動的に生成されたオブジェクトを扱うループの並列化手法

3E−2

稲垣 達氏　丹羽 純平　松本 尚　平木 敬

東京大学大学院理学系研究科情報科学専攻

1　はじめに

分散メモリ計算機において、ユーザーが定義する動的なオブジェクト単位の共有名前空間というインタフェースは、動的で不規則なデータ構造を持つ問題を記述できることに加え、オブジェクトというコンシステンシ管理の単位をアプリケーションからのヒントとして与えることができ、またコンパイラやライブラリによる通信の最適化を仮想化できるという利点がある。

これまでにも完全に動的な共有名前空間を提供したシステム [4] から、inspector/executor アルゴリズム [3] による不規則な配列のアクセスを伴う従来の SPMD スタイルのプログラミングモデルをユーザー定義のデータ構造に拡張したもの [1] まで、数多くの研究が成されている。これらのシステムではいずれも低レベルの通信に関する記述をユーザーに解放して最適化の余地を残し、オブジェクト単位のコンシステンシ管理をライブラリや言語システムによって仮想化している。我々は最適化コンパイラに力点を置いたアプローチによって、ユーザー定義の動的なデータ構造を扱う SPMD プログラム／逐次プログラムに対して大域的な名前空間を提供する研究 [5] を行なっている。本稿では inspector/executor アルゴリズムを適用できるようなオブジェクトを扱うループにおいて、ループ本体の計算と通信をオーバーラップするためのコード生成手法について述べる。

2　オブジェクトをアクセスするループの表現

大域オブジェクトはオブジェクトに対するポインタを介してアクセスされる。例えばある大域オブジェクトのリストをアクセスするループは図 1のようなコードとして表せる。 N は *Object* 型のリストの次の要素をアクセスする演算子である。ループボディの中で o を更新していなければ、静的単一代入形式のデータフローグラフは図 2になる。 網を掛けた部分がこのループでアクセスするイテレーションを生成するコードである。

struct *Object* $*o, *o_0$;
for $(o = o_0;\ o;\ o = N(o))$
  $S(o)$;

図 1: 大域オブジェクトをアクセスするループの例

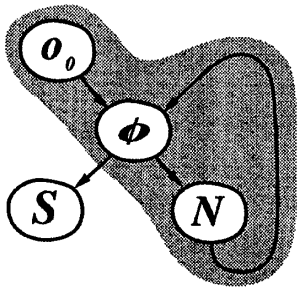

図 2: 静的単一代入形式による表現

3　通信と計算のオーバーラップ

このようなループに inspector/executor アルゴリズムを適用して、大域的な参照を予め列挙することを考える。このループがもし並列実行可能なループ、すなわちイテレーションの実行順序が変更可能なループであれば、inspector 実行時に遠隔データを参照するイテレーションと、局所データだけを参照するイテレーションとに分割することで、gather による通信と executor の局所計算をオーバーラップすることができる [3]。

各イテレーションを特定しているのは、イテレーションを生成している値、図 2では ϕ 関数のノードからループボディの文 S に達している値である。従ってイテレーションを分類する際には、この値がどの範疇に属するかを記録しておけばよい。図 1のコードであれば、*Object* 型へのポインタ o の値がイテレーションを代表するので、*Object* 型へのポインタを格納する配列を用意して遠隔データを参照するイテレーションと局所データだけを参照するイテレーションを分類すれば良い。これらの配列を inspector を起動する前に確保するには、元のループのイテレーションの総数を何らかの形で評価する必要がある。これはイテレーションを生成するコードがインデクスの

Parallelization of Traversal Loop for Dynamically Allocated Objects

Tatsushi INAGAKI, Junpei NIWA, Takashi MATSUMOTO, Kei HIRAKI

inagaki@is.s.u-tokyo.ac.jp

増減であれば簡単に静的にコードから判定できるが、リスト構造を走査するような場合は動的に求める必要がある。このためのコードはイテレーションを生成するコードのループボディをカウンタで置き換えたものになる。もし先に同じイテレーション生成コードを持つようなループが存在していれば、カウンタループをオブジェクトのデータ構造が変化していないコードの範囲で移動し、loop fusionを適用してイテレーション総数を求めるオーバーヘッドを削減することも可能である。

局所イテレーションと遠隔イテレーションへのポインタを一つの配列にまとめて、最終的に図3の示すコードが生成できる。

```
struct Schedule  *s;
struct Object  **O;
PackAndSend (s);
for (i = 0; i < n_local; i++)
  S(O[i]);
ReceiveAndUnpack (s);
for (i = 0; i < n_remote; i++)
  S(O[n_total - i - 1]);
```

図 3: 局所的な計算と通信をオーバーラップしたコード

4 考察

コード変換の効果を検証するため、以下の実験を行なった。当研究室で開発中のコンパイラシステムとランタイムライブラリ [5] を用いて、不規則なメッシュ格子点の値を隣接点の値を用いて反復計算するアプリケーション EM3D[2] のコード生成を行なった。現在の実装ではコンパイラは共有オブジェクト空間で計算を行なう SPMD コードを入力として、分散メモリ計算機用の通信コードを生成する。バックエンドコンパイラとして gcc を用いた。上記の手法に従って、生成されたコードに手動でコードの変換を行ない AP1000+ 上での実行時間を計測した。 表 1の入力データは頂点数 2,048、辺数 4,562(遠隔参照する辺の割合は 10%) で、 16 台で実行して executor の部分の実行時間を計った。反復の回数は 30 回である。 send/receive の時間は AP1000+ の cap_trace を用いて計測したものである。表の計算時間で示される項目は実際の計算を行なっている部分である。実行時間から計算時間と send/receive の時間を引いたものがオブジェクトから各プロセッサに送るメッセージを生成する pack/unpack 操作にかかる時間である。

	実行時間	計算時間	send	receive
base	86,819	14,336	14,949	21,441
overlap	85,174	14,641	14,085	20,428

表 1: EM3D の実行時間 (μs)

オーバーラップの効果によって受信の待ち時間が僅かに削減されているが、計算時間は増加している。計算のコストが増加してしまうのは、コード変換によって最内ループにおける間接参照が増え、メモリアクセスの数が増加したことと、メモリ参照の空間的な局所性が損なわれたことによる。

オーバーラップの効果が全体の実行時間にあまり現れていない理由は、現在の実装では計算とのオーバーラップができない、メッセージの pack/unpack にかかる時間がかなり大きいからである。さらに EM3D では個々のメッセージの長さが短いため、通信のレイテンシは pack/unpack にかかるコストに比べると非常に小さい。

メッセージが長く通信の遅延が大きい場合、オーバーラップできる局所的な計算の量は大きくなる。 pack/unpack のコストは論理的には通信のコストであるため、 pack/unpack 処理と計算がオーバーラップできれば、今回のような例においても本質的な計算以外のコストを緩和することが可能である。

参考文献

[1] Chialin Chang, et al. Object-Oriented Runtime Support for Complex Distributed Data Structures. Technical Report CS-TR-3428, University of Maryland, March 1995.

[2] David E. Culler, et al. Parallel Programming in Split-C. In *Proc. of Supercomputing '93*, pp. 262–273, November 1993.

[3] Charles Koelbel and Piyush Mehrotra. Compiling Global Name-Space Parallel Loops for Distributed Execution. *IEEE Trans. on Parallel and Distributed Sys.*, Vol. 2, No. 4, pp. 440–451, October 1991.

[4] Daniel J. Scales and Monica S. Lam. The Design and Evaluation of a Shared Object System for Distributed Memory Machines. In *Proc. of 1st Symp. on Operating Systems Design and Implementation*, November 1994.

[5] 丹羽 純平, 稲垣 達氏, 松本 尚, 平木 敬. 分散メモリ型並列計算機における共有オブジェクト空間の 実現. 情報処理学会研究報告, August 1996. (掲載予定).