並列計算の虚と実

茨木　俊秀

1. 並列化の時代

“並列”や“分散”が情報科学のキーワードとなって久しい．情報処理学会や電子情報通信学会などの研究発表をのぞくと，かなりの割合がこれらの話題に関連するものであるし，ＯＲ学会の発表でも決してめずらしくない．スーパーコンピュータという名称で（限定された形ではあるが）並列処理が身近なものとなっており，本格的な並列マシンも，あとで述べるように，大学の研究室から容易にアクセスできるようになってきている．最近では，コネクションマシンなど，数万台のプロセッサをもつ並列マシンが商品化されており［1，2］，通産省の次の大型プロジェクトである“新情報処理技術”は百万のオーダーのプロセッサを念頭においたものであるという．まさに並列化の時代である，

この状況を反映して，並列化によってすべての問題を解決できるという素朴な信仰が流布しているかにみえる．しかし，これは間違っている．並列化にも，アルゴリズムと物理的実現性の観点からおのずと限界があるという事実を認識することは重要である．限界を知ってはじめて可能性を正確に把握できるからである．

以下，並列計算の虚と実という形で，この限界を私なりに説明してみたい．

2. 並列化は計算量の壁を超えるか？

計算量の意味で困難な問題の例として，よく知られた巡回セールスマン問題を考えてみよう．n個の点をまわる巡回路の中で最短のものを求める問題である．これを解くには，巡回路は節点1から始まるとして一般性を失わないから，残りのn-1点について$(n-1)!$個のすべての順列を調べ，最短のものを残せばよい．しかし，この方法の難点は，個数$(n-1)!$が急速に大きくなり，すぐに手に負えなくなるという“組合せ的爆発”を生ずるところにある．

いばらき　としひで　京都大学　工学部　数理工学科
〒606 京都市左京区吉田本町

これに対し，いかに大きな値であっても，それに見合うだけのプロセッサを準備し並列化すれば対抗できると考える人がいる．本当にそうだろうか．$n=11$のとき，$(n-1)!$は3.6×10^6程度であるから，1つのプロセッサで間に合う．そこで10000台のプロセッサを準備して並列に処理してみよう．計算量は3.6×10^{10}ほどに増加するが，これは14!より少ない．つまり10000台のプロセッサで$n=11$から$n=15$まで拡大できるにすぎない．

いや10000台では少ないという意見もあろう．人間の脳には約10^{10}個のニューロンがあるといわれているので，このオーダーのプロセッサを用いてみよう．その結果は，$3.6\times10^6\times10^{10}\ll19!$，つまりようやく$n=20$程度である．この簡単な考察からも，プロセッサ台数の増加が決して本質点な解決にはならないことがわかる．

これに対し，巡回セールスマン問題の数学的性質にもとづくアルゴリズムでは，たとえば最近注目を集めている多面体アプローチによると，nが数百，数千という場合を（逐次計算で）厳密に解くことに成功している．すなわち，並列化では越せない壁を巧妙なアルゴリズムを考案することで突破しているのである．

それでは並列化による高速化は意味がないのかと問われると，それも正しくない．時間量が$n, n\log n, n^2, \cdots$程度の高速アルゴリズムをもつ問題に対しては，並列化はきわめて効果的である．たとえば，$n\log_{10}n$ならば，$n=10^6$のとき$n\log n=6\times10^6$であるので1台のプロセッサで処理できるとすると，nがこの100倍n^8になったとしても，$n\log n=8\times10^8$であるから，プロセッサ100台で対応できる．論理上，時間量が入力データ長nの多項式オーダーであるような（逐次）アルゴリズムをもつこれらの問題のクラスをPと記している．幸い，世の中には，実用上重要でしかもクラスPに属する問題が結構たくさんあることがわかっている．グラフの最小木，最短路，最大フロー，最小コストコロー，さらに線形計画問題などが代表例である．

3. どんな問題も並列化できるか？

上の議論では，p台のプロセッサを用いると計算時間

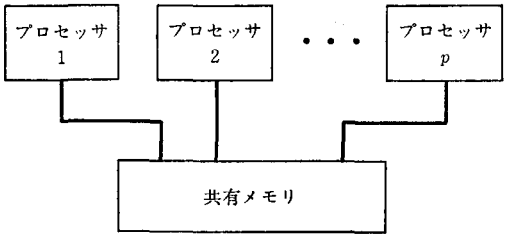


図 1 PRAM

が $1/p$ に短縮されると仮定していたふしがある．しかし，複数台のプロセッサを協力させて効率よく解を求めるのは決して容易ではない．これまでの経験から，逐次アルゴリズムの中で並列化できるところは並列化するという消極的な方法では，ある程度以上の効果は期待できないことがわかっている．やはり，並列処理に合った新しい並列向きアルゴリズムが必要である．

それでは，どのような問題でも工夫すれば効果的な並列アルゴリズムを見出すことができるだろうか．答はやはり否である．以下，そのような並列化にむかない問題を理論的に示す手段が築かれつつあることを紹介してみよう．

並列計算を理論的に扱う時，計算モデルとして図 1 の PRAM (parallel random access machine) を用いることが多い [3]．すなわち p 台の並列プロセッサは共通のクロックによって同期的に動作し，共有メモリを介して自由にいつでもデータを交換することができる．

PRAM によれば，たとえば p 個の数字 $x_i, i=1,2,\cdots,p$ の積を求めるという計算を次のように実行できる．x_i をプロセッサに受け持たせ，時刻 $t=1,2,\cdots$ に

$$x_i := x_i \times x_{i+2^{t-1}}$$

とすれば，図 2 に示すように，時刻 $t=[\log_2 p]$ のとき，x_1 に全 x_j の積が入る．

この並列計算の時間量は，1 台のプロセッサのみを用いたときの時間量 p の $1/p$ までには至っていないが，それに近い効果が得られている．このように(非常に)効率よい並列アルゴリズムをもつ問題を示すため，クラス NC (Nick's class, Nick はこの分野の研究者 Pippenger の第 1 名) が次のように定義されている．

問題Aの入力データ長がNであるような任意の問題例に対して，プロセッサ数 $p=O(N^k)$ (kは定数) をもつ PRAM によって，常に時間量 $O((\log N)^{k'})$ (k'も定数) で解を求めることができるならば，Aはクラス NC に属するという．

多項式オーダーのプロセッサ数 $O(N^k)$ は「あまり多くないプロセッサ数」，時間量 $O((\log N)^{k'})$ は「非常に早い」ことの理論的表現である．

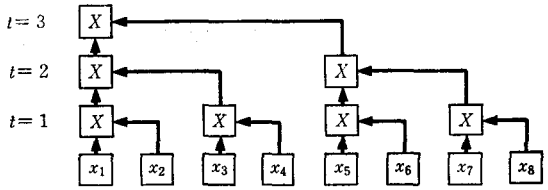


図 2 $x_1 \times x_2 \times \cdots \times x_8$

先の，「すべての問題は並列化できるか」という問いは，「すべての問題はクラス NC に属すか」と言いかえることができる．そこでまず，$A \in NC$ であるためには $A \in P$ でなければならないことに注意しよう．$O(N^k)$ 個ある全プロセッサの動作を逐次的に 1 ステップずつ追跡すると

$$O((\log N)^{k'})O(N^k) \leq O(N^{k+1}),$$

つまり，多項式オーダーの逐次計算で実行できるからである．これより，

$$NC \subseteq P$$

を得る．しかし，P がきわめて多様な問題を含んでいることから，そのすべてが並列化に向いているわけではない，つまり

$$NC \neq P$$

と予想されている（この予想は有名な P≠NP 予想とともに未解決である）．これが正しいとすれば，クラス P の中で最も難しい問題は NC に属さない（つまり並列化に向かない）ことになる．そのような問題は，（詳しい説明は略すが）P 完全問題と呼ばれ，多数存在することが明らかにされてきている．

クラス NC の問題と P 完全問題の違いをみるため，次のグラフ問題を考えよう．

極大独立集合

入力：グラフ $G=(V,E)$.

出力：Gの極大独立集合を 1 つ．

ただし，節点集合 $W \subseteq V$ が独立であるとは，$u,v \in W$ を満たす枝 $(u,v) \in E$ が存在しないことをいう．独立集合 W はそれを真に含む独立集合 W′ が存在しないとき極大である．

ところで，この問題は，次の欲張りアルゴリズムによって簡単に解くことができ，明らかに P に属す．

ステップ 0（初期解）：適当な節点 v を 1 つ選び，$I:=\{v\}$ とする．$k:=1$.

ステップ k（反復）：$u \in V-I$ で I 内のどの節点とも

接続しないものがあれば，その1つuを任意に選び，I:$=I\cup\{u\}$, $k:=k+1$ としてステップkへ戻る．そのようなuがなければ終了．Iは位数kの極大独立集合である．□

この問題は，さらにNCにも属している．そのための詳しい説明は略すが，ステップkのuの選択を並列に行ない，複数の独立節点をいちどにみつけIに追加することで，反復回数を$(\log n)^{k'}$に抑えることができるというのがポイントである．

しかし，独立集合に対し，さらに辞書式に先頭という条件を付すと事情は異なってくる．節点集合 $V=\{v_1, v_2, \cdots, v_n\}$ の任意の独立集合 $I=\{v_{i1}, v_{i2}, \cdots, v_{ik}\}$（一般性を失なうことなく $i_1<i_2<\cdots<i_k$ とする）を文字系列

$$I=i_1 i_2 \cdots i_k$$

とみなすとき，辞書式順序で先頭にくる極大独立集合を求める問題である．

この問題を解くには，逐次計算ならば，上の欲張り法のアルゴリズムを少し修正し，ステップ0のvをv_1に，ステップkのuを条件を満たすものの中で最小の添字iをもつv_iとするだけでよい．したがって，これもPに属す問題である．

しかし，この計算を並列化して，いちどにたくさんの独立節点を求めようとしても，辞書式に先頭であることを保証するには，各要素 v_{ij} の添字を $v_{i1}, v_{i2}, \cdots, v_{ij-1}$ を知った上で決定しなければならず，逐次的条件が介入してきてしまう．実際，この問題はP完全であることが知られている．

クラスNCの代表的な問題には，数の整列，最小木問題，最短路問題などがある．これに対し，P完全問題のリストには，最大フロー問題，線形計画問題などが含まれている［3, 4］．

4. 並列アルゴリズムはそのまま実現できるか？

PRAM上の並列アルゴリズムとそれに関する理論的性質をそのまま現実の並列アルゴリズムにあてはめることは許されるだろうか．

まず，PRAMは理論モデルであるから，そのプロセッサ台数Pを $P=N^k$ のように自由に置くことができる．さらに，$N\to\infty$ の場合の挙動を問題にすることすらある．当然ある程度の有限のPしか実現できない現実のコンピュータとの間にギャップがある．

もう1つの問題点は，プロセッサ間およびプロセッサとメモリの間のデータ通信であって，より深刻である．

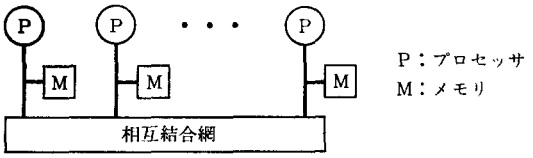


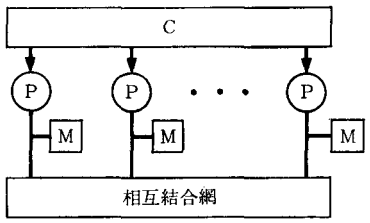
図3 メモリの分散による実現

すなわちPRAMでは任意の時刻に任意の個数のプロセッサが共有メモリを介してデータを交換できるとしているが，この機能を物理的に実現することは困難である．すなわち，データを送るには線を張らねばならず，その場所とそこから通るためにかかる時間が必要である．このため，たとえば単一バス結合による共有メモリの実現法では，十数台のプロセッサが限度とされている．

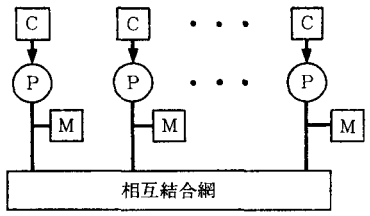
より多くの並列プロセッサをもつシステムを実現するには，メモリを各プロセッサに分散することが不可避であろう．つまり，**図3**のように，各プロセッサに局所メモリをもたせ，それらを相互結合網で結びつける方式が考えられる．相互結合網の具体的な形は，2次元あるいは3次元の格子，ハイパーキューブ，クロスバー，多段結合，木結合，その他数多くのアイデアがあり［1, 4］，商品化されているものもある．

さらに，これら多数のプロセッサをどのように制御するかについては，**図4**のSIMD (single instruction stream/multiple data stream) とMIMD (multiple instruction stream/multiple data stream) に大別される．前者は1つの制御部ですべてのプロセッサを制御する方式で，全プロセッサは同一の（あるいは若干の修飾を加えた）演算を一斉に実行する．もちろん，各プロセッサは自己のデータに対してその演算を実行し，その結果を相互結合網を通して交換できるので，全体として有機的にまとまった計算を実現できる．これに対し後者は，プロセッサごとに制御部をもち互いに独立に演算を実行する．したがって，1つのノード（プロセッサ，メモリ，制御部を合わせたもの）が複雑になるという欠点はあるが，汎用性はずっと高い．

これらのモデルは，理論モデルであるPRAMを物理的に可能な形で実現するための妥協の産物であるといってよい．しかし，その結果，データの分散保持，相互通信，各プロセッサの制御（つまりプログラム）など細部の設定が複雑になる．つまり，PRAM上での概念的な並列アルゴリズムと比べると，データの相互通信と全プロセッサの制御に整合性をもたせるための無駄（オーバヘッド）が生じる．その結果，PRAM上で得られてい



(a) SIMD



(b) MIMD

図 4　2つの制御方式(Cは制御部を示す)

た並列化によるスピード向上比がそのまま実現される保証はない.

もちろん, PRAM を想定した並列アルゴリズムが無意味であるわけではない. モデルが単純であるだけ並列計算の本質的な部分を把握しやすいからである. そのあとのチューニングの作業を容易にするため, 並列コンピュータのアーキテクチャとプログラム言語を工夫して, 個々のマシンの構造や特性を意識せずにプログラムしても, オーバヘッドを大きくしないよう, 地道な努力が払われている. これら, ハードとソフトの両面の現状については, 文献［2］などに詳しい.

5. 数理計画の例から

並列計算が理論だけのものでも対岸のものでもないことを理解していただくには, "われわれの研究室でも" 並列コンピュータを用いて計算を行なっていることを伝えるだけで十分であろう. じつは, われわれの数理工学教室には, 野木達夫先生と松下電器の共同開発になる ADENA が設置してあり, ネットワークを通して自由にアクセスできる. ADENA は256個のプロセッサをもつ MIMD タイプのマシンで, 偏微分方程式などに生じる配列で効率よく処理できるよう, 相互結合網の構成に特殊的な工夫がこらされている (詳しくは, 文献［2］中の解説記事参照). われわれユーザは FORTRAN の拡張である ADETRAN (これは ADENA を SIMD 的に動作させている) を用いてプログラムでき, ADENA の構造を意識する必要はほとんどない.

計算実験の1つとして, 町田君の卒業研究［5］の一部を紹介する. 研究指導は福島雅夫先生が行なったので当然非線形計画であって,

$$\text{目的関数}\quad \sum_{j=1}^{n}\left\{\frac{1}{2}x_j^T G_j x_j + h_j^T x_j\right\} \to \text{最小}$$

$$\text{制約条件}\quad \sum_{j=1}^{n} a_{ij}^T x_j + b_i \leq 0,\quad i=1,2,\cdots,m$$

$$x_j \in R^d,\quad j=1,2,\cdots,m$$

という分離形2次計画問題を対象としている. 各 x_j は d 次元の変数ベクトルであり, 目的関数は $d\times d$ 正定行列 G_j で定まる分離形2次関数, h_j, a_{ij}, b_i は係数ベクトルである.

この問題に文献［6］のアルゴリズムを適用したわけであるが, ここでは計算結果のみを示す. **図5**は上の $m=2$, $d=5$ の場合の問題例をランダムに発生し, いろいろな n に対する結果をプロットしたものである. 図の実現比は ADENA による計算時間と, 同じ計算を1台のプロセッサで行なった場合の計算時間の比, つまり, 速度比を示している. 最高で80近い実現比が得られている. これは ADENA のプロセッサ数256に比べると1/3弱で小さいようにもみえるが, アルゴリズムに自体逐次的な部分が含まれているために, 256 まで上げることは本質的に不可能である.

この点を明確にするため, 図5にはこのアルゴリズムの並列計算の部分が理想的に行なわれるとした場合の速度比を理論比として示している. 理論比は, n が 256 の倍数近くで最も大きく 160 程度になるが, 256 よりはかなり小さい. この理論比と実現比の間にはまだ約2倍の違いがあるが, これが物理的に実現された ADENA の

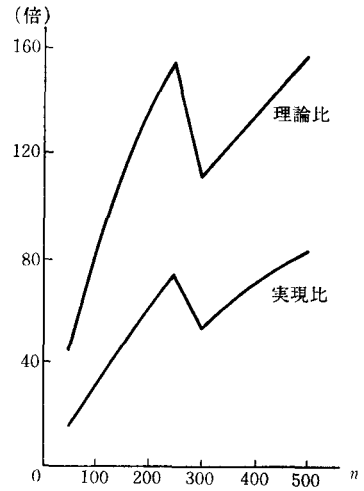


図 5　並列計算による速度比

オーバヘッドと解釈できる量である．他の並列コンピュータとの比較を行なったわけではないが，この程度のオーバヘッドで済むのはかなりよい結果だと考えている．

6. むすび

並列計算について日頃感じていることをわかりやすく説明したつもりであるが，どの程度成功しただろうか．要点をキャッチフレーズ的にまとめると次のようになろう．

- 並列化しても難しい問題はやはり難しい．
- しかし，やさしい問題はもっとやさしくなる．
- 並列コンピュータには並列向きのアルゴリズムを．
- 並列化に向く問題とそうでない問題がある．
- 理論と現実のギャップはまだ大きい．しかし，近づきつつある．
- 並列計算はすでに現実である．

私は決して並列計算の専門家ではないが，無視しているわけでもない．アルゴリズムをこれまでの逐次的な観点ではなく並列化という方向から眺めると，それまで気づかなかった新しい発見に出会い，新鮮な驚きを感じることがしばしばある．

幸い本特集には並列アルゴリズムをさまざまな観点から扱った興味深い記事が集められているので，読者の方々もぜひこの新しいおもしろさに触れていただきたい．

参考文献

[1] 富田眞治，末吉敏則，並列処理マシン，電子情報通信学会編，オーム社，1989.

[2] 特集：超並列マシンとその応用，情報処理，Vol. 32, No. 4, 1991年4月.

[3] A. Gibbons and W. Rytter, Efficient Parallel Algorithms, Cambridge University Press, Cambridge, 1988.

[4] 茨木俊秀，アルゴリズムと離散最適化，応用数学講座，岩波書店（近日出版）.

[5] 町田直樹，分離可能な凸計画問題に対する交互方向乗数法の並列計算，京都大学工学部数理工学教室特別研究報告書，1992.

[6] M. Fukushima, Application of the alternating direction method of multipliers to separable convex programming problems, Computational Optimization and Applications（掲載予定）.