計算幾何学と並列アルゴリズム

今井 浩*，今井 桂子**

1. はじめに

計算幾何学では，その応用の広がりとともに，解くべき比較的新しい問題がまだまだ多く，そのような問題に対する研究では，逐次アルゴリズムに関する研究が中心であるといえる．そういう中で，計算幾何の並列アルゴリズムに関する包括的な研究もいくつかあるが，一般的には計算幾何学での並列アルゴリズム研究というのは，成熟するにまだ至っていないような状況である．

しかし，一方で並列アルゴリズムの一般の手法を計算幾何の幅広い問題に適用して，効率的な逐次アルゴリズムを構成するということが，最近注目を浴びている．この手法は，中核となるアイデアの提案者であるMegiddo［5］の名で呼ばれたり，あるいはパラメリトックサーチ（parametric search）と呼ばれたりしている．この手法は，Megiddoの提案の後，Cole［3］によってさらに拡張されている．

本稿では，このような並列アルゴリズムと逐次アルゴリズムの絡みで生まれてきた手法の説明をし，それによって効率よく解ける計算幾何の問題のいくつかについて触れていく．これは，並列計算機でより高速に問題を解くという話ではなく，今の逐次計算機上で問題を高速に解くという話である．最終的に得られるものは逐次アルゴリズムであるが，その設計過程で並列的な発想を必要とするものである．そういった意味で，これを計算幾何での並列アルゴリズムの話として述べるのはおかしいかもしれないが，逆に見れば，出てくるものが効率的な逐次アルゴリズムということですぐに役立つものであるので，ここでは並列性を内部にもった逐次アルゴリズムとしてとりあげていく．

この手法によって従来の方法より効率的に解ける計算幾何問題の中には，多くの施設配置問題も含まれており，この手法自体，数理計画と密接な関係をもっているという意味でも興味深いと思われる．

2. パラメトリックサーチ

まず，Megiddoの与えたパラメトリックサーチの手法について説明しよう．具体的な例題で説明した方がよいが，この手法の面白さの1つは，その考えの漠然としたところから生まれる汎用性でもあるので，一般的な枠組だけはまず大きな枠組で説明しておく．

Megiddoのこの手法は，次のようなものである．並列計算環境としては，理想的なものを考える．すなわちP台プロセッサがあれば，アルゴリズムさえ並列化可能ならば，速度がP倍速くなるというものである．

ある問題Aを解く次のような性能の逐次アルゴリズムと並列アルゴリズムがあったとする：

(1) 問題Aは逐次的に解けばT_s時間の手間で解け，

(2) 問題AはPプロセッサでT_p時間の手間で解ける．

一般に，(2)のような並列アルゴリズムを，1プロセッサしかない逐次計算機の上で$O(PT_p)$時間かけて実行すること（シミュレートすること）は容易である．（逆に，T_s時間の逐次アルゴリズムを自動的に変換して，Pプロセッサの並列計算機で$O(T_s/P)$時間で問題を解くことは，一般には非常に難しく，このことこそが，単に逐次アルゴリズムを研究するだけでなく，並列アルゴリズムを研究する必要性を示している）

この問題Aを繰り返し解くことにより解ける問題Bを考える．問題Bがその中で問題Aを解く回数をCとすると，Aを解くのに逐次アルゴリズムを用いると，問題Bは$O(CT_s)$の手間で解けることになる．一方，Aを解くための逐次アルゴリズムの代わりに並列なアルゴリズムの方（逐次的に実行して）を用いることによって，問題Bを逐次的に$O(CT_pP)$時間で解くことは自明である．しかし，じつは（Pがあまり大きすぎないなどといった条件が成り立っているときには）逐次的に$O(T_pP+CT_p\log P)$時間で問題Bを解くことができるということをMegiddoは示した．

Megiddoのこのパラメトリックサーチ手法は，問題Aとして整列（ソーティング）とそれに関した問題をよく用いる．ただし，問題Aはこれらの問題のみに限られるのでなく，一般に効率のよい並列アルゴリズムが存在する問題であればなんでもよい．ここでは，例として，ある最適化問題Bで，整列問題Aを部分問題とするようなものを考える．

* いまい ひろし 東京大学理学部情報科学科
** いまい けいこ 中央大学理工学部情報工学科

今，整列問題Aに対して速い並列アルゴリズムが得られているとする（実際，存在する）．また，問題Bに対してはAを解くことによって，あとは高速の逐次アルゴリズムを得られるとし，次のような仮定1，2をおく．

（**仮定1**） 問題Bは，整列を行なったのちは，簡単に解くことができるが，整列の過程での2つの要素の比較はある問題を解くことに対応して高価であり，それには定数時間より長い$C(n)$の手間がかかる．普通，$C(n)$は$O(n)$とか$O(n\log n)$である．□

（**仮定2**）（一括処理条件）m個の比較$C_1,\cdots,C_m$を考えた時，これらの比較の集合に次のような意味での順序$C_{\pi(1)}\leq\cdots\leq C_{\pi(m)}$をつけることができる（ここでの比較$C$とは答が“yes”か“no”であるような問い“$c_1<c_2$?”としている）．ここでつけた順序によれば，もし$C_{\pi(j)}$に対する答が“yes”なら，$C_{\pi(1)},\cdots,C_{\pi(j-1)}$の答は“yes”であり，$C_{\pi(j)}$に対する答が“no”なら，$C_{\pi(j+1)},\cdots,C_{\pi(m)}$の答は“no”でなければならない．この条件が成り立っていると，たとえば$C_{\pi(m/2)}$の答をひとつ計算するだけで，少なくとも他の半分の答を決定することができ，これをうまく行なっていくことにより，2つの比較の相対的な順序の決定はかなり速く行なえるようにできる．□

ここまでは，一般的な枠組で話を進めてきたが，ここからは手法が具体的にどのように進むかを例で見ていこう．n個の線形増加関数$f_i(x)=a_ix+b_i$，$a_i>0, 1\leq i\leq n$を考える．中央値関数を$f(x)=\mathrm{median}[f_1(x),\cdots,f_n(x)]$と定義する．$f(x)$も増加関数であることに注意する．ここで，$f(x^*)=0$となる$x^*$を求めることを問題とする（問題$B$）．もし，$x^*$を求めずに$f_i(x^*)=f(x^*)$となる$i$を見つけることができれば，$x^*$を求めたことになる（あと，$f_i(x)=0$となるような$x$を求めればよいから）．

ではどうやってx^*を求めずに$f_i(x^*)=f(x^*)$となるiを求めることができるのだろうか．この$f_i(x^*)=\mathrm{median}[f_1(x^*),\cdots,f_n(x^*)]$となる$i$を知るためには$x^*$を求めることなしに，$f_1(x^*),\cdots,f_n(x^*)$の順序だけ（値ではない）がわかれば，その中央値を取るiが求まる．

つまり問題Bは部分問題A：「x^*を求めることなしに$f_1(x^*),\cdots,f_n(x^*)$をどうやって整列するか？」を解くことにより，解けてしまう．この整列問題Aは「x^*がわからなくても任意の2つの値$f_1(x^*)$と$f_2(x^*)$を比較することができる」ならば，その手続きを比較方法として，適当な整列アルゴリズムに挿入することができる．

そこで，x^*がわからなくても，任意の2つの値$f_1(x^*)$と$f_2(x^*)$を比較することができることを説明しよう．もし，直線$f_1(x)$と$f_2(x)$が並行であればすべてのxに対して$f_1(x)>f_2(x), f_1(x)=f_2(x), f_1(x)<f_2(x)$のどれかが成り立つ．$x=x^*$でもそうである．$f_1(x)$と$f_2(x)$が並行でないとし，一般性を失なうことなく$a_1>a_2$と仮定してかまわない．このときは$f_1(x)=f_2(x)$となる$x=x_0$が一意的に存在する．そして，$x>x_0$に対しては$f_1(x)>f_2(x)$であるし，$x<x_0$に対しては$f_1(x)<f_2(x)$となることがわかる．ここで，もし$x^*>x_0$であれば，$f_1(x^*)>f_2(x^*)$，$x^*=x_0$であれば，$f_1(x^*)=f_2(x^*)$，$x^*<x_0$であれば，$f_1(x^*)<f_2(x^*)$となる．このように比較に答えることは，$x^*<x_0$，$x^*=x_0$，$x^*>x_0$のどれが成り立つかを決定することに帰着される．このためには，$f(x_0)$を（$O(n)$の手間で）評価し，$f(x)$は増加関数であることから，$f(x_0)<0$なら$x^*>x_0$であるし，$f(x_0)=0$なら$x^*=x_0$，$f(x_0)>0$なら$x^*<x_0$であることがわかる．したがって$O(n)$時間（$=O(C(n))$時間）で比較が行なえる．これで，一般的な枠組での仮定1が満たされていることがわかった．

次に，仮定2の一括処理条件が満たされていることを見よう．一括処理条件を示すためには，比較が順序づけられることを示さなければならない．“$f_{i1}(x^*)<f_{i2}(x^*)$?”というm個の比較C_iを考える．ここで$a_{i1}>a_{i2}$，$1\leq i\leq m$とする．各比較は，$x^*>x_i$，$x^*=x_i$，$x^*<x_i$を決めるための値x_iを決定する．これらの値を順番に並べることができる：$x_{\pi(1)}\leq\cdots\leq x_{\pi(m)}$．比較$C_i$にも同じ順序をつける：$C_{\pi(1)}\leq\cdots\leq C_{\pi(m)}$．ここで$x^*>x_{\pi(j)}$なら$i\leq j$に対して$x^*<x_{\pi(i)}$，$x^*<x_{\pi(j)}$なら$i\geq j$に対して$x^*>x_{\pi(i)}$となることに注意すると，$C_{\pi(j)}$の答が得られれば，$C_{\pi(1)},\cdots,C_{\pi(j-1)}$，または$C_{\pi(j+1)},\cdots,C_{\pi(m)}$の答が得られたことになる．すなわち，この答えがわかった側については，具体的に各比較を評価することなくすんでしまうことになる．

以上のような枠組の下でパラメトリックサーチがどのように問題Bを解いていくかを見ていこう．今までBを解くために標準的な整列アルゴリズム（$O(n\log n)$）を用いていたとしよう．そうするとBに対する$O(C(n)n\log n)$の手間のアルゴリズムが得られる．しかしこれではせっかく色々とある性質を全然使いきれていない．

一方，$P(n)$個のプロセッサを用いて$T(n)$並列時間で並列整列アルゴリズムが実行できるとすると，並列アルゴリズムの各ステップでは，高々$P(n)$個の比較を行なっている．これを逐次的に1つ1つ比較していくかわ

りに，一括処理条件を使う方法を考える．すなわち，この $P(n)$ 個の比較に対応する値の中間値を求め，それに関するテストを行なうと，その答から一括処理条件により他の半分の比較の答がわかってしまう．このことを，答のわからなかった片方の半分に対して再帰的に繰り返していくと，$\log P(n)$ 回繰り返したときには（これは，通常 $O(\log n)$ 回に相当する），すべての $P(n)$ 個の比較の答が計算されている．

このアルゴリズムでは，並列整列アルゴリズムでの各ステップで，$\log P(n)$ 回実際に比較を行なうので，これ関して全体で $O(T(n)C(n)\log P(n))$ の手間がかかる．中間値を求める部分については，中央値を求めるための線形時間アルゴリズムを用いる．すると，$\log P(n)$ 回の繰り返しで，まず最初の $P(n)$ 個の中間値が $O(P(n))$ で，以下の $P(n)/2, \cdots, P(n)/2^i, \cdots$ 個それぞれの中央値が $O(P(n)/2), \cdots, O(P(n)/2^i), \cdots$ の時間で求まり，全体で $O(P(n)+P(n)/2+P(n)/4+\cdots)=O(P(n))$ の手間ですべての中間値が求まる．これが並列整列アルゴリズムでの各ステップで行なわれるので，この部分全体で時間 $O(P(n)T(n))$ かかる．

以上をまとめると，問題 B を解くには，$O(P(n)T(n)+T(n)C(n)\log P(n))$ の手間がかかることになる．通常，$P(n)=O(n\log n)$，$T(n)=O(\log n)$ や $P(n)=O(n)$，$T(n)=O(\log n)$ が用いられる．$C(n)=O(n)$ や $C(n)=O(n\log n)$ の場合には，B を解く時間はそれぞれ $O(n\log^2 n)$，$O(n\log^3 n)$ となる．

Megiddo [5] はこのような手法が多くの問題に対して有効であることを示している．ここではその内の1つのみ触れておく．他の例についてはこの Megiddo の手法をさらに拡張した Cole の方法を説明した後で示す．

2種の重みに関してバランスした木を求める問題

グラフの各辺 e に2種の正の重み a_e と b_e が与えられているとき一般に a_e に対する最小重み木と b_e に関する最小重み木は異なったものになる．この2種の重みに関してバランスした最小重み木を求める問題は，線形関数で表わされるような重み $w_e=w_e(\lambda)=a_e+\lambda b_e$ を新たに考え，$F(\lambda)$ で各辺に重み $w_e(\lambda)$ のついたグラフの極大木の重みの総和の最小を表わすとすると，$F(\lambda)=0$ の解を求める問題とみなせる．これはこれらの重みに関する最小比の重み木を求める問題 [2] に対応する．グラフの辺と頂点をそれぞれ E, V としたときこの問題は，上記のように並列計算を概念的に用いることによって $O(E(\log V)^2\log\log V)$ 時間で解くことができる．

3. パラメトリックサーチの拡張

前節で説明した Megiddo の手法は，さらに改良できるだろうか？ 改良できるとすると，どこかに最大限には効率を出し切れていないところがあって，そこを改良するのが常套手段である．アルゴリズム設計とかに詳しい人にとっては，この手法で最大限効率が出し切れていないところを見つけるのはそう難しくない．そのような非効率性があるところは，$P(n)$ 個の比較を $\log P(n)$ 回のステップで行なうときに，最初は $P(n)/2$ 個の比較が行なえるのに対し，ステップが進む毎に実質的に比較が実行される数が半減していき，最終的には定数個の比較しか判定していないところである．ただ，これに気がついたとしても，実際に改良できるかとなると，また別問題である．

Cole [3] は，まさしくこのような観点から，Megiddo の手法の計算時間からさらに $\log n$ 倍速くできることを示した．この手法では，整列を次のような幅 $n/2$，深さ $f(n)$ の整列ネットワーク上のゲームとしてとらえ，ゲームの終了する（整列が完了する）ステップを見積もる．以下，概略のみ説明する．ゲームは n 個の項目に対して，整列ネットワーク上で2人のプレーヤー（ソーターと相手）によって行なわれる．2人のプレーヤーは交互に行ない，まずソーターは相手にいくつかの比較を要求する．すると相手はこれらの比較のいくつかを解く．

ネットワーク上の比較器は，それに対する入力がわかっていて，比較した結果がまだ決定されていない時，アクティブと呼ぶ．ゲームの細かいルールを説明しよう．2人のプレーヤーは次のことを交互に行なう．

(1) ソーターはアクティブな比較器に重みを割り当てる．重みの総和（アクティブ重み）を W とする．C を比較器 $\bar{C}$ に対応する比較とする．$\bar{C}$ に重み w が割り当てられている時，C にも w が割り当てられていると考える．

(2) 相手は解いていない比較の重みの総和が少なくとも $W/2$ になるように重みがついた比較を十分多く行なう．

ゲームが終るのは整列が完了した時である．ソーターの目標はゲームを速く終らせることである．各プレーヤーが1回ずつプレーを行なうのを1ステップと数えるとソーターはこのゲームを $O(\log n+f(n))$ ステップで終らせることができるということを示せる．ソーターの戦略は比較器に次のようなルールで重みを割り当てることである．ネットワーク上の深さ j のアクティブな比較器に重み 4^{-j} を割り当てる．すると次の補題が成り立つ．

補題1 $k\geq 0$ に対して最初の $k+1$ ステップ後のアクティブ重みは $(3/4)^k \cdot n/2$ で押えられる．□

補題2 $k\geq 5(j+1/2\log n)$ に対して，$k+1$ 番目のステップの間，深さ j の比較器はアクティブではない．□

補題2より $5(f(n)+1/2\log n)$ ステップ後，アクティブな比較器はない．ということは，すべての比較は解かれ，整列は完了している．したがって，ゲームは $O(f(n)+\log n)$ ステップで終了する．

この考え方を Megiddo の手法に取り入れて，問題 B を解く時間 $O(n\log^2 n)$，$O(n\log^3 n)$ の中の $\log n$ を1つ取ることを考える．相手が比較の半分の重みを解くことを求められている時，重みの中央値をもつ比較を解く．すると，一括処理条件によって比較の半分の重みを解いたことになる．n 個のうち重みの中央値をもつものは $O(n)$ 時間で求められる．ゲームを行なう時間は，比較を除けば，$O(n\log n)$ である．AKS ネットワーク［1］は，係数は大きいが $O(n\log n)$ 時間で構築できる［1］．AKS ネットワークの深さは $O(\log n)$ であり，ここでは $O(\log n)$ の比較を行なえばよい．よって，$O(C(n)\log n+n\log n)$ 時間となり，$C(n)=O(n)$ に対しては $O(n\log n)$，$C(n)=O(n\log n)$ に対しては $O(n\log^2 n)$ となる．以下に，この Cole の拡張手法を使って，高速に解ける計算幾何の問題を2つを示す．

傾き選択問題［4］

平面に n 個の点 $(x_1,y_1),\cdots,(x_n,y_n)$ が与えられているとし，これらのうちの2点を結ぶ $N=\binom{n}{2}$ 個の直線 $y=a_{ij}x+b_{ij}, 1\leq i<j\leq n$ を考える．これらの直線のうち，傾きが k 番目のものを求めたい．つまり，$S=\{a_{ij}, a\leq i<j\leq n\}$ の k 番目に小さい値を求めることが問題である．この問題は，Megiddo の手法によって $O(n(\log n)^2)$ 時間で，Cole の改良を用いることによって $\log n$ が取れて $O(n\log n)$ 時間で解くことができる．この問題を解くには，どうしても $\Omega(n\log n)$ 時間必要なことがわかっているので，これはオーダーに関して最適な結果である．

よりロバストな回帰直線［6］

傾き選択問題に似た問題として，次のように定義される回帰直線（repeated median regression line）の傾きを求める問題がある．傾き選択問題の場合と同様に，平面に n 個の点 $(x_1,y_1),\cdots,(x_n,y_n)$ が与えられ，これらのうちの2点を結ぶ $N=\binom{n}{2}$ 個の直線 $y=a_{ij}x+b_{ij}$, $1\leq i<j\leq n$ を考える．この場合の回帰直線の傾きとは，

$$\hat{a}=\mathrm{med}_i \mathrm{med}_{j\neq i} a_{ij}$$

で定義されるものである．つまり，i を固定したとき i から他の点への傾きの中央値を求め，次に i を動かしてそれらの中央値の中の中央値を求めるという問題である．この問題も Megiddo の手法，さらに Cole の改良を用いることにより $O(n(\log n)^2)$ 時間で解ける．

4. おわりに

本稿は解説としてはアルゴリズムの中身の話になってしまって，その成果の幅広さ・有用さを述べるところまではいかなかった．ただ並列アルゴリズムの考えを用いて高速な逐次アルゴリズムを構成するというところの面白さまででも，なんとか説明できていると幸いである．

このパラメトリックサーチの手法の有用さは，大きさ n の最適化問題があったときに，素朴なアルゴリズムに比べて n に関して1乗（ある場合には2以上の）オーダーの下がったアルゴリズムを提供することにある．ただ，この解説中での AKS ネットワークなどの話は，実用的には役に立たないほど定数の大きな話である．今後この手法の有用性を実地検証していくことが必要であろう．

参考文献

［1］ M. Ajtai, J. Komlós and E. Szemerédi : Sorting in $c\log n$ Parallel Steps. *Combinatorica*, Vol.3, No.1 (1983), pp.1-19.

［2］ R. Chandrasekaran : Minimum Ratio Spanning Trees. *Networks*, Vol.7 (1977), pp.335-342.

［3］ R. Cole : Slowing Down Sorting Networks to Obtain Faster Sorting Algorithms. *Journal of the Association for Computing Machinery*, Vol.34, No.1 (1987), pp.200-208.

［4］ R. Cole, J. S. Salower, W. L. Steiger and E. Szemerédi : An Optimal-Time Algorithm for Slope Selection. *SIAM Journal on Computing*, Vol.18, No.4 (1989), pp.792-810.

［5］ N. Megiddo : Applying Parallel Computation Algorithms in the Design of Serial Algorithms. *Journal of the Association for Computing Machinery*, Vol.30, No.4 (1983), pp.852-865.

［6］ A. Stein and M. Wermann : Finding the Repeated Median Regression Line. *Proceedings of the 3rd SIAM-ACM Symposium on Discrete Algorithms*, 1922, pp.409-413.