2U－6

超並列オブジェクトベース言語 OCoreにおける データ並列計算の記述

小中 裕喜　石川 裕　友清 孝志　前田 宗則　堀 敦史
新情報処理開発機構つくば研究センタ

1 はじめに

われわれは超並列計算モデル/通信モデルやその最適化実行のための Research Vehicle として、超並列オブジェクトベース言語 *OCore*を提案している。*OCore*では、並列オブジェクトの提供するコントロール並列性に加え、共同体の概念を導入することにより、データ並列計算や「場」の記述性を高めている。後者の例としては Artificial Life シミュレーション [2] などがあるが、本稿では共同体を用いたデータ並列計算の記述について述べる。SPLASH ベンチマーク [1] の中の分子動力学シミュレーションプログラム Water を例とし、*OCore*による記述の概要とその予備的な性能評価を示す。

2 *OCore*

*OCore*は簡素で拡張性の高い並列オブジェクトベース言語である。その特徴として、i) 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、ii) アルゴリズムの記述と、資源管理、実行最適化、例外処理などの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、iii) 同期構造体を用いた通信により、柔軟な同期を可能とする、iv) 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、v) グローバル GC を有する、などがあげられる。

*OCore*のオブジェクトは同期/非同期のメッセージパッシングを行ないながら処理を進める計算主体である。オブジェクト内のメッセージ処理は逐次的に行なわれる。

一方、共同体はオブジェクトの集合を多次元空間の中で構造化するものである。共同体インスタンスに属するオブジェクト (メンバオブジェクトと呼ぶ) の間ではバリア同期やリダクションなどのグローバルオペレーションが可能であり、また他のメンバオブジェクトのスロットの値を読む (`comm-get`) ことが許される。さらに共同体手続きと呼ばれるインタフェースを提供することにより、共同体の論理的な構造に従ったメンバオブジェクトへのアクセスを効率的にサポートする。

また、共同体インスタンスの全メンバオブジェクトにメッセージをブロードキャスト (`broadcast`) することも可能であり、大域的操作とともにデータ並列計算の記述を容易にしている。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で、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はメソッドと類似しているが、大きな違いは同期通信の場合に生じる。すなわち、返答を行な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なわれた後、任意の1つのメンバオブジェクトだけが実際に返答を行なう。また、バリア同期の代わりにリダクションを行ない、その結果を返答することも可能である。

現在、*OCore*の言語処理系プロトタイプは Intel Paragon XP/S 及び Sun (PVM を用いた分散処理も可) の上で稼働している [3]。

3 *OCore*による Water の記述

Water は分子動力学シミュレーションプログラムである。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なった後、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する。

SPLASH の Water は共有メモリモデルに基づいて記述されていて、複数のプロセスが全分子の状態など共有メモリ上のデータをアクセスしながら処理を進めていく。ループの先頭、ii) と iii) の間、iii) における力の更新の終了時、vi) における各分子の運動エネルギーの計算終了時にはバリア同期が行なわれる。また、各分子の状態の更新や系全体の運動エネルギーなどを求めるために、総和をとる場合などはロックをかけながら行なう。

一方、今回の *OCore*によるプログラミングでは水分子 `Mol` のオブジェクトをメンバとする共同体 `Water` によって系を表現した。中心的なループの部分を以下に示す。

`:predic-intraf` は i),ii) を、`:correct-bnd-kin` は iv),v),vi) をそれぞれまとめて行なう `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、後者は運動

Data parallel programming in massively parallel object-based language *OCore*
Hiroki Konaka, Yutaka Ishikawa, Takashi Tomokiyo,
Munenori Maeda, Atsushi Hori
Tsukuba Research Center, Real World Computing Partnership
Mitsui Bldg. 16F, 1-6-1 Takezono, Tsukuba, Ibaraki 305, Japan

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

エネルギーをリダクションして返す。

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。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なう。

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) 逆に自分の情報を相手に送りつける、などが考えられる。(A) は自然な表現であるが、相手が別 PE に存在すると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。(B) はメッセージ数が少なくてすむが、処理の流れはやや複雑になる。

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、(B) の場合の状態通知メソッドをすべて非同期とし、各分子で行なうべき処理をカウントして最後にバリアをとり、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 (5行めはその終了を待つ)。

上記の 2 つの方法で実現した *OCore*プログラムを Intel Paragon XP/S 上で PE 数を変えながら実行した。分子数 216 で上記ループを 100 回実行する時間を示したのが図 1である。

4 検討

1PE の場合は全ての comm-get が単なるローカルメモリアクセスとなるが、その他の場合はリモートメモリアクセスとなるため、予想通り 1PE の場合を除いて (A) の方が実行時間が長くなった。しかし、複数の comm-get をまとめられれば、オーバヘッドは軽減される。

今回の両プログラムではリモートから得られる情報の共有が行なわれていない。(A) の場合 comm-get で得られた情報をキャッシュすることにより、性能向上が期待される。また、(B) の場合はマルチキャストの効率的な実装がこれに対応する。

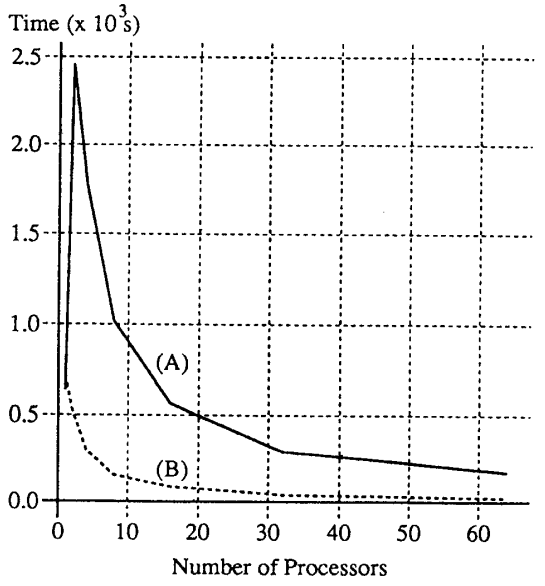

図 1: Water の実行時間

5 おわりに

*OCore*によるデータ並列計算の記述について分子動力学シミュレーションプログラム Water を例にとって述べた。共同体はデータ並列プログラミングにも対応する有用な概念であるが、Paragon 上での実行結果などから、さらに実行効率を向上させるための言語仕様及び処理系へのヒントが得られた。今後、Water を含む多くのアプリケーションを通じ、共同体やメタレベルアーキテクチャなどの仕様/実装の強化を行なっていく予定である。

参考文献

[1] J. P. Singh, W.-D. Weber, and A. Gupta. SPLASH: Stanford parallel applications for shared-memory. Technical Report CSL-TR-92-526, Computer Systems Laboratory, Stanford University, June 1992.

[2] 小中, 石川, 前田, 友清, 堀. 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*, Feb. 1994.

[3] 小中, 石川, 前田, 友清, 堀. 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.