

解説

2. ハードウェアアルゴリズムの基礎理論



2.1 並列計算モデルと計算の複雑さ†

笠井 琢 美††

1. はじめに

一つ一つの命令を順次実行していくという、逐次型計算に関しては、計算モデルも安定したものがあり、これまでに多くの効率のよいアルゴリズムが開発されてきた。また、アルゴリズムの効率を測る尺度としての計算量も、その時代の技術にあまり依存しない理論的には安定した不変量とみなしてよいという認識を多くの人が持つようになってきた。一方、近年のVLSI技術の発展にともない、並列計算に関する関心が高まり、数多くの並列計算モデルが提案され、そのモデル上での高速な並列アルゴリズムが発表されている。ここでは、一般的な汎用計算機としての並列計算モデルを中心に解説する。

計算機モデルの性能を測る尺度を計算量という。計算量として重要なものに、時間量 (time complexity) と領域量 (space complexity) がある。時間量とは計算に必要な実行時間のことで、領域量とは計算に必要な記憶領域の量のことである。計算量は、入力の大さを表すパラメータの関数として表わされる。たとえば、 M を計算機モデル、 M を M のプログラムとする。 M の時間量が $T(n)$ であるとは、大きさ n の任意の入力に対し、 M は高々 $T(n)$ 時間で停止することをいう。この解説では次の問を中心に議論を進める。

(1) 並列計算は逐次型計算より強力か。もし強力ならどの程度強力か。

実はこの問は、いくつかの重要な未解決問題を含んでおり、現在のところ明確な形で答えることができない。この問の困難さは、計算量の下限を示すところ、すなわち次のような否定的な結果を示すところにある。

(2) 問題Aは計算機モデル M では T 時間で解け

ない。

最近、いろいろな具体的な問題に対し、(2)のような計算量の下限が示されてきてはいるが、(1)を示すほど強力な手法はまだない。一方、ある計算機モデル M_1 を他の計算機モデル M_2 で効率よく模倣できることを示すことは、比較的容易なことが多い。 $p(\cdot)$ を関数とする。 M_2 が M_1 を高々 p 倍の時間量で模倣できるとは、次が成立するときをいう： M_1 の任意のプログラム M_1 に対し、 M_1 の時間量が $T(\cdot)$ なら、 M_1 と同値な (すなわち、同じ入力に対し同じ出力をする) M_2 のプログラム M_2 で、時間量が $p(T(\cdot))$ のものを構成することができる。計算機モデル間の比較においては、次のように多項式倍の時間量の差を無視するという、非常に大ざっぱな見方をすることがよくある。2つの計算機モデルが多項式時間同級であるとは、互いに他を高々多項式倍の時間量で模倣できるときをいう。これまでに提案された逐次型の計算機モデルはすべて、このような大ざっぱな見方のもとでは同値なものとなる。さらに、並列計算機モデルも数多くのものが提案されているが、ほとんどのものは多項式時間同級である。それでは並列計算機モデルもまた、逐次型計算機モデルと比較して、高々多項式倍の差しかないであろうか。以下ではこのことについて議論する。

2. 逐次型計算機モデル

並列計算機モデルについて述べる前に、従来の逐次型の計算機モデルについて述べよう。計算機モデルの定義の仕方はいろいろあるが、ここでは計算機モデル M を基本命令の集合として定義する。 M のプログラムとは基本命令の有限列のことを言う。基本命令は基本実行命令と基本判定命令の2つに分類される。基本判定命令は次の形をした命令である。

if 条件 goto l

ここで l は正の整数で、以後 l のことをラベルと呼

† Computational Complexity and Models of Parallel Computation by Takumi KASAI (Department of Computer Science, The University of Electro-communications).

†† 電気通信大学計算機科学科

ぶ。この命令が実行されたとき、もし条件が満たされるなら、制御は l 番目の命令に移り、そうでないなら次の命令に移る。各計算機モデルは記憶装置を持っている。記憶装置は記憶セルと呼ばれるものから成る。各記憶セルは情報を記憶する単位で、基本実行命令は記憶セルから情報を読み取ったり、情報を書き込んだりする。

まず、Turing 機械について述べよう。Turing 機械は原始的なモデルではあるが、理論的に最も安定したモデルである。Turing 機械の記憶装置は両方向に無限に長い 1 本のテープである。テープはセルに仕切られており、各セルは有限個の情報が記憶できる。この有限個の情報の各々を文字とか記号と呼ぶ。テープにはヘッドが 1 つ付いており、文字を読んだり書いたりする。ヘッドは 1 回ずつ、左または右に移動できる。Turing 機械の基本命令は次から成る。

- (i) **rshift** (ii) **lshift** (iii) **print "a"**
 (iv) **halt** (v) **if "a" goto l**

(i) から (iv) は基本実行命令で、**rshift** はヘッドを右へ 1 回移動すること、**lshift** は左へ移動すること、**halt** は計算の実行を停止すること、**print "a"** は現在ヘッドがいるセルに文字 **a** を書き込むことを意味する。(v) は基本判定命令で、現在ヘッドが読んでいる文字が **a** なら、次に l 番目の命令へ飛ぶ。時間量は、各基本命令の実行時間を 1 単位時間として測る。すなわち、停止するまでに実行した基本命令の個数である。領域量は使用したセルの個数である。

Turing 機械のほかによく用いられるモデルとして RAM (random access machine) と呼ばれるものがある。RAM は一見すると現在の計算機の忠実なモデルに見えるが、形式化にいくつか問題点がある。RAM の記憶装置は可算無限個の記憶セル $x_0, x_1, \dots$ から成る。各記憶セルは任意の自然数を記憶できる。記憶セルのことをレジスタと呼ぶこともある。RAM の基本命令は次から成る。

- (i) $x_i \leftarrow c$ (ii) $x_i \leftarrow x_j \theta x_j$
 (iii) $x_i \leftarrow x[x_j]$ (iv) $x[x_i] \leftarrow x_j$
 (v) $x_i \leftarrow x_j$ (vi) **halt**
 (vii) **if $x_i = 0$ goto l**

ここで c は自然数であり、(i) は x_i の内容を c にセットすることを意味する。 θ は四則演算(加減乗除)で x_i と x_j の内容に θ を施した結果を x_i に格納する。(iii) と (iv) は間接番地指定の命令で、(iii) は x_j の内容が示す記憶セルの内容を x_i に格納することを、

(iv) は x_j の内容を x_i が示す記憶セルに格納することを意味する。(vii) は基本判定命令で、 x_i の内容が 0 なら次に l 番目の命令を実行する。

ここで RAM モデルの問題点について述べておこう。RAM モデルの不自然さは、各記憶セルが任意の自然数を記憶できるという点にある。自然数は、たとえば 2 進数と解釈し、文字列データとして扱うことができる。現在の計算機の記憶セルは、数値データとしてならともかく、文字列データとして任意の長さの文字列を記憶できると形式化できるほど大きな記憶量を持ってはいない。したがって、プログラムを数値とみなして 1 つの記憶セルに格納したり、多数の記憶セルにある内容を符号化して 1 つの記憶セルに格納するといったことは避けなければならない。実際、上で述べた各基本命令が 1 単位時間で計算できると仮定すると、現在大きな計算量を持つと予想されている数多くの問題が、効率よく計算できてしまうという奇妙な結果が導かれる⁸⁾。このことについては、後でもう一度述べる。したがって RAM モデル上で議論するときは次のいずれかの立場を取る。一つは (ii) の四則演算として乗算を禁止すること。乗算さえ使わなければ、扱うデータの桁数が指数関数的に増加することを防ぐことができる。もう一つの立場は、各基本命令に 1 単位時間以上の負荷を課すること。Cook は対象となる記憶セルの内容の桁数だけの負荷を課そうと提案している。これを **Cook の負荷体系** という²⁾。 $\bar{l}(x)$ で記憶セル x の内容を表わす。 $\bar{l}(x)$ で自然数 x の 2 進表現の桁数を表わす。すると、たとえば (ii) の基本命令の実行時間は $\bar{l}(x_i) + \bar{l}(x_j)$ と、(iii) の実行時間は $\bar{l}(x_i) + \bar{l}(x_j)$ となる。時間量は実行された基本命令の負荷の合計である。以下では RAM は、この 2 つのうちのいずれかの立場を取るものとする。どちらを取っても同じで、次が成立する。

(3) Turing 機械と RAM は多項式時間同級である。

3. 並列 RAM

並列 RAM とは 2 章で述べた RAM を並列に動作させるもので、いろいろな種類のものがある。ここではそのうちの代表的なものについて述べる。これらのモデルがどの程度の能力を持つかについては 4 章で議論する。

大域記憶を持つもの まず最初に図-1 のように単純に RAM を並べたモデルを考えよう。ここで P_0 ,

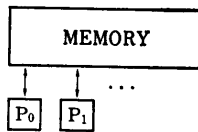


図-1

$P_0, P_1, \dots$ は RAM で、各々のことをプロセッサと呼ぶ。プロセッサの個数は可算無限個（と形式化してよいぐらい）あるとする。また図-1 で示すような共通の記憶装置のほかに、各プロセッサは固有の記憶装置を持つ。共通の記憶を大域記憶、固有の記憶を局所記憶と呼ぶ。2つのプロセッサが大域記憶の同じ記憶セルに同時に書き込みを行った場合の動作の定義の仕方はいろいろあるが、どれを取っても大差はない。ここでは最も番号の小さいプロセッサが優先されるものとする。特殊な議論をするとき以外は、たいてい各プロセッサは同一のプログラムを実行すると仮定している。

今述べたモデルでは、各プロセッサは同じプログラムを実行するが、ある時点で各々のプロセッサが実行している命令は同一ではない。次に述べるモデルは、常にすべてのプロセッサが同一の命令を実行するように制限したものである。このモデルは SIMDAG (single instruction stream, multiple data stream, global memory) と呼ばれている。基本命令は2章で述べた RAM の基本命令のほかに次のものを持つ。

- (i) $y_i \leftarrow c$ [...] (ii) $y_i \leftarrow y_j \theta y_k$ [...]
- (iii) $y_i \leftarrow v[y_j]$ [...] (iv) $v[y_i] \leftarrow y_j$ [...]
- (v) $y_i \leftarrow y_j$ [...] (vi) $y_i \leftarrow \text{SIG}$ [...]
- (vii) $y_i \leftarrow x[y_j]$ [...] (viii) $x[y_i] \leftarrow y_j$ [...]

ここで y_i は局所記憶の i 番目の記憶セルの名前であり、 x_i は大域記憶の記憶セルの名前である。[...] の部分は次のいずれかの形の条件である。

if $\text{SIG} < x_i$

if $\text{SIG} < x_i$ かつ $y_i > 0$

各プロセッサは SIG (signature) と呼ばれる特別のレジスタを持っており、SIG にはそのプロセッサの番号が入っている。各命令に付いている条件 $\text{SIG} < x_i$ は、0 から x_{i-1} までの x_i 個のプロセッサが同時にその命令を実行することを意味する。各プロセッサ P_i は、(vi) の命令で自分の番号 i を知ることができ、プロセッサごとに異なる仕事をさせることができる。各命令の意味は、2章の RAM の命令と同様である。2章で述べたように、四則演算 θ として乗算を禁止す

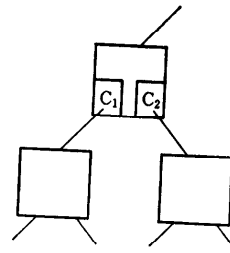


図-2

るか、あるいは各命令にデータの桁数に比例した負荷を課すことにする。Goldshlager は、実際にこのようなモデルを構成する場合、各命令にどの程度の負荷を課するのが現実的かについて議論している⁶⁾。

特殊な結合方式を持つモデル 次に述べるモデルはプロセッサ間の情報のやり取りを直接プロセッサ間で行うものである。逐次型計算の多項式倍を超える効率を達成するためには、あるプロセッサが n ステップ後に影響を与えるプロセッサの数は、 n の多項式を超える個数でなくてはならない。このようにするには、たとえば図-2 で示すように木の形にプロセッサを結合すればよい。このようなモデルで大域記憶を許すものもあるが、ここでは情報のやりとりは結ばれたプロセッサ間だけに制限することにする。また、最初は何も結合されていず、必要になったとき使用していないプロセッサを次々と木の形に結合していくものもあるが、論理的にはどちらも同じである。各プロセッサはチャンネルレジスタと呼ばれる2つのレジスタ C_1 と C_2 を持っており、このレジスタで2つの子供のプロセッサとの情報の受渡しを行う。基本命令は2章で述べた命令のほかに次のものを持つ。

- (i) **fork** σ l (ii) **return** x_i
- (iii) $x_i \leftarrow C_\sigma$ (iv) $C_\sigma \leftarrow x_i$
- (v) **if** σ **returned goto** l

ここで x_i は各プロセッサの局所記憶の i 番目の記憶セルの名前であり、 σ は1または2である。**fork** σ で σ 番目の子供を起動させる。 σ 番目の子供が動作中のときは停止するまで待つ。**fork** σ l が実行されると、 C_σ の内容が σ 番目の子供の子供の x_0 にコピーされ、子供は l 番目の命令から実行を開始する。 σ 番目の子供が **return** x_i を実行すると、 x_i の内容が親の C_σ にコピーされ、子供は次に起動されるまで休止する。(v) の命令は σ 番目の子供が休止中かどうかをテストするもので、もし休止中なら制御を l 番目の命令に移す。

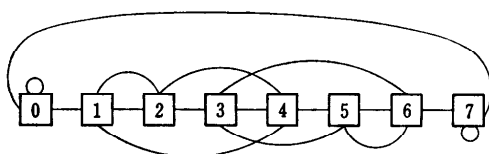


図-3

本質的に木と同程度の能力を有するもう一つの結合方式について述べよう。これは Ultracomputer¹⁵⁾に使用された結合方式である。 n を 2 のべき乗の形の整数、すなわち $n=2^k$ とする。このとき n 個のプロセッサ $P_0, \dots, P_{n-1}$ を次のように結合する：プロセッサ P_i を $P_{\rho(i)}, P_{\rho^{-1}(i)}, P_{\sigma(i)}, P_{\sigma^{-1}(i)}$ と結合する。ここで ρ と σ は次で定義される $\{0, 1, \dots, n-1\}$ 上の順列である。

$$\rho(i) = i+1 \bmod n$$

$$\sigma(i) = \text{if } i < n/2 \text{ then } 2i \text{ else } 2i-n+1$$

$n=8$ の場合を図-3 に示す。 $P_{\rho(i)}$ は P_i の右隣り、 $P_{\rho^{-1}(i)}$ は左隣りである。 σ のことをシャッフルと呼ぶ。これは n 枚のトランプのカードを 2 等分し、1 枚ずつ交互にくるように並べなおしたとき、最初 i 番目にあったカードが $\sigma(i)$ 番目に来ることから名付けられた。このモデルでは、2 章の基本命令のほかに、 $\rho, \rho^{-1}, \sigma, \sigma^{-1}$ の 4 つの基本命令を持っている。たとえば ρ は、各プロセッサが右隣りのプロセッサに、すなわちプロセッサ P_i はプロセッサ $P_{\rho(i)}$ に、データを転送することを意味する。他の命令 $\rho^{-1}, \sigma, \sigma^{-1}$ も同様に定義される。 $X = \langle x_0, x_1, \dots, x_{n-1} \rangle$ をデータの n 組、 π を $\{0, 1, \dots, n-1\}$ 上の順列とする。このとき $\pi(X)$ を

$$\pi(X) = \langle x_{\pi(0)}, x_{\pi(1)}, \dots, x_{\pi(n-1)} \rangle$$

と定義する。この結合方式の強力さは次の事実に基づく： $\rho(X), \rho^{-1}(X), \sigma(X), \sigma^{-1}(X)$ が 1 単位時間で計算できると仮定すると、任意の順列 π に対し、 $\pi(X)$ は $O(\log n)$ 時間で計算できる。言い換えると、任意の順列 π は $\rho, \rho^{-1}, \sigma, \sigma^{-1}$ の高々 $O(\log n)$ 個の合成として表わせる。

このモデルを他のモデルと比較するとき、プロセッサの個数 n は、(入力のサイズに依存して) いくらでも大きく取れるものと仮定しなくてはならない。

並列演算型モデル 記憶装置の外に 2 組のレジスタの列 $X = \langle x_0, \dots, x_{n-1} \rangle$ と $Y = \langle y_0, \dots, y_{n-1} \rangle$ が使用できるものとする。このレジスタの列に対し、Ultracomputer で述べたシフト演算 ρ と、シャッフル演算

σ と、その逆演算 ρ^{-1}, σ^{-1} が適用できるものとする。さらに X と Y との並列演算と、 X, Y と記憶装置との間の適当な転送命令があるとすると、これ等の命令が効率よく実現できれば、Ultracomputer と、したがって他の並列 RAM と、ほぼ同程度の能力を有するものとなる。すなわち並列計算は、並列演算の中に埋め込むことができるのである。2 章で RAM に乗算を許し、乗算が 1 単位時間で実行できるものと仮定すると、モデルとして不自然なものになると述べた。このことは、非常に桁数の大きい 2 つの数の乗算が効率よく計算できると仮定すると、記憶装置全体の情報を一つの記憶セルに符号化することによって、並列計算が効率よく模倣できてしまう点にある¹⁾。それではどの程度強力な演算を許すと、逐次計算は並列計算と同じ能力を持つようになるのであろうか。そのようなもので最も単純なものが次に述べるベクトル機械 (vector machine, VM と略す) である¹¹⁾。VM は有限個のベクトルレジスタ $x_0, x_1, \dots$ と有限個のインデックスレジスタ $i_0, i_1, \dots$ を持つ。VM の基本命令は次で示される。

- | | |
|--|--|
| (i) $i_i \leftarrow c$ | (ii) $i_i \leftarrow i_i + i_k$ |
| (iii) $i_i \leftarrow i_i - i_k$ | (iv) $x_i \leftarrow c$ |
| (v) $x_i \leftarrow x_i \wedge x_k$ | (vi) $x_i \leftarrow \neg x_i$ |
| (vii) $x_i \leftarrow x_i \uparrow i_k$ | (viii) $x_i \leftarrow x_i \downarrow i_k$ |
| (ix) if $i_i = 0$ goto l | (x) if $x_i = 0$ goto l |

ここで c は定数、 $\wedge$ は (ビットごとの) 論理積、 $\neg$ は (ビットごとの) 否定である。(vii) は x_i の表わすビット列を i_k ビットだけ左にシフトすることを、(viii) は右に i_k ビットシフトすることを表わす。ビット列を 2 進数とみなす (右端を最下位のビットとする) なら、(vii) は $x_i \leftarrow x_i * 2^{i_k}$ と、(viii) は $x_i \leftarrow x_i / 2^{i_k}$ と同じである。理論的モデルとしては、各レジスタは任意の整数 ($\dots 111w$ と左側に無限に 1 が続くビット列は負の数を表わす) を記憶できるものとするが、現実的なイメージとしては、インデックスレジスタは通常の桁数で、ベクトルレジスタの桁数は非常に大きいものと考えればよい。各基本命令は 1 単位時間で実行できるものとする。このモデルでは、レジスタ (記憶セル) の個数は有限であり、間接番地形の命令 ($x_i \leftarrow x[x_j], x[x_i] \leftarrow x_j$) を持っていないことに注意する。

4. 理論的モデル

3 章で述べた並列計算モデルはどれほど強力である

うか。現在の理論でどの程度のことと言えるかについて述べよう。議論を単純にするために、問題を決定問題、すなわち真か偽を答える問題に制限する。 Σ をアルファベットとし、 Σ^* で Σ 上の語全体から成る集合を表わす。 Σ^* の部分集合のことを問題と呼ぶ。(実際、たいていの決定問題は、 Σ^* の部分集合に対する決定問題として形式化できる。)ここでは計算機モデルは2章で述べた Turing 機械とする。Turing 機械に真か偽を答えさせるために、停止命令 halt のかわりに次の2つの停止命令を用意する。

(i) **accept** (ii) **reject**

accept は真と、**reject** は偽と答えることを意味する。 M をプログラムとする。 Σ^* の元 x が入力としてテープに与えられたとする。 M が計算を開始する直前は、入力がかかれてセル以外は空白と呼ばれる特別な文字がかかれており、ヘッドは入力 x の左端の文字の位置にある。 M が **accept** で停止するなら、 x は M に受理されるといい、**reject** で停止するなら、拒否されるという。 M の時間量が $T(n)$ であるとは、任意の長さ n の入力に対し、 M が高々 $T(n)$ ステップで停止するときを言う。領域量が $S(n)$ であるとは、使用するセルが高々 $S(n)$ のときを言う。 $A \subset \Sigma^*$ を問題とする。 M が A を受理する(あるいは A を解く)とは、任意の x に対し、 $x \in A$ なら x を受理し、 $x \notin A$ なら x を拒否するときを言う。問題 A の時間量が T (領域量が S) であるとは、 A を受理する時間量が T (領域量が S) のプログラムが存在するときを言う。

さて、碁とか将棋が先手必勝かどうかといった問題のように、原理的には計算可能であるが、実際には時間がかかりすぎて計算できないといった問題は多い。それでは“実際の”な問題とはどのように定義したらよいであろうか。次の立場を取ることは、たいていの場合妥当であり、また現在多くの人がこの立場を取っている。

(4) 多項式時間で解けない問題は実際のでない。

(4)は実際のでないことの一つの判定基準であって、多項式時間で解けることが实际的であると主張しているのではないことに注意する。

P を Turing 機械で多項式時間で解ける問題のクラスとする。Turing 機械は、逐次型計算機の一般的モデルとみなしてよい。したがって、もし P に属さない問題で、並列計算機で多項式時間で解けるものの存在を示すことができるなら、並列計算は逐次型計算より

真に強力であることが示せたことになる。

非決定性機械 非決定性という概念は古くからある概念で、一種の並列計算と見ることができる。非決定性 Turing 機械は2章で述べた Turing 機械の基本命令の外に次の命令を持つ。

$\exists \text{ fork } l_1, l_2$

ここで l_1 と l_2 はラベルである。この命令に出会うと機械は2つの子供を産む。子供のテープの内容、およびヘッドの位置は親と同じである。一方の子供は l_1 番目の命令より、もう一方の子供は l_2 番目の命令より実行を開始する。親は子供から答えが返ってくるまで休む。一方の子供から受理の答えが返った場合、直ちに自分の親に受理と答える。両方の子供から拒否の答えが返った場合は親に拒否と答える。最初1つの機械が計算を開始し、途中で2つの子供を産み、その子供がまた子供を産み、どんどん子供がふえていく。ある時点である子供が **accept** 命令に達したなら、次々と親に受理の信号を送り、入力は受理される。時間量は、計算を開始してから、木の根にあたる最初の機械に受理または拒否の信号がとどくまでの時間である。領域量は、各々の機械が使用するセルの個数の最大値である。

非決定性 Turing 機械で多項式時間で受理される問題のクラスを NP で表わす。次は重要な未解決問題の一つである。

(5) $P=NP$ か?

非決定性機械は非常に強力で、多くの問題を効率よく解くことができる。 NP に属するが、まだ多項式時間のアルゴリズムが知られていない膨大な数の問題のリストが発表されている⁹⁾。その中には古典的な問題も含まれており、現在では多くの人が $P \neq NP$ と予想している。領域量に関しては次が知られている¹²⁾。

(6) 非決定性で $S(n)$ 領域 $\subset S^2(n)$ 領域

左辺は非決定性 Turing 機械で $S(n)$ 領域で計算できる問題のクラスを表わし、右辺は Turing 機械で $S^2(n)$ 領域で計算できる問題のクラスを表わす。

$PSPACE$ で多項式領域で解ける問題のクラスを表わす。すると、(6)と定義より次が成立する。

(7) $P \subset NP \subset PSPACE$

交互機械 非決定性機械は受理のときのふるまいと拒否のときのふるまいが双対的でない。交互 Turing 機械 (alternating Turing machine, 以後 ATM と略す) とは非決定性 Turing 機械に次を基本命令として加えたものである。

$\forall \text{fork } l_1, l_2$

この命令に出会うと、 $\exists \text{fork}$ 命令のときと同様に、2つの子供を産み、子供から答えが返ってくるまで休む。2つの子供的一方が拒否の答えを返したら、親に拒否の答えを返し、2つの子供から受理の答えを受け取ったときは親に受理の答えを返す。木の根の所にある最初の機械が受理の信号を受け取ったなら入力を受け取り、拒否の信号を受け取ったなら入力を拒否する。したがって、ATM では子供を産んで子供から返事を待っている機械は、 $\exists \text{fork}$ 命令を実行しどちらか一方からの受理信号を待っているものと、 $\forall \text{fork}$ 命令を実行し両方からの受理信号を待っているものの2種類ある。時間量、領域量の定義は非決定性機械のときと同様である。ATM は理論的に興味あるモデルで、問題のクラスの間のいくつもの美しい関係を与える^{9), 10), 16)}。とりわけ次は重要である。

(8) $P = \text{ATM}$ で対数領域計算可能

(9) $\text{PSPACE} = \text{ATM}$ で多項式時間計算可能
対数領域計算可能性の厳密な定義は略すが、直観的には、長さ n の任意の入力に対し、入力以外に本質的に使用するセルの個数が高々 $\log n$ のときをいう。

ATM は並列計算機の一般的モデルとみることができる。実際3章で述べた並列計算モデルはすべて ATM と同程度の能力である。したがって(9)は次のように言い換えることができる。

(10) $\text{PSPACE} = \text{並列計算機で多項式時間}$

5. おわりに

前にも述べたように、NP は数多くの重要な問題を含んでいる。(7)と(10)より、これらの問題は並列計算機で多項式時間で解ける。それでは近い将来並列計算機が実現し、現在困難だと予想されている NP に属する多くの問題が効率よく解けるようになるのだろうか。これは現時点ではあまり期待できない。一つの理由は、ここで述べたモデルはどれも理想化された理論的モデルであって、実際に実現するためにはいくつかの問題点を含んでいるからである。本特集の VLSI モデルに関する解説で論ぜられると思うが、実際に機械を構成しようとする場合、平面とか立方体の中に配置しなければならない、また情報の伝達する速度も考慮しなければならない。実際、このような物理的制約から見て、どのような並列計算機も Turing 機械と多項式時間同級であるという議論もある¹⁾。しかしこのことは、情報の伝達速度がその他の時間と比べ無視で

きる程度のものであればよいわけで、現時点で結論を出すことはできないと思う。実際に実現するにあたっての、より具体的な研究がさらに必要であろう。もう一つの理由は、非決定性の実現方法にある。4章で述べた定義を直接に模倣するような方法だと、産み出される機械の数は指数関数的なものとなり、結局は破綻する。

これまで多項式倍の時間量の差を無視するという立場を取ってきた。一つの理由は、4章で述べたように、並列計算が従来の計算量理論と密接な関係にあることを述べたかったからであり、もう一つの理由は、理論の弱さのためである。換言すれば、多項式時間同級という大ざっぱな見方をしても、現在のところ逐次型計算と並列計算の同値性を証明することができない。しかし、並列計算に多項式倍以上の効率のよさを要求するのは、いささか法外な気もする。最近、論理回路モデルや VLSI モデルにおいて、計算量の下界を示す手法が開発されており、将来もっと細かい計算機モデル間の能力の比較ができるようになるものと思う。

逐次型計算で多くの計算量を必要とする問題で、並列計算を導入してもそれほど計算量が減少しないものも多くあるであろう。NP の問題の多くはそうであるかも知れない。しかし重要なことは、並列計算によって大幅に計算量が減少する問題が多数存在するかどうかであり、最近効率のよい並列アルゴリズムが多数発表されていることは、このことを強く示唆している。これらのアルゴリズムの多くのものは、時間量が $O(\log n)$ とか $O(\log^2 n)$ などといった、入力サイズ n より小さい時間量を持つものである。逐次型計算では、たいていの場合入力を全部調べなければならないから、時間量を n より小さくすることはできない。したがって、これらの問題に対しては明らかに逐次型計算より並列計算の方が強力である。現在のところ、理論はこういった低いレベルの計算量を扱えるまでに至っていないように思う。Turing 機械は n^2 以下の時間量や、 $\log n$ より小さい領域量を扱うのに十分説得力のあるモデルであると断言できない。ヘッドを一挙に遠くに飛ばす能力を加えたり、複数個のヘッドを持たせたりするような、モデルのちょっとした変更のもとで、この程度の計算量は安定したものとはいえない。むしろ論理回路モデルとか VLSI モデルの方がより適切ではないかと思う。この分野の今後の発展が、計算の機構に関するいろいろな事柄を明らかにしてくれることを期待する。

参 考 文 献

- 1) Chazell, B. and Monier, L. : Unbounded Hardware is Equivalent to Deterministic Turing Machines, *Theor. Comput. Sci.*, Vol. 24, pp. 123-130 (1983).
- 2) Cook, S. A. and Reckhow, R. A. : Time Bounded Random Access Machines, *J. Comput. Syst. Sci.*, Vol. 7 (1973).
- 3) Chandra, A. K., Kozen, D. C. and Stockmeyer, L. J. : Alternation, *J. ACM*, Vol. 28, pp. 114-133 (1981).
- 4) Dymond, P. and Cook, S. A. : Hardware Complexity and Parallel Computation, *Proc. IEEE*, Vol. 21, pp. 360-372 (1980).
- 5) Garey, M. R. and Johnson, D. S. : *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company (1979).
- 6) Goldshlager, L. M. : A Universal Interconnection Pattern for Parallel Computers, *J. ACM*, Vol. 29, pp. 1073-1086 (1982).
- 7) Fortune, S. and Wyllie, J. : Parallelism in Random Access Machines, *STOC*, Vol. 10, pp. 114-118 (1978).
- 8) Hartmanis, J. and Simon, J. : On the Power of Multiplication in Random Access Machines, *Proc. IEEE*, Vol. 15, pp. 13-23 (1974).
- 9) Kannan, R. : Alternation and the Power of Nondeterminism, *STOCK* (1983).
- 10) Paul, W. and Reischuk, R. : On Alternation, *Acta Informat.*, Vol. 14, pp. 243-255 (1980).
- 11) Pratt, V. R. and Stockmeyer, L. J. : A Characterization of the Power of Vector Machine, *J. Comput. Syst. Sci.*, Vol. 12, pp. 198-211 (1976).
- 12) Savitch, W. J. : Relationships Between Nondeterministic and Deterministic Tape Complexities, *J. Comput. Syst. Sci.*, Vol. 4, pp. 177-192 (1970).
- 13) Savitch, W. J. : Parallel Random Access Machines with Powerful Instruction Set, *Math. Systems Theory*, Vol. 15, pp. 191-210 (1982).
- 14) Savitch, W. J. and Stimson, M. J. : Time Bounded Random Access Machines with Parallel Processing, *J. ACM*, Vol. 26, pp. 103-118 (1979).
- 15) Schwartz, J. T. : Ultracomputers, *ACM Trans. Prog. Lang. Syst.*, Vol. 2, pp. 484-521 (1980).
- 16) Ruzzo, W. L. : Tree-size Bounded Alternation, *J. Comput. Syst. Sci.*, Vol. 21, pp. 218-235 (1980).

(昭和 60 年 1 月 7 日受付)