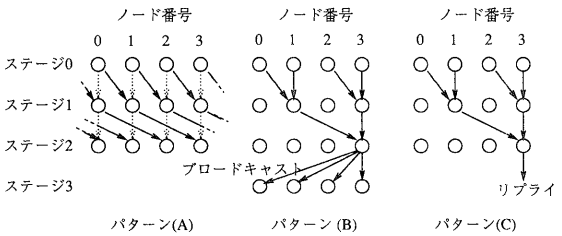

図4 大域的操作におけるノード間メッセージパターン
Fig. 4 Inter-node message patterns for global operations.

ングで実現しているが，その通信パターンとして**図4**に示す3つの方法を用意している．

ノード数をNとすると，(A)では各ノードnがステージsでノード$(n+2^s) \bmod N$に送信して，ステージごとに同期をとりながらN本の木を並列に構築していく．一方(B)では木を1本構築した後，結果をブロードキャストする．これらは一般の大域的操作に用いられ，メタレベルの記述により選択される．デフォルトでは(A)が用いられる．メッセージ数は(A)の場合$O(N\log(N))$，(B)は$O(N)$と(A)の方が多いが，他にアクティビティがない場合は処理時間が短くてすむ．一方(B)は他にアクティビティが存在して大域的操作のレイテンシが隠蔽可能な場合に有効である．また同時に複数実行されることがない同期型ブロードキャストハンドラの場合，リプライを返すメンバオブジェクト以外は同期の成立を知る必要がないので，(B)の最後のブロードキャストが省略された(C)が用いられる．リプライを返すのは，同期が成立するノード（図ではノード3）に存在する任意のメンバオブジェクト（実際には最後に同期ポイントに達したオブジェクト）である．

共同体の全メンバオブジェクトに対するブロードキャストにおいては，メッセージがある程度大きい場合，それを全メンバオブジェクトにコピーするのは非効率的である．そこで，ノードごとに1つだけコピーを置き，各メンバオブジェクトのメッセージキューにはそのコピーの利用を示す特別なメッセージをキューイングしている．コピーにはローカルなメンバオブジェクトの総数が参照カウントとしてセットされ，全メンバオブジェクトがその処理を終えた時点で領域が解放されるようにしている．

comm-getの実現はローカル/リモートメモリアクセスに，共同体手続きの呼び出しは実行フレームの割り当て/解放とC++のメンバ関数呼び出しに，それぞれ還元される．

表1 基本的操作の実行時間
Table 1 Basic operation times.

操作	時間 (μs)
コンテキストスイッチ	4.2
オブジェクト生成	
ローカル	11.5
リモート	154.7
非同期送信	
ローカル *dormant* オブジェクトへ	5.2
ローカル *running* オブジェクトへ	2.1
リモートオブジェクトへ	49.0
同期通信	
ローカルオブジェクトへ	12.7
リモートオブジェクトへ	156.3
共同体操作 （メンバ数1,024，ノード数64）	
ローカル comm-get	0.2
リモート comm-get	142.0
共同体手続き呼び出し	1.3
同期型ブロードキャスト	1223.4
バリア同期	1114.5

なお，現状では例外処理などメタレベルの多くの部分やGCは未実装である．

4. 性能評価

以上述べた実装方式によるOCoreの基本的な操作の実行性能をParagon XP/Sを用いて測定した．各ノードの主記憶は16 MBである．NXのパケットサイズは64バイトとした．また，OCore実行フレームは1 KB固定長，OCoreメッセージの格納領域は64バイト固定長とした．結果を**表1**に示す．

コンテキストスイッチにはレジスタのセーブ/リストア，他のノードからの受信メッセージがあるかどうかのチェック，他の実行可能なスレッドがあるかどうかのチェックの時間が含まれる．他のノードからのメッセージが到着していればその処理を行う．受信メッセージがなく，実行可能スレッドが存在すれば，それをスケジュールする．

オブジェクトの生成では整数型のスロットを1つだけもつものを作っている．ここではメモリアロケーションに一般のmalloc()を用いているが，より効率的なメモリ管理を行うことによりオーバヘッドを低減することが可能である．

オブジェクトへの送信では無引数のメッセージを送信している．*dormant*状態のオブジェクトへのローカル送信の場合は，実行フレームを割り当ててスレッドを実行可能状態とするが，*running*（*waiting*）状態のオブジェクトへのローカル送信ではオブジェクトのメッセージキューにメッセージをキューイングするだけである．

リモートへの操作の場合，comm-get の実行時間はほぼ NX によるメッセージ往復の通信時間であり，オブジェクト生成や同期送信なども通信時間が支配的であることが示されている．

大域的操作については，1,024 オブジェクトからなる共同体において，図 4 のパターン(C)による整数和をとる同期型ブロードキャストと，パターン(A)によるバリア同期の実行時間を 64 ノードで測定した．

これらを以前の報告[11]と比較すると，例えばコンテキストスイッチは約 7.2 倍，ローカルな同期通信は約 5.8 倍，リモート同期通信は約 4.3 倍，同期型ブロードキャストは約 3.7 倍高速になっている．これは，i) OS のバージョンが R 1.2 となり，Paragon の通信用プロセッサが稼働して NX の性能が向上した，ii) OCore ランタイムシステムが同期受信（crecv ()）ではなく非同期受信（irecv ()）を利用するようになった，iii) ローカルなメッセージのコピーを最小にするようトランスレータとランタイムシステムを改良した，などの理由があげられる．例えばリモート同期通信では，i) により約 2.3 倍，ii) によりさらに約 1.9 倍高速化されている．

5. 共同体を利用したプログラミング

共同体のメンバオブジェクトを他のオブジェクト同士のメッセージ交換の媒体として用いることにより，場のような概念の記述[12]や抽象化，分散化，階層化などさまざまな特徴をもった通信モデルの記述が可能となる[13]．また，ブロードキャストと，バリア同期やリダクションなどの大域的操作を用いることにより，データ並列プログラミングを行うことが可能である．ここでは後者の例として，SPLASH ベンチマーク[7]の中の分子動力学シミュレーションプログラム Water を例題とし，OCore での記述の概要と実行結果を示す．

5.1 OCore による Water の記述

Water は周期的な境界条件を持った立方体の中の水分子の系のふるまいを，Gear の predictor-corrector 法を用いて計算する．プログラムは初期化を行った後，i) 各原子の状態の予測，ii) 分子内力の計算，iii) 分子間力の計算，iv) 各原子の状態の修正，v) 境界条件による分子位置の修正，vi) 運動エネルギーなどの計算，という一連の処理を指定回数繰り返す．なお，カットオフ半径（ここでは立方体の一辺の半分）以上離れた分子間力は無視している．

SPLASH の Water は共有メモリモデルによるプログラムだが，ここでは水分子クラス Mol のオブジェクトをメンバとする共同体 Water によって系を表現する．中心的なループの部分を以下に示す．

```
1  (loop
2    (broadcast water [: predic-intraf])
3    (broadcast water [: interf FORCES])
4    (qread done)
5    (set sum
6        (broadcast water [: cor-bnd-kin]))
7    ;; calculations of energy, etc. (omitted)
8  )
```

broadcast は[]で括ったメッセージを共同体インスタンスにブロードキャストする．: predic-intraf は i)，ii) を，: cor-bnd-kin は iv)，v)，vi) をそれぞれまとめて行う Mol クラスの同期型ブロードキャストハンドラである．特に，後者はリダクションを行って全運動エネルギーを求めており，その結果が変数 sum に代入されている．

これらの処理が基本的に分子ごとに独立しているのに対し，: interf で起動される iii) の処理は分子状態の相互参照が必要である．ある分子間の力は一方の分子が計算すればよいため，各分子は残りの分子の約半分についてそれぞれ距離を計算し，カットオフ半径内ならば相手と自分の状態から分子間力を計算して双方の状態の更新を行う．

共同体におけるこのような参照パターンの実現として，

(a) comm-get を利用して相手の状態を参照する．
(b) (a)において comm-get したデータのキャッシュを行う．
(c) 自分の情報を相手に送信する．
(d) 自分の情報をブロードキャストする．

などが考えられる．

(a)は自然な表現であるが，相手が別ノードに存在する場合，リモートメモリアクセスのためのメッセージが往復する．また，現在の OCore 処理系では相手の状態を 1 度に 1 要素しかアクセスできないので，さらに多くのメッセージが往復することになる．このようなオーバヘッドを軽減するために実験的にキャッシュを導入したものが(b)である．comm-get するスロットのアドレスとデータをランタイムシステムがキャッシュする．invalidation はシミュレーションの各時刻の終りにメンバオブジェクト間の大域的操作を介して行っている．

(c)は(a)と比較すると総メッセージ数は少なくてすむが，処理の流れがやや複雑となる．(c)が自分の

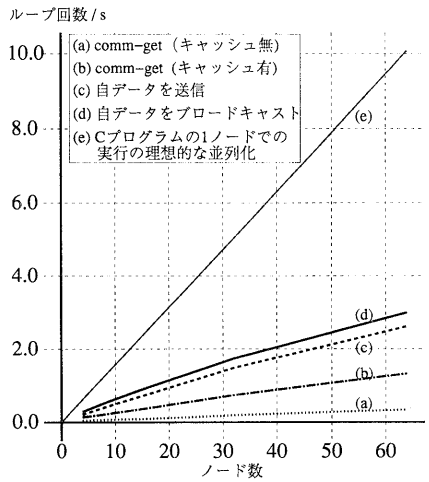

図5 Paragon XP/S上でのWaterの実行結果
Fig. 5 Performance of Water programs on Paragon XP/S.

データを必要とするオブジェクトに対してのみメッセージを送信するのに対し，(d)では個々のメンバオブジェクトにおける無駄なメッセージ処理が増加するものの，3.2節で述べたような受信メッセージデータの共有が行われ，ノード数に比べてメンバオブジェクト数が多い場合にネットワーク上のメッセージ数がさらに減少する．

いずれの実現においても，ブロードキャストハンドラ：interfの処理の後に，他からの状態更新などのメッセージを処理しなければならない場合があるため，：interfの最後にバリア同期をとって分子間力の計算終了を知ろうとするとデッドロックを生じる．しかし，各分子に到達する状態更新メッセージの総数は（カットオフの場合もしかるべきメッセージを送るとすれば）わかっている．そこで，今回のプログラムでは：interfや状態(非)更新メソッド，(c)，(d)の場合の状態通知メソッド/ブロードキャストハンドラをすべて非同期とし，各分子では行うべき処理をカウントして最後に全体でバリア同期をとり，代表分子がグローバルな同期構造体を用いて分子間力の計算終了を伝えるようにしている（4行めはその終了を待つ同期構造体の読み出しである）．

5.2 実行結果

分子数343で実行ノード数を変えながら各プログラムを実行した時の1秒あたりのループ実行回数を図5に示す．比較のために，SPLASHベンチマーク中のCプログラムを1ノードで実行し，その実行速度を基準として理想的な並列化が行われた場合の直線を(e)として示している．

(b)におけるcomm-get用のキャッシュはラインサイズ256バイト，ライン数1,024である．キャッシュを導入することにより，(a)のようなナイーブなプログラムでもかなり効率が改善されることが示された．一方(c)，(d)はさらに効率化されているが，実行には大容量のメッセージバッファを必要とした．特に(c)の場合，一時期に大量のメッセージが交換されるため，ネットワークの輻輳や通信バッファの容量不足が問題となりやすい．そこで状態通知の際に適当にスレッドを中断して，リモートからのメッセージの処理を滞らせないようにしている．

6. おわりに

超並列オブジェクトベース言語OCoreは，オブジェクトの集合の構造化と通信の分散，効率的な実装などを目的として「共同体」という概念を導入しており，多様な並列性をもったアプリケーションの記述と並列計算機上での効率的な実行を容易にしている．

本論文ではOCoreの概要とマルチコンピュータIntel Paragon XP/S上での言語処理系の実装方式，およびその予備的な性能評価について述べた．また，分子動力学シミュレーションプログラムWaterの記述と実行を通じて，共同体の枠組の有用性を示した．

現在，共同体にはいくつか実装上の制約がある．今後の拡張には以下のようなものが考えられる：

- メンバオブジェクトの集合の動的な変化の実現
- マッピングの変更などメタレベルの実現
- 非一様な論理構造の実装
- comm-getにおけるキャッシュのサポート

言語処理系のプロトタイプ実装は，Paragon XP/S以外にもSun Sparc Stationを対象としても行われており，PVM[3]を介した分散環境での実行も可能である．今後は，これらの処理系を用いて実アプリケーションによる評価と言語仕様の改良を行うとともに，グローバルGCやメタレベルアーキテクチャの実装，RWC-1[10]などへの移植を行う予定である．

謝辞 本研究を行うにあたりご指導，ご討論をいただいたRWC超並列ソフトウェアWSおよび超並列言語WGのメンバーの方々に感謝いたします．

参考文献

1) Chien, A.A.: *Concurrent Aggregates*, The MIT Press (1993).
2) Gannon, D.: Libraries and Tools for Object Parallel Programming, *Proc. CNRS-NSF Work-*

shop on Environments and Tools For Parallel Scientific Computing (1992).
3) Geist, G.A., Beguelin, A.L., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V.: PVM 3 User's Guide and Reference Manual. Tech. Report TM-12187, Oak Ridge National Laboratory (1993).
4) Intel Supercomputer Systems Division: *Paragon OSF/1 C System Calls Reference Manual* (1993).
5) Nikhil, R.S. and Pingali, K.K.: I-Structure: Data Structures for Parallel Computing, *ACM Trans. Prog. Lang. Syst.*, Vol. 11, No. 4, pp. 598-639 (1989).
6) Sato, M., Kodama, Y., Sakai, S., Yamaguchi, Y. and Sekiguchi, S.: Distributed Data Structure in Thread-based Programming for a Highly Parallel Dataflow Machine EM-4, *Proc. of ISCA 92 Dataflow Workshop* (1992).
7) Singh, J.P., Weber, W.D. and Gupta, A.: SPLASH: Stanford Parallel Applications for Shared-memory, *Technical Report CSL-TR-92-526*, Computer Systems Laboratory, Stanford Univ. (1992).
8) Tomokiyo, T., Konaka, H., Maeda, M., Hori, A. and Ishikawa, Y.: Meta-level Architecture in *OCore, OOPSLA '93 Object-Oriented Reflection Workshop* (1993).
9) von Eicken, T., Culler, D.E., Goldstein, S.C. and Schauser, K.E.: Active Messages: A Mechanism for Integrated Communication and Computation, *Proc. of the 19th Intl. Symp. on Computer Architecture*, pp. 256-266 (1992).
10) 坂井修一, 岡本一晃, 松岡浩司, 広野英雄, 児玉祐悦, 佐藤三久, 横田隆史: 超並列計算機 RWC-1 の基本構想, 並列処理シンポジウム JSPP '93, pp. 87-94 (1993).
11) 小中裕喜, 石川 裕, 前田宗則, 友清孝志, 堀 敦史: 超並列オブジェクトベース言語 *OCore* の商用並列計算機上での実装, 並列処理シンポジウム JSPP '94, pp. 113-120 (1994).
12) 小中裕喜, 石川 裕, 前田宗則, 友清孝志, 堀 敦史: 超並列オブジェクトベース言語 *OCore* における共同体プログラミング, *WOOC '94* (1994).
13) 小中裕喜, 横田隆史, 瀬尾和男: 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, 情報処理学会プログラミング—言語・基礎・実践—研究会, Vol. 9, No. 3, pp. 17-24 (1992).

(平成 6 年 9 月 16 日受付)
(平成 6 年 12 月 5 日採録)

小中 裕喜 (正会員)

1964 年生. 1987 年東京大学工学部電子工学科卒業. 1989 年同大学院工学系研究科電気工学専攻修士課程修了. 同年三菱電機(株)入社. 1992 年より技術研究組合新情報処理開発機構に出向. 現在に至る. 並列プログラミング言語の研究に従事. 並列・分散処理, 計算機アーキテクチャ, プログラミング環境などに興味を持つ.

石川 裕

1987 年慶應義塾大学理工学部電気工学科博士課程修了. 工学博士. 同年電子技術総合研究所入所. 1988 年～1989 年カーネギー・メロン大学客員研究員. 1990 年日本ソフトウェア科学会高橋奨励賞を授賞. 1993 年から新情報処理開発機構に出向. 並列・分散システム, 適応可能並列プログラミング言語／環境／処理系, リアルタイム処理, 等に興味を持つ. ソフトウェア科学会, ACM, IEEE 各会員.

前田 宗則 (正会員)

1964 年生. 1989 年大阪大学基礎工学部情報工学修士課程修了. 同年富士通(株)入社. 1992 年より技術研究組合新情報処理開発機構に出向. 現在に至る. 並列ガーベジコレクション等の研究に従事.

友清 孝志 (正会員)

1967 年生. 1989 年九州大学工学部電気工学科卒業. 1991 年大学院総合理工学研究科情報システム学専攻修士課程修了. 同年(株)東芝入社. 1992 年より技術研究組合新情報処理開発機構に出向. 現在に至る. 並列・分散システム, 並列オブジェクト指向言語, 拡張可能な並列プログラミング言語, 関数型言語などに興味を持つ.

堀　敦史（正会員）

1956年生．1979年早稲田大学理工学部電気工学科卒業．1981年同大大学院理工学研究科計測制御工学専攻修士課程修了．同年(株)三菱総合研究所入社．1992年より技術研究組合新情報処理開発機構に出向．現在に至る．並列オペレーティングシステムの研究に従事．並列プログラミング言語，並列アーキテクチャなどに興味を持つ．