論文の内容の要旨

論文題目　　大量のコンピューティングリソースを活用するためのソフトウェア基盤
氏　　名　　横　山　大　作

本研究は、大規模な並列・分散環境において利用可能な大量のコンピューティングリソースを、有効に活用できるようなソフトウェア開発のための基盤技術を構築することで、実社会の大規模問題の専門家が、効率よく問題を解けるようになることを目標としている。

近年、並列・分散計算環境は急激に普及している。一般のパーソナルコンピュータでもマルチコアプロセッサを搭載することが多くなり、コモデティ化した計算機を集めた PC クラスタが多くの場所で導入され、身近なものになっている。ネットワーク技術の発達により、インターネットを用いて高性能計算機を接続した GRID や、遊休計算機を集めたデスクトップグリッドなど、より広域に広がった分散計算環境も実用的な存在となってきた。このように、近年は多数の計算資源を一度に利用できるような並列環境が普及してきており、今後はさらに多くの計算資源を、より多くの場面で利用できるように発展していくと考えられる。

しかし、現在のソフトウェア開発技術を用いてこのような環境を十分に活用することは難しい。「活用」とは、

(1)計算資源を効率よく使えること

(2)正しい並列プログラムが書けること

の両方が成り立っていなければならないが、その両方ともに現在のソフトウェア開発技術では未解決な問題点が存在する。本研究では、(1)についての問題点の一つとして「並列アルゴリズムの計算コストを正しく見積もれる計算量モデルの不在」に注目し、これを解決することを第一の主題とする。さらに、(2)の難しさを緩和し、実行効率の良いプログラム

を正しく簡単に構築可能にすることを第二の主題とする。このような二つのソフトウェア基盤技術を提案・実装することで、大量の計算資源を利用する際の障害を総合的に緩和し、多くの人が、簡単に大量の計算資源を活用できるようになることを狙うものである。

まず第 2 章では、計算資源を効率よく使うようなアルゴリズム開発に役立つ、新しい計算量モデルの提案を行う。

実行効率の良いプログラムを設計するためには、あるアルゴリズムがある環境ではどれくらいの時間で実行できるのか、を正確に把握するための計算量モデルが必要になってくる。これまでにいくつかの並列計算量モデルが提案されてはいるが、これらは単純過ぎて現実の計算環境を表現できなかったり、特定の環境や特定のアルゴリズムに限定したモデルになっていたりと、満足できるものにはなっていない。アルゴリズム設計において必要とされるのは、特定の環境における実行時間の精密な見積もりよりは、現在のさまざまな計算環境とこれから使用可能になるであろう計算環境とで、対象アルゴリズムがどのような振る舞いを示すのか、を定性的に示すようなモデルである。現在、規模もネットワーク構成も大きく異なる多種の計算環境が利用可能であり、これら並列環境の構成方式のトレンドも日々変化している。このような多種多様な環境を、何らかのモデル化によって統一的に扱い、与えられたアルゴリズムが共通にどのような振る舞いを起こすのか、という実行性能予測を可能にするような計算量モデルが求められているのである。

そこで、計算コストは通信コストであるという基本理念の元に構築した新しい並列計算量モデル「アクセス計算量モデル」を提案し、その有効性を実証することを試みた。アクセス計算量モデルの提案では、単体計算機内部のメモリ階層から、LAN、インターネットといったグローバルなネットワークを用いた通信までを、統一的にかつ簡潔に表現し、ネットワークトポロジなど並列計算環境の具体的な構造に依存しない、一般性の高い計算モデルを構築しようと考えた。

アクセス計算量モデルでは、メモリがある密度で広がる空間を考え、計算はメモリ上の任意の地点で、任意の密度で行えるとする。メモリ空間には何らかの距離の指標が定義されており、計算が行われている場所からメモリ上のある地点に存在するデータへアクセスする際には、その距離に従ったアクセスコストがかかる。計算にかかるコストは、演算に必要なデータを計算場所まで移動し、演算結果を所定のメモリ位置に書き込む、という通信にかかる時間コストのみであり、演算そのものにはコストがかからないものとする。計算自体は同一点でいくらでも行うことができるが、通信路には容量があり、通信路の混雑によって計算に必要なデータが届かないため、同一点でいくらでも多くの計算を同時に実行できるわけではない。

この章では、以上の概念に基づくモデルを提案した。より詳細に、1 次元のメモリ空間と、局所的な負荷が表現できる通信路からなるモデルを定め、そのようなモデル上でのプログラムを表現できる仮想機械を定義、設計した。また、よく知られたいくつかの並列アルゴ

リズムについて解析的に計算量を求め、実計算機上の実行結果を用いた評価を通して、モデルの妥当性と実用性を示した。

次に、第 3 章で、効率よく動作する正しい並列プログラムを簡単に開発するための、並列化フレームワークを提案する。

バグのない並列プログラムを書くことは、それ自身難しいことである。並列計算環境にはさまざまな種類があり、それぞれの構成において計算性能を上げやすいプログラミングモデル、通信ライブラリなどが異なっている。ある環境である問題を並列に解きたいときにどのような計算モデルや通信ライブラリを選択・組み合わせるのか、という決断はプログラマに深い知識と経験を要求する。また、計算環境が変化するたびにプログラムの変更が必要になるようでは生産性も低く、バグも入りやすい。よって、さまざまな環境で同一のプログラムが正しく動くようなプログラミング手法が求められる。さらに、大規模問題を解くためには長期間、多数の計算機を使う必要があるが、長期間同じ計算機を占有し続けられる状況は珍しく、多くの場合は使用可能な計算機が増減する。また、多数の計算機を使うとどこかで故障が発生する確率が高まるため、部分的な故障で計算全体が停止してしまうようなプログラムでは計算が進まない。よって、動的な計算資源の増減への対応、耐故障性の実現などが求められるのであるが、プログラマが自分で実現するには深い知識と多大な労力が必要となる。

プログラマの負担を減らすための一つの方法に、並列計算フレームワークがある。ある問題領域に適用範囲を絞り、その問題に特有の並列化手法を専門家があらかじめ実装しておくことで、プログラマは並列化手法の詳細について悩むことなく並列プログラムを書くことができる。また、このフレームワークは並列環境の差異を隠蔽し、それぞれの環境で効率のよい実装を使い分けるなどして実装されているため、ひとつのプログラムで、さまざまな環境で効率よく計算することが可能になる。耐故障性の導入なども、問題領域を絞ることでプログラマへの負担を軽減した形で実現できる。

科学技術計算などの構造が単純な問題においてはこのようなフレームワークが存在し、高性能計算を簡単なものにしているが、より複雑な構造を持つ問題、例えば組合せ最適化やゲーム木探索などに代表される探索問題については、まだ研究が不十分な点も多い。探索問題は実社会においても応用範囲が広く、規模が大きくなると莫大な計算量を必要とするため、並列計算の技法を用いて実用的な時間でより大きな問題を解けるようにすることが強く求められている。探索問題において特に問題になるのは、共通の部分問題に依存した多くの部分問題が存在する点である。単純に並列化を行うと、これらの共通問題を別々の計算機で重複して計算してしまうため、無駄な計算が増えて探索性能の向上に結びつかない。共通部分問題を効率よく発見し、計算結果を再利用しながら並列計算を行うような手法が求められている。そこで、このような性質を持つ探索問題をターゲットとする並列計算フレームワークを設計・構築し、並列プログラム開発のための基盤技術とすることを

試みた。

提案する並列探索手法は、親問題が子問題を再帰的に生成し、子の計算結果を使って親の結果を計算するような構造の問題を対象とし、分散ハッシュ表(DHT)の考え方を用いて部分問題を管理する。対象とする問題領域では、部分問題は共通性が発見できるよう一意にコード化されるので、ハッシュ表を用いて共通部分問題を効率よく発見し、計算結果を再利用しながら探索を進める。DHT は近年研究が盛んであるが、多数の計算機でファイルなどの情報を保持するための手法として捉えられていることが多い。また、DHT の考え方を探索に用いる研究もあったが、分散システムの重要な特性である耐故障性能について考慮したものは存在していなかった。今回の提案では、故障時には失われた部分問題を必要とする親問題が子問題を再実行し、必要な探索途中結果を DHT 上に再構成する、という方法で耐故障性を実現する。

この章では、提案する手法が、共通部分問題を含む問題領域において良い探索効率を与えること、故障が発生した際も性能悪化が少ないことを、既存の分散計算手法であるマスタワーカとの比較シミュレーションによって示した。また、提案する手法をフレームワークとして利用できるようにインタフェースを整理し、実装を行い、シンプルな探索問題を用いて評価を行った。

これら二つの領域の問題点の両方に対する研究がそろって初めて、現在身近になり、これからさらに普及し続けると考えられる並列計算環境を「活用」することができると考えている。本論文では、新しい並列計算量理論と、共通部分計算を持つ探索問題の大規模耐故障並列化フレームワークの提案を行うことで、並列プログラムをさまざまな環境で効率よく、正しく簡単に実装できるような基盤技術を拡充することができた。