超並列オブジェクトベース言語 *OCore* による *N*体問題の記述

2U－7

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

1 はじめに

我々が開発している超並列オブジェクトベース言語 *OCore* [3] は, 構造化されたオブジェクト間の通信を高速に処理するとともに, 従来のプログラミング言語が苦手としていた不規則かつ動的な通信についても高い記述性/実行効率を目指している.

*N*体問題は, 複数の粒子が互いに影響を与えあう系の時間発展を求めるシミュレーションであり, このような特徴を持つ問題の1つである. このような不規則な通信パタンをもつ問題は, 各粒子を自律的に動作する並列オブジェクトとすることにより素直にモデル化できる.

しかし, *N*体問題の質点のような相互に影響力を及ぼしあう個体を並列オブジェクトとして実現する際には, i) 粒子間の相互作用を, 同期型通信によりスロットの内容を相互参照しあうようにするとデッドロックしてしまう, ii) 物理空間で近傍に存在する粒子の参照データに関する局所性を利用できず, 冗長な通信が起きる, などの問題が発生する.

OCore では, i) 同じ共同体 [3] に属するオブジェクトには互いのスロットをアクセスすることを許す (`comm-get`), ii) リモートアクセスのキャッシングおよび通信レイテンシ隠蔽のために, Q-structure などの同期構造体 [3] を活用する, iii) オブジェクトの PE への分散をメタレベルで記述する, などによりデータの局所性を意識したプログラムの開発効率をあげることを目指している.

本稿では, 共有メモリベンチマークである SPLASH[1] 中の Barnes-Hut を例題として, 局所性を意識した *OCore* でのプログラミングについて述べる.

2 Barnes-Hut アルゴリズム

SPLASH の Barnes-Hut は, 重力の影響を互いに受ける質点系のシミュレーションである. これは分子シミュレーションなどの場合と異なりポテンシャルのカットオフ半径を仮定することができないために, $O(N^2)$ の計算を必要とする. ここで使用されている Barnes-Hut(BH) アルゴリズムは, 3 次元の物理空間を階層的に分割された木構造 (以下 BH 木) として表現し, 遠くの質点群を重心で近似する $O(N \log N)$ のアルゴリズムである.

図1に2次元の場合のシミュレーション空間の分割の様子 (a) と, その BH 木による表現 (c) を示す. 木の節点はそれぞれ物理空間のセル (小領域) に対応し, それが端点の場合は質点を, 内点の場合には擬似質点 (=セルに含まれる質点全体の重心) を示す.

力の計算時には, 各質点は BH 木をルートから辿りながら力を加算していく. このとき対象となるセルの幅を l, 質点とセルの重心の距離を d とし, ある閾値θに対して $l/d < \theta$が成り立つならば, そのセルに含まれる質点を重心で置き換えることができる.

プログラムは, a) 木の生成, b) 擬似質点データの計算, c) 力の計算, d) 質点の位置および速度の更新, e) 座標の最小/最大値を求め領域の大きさを計算, という一連の処理を指定回数くりかえす. オリジナルのプログラムでは共有メモリを前提とし, BH 木は配列で表され, 各要素の更新はロックをかけながら行なわれる. また上記各フェーズ毎にバリア同期をとっている.

本アプリケーションは, 以下のような特徴を持つ. i) ある質点 (または擬似質点) のデータが必要かどうかは, 影響を与える質点との間で近似条件が成り立つかどうかを計算してはじめて分かる, ii) 物理的に近傍に位置する各質点どうしは, 互いに似通ったリモートデータアクセスパタンを示す, iii) 質点の疎密があると, 各質点の計算量にばらつきがでるため, 質点を各 PE に均等に割り付けるだけでは十分な負荷分散を行なうことができない (しかし, 細粒度の動的負荷分散は必要なく, 定期的にデータの割り付け直しを行なえば済む).

3 *OCore* による Barnes-Hut アルゴリズム

OCore では, BH 木の各節点をオブジェクトとして実現する. 節点オブジェクトは質点オブジェクトと擬似質点オブジェクトに分類される (*OCore* では現在継承がないため, これらを同一の節点オブジェクトで表しタグで区別している). 質点オブジェクトは, 粒子の質量, 3 次元の位置座標, 速度, 加速度をスロットに持つ. 同様に, 擬似質点オブジェクトはその重心, 位置, および 8 つの直系子孫の節点への参照を格納する配列をもつ.

BH 木を構成する方法には, 木をプロセッサにどのように割り当てるか, また木のデータをプロセッサ間でどの程度の冗長性をもたせるかにより, さまざまな方法が考えられる. *OCore* の変数のスコープには, PE 内に局所的なものと, 全体にわたって大域的なものがあり, これらを使い分けることによって以下の各方法を記述する.

まず, 最もナイーブな方法は, 予め各 PE に分散して生成された各質点オブジェクトが, 大域データであるルートセルを根とする大域的に存在する BH 木を辿りながら, 自分自身を木に挿入していく方法である. この際, 擬似質点オブジェクトの生成される PE 番号はオブジェク

"N-body Simulation in the Massively Parallel Object-Based Language *OCore* ," Takashi TOMOKIYO, Hiroki KONAKA, Munenori MAEDA, Yutaka ISHIKAWA, and Atsushi HORI.
Tsukuba Research Center, Real World Computing Partnership
Mitsui Bldg. 16F, 1-6-1 Takezono, Tsukuba, Ibaraki 305, Japan

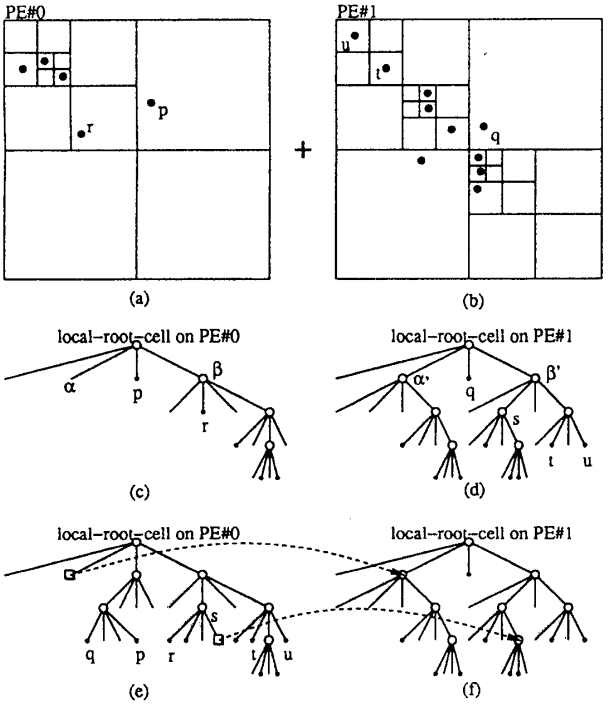

図 1: 2 次元の場合の領域分割とその BH 木表現 (4 分木)

ト生成時に指定される．この場合の力の計算は，各質点がルートセルから BH 木を辿り十分近似可能な擬似質点データを質点に送りつけることで行なう．力の計算の終了判定は，擬似質点に予め記録されている子孫の数を質点が受けとり，受けとった数が質点の総数になれば終了とする．この方式ではルートセルでのメッセージ処理がボトルネックとなり，また参照の局所性が考慮されていないために PE 内で同一のリモートデータを何度も要求することになってしまう．実際，この方式のプログラムを Intel Paragon XP/S 上に実装された *OCore* のプロトタイプ処理系上で実行した結果，高い台数効果は得られなかった．

第 2 の方法は，冗長性を最大限に持たせたもので，各 PE がすべての質点のデータをもち，BH 木全体をローカルにつくる．力の計算は，各質点オブジェクトが自 PE のローカルな木を辿りながら行なうため，通信は発生しない．更新された質点データは PE 間で大域操作により交換しあう．このときに通信をまとめることができるため通信レイテンシの大きな計算機向きといえる．しかし，i) メモリネックとなり大規模な問題に対するスケーラビリティがない，ii) 計算と通信をオーバラップしにくい，などの欠点をもつ．

第 3 の方法は，各 PE が予め割り当てられた質点のみをもち，それらを端点とし，local-root-cell を根する部分木をローカルに生成する．その後各 PE は他の PE と local-root-cell を交換しあい，部分木をマージする．図 1 にこの様子を示す．いま 2 台のプロセッサ PE#0, PE#1 に (a), (b) の粒子が割り当てられたとすると，それぞれの BH 木による表現は (c), (d) のようになる．ここで PE#0 上で (d) を (c) とマージすることを考える．図中 α のように対応するセルが空である場合には，それ以下の部分木はコピーせず，本体をスロットとしてもつ proxy オブジェクトを格納しておく (図中 (e) から (f) への点線矢印)．一方もし対応するセルが空でない場合は，リモートデータを再帰的に local-tree に挿入していく．

この方法では力の計算時にリモートデータをキャッシングすることが可能となる．力の計算は local-root-cell から :traverse-tree メッセージを木に流し BH 木を辿ることにより行なわれる．proxy オブジェクトは，そのメッセージを受けると，アクセス中を示すフラグをたて，本体からデータをフェッチしてくる．その際に同期構造体 [3] を用いることによりレイテンシを隠蔽することが可能となる．すなわち，同一 PE 内の他の質点オブジェクトからの当該データへの後続する要求はキューイングされ，他の active なコンテキストへと処理が切替えられる．当該待ちデータが到着すると，待ち状態にあった全スレッドはリジュームされる．

本手法によるプログラムを実行した結果，十分な台数効果が得られた．

なお，proxy によるキャッシングが有効に働くためには，各 PE の持つ部分木が disjoint なものである必要がある．ここでは [2] と同様の方法を用いて，3 次元の浮動小数点座標を，空間的に連続な 1 次元の整数値へと変換し，この値により初期データを分散している．さらに負荷分散のためには，前時刻での力の作用数を負荷とし，それが均一となるよう質点データを割り当て直す必要があるが現在はまだ行なっていない．

4 おわりに

OCore による N 体問題のプログラミングについて述べた．並列プログラムを高速化するには，データ参照の局所性を活用する必要があるが，同期構造体を利用することにより，不規則な通信パタンをもつプログラムに対しても，リモートデータのアクセスに伴う通信遅延を隠蔽しつつデータを再利用することができた．

今後，これらの効率的な記述と実行ならびにコードの再利用を目指し，メタレベルアーキテクチャや共同体などを拡張していく予定である．

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, Stanford University, June 1992.

[2] M. S. Warren and J. K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Proceedings of Supercomputing '93*, Mar. 1993.

[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.