マイグレーションを支援する分散集合オブジェクト

近山・田浦研究室　　電子情報工学科 303838 高橋 慧

1　背景と目的

インターネットの普及により、様々な環境の計算機を接続し、一つの大きな計算資源とする、グリッドコンピューティングが脚光を浴びている。現在までにも専用並列計算機に匹敵する成果が上がっているが、さらに大きな成果を上げるためにいくつかの要請がある。

一つは、プロセッサ数の増減に対応することである。現状のグリッドは研究所のクラスタを用いる方法が主流だが、複数のクラスタを接続すればより大きな問題を解くことが出来る。しかし、各クラスタを占有できる時間は限られている場合が多い。この場合、計算途中にクラスタに新たに展開したり撤退したりできないと、計算資源を十分に活用できない。また、耐故障性の確保や、普通のパソコンを計算に参加させるPC グリッドのためにも、プロセッサ数の変化に対応が必要である。

もう一つは、記述を簡単に行いたいという要請である。現状では多くの並列プログラムは通信 (メッセージパッシング) を用いて記述されているが、これは下層の処理に近すぎて、簡潔にアルゴリズムを表現できない場合が多い。これに対し、逐次プログラムで実績のあるオブジェクト指向を用いれば、通信を隠蔽し、簡単に複雑な処理を記述できる。

しかし現状では、オブジェクト指向で並列プログラムを記述できる分散オブジェクト技術は存在するものの、多くの並列プログラムはメッセージパッシングで記述されている。これは、分散オブジェクト技術でプロセッサの増減に対応しようとすると、無駄な通信が多く発生し、性能が悪化してしまうためである。このため、プロセッサ数の増減に対応したプログラムは、多くが通信を用いて手続き的に記述されている。

本研究では、動的なプロセッサ数の増減に対応し、オブジェクト指向で記述が容易で、無駄な通信が発生しない記述モデルとして、「分散集合オブジェクト」を提案する。

2　関連研究

2.1　メッセージパッシング

メッセージパッシングモデルは、現在現在最も広く使われている並列プログラム記述のフレームワークである。代表的な実装としては MPI[1] が挙げられる。これは相手プロセッサを指定してメッセージを送受信するもので、下層の通信 API に近いものになっている。このためは無駄のないプログラムが書ける一方、記述が手続き的で簡単ではない。また、通常のメッセージパッシングではプロセッサは番号で識別するため、プロセッサ数が増減した場合、メッセージの送信先を動的に変えなければならない。

2.1.1　Phoenix モデル

Phoenix モデル [2] では、メッセージパッシングを基本としつつ、メッセージの送信先のプロセッサの指定方法として、番号の集合を用いることにより、プロセッサの増減によらず確実にメッセージの送受信が行えるようになっている。

計算前に決まった番号の集合を考え、これを計算を担当するプロセッサで過不足無く分割する。計算に参加する場合は、他のプロセッサから集合の一部を割り当てられ、離れる場合は他のプロセッサに集合を委譲する。ここで、メッセージの送信はこの番号を指定して行い、送信されたメッセージは指定されたを持つ番号をプロセッサによって受信される。プロセッサが担当する番号は変化し得るが、メッセージが失われることは無い。

このモデルでは、プロセッサの増減に対応できる。また、通信の際に特定の親を持たないので、ルーティングに関するボトルネックは無いが、記述効率は従来のメッセージパッシングから改善されていない。

2.2　分散オブジェクトモデル

複数のプロセッサで使えるオブジェクト指向の記述フレームワークとしては、分散オブジェクトモデルが提案されている。並列計算ではあまり使われていないが、CORBA[3] がアプリケーションサーバーの接続などで成功を収めている。

プログラマはまずクラスを定義する。このクラスのインスタンスをあるプロセッサ上で作り、他のプロセッサにこれへの参照を持たせる。すると、この参照からリモートのオブジェクトのメソッドが呼び出せる。これを RMI (remote method invocation) と呼ぶ。このメソッドの処理は、呼び出されたオブジェクトを持つプロセッサが行う。

プロセッサの増減には、マイグレーションで対応できる。これは、あるプロセッサで生成されたオブジェクトを、動的に他のプロセッサに移せる技術である。これにより、計算に参加したプロセッサがオブジェクトの割り当てを受けたり、計算から脱退するプロセッサが保持しているオブジェクトを退避させたりできる。

しかし、このモデルを用いてプロセッサ数の増減に対応した並列プログラムを書こうとすると、記述性か性能が損なわれてしまう。大きな配列を扱う例でこれを説明する。

例えば、大きな配列を更新していくプログラムについて考えてみる。配列データは各プロセッサに配分することになるが、プロセッサの増減に対応するには、外側からの見え方は変わらないようにするのが望ましい。そこで、次のようなモデルが考えられる。配列は実際のデータを持つ「断片オブジェクト」と、要素とプロセッサの対応を把握する「全体オブジェクト」から構成される。メソッドの呼び出しは全て全体オブジェクトを通じて行う。プロセッサが増減した場合には、断

片オブジェクトを分割・併合・マイグレートすることで配分を変更する。この変更は、全体オブジェクトが管理する。

このように記述すると、プロセッサ数の増減に対応したプログラムを簡単に書ける一方、どのようなメソッドを呼ぶ際にも「全体オブジェクト」にメッセージが送信されるため、ここが性能上ボトルネックとなる。このボトルネックを解消するには、親プロセッサを無くして、プログラマが断片オブジェクトを直接把握する方法が考えられるが、これはプロセッサ数が変化する環境で書きやすいモデルとは言い難い。

3 分散集合オブジェクトの概要

本研究では、大きな配列などインデックスにより識別される集合を扱う際に、プロセッサ数の変化に対応した並列プログラムを簡単に記述できる、「分散集合オブジェクト」を提案する。これは、配列のようにインデックスによって識別されるデータを扱う際、データが複数のプロセッサに分散していても、プログラムの記述上は一つの大きなオブジェクトに見えるものである。

3.1 記述方法

プログラマはまず断片クラスを定義する。ここで、メソッドは引数に「10 番から 20 番」のようなインデックスの集合を取るものとする。これらのメソッドは、仮想的な全体オブジェクトへの参照からアクセスする。全体オブジェクトへの参照はコンストラクタによって得られるほか、断片クラス内部からは whole キーワードによってアクセスできる。

メソッドの呼び出しは、「全体オブジェクト」に対し、メソッド名と操作の対象となるインデックスの集合を指定する。すると、インデックス集合と重なりを持つインデックス集合を持つ全ての断片オブジェクトで、インデックス集合を引数としてメソッドが呼び出される。例えば、[0-10)、[10-20) のインデックス集合を持つオブジェクトに対し、[5-15) を指定してメソッドを呼び出すと、二つのプロセッサ 1 で [5-10)、[10-15) を引数としてメソッドが呼び出される。(カッコは整数の範囲を表す)

マイグレーションについては、split()・merge()・serialize() のメソッドを記述すると、join() 関数と leave() 関数が自動生成される。この関数により、そのプロセッサ内に断片オブジェクトを受け入れたり、オブジェクトを他のプロセッサに退避させたり出来るようになる。

3.2 特徴

本フレームワークの特徴は、インデックスによって識別できる集合を用いる並列プログラムを簡単に記述でき、プロセッサ数の増減に対応できることである。

記述に関しては、オブジェクト指向の記述ができ、意識せずにデータとタスクの関連性が保たれる。一方で、メソッドの呼び出しはインデックス集合によって行われるが、この際呼び出しのメッセージは一度ある「全体オブジェクト」に届くのではなく、直接各断片オブジェクトに届く。このため、メソッド呼び出しの際に全体オブジェクトがボトルネックにならない。

マイグレーションについても、オブジェクト自体を分割・併合でき、しかもメソッドの呼び出しは分割の単位を意識せずに行えるので、常にプロセッサ数に適合したデータ配分で計算を行うことが出来る。

4 現状と今後の予定

現在、リモートからメソッドが呼び出し可能な分散配列オブジェクトを作成している。これに適当なメソッドを追加することで、マイグレート可能な FFT や SOR のプログラムが記述できる。

今後は、実際のアプリケーション記述において必要になる機能を考えつつ、システムについての実装を進めていく。

まずは先述の分散配列オブジェクトのテンプレートを完成させ、アプリケーションを記述する。その後、一般的なクラス記述に用いるテンプレートの形にし、またメッセージの受信スレッド、メッセージ定義などを自動生成できるような書式を考え、システムを構築する。

先述したように返り値については当面サポートせず、上記のアプリケーションも返り値を用いない形で記述する。その後、プログラムの変換により返り値をサポートするような書式・システムを考えていきたい。

参考文献

[1] The Message Passing Interface(MPI). http://www-unix.mcs.anl.gov/mpi/ .

[2] Kenjiro Taura. Phoenix: A Parallel Programming Platform Supporting Dynamically Joining/Leaving Resources (In Japanese). In IPSJ SIG Notes(HPC-87-24), July 2001.

[3] Object Management Group. http://www.omg.org/ .