Vol. 41　No. 6　　情報処理学会論文誌　　June 2000

アクセス計算量: 新しい並列計算量の枠組の提案

横 山 大 作† 　近 山 　 隆†

近年の計算機ハードウェアの進化により，既存の計算量理論ではアルゴリズムの解析に十分とはいえない局面が多くなってきている．計算機のメモリ階層は深化し，RAM モデルと現実とは著しく乖離している．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「アクセス計算量モデル」を提案した．このモデルは，計算のコストの本質は演算ではなく通信にこそ存在するという立場をとる．このモデルは十分簡潔なものであり，並列アルゴリズムの計算量を解析的に求めることができることを，いくつかのアルゴリズムで実証することができた．また，bitonic sort アルゴリズムの実計算機上での実行と比較することで，このモデルの妥当性を示すことができた．

Access Complexity: A New Framework for Complexity of Parallel Computation

Daisaku Yokoyama† and Takashi Chikayama†

Recent advances in computer hardware has been making existing computational complexity theory inappropriate for many cases. The random access memory (RAM) model was made unrealistic by large speed gap between the processing units and main memory systems. Distributed computing environments have obsoleted traditional models for parallel computation due to non-negligible diversity in communication delay. In this paper, we propose a new framework for computational complexity, named *access complexity*, in which the cost lies in data transfer rather than computation itself. The model tries to capture all levels of system hierarchy, from cache systems to globally distributed environments. It models these diverse access costs in a simple and uniform way. We apply the model to analyze some parallel algorithms, to show that the model can analyze well-known algorithms easily. We also show the appropriateness of the model, through comparing the predicted and the measured performance of bitonic sort algorithm.

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著しいものになり，RAM モデルはもはや適用できないのが現実である．

例えば，図 1 は SPARC III 1.2GHz, 8CPU の計算機において配列の要素へランダムにアクセスする際の 1 要素あたりのアクセス時間を測定したものである．詳細は後述するが，横軸の要素数 (の log) が増えるに従い，アクセス時間は 1000 倍近くまで増大する．この振舞いは，とても RAM モデルでは表現できない．

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，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を考え，それらがバスで相互に接続されたものをメ

† 東京大学新領域創成科学研究科
Graduate School of Frontier Science, the University of Tokyo

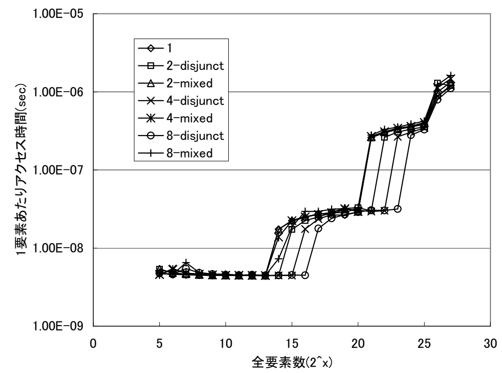

図 1　ランダムアクセステスト (SPARC)

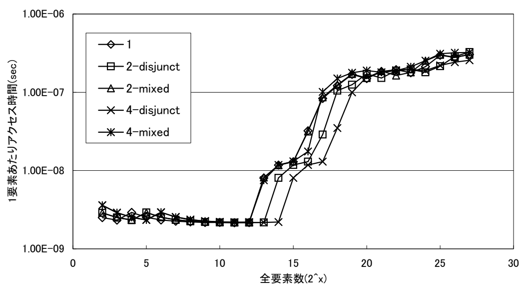

図 2　ランダムアクセステスト (Opteron)

モリ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，次第に大きく遅いモジュールが連なっていく，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，これらは全て通信路のモデル化に留まっている．

また，Coarse Grained Multicomputers (CGM)[4] や，Bulk-synchronous Parallel (BSP)[5] といった並列計算モデルもある．これらは，ある程度の量のメモリをローカルに持つプロセッサが，何らかのネットワークで相互に接続されている，と計算環境のモデル化を行ない，その上で「ローカルな計算フェーズ」と「通信フェーズ」のコストを別々に考え，加算することで全体の計算量を把握しようというモデルである．

また，BSP を基本に，通信路の振舞いや計算機性能が場所によって異なるような計算機環境をモデル化する Heterogeneous BSP[6] も提案されている．計算フェーズ，通信フェーズが，並列計算全体で同期している BSP モデルであるため，通信が最も遅いところに揃うことで通信路の構成に起因する振舞いをとらえようとしている．ただし，このモデルでの通信時間はプロセッサ入口のバンド幅と送受信するデータ量によってのみ決定されるというシンプルなものであり，通信路の構造，通信の局所性などは考慮していない．

これらのモデルのように，単体計算機内の計算と計算機間の通信とを別々に考慮する場合，どうしてもモデルの組合せによる複雑さが生じる．また，単体の CPU 内のメモリ階層による影響を表現するモデルと，CPU 間の通信路を精密に表現するような別々のモデルを組み合わせるという複雑な手順を取って，ある特定の計算環境を精度よく表現することが可能になったとしても，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環境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は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「計算のコストとは，演算にかかる時間にあるのではなく，演算に必要なデータを取得し，演算結果を格納するという通信過程こそに存在する」という基本理念に基づいて，計算量を統一的に捉え直そうと考えた．メモリ階層はデータへのアクセス遅延時間の違いが生み出すものであり，ネットワークトポロジの違いを考慮すべきなのも，他の計算機にあるデータへのアクセス遅延時間の違いが無視できないからである．

この基本理念をもとに「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 つの仮想計算機として表現したモデル」によって，新しい計算量モデルを構築することを考えた．このモデルでは全てのメモリ間に距離 x が定義できるものとし，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，任意のネットワークの振舞いを，統一的にかつ簡潔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．

このモデルは，特定の計算機環境における計算時間を極めて精密に予測するというよりは，多くの並列計算機環境で共通した振舞いをよく予測できるような，一般性の高い指標を目指している．図 2 は，図 1 の実験を Opteron 1.4GHz, 4 CPU の計算機で実行した結果である．メモリアクセス時間の増大はキャッシュなどの構造に依るので，2 つの環境では振舞いもかなり違っているが，全般的な増大の仕方に共通点もある．さらに，この実験はローカルメモリの大きさまでの測定だったが，ネットワークでつながった別の計算機のメモリを使って同じ実験を行なえばこのグラフの先もやはりアクセス時間は増大していくと予想される．その際も，ネットワーク性能やネットワークトポロジなどの違いによって細かい振舞いは異なる．しかし，環境に依存しない増大の仕方を，ローカルメモリからネットワーク越しのメモリの範囲まで全て，$f(x)$ という単純な関数で表現するよう割り切ってしまおう，というのがこのモデルの基本的な方針である．

Thomas rauber らの Locality measure[7] は，メモリアクセス履歴を用いた簡単な指標で，並列アルゴリズムの実行環境に依存しない実行性能を表現しようとするものであるが，我々のモデルも同じ方向を指向している．ただし，Locality measure がメモリアクセスの履歴から間接的な定性的要素を取り出しているのに対し，我々のモデルはより直接的に，計算機の構造を意識した指標を作ろうとしている．

このような並列計算コストのモデルを，我々は「アクセス計算量」と名付けた．本稿では，このアクセス計算量の概念と，仮想機械のアーキテクチャの設計，仮想機械語の設計を示す．また，いくつかの並列アルゴリズムへの計算量解析への適用を通してモデルの妥当性と適用可能性について示す．

2. アーキテクチャと計算の概観

2.1 モデル概要

アクセス計算量とは，以下のようなモデルに従う計算量である．

- 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．メモリ上には「計算実体」と呼ばれる，計算を行なう「もの」が存在し，メモリ上の好きな地点で計算を行なうことができる．
- 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したり 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，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は衝突しないが，メモリアクセスの際に使われる通信路が局所的に混雑すると，競合によりメモリアクセスが遅くなると考える．これは，データを集中させず，適宜分散させて計算した方が速い場合があることを表現するためである．
- 計算実体は「プログラムカウンタ」と「実行場所」のみを持っており，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．計算実体自身にレジスタのようなメモリはない．
- 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 $f(x)+l$ 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．
- 通信路にはある有限の容量があり，容量を越えた通信についてはペナルティが追加された時間コストがかかるものとする．これにより，計算主体自身は任意の並列性で存在できたとしても，計算全体の速度向上上は通信路の混雑により制限される．

今，1CPU しかない普通の計算機をアクセス計算量モデルで表現すると，1 次元のメモリ上に計算実体が 1 つだけ存在する状態となる．計算実体の直近のメモリは，アクセスする際の距離が小さい，読み書きのレイテンシが最も短い場所であり，現実の計算機ではレジスタにあたる．その隣には，少し距離が大きくなっ

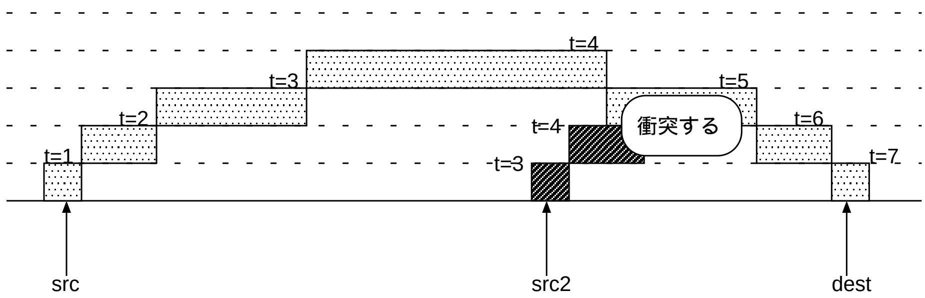

図 3 通信路と衝突

た L1 キャッシュ相当のメモリがあり，以後段々遠くに L2 キャッシュ，メインメモリが置かれていると考える．現実の計算機ではレジスタ，L1 キャッシュ，L2 キャッシュ，メインメモリとレイテンシが増大するに従って容量は大きくなる．このレイテンシと容量に合わせて $f(x)$ を定義すれば，現実の計算機のアクセスコストを模倣することができる．一般的には，レイテンシと容量では容量の方が急速に増大する傾向にあり，HMM の研究では $f(x) = \log(x)$ 程度が適切であった．ただし，近年はレイテンシの増大が激しくなっており，適切な $f(x)$ の設定については研究を要する．

現実の計算機では，メインメモリにアクセスするとアクセス対象の近傍のメモリがキャッシュメモリに移され，以後のアクセスが高速になるが，アクセス計算量モデルではキャッシュはアルゴリズム設計者が明示的に使用しなければならない．つまり，このモデルの仮想機械が持つブロック転送の機能を用いて遠いメモリから近くのメモリへとデータを移し，近くのメモリを使って計算を行なう，という動作を明示的に行なわせる必要がある．ただ，このモデルでは計算実体が自由に場所を移せるため，データを取ってくる代わりに，計算実体自身をデータの近くへと移すことでもキャッシュと同じ効果が得られる．

次に，SMP マシンをこのモデルで表現すると，メモリ上にプロセッサ数だけの計算実体が置かれた状態になる．図 1 と 2 の実験を例に，SMP マシンの動作がこのモデルでどのように表されるかを説明する．

図 1，2 の実験は，連続したメモリ領域の要素が 1 本のリストとしてつながっており，リストをたどって全要素を読むことで全メモリ領域をランダムに 1 回ずつ読む，というものである．系列 1 は 1 スレッド，2- から始まる系列は 2 スレッド，のように複数スレッドで実験を行ない，複数スレッドでも全要素数 N は 1 スレッドと変わらず，各スレッドが (スレッド数 P として)N/P 個の要素を読む．-disjunct の系列は，N の連続メモリ領域を N/P 個ずつの連続領域に分け，各スレッドはそれぞれの部分領域の中だけでアクセスをした場合で，-mixed の系列は各スレッドが N の中からランダムに N/P 個のアクセスをした場合となる．

1 スレッドの場合，わざわざレイテンシの大きいメモリを使う必要はないので，計算実体は N の要素のどこかに居る．アクセス対象メモリまでの平均距離は $N/2$ であり，要素数が増えるとアクセス時間は $f(N/2)$ に従って増大していくのがわかる．-disjunct の場合，計算実体は N/P の要素のどこかに居るのが自然だろう．この時，計算実体からアクセス対象メモリまでの平均距離は $N/2P$ である．それ以外は 1 スレッド実行の時と違いがない．そのため，図を見るとわかる通り，-disjunct のアクセス時間は要素数 $1/P$ の 1 スレッド実行時のグラフときれいに同じ形をしている．一方，-mixed の場合，計算実体が-disjunct と同じ場所に居るとしても，アクセス対象までの平均距離は 1 スレッド実行と同じ $N/2$ となる．そのため，-mixed のグラフは 1 スレッド実行のグラフときれいに重なっている．このように，このモデルは並列実行の様子を比較的に直接的な形で把握できることを目指している．

ところで，メモリが 1 次元であることによる現実との乖離ももちろん存在する．3 個並んだ計算実体が互いに相手の近くのメモリを読む時，端の 2 つの間の方が現実 SMP でのレイテンシが大きいとは考えにくい．クラスタのような分散環境でも同様である．また分散環境の場合，計算機間に複数の通信経路がある場合や，通信方向によってレイテンシに差がある場合などもありうる．このような現実を，1 次元のメモリモデルでは扱うことができない．しかし，例えばメモリと通信路を 2 次元以上に拡張し，2 地点間の距離と通信経路を適切に定めることができれば，このような現実も扱うことが可能になる．ただし，多次元の拡張を行なうと距離の定義が難しく，混雑を迂回する通信路も考えられるため解析が困難になる．一般的なネットワークの表現には何次元が適切なのかもよくわからない．以上のような理由で，現状のアクセス計算量モデ

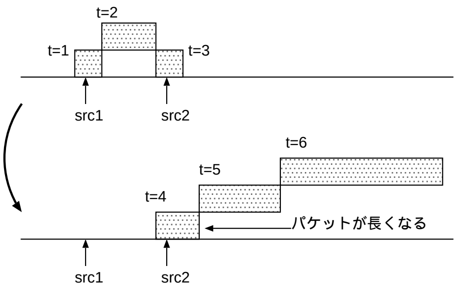

図 4　2 ワードを 1 パケットにまとめて送信する場合

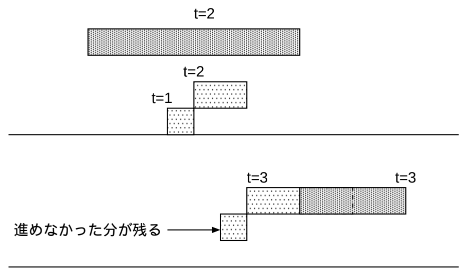

図 5　通信路がふさがっている場合

ルでは，メモリと通信路は 1 次元であると仮定する．

2.2　複層の通信路モデル

上述のモデルでは，並列実行時の計算の振舞いは通信路によって大きく規定される．計算と同様，通信路の混雑の局所性も計算量を見積もる際に大きな影響を与える，という立場から，我々のモデルでは全ての通信をパケットととらえ，パケットの動きを追跡することで通信路の振舞いを把握することにした．以下に，この「複層の通信路モデル」の概要を示す．

- 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．正方向と負方向に送られるパケット相互は衝突しない．
- 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 $v(h)$，k を 1 より大きい適切な定数として，$v(h) = kv(h-1)$ とすればよい．(図 3 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．1 はパケットが持つ定数のコストを示しており，n が大きいパケットを用いるほど通信路の利用効率が上がることを表現している．
- 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 4 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，パケットは空きを待つ．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 (パケットが減る方向の) パケットを優先させる．(図 5 参照)

このような通信路は，いわばキャッシュメモリからメインメモリへと連なるメモリバス，さらに Local Area Network，Wide Area Network へという階層的な物理的通信路のモデル化を狙ったものである．一番底の層が CPU 内部にあるレジスタやキャッシュとのバス，次の層がメインメモリのためのバスで，その上が LAN，WAN という意識である．上の層の方が速度が速いが，その一方 1 つのパケットがより広範囲に影響を及ぼすようになるのは，イーサネットで隣の計算機と通信ができるが，複数台の計算機が一斉に通信を始めると輻輳を起こしてしまう，といった振舞いをモデル化しようとしたものである．このような構造は，CPU 内部のメモリバスでも，大規模なネットワーク環境でも，共通して見られる構造であると考えている．

2.3　通信負荷累積モデル

我々は，このような計算モデルに基づく仮想機械を設計 (後述) し，このモデルでの計算の様子を把握できるシミュレータを構築した．また，この仮想機械上でのプログラミング手法を提案し，いくつかのアルゴリズムをシミュレータ上で実行することで計算モデルの妥当性や有効性を検討してきた．その結果，複層の通信路モデルは現実のシステムを確かに良く表現するが，解析的に計算量を求めるにはやや複雑すぎることもわかった．そこで，通信路の混雑度合をより単純にモデル化した「通信負荷累積モデル」を作成し，解析的手法による計算量の見積もりを可能にしようと試みた．

通信負荷累積モデルでも，通信を担うパケットの各時点での速度，及びそのパケットが影響を与える範囲は複層の通信路モデルと同じである．結果，距離 x だけ離れた 2 地点間の通信は少なくとも $f(x)$ だけの時間がかかる．2 つのモデルは，通信の集中による通信路の混雑の表現についてのみ異なっている．

パケットは現在占めている通信路の領域にある負荷

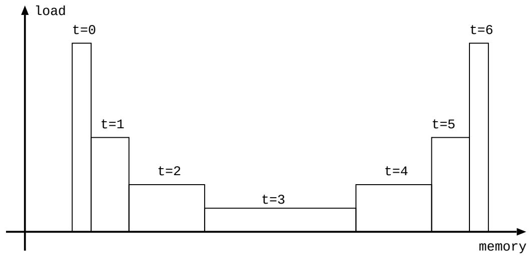

図 6 通信負荷累積モデルでの 1 つのパケットの負荷

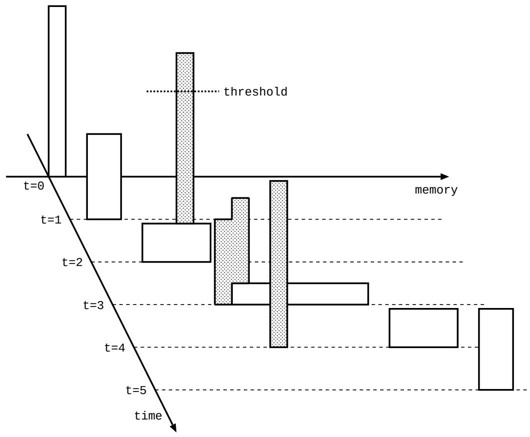

図 7 通信負荷累積モデルでの複数のパケットの負荷

を与える．複数のパケットが同じ領域に存在する時，それぞれの負荷が加算されてその領域に与えられる．通信路には容量が定められており，ある領域の負荷が容量を越えた時，その時刻の通信路全体に渡って混雑による通信遅延が起きるものと考える．

パケットは，速度に応じて負荷を与える範囲が変わる．単位量の通信パケットは全体としてはある一定の負荷を持っており，ある時刻での地点 x での負荷を $l(x)$ とすると，

$$\int l(x)dx = constant \qquad (1)$$

が成り立つ．つまり，速度が上がると広い範囲に負荷を与えるが，その場合ある地点に与える負荷の値は小さくなる．このような $l(x)$ の形はさまざまなものが考えられ得るが，以降の議論では，ある時刻のパケットの速度を v とした時，v の範囲に均一に $\frac{1}{v}$ の負荷が与えられる，という単純なモデルを採用する．

図 6 に，1 つのパケットのみが存在する時各時刻において通信路に与えられる負荷を示す．パケットの速度，存在するメモリ上の位置は図 3 と同様に変化しているが，通信路に層は存在せず，1 つの通信路に与える負荷 ($load$) が速度に応じて変化している．

図 7 は，2 つのパケットが衝突する様子を示している．$t = 0$ で 1 つ目の，$t = 2$ で 2 つ目のパケットが発生し，ともに右に向かって進んでいく様子を表しているが，$t = 2, 3$ では 2 つのパケットが通信路の同じ領域に存在するため，2 つの負荷を加算したものがその時刻の通信路負荷となる．$t = 2$ で，2 つの負荷を合計したものがある通信路容量 $threshold$ を越えているため，この時刻に通信路中に存在する全ての通信はペナルティを受けて遅くなる．ペナルティについては，負荷 l が容量 T を越えた時，通信は通常の l/T 倍だけ時間がかかるというモデルを想定している．

このような通信路は「複層になった通信路モデル」と比較して，現実の通信路の特性をいくつか無視したものとなっている．例えば，複数のパケットの負荷を単純に加算してある地点の通信路負荷を求めているため，接続された LAN 内の通信が輻輳を起こしているとローカルのメモリの読み書きが遅くなる，というモデルであると言える．しかし，このような単純化を行なうことで，複数の通信が同時に行なわれる時の通信路の振舞いを簡単に表現でき，解析的手法で計算量を求めることが可能になると期待される．

3. 仮想機械

これまで述べてきた計算モデルは，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*> は即値及びダイレクト，インダイレクト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・メモリ

(<*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, 等の基本演算

(**copy** <*src*> <*dest*>) $src \rightarrow dest$ というメモリ内容コピーを行なう．ブロック転送も可能．

3.2 実行場所の制御

(**next_place** <*next-place*>) 次の実行場所を現在との相対位置で指示．

3.3 制 御

(**jump** <*program-point*>) 無条件ジャンプ．<*program-point*>の命令を実行．

(**branch** <*input*> <*program-point*>) <*input*> の場所のメモリの値によって分岐．

(**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．

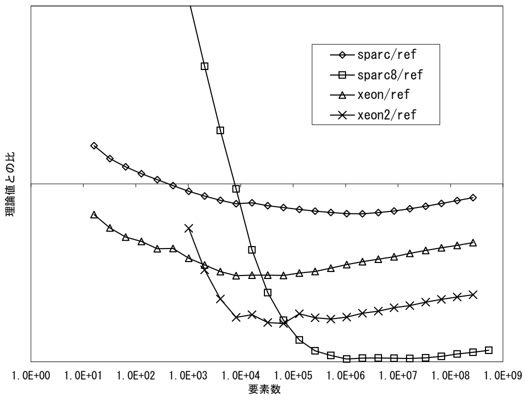

図 8 実計算機と RAM モデルの計算量比較

(compare_and_swap <*src*> <*org*> <*update*> <*pp*>**)**
<*src*>の内容を compare and swap．

(vanish) 計算実体が消える．

これらの命令セットを用いてアルゴリズムを記述することで，全ての計算に必要なデータの配置とその間の通信を明記することになり，アクセス計算量モデルに基づいた計算量を求めることができる．

我々は，この仮想機械プログラムを解釈実行し，その計算の振舞いを把握できるシミュレータを実装した．また，仮想機械上でやや高水準なプログラミングが行なえる言語 CEMA[8] を設計，実装した．複雑な並列アルゴリズムの計算量を考察する場合においては，このようなシミュレーションを用いていく．

4. 並列アルゴリズムの解析

ここでは，よく知られた並列アルゴリズムである，bitonic sort，merge sort，FFT を例にアクセス計算量モデルでの計算量解析を行なう．また，bitonic sort を実計算機上で実行し，(P)RAM モデルとアクセス計算量モデルの解析結果と比較する．

4.1 bitonic sort

bitonic sort アルゴリズムは，sorting network の一種である．規則的な比較演算を全データに対して繰り返すアルゴリズムであり，比較を行なう要素の対がデータに関わらず固定されているため，並列計算の適用も比較的容易である．

n 要素の bitonic sort は $\frac{n}{2}\frac{\log n(\log n+1)}{2}$ 回の比較演算を含むため，RAM モデルでの計算量は $O(n(\log n)^2)$ となる．また，並列計算の際には $\frac{n}{2}$ 回の比較操作を並列に実行できるため，並列度 R のとき $O(n(\log n)^2/R)$，最高で $O((\log n)^2)$ の計算量となる．

実計算機上での実験結果を図 8 に示す．計算環境は

- Ultra SPARC III Cu 1.2GHz, 8 CPU
- Pentium4 Xeon 2.4GHz, 2 CPU

である．いずれも pthread ライブラリを用い，共有メモリを使って並列化を行なっている．4 つのグラフ系列は，2 つのアーキテクチャそれぞれで single thread 実行時と 8(2)thread 実行時の 2 通りの計算時間を示している．横軸はデータ数 n，縦軸は RAM モデルでの理論計算量と実際の計算時間の比 (分母が理論計算量) である．ただし，縦軸は適当に倍率を決めたものであり，SPARC どうし，Xeon どうしの系列の値の比は計算時間の比を示しているが，SPARC と Xeon の系列の縦軸の値の比，ならびにその絶対値は何の意味も持たない．

図 8 の通り，データの量が大きくなるにつれて理論値との比が単調に増加している．つまり，RAM による単純なモデル化では把握できないメモリ階層などの要素が現実の計算時間を押し上げており，RAM モデルの計算量と現実との乖離が起きているのである．

次に「アクセス計算量モデル」による解析を行なう．アクセス計算量モデルでは，データの配置と計算実体の位置が計算量を決定する．実験に用いたプログラムでは，データは最初に連続領域に配置されその領域内でソートされる．このとき，n 要素 bitonic sort に対して一度呼ばれる n 要素 bitonic merge は，$i=0$ から $n/2-1$ まで i を順に変えつつ i 番目と $i+n/2$ 番目要素の比較を行なう．

これをアクセス計算量モデルで解析する時，データ配置は元プログラムの配置に従えばよいが，計算実体の場所は通常のプログラムには概念が存在しないため，解析者が適切に定める必要がある．今回の場合，要素に順番にアクセスするためメモリキャッシュにデータが乗っていることが多いと考えられる．より正確な解析のためには，次に比較に使う 2 つの要素付近のメモリを計算実体そばにブロック転送し，キャッシュの効果を再現する方がよいが，今回は比較に使う片方の要素の上に計算実体が移動することでキャッシュを再現することにした．この場合，もう片方の要素はやや遠い位置に存在することになるが，その距離は常に一定になるため，それほどひどいアクセスコストを生じず「順番に行なうアクセス」の振舞いを表現できるのではないかと考えた．また，メモリの距離 x に対するアクセスコストは HMM の結果より $\log x$ と定めた．

このとき，n 要素 bitonic sort に対して一度呼ばれる n 要素 bitonic merge 内での比較演算は距離 $n/2$ のメモリアクセスを行なっているので，$O(\log n)$ の遅延が起きることになる．これを再帰的に適用していく

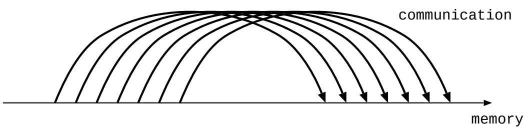

図 9　連続した点からの同一方向への一斉通信

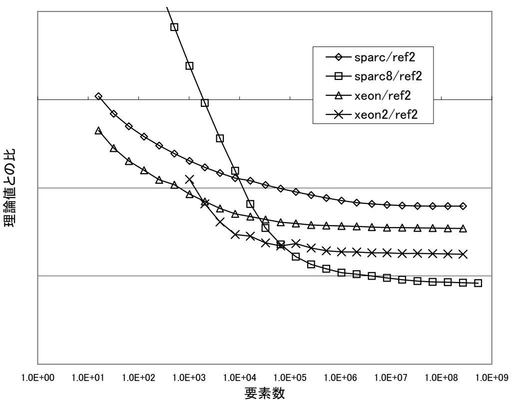

図 10　実計算機とアクセス計算量モデルの計算量比較

と，bitonic sort 全体の計算量は $O(n(\log n)^3)$ となる．

また，P プロセッサでの並列実行時には $n/2P$ だけ離れた箇所で同じ距離の通信が並行して行なわれる．最も通信路が混雑するのは $P = n/2$ の時であり，連続した P 個のメモリ上の点から，P だけ離れた点への通信が一斉に行なわれることになる．(図 9 参照)

このような場合について，通信路の「通信負荷累積モデル」を用いて解析する．図 9 の状況において，任意の地点 x の通信路の負荷は，同一の負荷の形をしたパケットが距離 1 ずつずれながら重なっているので，1 つのパケットのある時刻の負荷を $l(x)$ として，$\sum^d l(x-d)$ で表される．これは，1 つのパケットの持つ負荷の総量が一定であるという制限 (式 1) より，一定値であるといえる．つまり，複数のパケットが生まれていたとしても，あらゆる地点の負荷は単一のパケットのみが存在した時の負荷を越えず，このような通信パターンは通信路を混雑させないと解析される．

よって，並列計算時でも通信路の混雑は起きず，計算量は $O(n(\log n)^3/R)$，最高で $O((\log n)^3)$ となる．

先の実験の実行時間を，アクセス計算量モデルの理論計算量と比較したものを図 10 に示す．縦軸・横軸は図 8 と同じ意味である．図 8 の RAM モデルでは表現できていなかった要素数の大きい領域において，計算時間と理論値との比は一定に収束していっている．つまり，アクセス計算量モデルは bitonic sort の計算量をうまく見積もれたといえる．

4.2　merge sort

merge sort は並列化手法を決定的に決めることができるアルゴリズムであり，解析は容易である．RAM モデルでの計算量は $O(n \log n)$，並列化すると最も多くのプロセッサを使った場合で $O(n)$ となる．

次に，アクセス計算量モデルでの計算量を考える．まず，n の連続した入力データに対し，隣に同じ大きさのバッファを用意し，その 2 つのメモリ領域を交互に使ってソートを行なうものとする．通信は 2 つのソートされた領域の要素 (2 領域の合計要素数を k とする) を順々に比較する時，及び n だけ離れたバッファに比較後の結果をコピーする時の両方で起きる．これらの通信の中では距離 n の通信が支配的であり，これは k 要素全てがソートされていて，全ての要素が図 9 のような距離 n の通信を行なう時に最悪レイテンシの通信順序になると考えられる．もし，どこかの要素の順序が入れ替わっている時，その部分の通信は「距離 n を 2 回」から「$n-1$ と $n+1$ の距離の通信を 1 回ずつ」へと変化する．この時の通信遅延は $2f(n)$ と $f(n-1)+f(n+1)$ であるが，$f(x)$ が上に凸な関数なので $2f(n) > f(n-1)+f(n+1)$ である．よって，最も遅くなるようなアクセスは全ての要素が距離 n の通信をする時で，通信コストは $k \log n$ である．

この時，ソートされた領域の比較には $2\sum_{i=1}^{k/2} \log i = k \log k - 2(k-1)$ の通信が必要となり，k 要素の merge sort の時間 $T(k)$ についての漸化式 (全体の要素数 n) は

$$T(k) = 2T(k/2) + k \log n + k \log k - 2(k-1)$$

となる．これを解くと計算量は $O(n(\log n)^2)$ となる．また，並列化について考えると，bitonic sort と同様の考え方で通信路は混雑しないと結論づけられ，十分多いプロセッサが存在する場合は $O(n \log n)$ の計算量となる．なお，異なったデータ配置でのアルゴリズム，例えば入力データと作業用バッファを一要素ずつ交互に配置する方式も考えられるが，この配置の場合でもアクセス計算量モデルでの計算量は変わらなかった．

実計算機での実験結果を図 11，12 に示す．RAM モデルとの比較では，SPARC 1CPU，Xeon 2CPU では要素数が大きい領域で理論値比が緩やかに上昇しているが，Xeon 1CPU では理論値とよく一致している．一方アクセス計算量モデルとの比を見ると，SPARC 1CPU や Xeon 2CPU では要素数の大きい領域で予測と一致しているが，全般的に理論値比が低下する方向にある．つまり，本実験のプログラムは，RAM モデルとアクセス計算量モデルの間程度の計算量を持ち，アクセス計算量モデルと完全には一致していない．

これは，小さい領域を連続アクセスし，次はその小

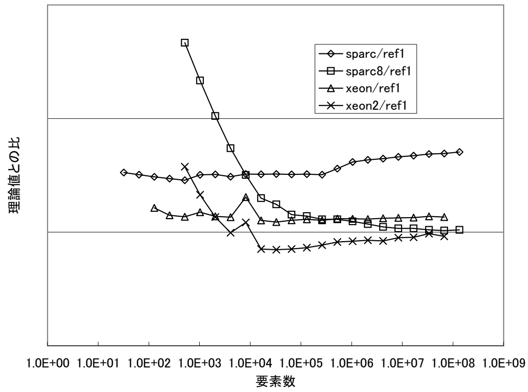

図 **11**　実計算機と RAM モデルの計算量比較

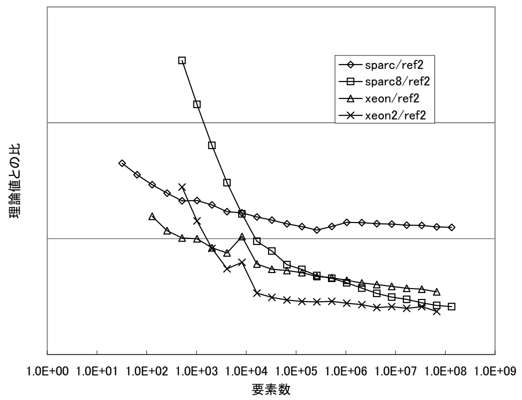

図 **12**　実計算機とアクセス計算量モデルの計算量比較

さい領域 2 つを連続してアクセスする，という merge sort のメモリアクセスパターンが実計算機のキャッシュ機構にうまく当たるのに対し，アクセス計算量モデルで解析を行なう際明示的にキャッシュを使うようなアルゴリズムを採用しなかったためであると考えられる．

4.3 FFT

FFT については，メモリアクセスのローカリティを高めるためのさまざまな手法が知られている．また，演算数を減らすための手法も多い．しかし，工夫を行なっても演算数のオーダーそのものは変化しないこと，また演算数を減らす手法は並列化した時の解析が面倒になることを考慮し，今回は一番単純な形の FFT，つまり n 入力の FFT を「バタフライ回路 $\rightarrow$2 個の $n/2$ 入力 FFT$\rightarrow$ シャッフル回路」と計算する方式について解析を行なう．RAM モデルでの計算量は $O(n\log n)$，十分にプロセッサが存在する時は $O(\log n)$ である．

アクセス計算量モデルでは，n 個の入力データは連続したメモリ上に置かれ，隣接した同じ大きさの作業

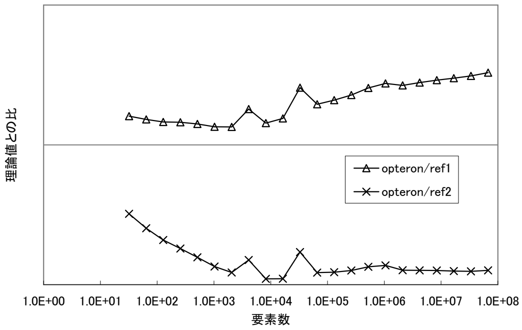

図 **13**　実計算機と各モデルの計算量比較

領域を使って書き換えられるものとする．全体が n 要素の FFT で，k 要素の部分 FFT を考える時，バタフライ回路の通信は $\log(k/2)$ の遅延が k 回，作業領域への書き込みが遅延 $\log(n)$ を k 回，シャッフル回路の通信コストは作業領域から入力データ領域への移動なので $k\log(n)$ である．よって，計算時間の漸化式は

$$T(k) = 2T(k/2) + k\log(k/2) + 2k\log(n)$$

となり，これを解いて全体の計算量は $O(n(\log n)^2)$ となる．また，並列計算時は，bitonic sort と同様の議論で通信路が混雑しないことが示され，十分にプロセッサが存在する場合で $O((\log n)^2)$ となる．

簡単な実験を行なったものとモデルとの比較を図 13 に示す．これは厳密な FFT ではなく，バタフライで三角関数ではなく定数をかけている．通常のライブラリの三角関数は重いため表引きなどの高速化が必要だが，解析の際には高速化に用いるメモリ量を考慮しなくてはならず，アルゴリズムの選定が難しい．よって，今回の簡単な解析ではその部分を無視することにした．参考程度の実験であるので並列化は行なっておらず，実験環境は Opteron 1.4GHz である．opteron/ref1 が RAM モデル理論値との比，opteron/ref2 がアクセス計算量との比である．図 13 の通り，RAM モデルでは理論値との比が増大してしまっているが，アクセス計算量モデルでは正しく実行時間を把握できている．

merge sort と異なり，今回の単純な FFT の場合はシャッフル操作がほとんどランダムアクセスで，実計算機ではキャッシュをうまく使えない．そのため，今回のキャッシュを意識しない解析方法でもうまく計算量が見積もれたと考えられる．

4.4 通信路の輻輳実験

4.1 の bitonic sort の共有メモリによる並列計算では，効率はそれほど落ちることなく並列化が行なわれている．これは連続体モデルの計算量とも合致しているが，通信路のバンド幅がもっと狭い，ネットワーク

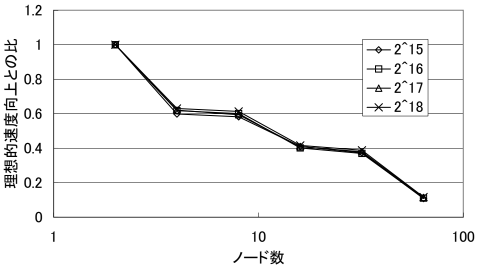

図 **14**　同時通信数と通信速度

接続された分散計算機のような環境では同様の結果となるか，についても確認する必要がある．

そこで，bitonic sort で起きる通信パターン，同じ距離の通信が複数密集して起きる図 9 のような状態を，PC クラスタ上で再現した実験を行なった．実験環境は Xeon 2.4GHz dual, 64 node のクラスタであり，ネットワークは Gbit-ether，通信ライブラリに phoenix[9) を用いた．全体のデータサイズを一定とし，通信を行なうノードの数を変化させた時の通信時間を測定した．この状況では，理想的には台数に比例した速度向上が得られると期待される．結果を図 14 に示す．グラフの系列は全体のデータサイズの違いを示しているが，結果は 4 系列ともほとんど変わらない．横軸はノード数，縦軸は期待される理想速度向上と実際の計算時間との比を示している．図 14 の通り，ノード数が増えるに従って速度向上率が落ちる結果となっている．

つまり，ネットワークの帯域幅が十分に大きい環境では現在の通信負荷累積モデルで精度良く振舞いを予測できるが，帯域幅が小さいところは現在のモデルではうまく説明できていないと言える．通信路を全て統一的に表現することを目指し，通信路のモデル化についてはさらに検討を続ける必要がある．

5. ま と め

我々は，計算のコストの本質は演算ではなく通信にこそ存在するという基本概念から，アルゴリズムの計算量を均一に広がったメモリ上に存在するデータ間の通信の遅延の総和であるとする，アクセス計算量モデルを提案する．このモデルを用いることで，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，かつ簡潔に記述することができる．このモデルは十分簡潔であり，並列アルゴリズムの計算量を解析的に求められることを，merge sort，FFT のアルゴリズムで検証した．また，bitonic sort アルゴリズムの実計算機上での実行と比較することで，従来の PRAM モデルが表現しきれなかった振舞いを，このモデルがよく表現できていることが示された．

ただし，merge sort の解析結果が実計算機とあまり一致しなかったことからわかる通り，キャッシュを明示的に操作するアルゴリズムで解析しなければならない場合も多いと考えられる．さらに，通信路のモデル化には，帯域幅が十分でない環境において，通信の輻輳を必ずしも十分に表現しきれていないという側面もある．これらを今後の検討課題としたい．また，より多くのアルゴリズムやより大規模なアルゴリズムに適用することで，このモデルによる計算量解析の妥当性と適用可能性を検証していきたい．

謝辞　本研究の一部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) Aggarwal, A., Alpern, B., Chandra, A. and Snir, M.: A Model for Hierarchical Memory, *Proc. of the 19th Annual ACM Symposium on Theory of Computing*, pp. 305–314 (1987).
2) Alpern, B., Carter, L., Feig, E. and Selker, T.: The Uniform Memory Hierarchy Model of Computation, *Algorithmica*, Vol. 12, No. 2/3, pp. 72–109 (1994).
3) Culler, D. E., Karp, R. M., Patterson, D. A., Sahay, A., Schauser, K. E., Santos, E., Subramonian, R. and von Eicken, T.: LogP: Towards a Realistic Model of Parallel Computation, *PPoPP*, pp. 1–12 (1993).
4) Dehne, F. K. H. A., Fabri, A. and Rau-Chaplin, A.: Scalable parallel computational geometry for coarse grained multicomputers, *Intl. Journal of Computational Geometry and Applications*, Vol. 6, No. 3, pp. 379–400 (1996).
5) Valiant, L. G.: A bridging model for parallel computation, *Communications of the ACM archive*, Vol. 33, No. 8, pp. 103–111 (1990).
6) Williams, T. L. and Parsons, R. J.: The Heterogeneous Bulk Synchronous Parallel Model, *LNCS*, Vol. 1800, pp. 102–108 (2000).
7) Rauber, T. and Rünger, G.: Program-Based Locality Measures for Scientific Computing, *IPDPS*, p. 164 (2003).
8) 渡邊誠也, 横山大作, 近山隆, 小宮常康, 湯淺太一: メモリ上の配置を意識する並列処理向き高水準機械語の設計と実装, ソフトウェア科学会第 20 回大会 (2003).
9) Taura, K., Endo, T., Kaneda, K. and Yonezawa, A.: Phoenix: a Parallel Programming Model for Accommodating Dynamically Joining/Leaving Resources, *PPoPP* (2003).