計算連続体モデルに基づく仮想機械の仕様

An Abstract Machine on the Computing Continuum Model

横山 大作 † 近山 隆 †
Daisaku YOKOYAMA Takashi CHIKAYAMA

† 東京大学新領域創成科学研究科
Graduate School of Frontier Sciences, the University of Tokyo
{yokoyama, chikayama}@logos.k.u-tokyo.ac.jp

近年、計算機のメモリ階層は深化し、RAM モデルと現実との乖離は著しいものがある。また、クラスタや大規模分散計算等、各計算機間の通信遅延やその違いを無視できない並列環境が一般的になっているが、現状の並列計算コストモデルは、通信遅延の差異を考慮しないものや特定のネットワークトポロジに特化したものしか存在しない。我々は、単一計算機内のメモリ階層から計算機間のネットワーク遅延の差異までを統一的に記述できる計算量モデル、「計算連続体モデル」を提案した。本稿では、このモデルの検証の基礎となる仮想機械の仕様について述べる。

1 はじめに

あるアルゴリズムの計算コストを考える際、すべてのメモリが単位時間でアクセスできるとする RAM モデルが多くの場合で使われ続けてきた。しかし、近年の計算機アーキテクチャは、小量の速いメモリと大量の遅いメモリが存在するという階層構造を持ち、しかもそのメモリ階層は年々深くなり続けている。このため、RAM モデルでの計算コストの見積りと、現実の計算機での実行結果との乖離が大きなものとなってしまい、RAM モデルはもはや適用できないのが現実である。

これに対し、メモリへのアクセスコストが一様ではないとするモデルがいくつか提案されてきた。Hierarchical Memory Model(HMM) [1] や Uniform Memory Hierarchy Model(UMH) [2] がそれである。HMM は、メモリのアクセスコストがアドレス x に対して $f(x)$ で与えられるというものであり、メモリの中でアドレスの小さい部分は CPU に近く、速くアクセスすることができるが、大量のメモリを使おうと思うと次第に CPU から遠い、遅いメモリを使うようになる、ということを表現できるようにしたモデルである。UMH は、大きさやレイテンシが再帰的に決められた一連のメモリモジュール群というものを考え、それらがバスで相互に接続されたものがメモリである、とするモデルである。つまり、CPU の近くには小さくて速いメモリモジュールがあり、次第に大きくて遅いモジュールがつながっていく、という構造である。これらのモデルはアルゴリズムの解析に成果を挙げてきたが、並列アルゴリズムは対象としていなかった。

並列計算のモデルでは、LogP[3] が良く知られている。これは、計算機間が 4 つのパラメータを用いて表現された均一な通信路によって相互接続されている、とするモデルである。しかし、現在普及しつつある大規模な分散計算環境では、各計算機間の通信路が均一ではなく、遅延が大きいプロセッサとそうでないプロセッサ、すなわちプロセッサの「遠い近い」が計算時間に無視できない影響を与えてしまうため、このモデルも現実との乖離が大きくなりつつある。LogP の発展モデルとして、さらにパラメータを付け加え、詳細に通信路特性を記述しようとする試みもいくつか見られるが、これらはすべて通信路のモデル化に留まっている。

今、並列計算アルゴリズムの優劣を議論しようとした場合、単体の CPU 内のメモリ階層による影響と、CPU 間の通信による影響を別々のモデルによって評価し、それを合わせて全体の計算コストを見積もるという、複雑な手順を取らなければならない。また、単体の CPU の構成が違う、あるいはネットワークトポロジが違うものを考えようとすると、モデルの再設定、及び計算コストの評価のやり直しが必要となってしまう。計算機構成のトレンドは年々変化しており、トレンドが変化するたびにモデルを変更しなければならないのでは、アルゴリズムの設計に対する指針としては使いにくい。

さらに、非常に多数の計算機が結合した環境で、す

べての通信路を独立に扱うことがもはや困難になる場合などもある。このように、並列計算環境を「複数の計算機がネットワークでつながったもの」としてとらえることは、モデルの複雑化を招き、またこれから大いに発展すると思われる大規模分散環境に対応できないアプローチであると考えられる。

我々は「計算機の内側と外側の区別、単体計算機内の計算とネットワーク上の通信の区別をなくし、それらを統合した 1 つの計算機モデルとして表現したら良いのではないか」と考えた。これにより、アルゴリズム設計の際、計算コスト評価の労力が著しく緩和される。また、このモデルではすべてのメモリ間に距離 x が定義できるものとし、それぞれのメモリ間の通信コストが「距離による単純な関数 $f(x)$」で表されるものと仮定する。この $f(x)$ によって、任意の単体計算機内部のメモリ階層、任意のトポロジによるネットワークのふるまいを、統一的にかつ簡潔にに表現し得るのではないか、というのが我々の主張である。

このような並列計算のコストモデルを、我々は「計算連続体」と名付けた。本稿では、この計算連続体の概念を示し、そのモデル上の計算を実現する仮想機械の設計について述べる。

2 アーキテクチャと計算の概観

計算連続体とは、以下のようなモデルである。

- メモリはある密度で連続的に (現在のところ 1 次元で) 分布したものであるとする。
- 計算する資源はメモリ上の至る所に存在している。
- プログラムの実行時間はメモリアクセスの時間が支配的であり、演算そのものにかかる時間は無視できるとする。ただし、演算のためには必ずメモリ上からデータを持ってくる必要があるので、見掛け上無限に速い演算ができるわけではない。
- メモリへのアクセスには「格納された場所」から「計算する場所」までの「距離」に従ったコストがかかる。x だけ離れた場所のメモリにアクセスするときは、指令からある時間 $f(x)$ だけたった後にアクセスが行われる。x の点でのアクセスにはある一定の時間 $l, l > 0$ がかかる。さらに、読んだデータを送り返す時間または ack を返す時間が $f(x)$ だけかかるため、メモリアクセス命令全体では $2f(x) + l$ だけの時間がかかる。ack を返さない書き込み命令はない。
- $f(x)$ は単調増加関数であり、$f(0) = 0$ が成り立つ。また、$0 < x < y$ としたとき $f(y) < f(x) + f(y - x)$ が成り立つ、すなわち上に凸な関数である。これはつまり、どこかで通信を中継することで速いアクセスが可能になるということはない、ということを示している。
- メモリアクセスそのものの衝突は考えない。メモリアクセスの際に使われる通信路が混雑することで、アクセス競合の振る舞いを捉えようと考える。これは、データを近傍に並べるのではなく、適宜分散させて計算した方が速い場合があることを反映させようという狙いである。
- 計算の実体は「プログラムカウンタ」と「実行場所」を持っており、PC を進めつつ、メモリ中で実行場所を移動させて計算を行う。
- 命令を取ってくる時のコストは考えない。
- 計算の実行場所が移るときもメモリアクセスと同じ時間コスト $f(x)$ がかかる。メモリアクセスを計算実体が追い越すようなことはできない。
- メモリが 1 次元であるモデルの場合、通信路は正方向行き/負方向行きの 2 つが独立して存在する。正方向へ送られるパケットと負方向に送られるパケットは衝突しない。
- 通信路は層状に積み重なっている。通信の衝突は各層内でのみ発生する。各層の通信速度は、底の方ほど遅い。通信は一番底の通信路からスタートし、一定時間毎に隣の層へと移動していく。この時、世界全体で共有されたパルスが存在すると仮定している。一定時間毎に世界全体でパルスが発生し、そのパルスに従って通信路が一斉に状態を変える。
- 各通信路の速度は $f(x)$ によって規定される。例えば $f(x) = O(\log(x))$ の場合、高さ h の通信層の速度を $v(h)$ として、$v(h) = kv(h-1)$ であれば $f(x)$ は実現できる。(図 1 参照)
- 通信は、現在使っている通信層の速度 $v(h)$、データ長 n に対して、$(n+1)v(h)$ に相当する広さの領域を占有する。

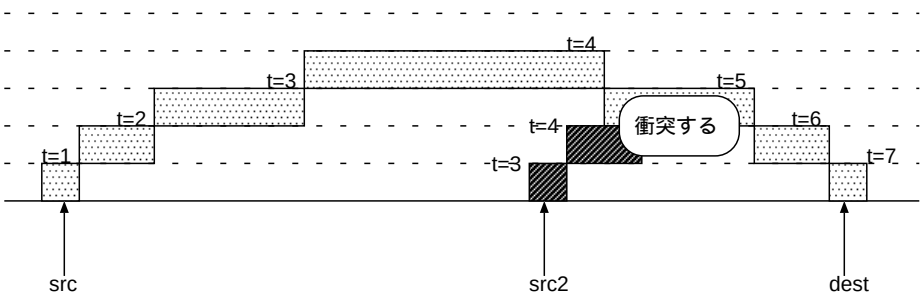

図 1: 通信路と衝突

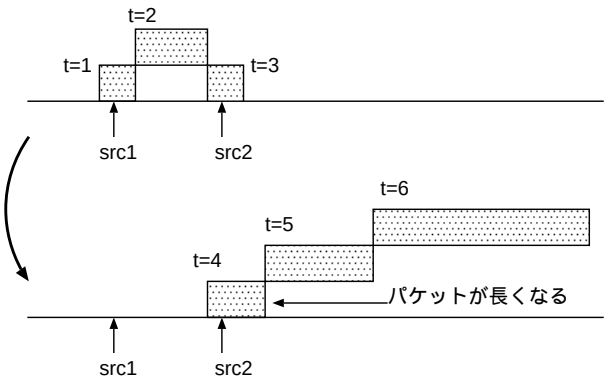

図 2: 2 ワードを 1 パケットにまとめて送信する場合

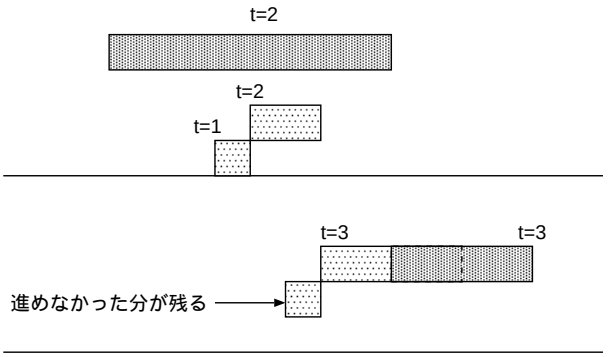

図 3: 通信路がふさがっている場合

- パケットは任意のデータ長を一度に送ることができる。複数ワードを送信する際は、送信するデータが置かれた場所を次々にパケットが訪問し、次第に長いパケットに成長していく。データを受信する側でも同様に次第にパケットが短くなりながらメモリ上に書き込みが行なわれる。(図 2 参照)
- 次の単位時間に占有するべき領域が空いていない時、その領域が空くまでパケットは待つ。空きが足りない場合は、できる限り進もうとする。2 つのパケットが衝突している時は、底の通信路へ降りていくパケットを優先させる。(パケットが減る方向を優先する) (図 3 参照)

3 仮想機械の命令セット

計算連続体モデルは、次のような命令セットを持つ仮想機械であると定義できる。

以下の説明で、<*input*> , <*output*> は即値 (const <*value*>) 及び、ダイレクト (direct <*address*>)、インダイレクト (indirect <*address*>) のアドレッシングモードをとるメモリ内容である。なお、アドレスは現在の計算実体が存在している場所からの相対位置を指定する。

各々の命令は現在の実行場所で実行され、実行後 PC を 1 つ進める。

3.1 演算

- (<*operation-name*> <$input_1$> <$input_2$> <*output*>)

  $input_1 \cdot input_2 \rightarrow output$ という演算を行なう。

  <*operation-name*> = add, sub, mul, div, 等の基本演算

3.2 メモリ

- (copy <*src*> <*dest*>)

  $src \rightarrow dest$ というメモリ内容コピーを行なう。メモリ上に定数を書き込みたい時もこの命令を使う (*src* が即値になる)

  <*src*> , <*dest*> は

- 即値 (const <*value*>)
- (area <*memory*> <*size*> <*stride*>)

  <*memory*>の場所にある<*stride*>間隔で並べられた<*size*>個のデータ。

  <*memory*>の場所にある<*size*>のデータ。<*memory*>はダイレクト (direct <*address*>)、インダイレクト (indirect <*address*>) の 2 つのアドレッシングモードをとる。アドレスは現在の実行場所からの相対指定。<*size*>は即値。<*stride*>は 1 以上の整数値。

  全体で $\{address, address + 1 \times stride, ..., address + (size - 1) \times stride\}$ のデータを指す。

<*src*> のみ、さらに

- プログラムカウンタ (pc)
- 現在の実行場所 (place)

も指定できる。

3.3 実行場所の制御

- (next_place <*next-place*>)

  次の実行場所を指示。<*next-place*>は (place <*diff*>) であり、現在の場所から相対的に記述。

3.4 制御

- (jump <*program-point*>)

  無条件ジャンプ。<*program-point*>の命令を実行。

  <*program-point*> は (pc <*program-counter*>) または (label <*label*>)

- (branch <*input*> <*program-point*>)

  <*input*> の場所のメモリを読み込み、その値によって taken ならば<*program-point*>の方へ分岐。

- (fork <*program-point*>)

  別の計算実体を作成し、<*program-point*>の命令を、現在の実行場所で実行させる。自分は次の命令を現在の実行場所で実行する。

- (compare_and_swap <*src*> <*org-value*> <*update-value*> <*program-point*>)

  <*src*>場所のメモリを読み、<*org-value*>と一致したら<*update-value*>で置き換え、<*program-point*>の計算を続ける。<*org-value*>と一致しなかったら<*org-value*>を<*src*>場所の内容で置き換え、not taken で下に抜ける。

- (vanish)

  計算実体が消える。

4 まとめと展望

本稿では、単体計算機とネットワークとを統一的に表現し、個々の計算機の内部構成やネットワークトポロジを簡潔に抽象化した、「計算連続体モデル」について、その概念とモデルを表す仮想機械の設計について述べた。今後、このモデルの妥当性を評価するため、仮想機械のシミュレータの構築、その上でのさまざまなアルゴリズムの評価と現実の並列計算環境との比較を行なっていく。

謝辞

本研究は文部科学省科学研究費特定領域研究「ＩＴの深化の基盤を拓く情報学研究」の一環として行った。

参考文献

[1] A. Aggarwal, B. Alpern, A. Chandra, and M. Snir. A model for hierarchical memory. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pp. 305–314, May 1987.

[2] B. Alpern, L. Carter, E. Feig, and T. Selker. The uniform memory hierarchy model of computation, 1987.

[3] David E. Culler, Richard M. Karp, David A. Patterson, Abhijit Sahay, Klaus E. Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: Towards a realistic model of parallel computation. In *Principles Practice of Parallel Programming*, pp. 1–12, 1993.