アルゴリズム 21－3
（1991. 5. 29）

到達可能性及び充足可能性判定問題に対するメモリ型並列アルゴリズム

高木一義　　武永康彦　　矢島脩三

京都大学工学部 情報工学教室

本報告では、ランダムアクセス機械に機能メモリを付け加えた計算モデルの上での並列アルゴリズムを提案する。ここで考える機能メモリは、複数ワードへの、部分一致による並列アクセスが可能である。到達可能性判定問題に対するアルゴリズムでは、節点数 n に対して $(n+1)$ ビット $\times$ $2n$ ワードの機能メモリを用いて、$O(n \log n)$ 時間で、有向グラフの可到達性行列を求める。和積標準形論理式の充足可能性判定問題に対しては、定数ワードの機能メモリを用いて大きな規模の問題を解くアルゴリズムを考える。このアルゴリズムでは、利用できる機能メモリの大きさに応じた並列性を得ることができる。

Memory-Based Parallel Algorithms for Reachability and Satisfiability Problems

Kazuyoshi TAKAGI　　Yasuhiko TAKENAGA　　Shuzo YAJIMA

Department of Information Science, Faculty of Engineering, Kyoto University

Yoshida-Honmachi, Sakyo-ku, Kyoto 606, Japan

In this report, we propose parallel algorithms on a computation model which consists of a random access machine and a functional memory. The functional memory allows parallel access on each word through masked search. First, we describe an algorithm for a reachability problem, which computes the reachability matrix of the given digraph with n nodes. This algorithm requires the functional memory of $(n+1)$ bits $\times$ $2n$ words and $O(n \log n)$ time. Next, we describe an algorithm for CNF-SAT, which works on a functional memory with constant words. We can fully use the parallelism of the functional memory in the algorithm.

1 はじめに

並列計算は、計算量の大きな問題を扱うための重要な手段の一つである。このため、様々な並列計算モデルが提案され、その上でのアルゴリズムの研究がなされている。

並列計算に利用できるハードウェアの一つに機能メモリがある。機能メモリとは、通常のランダムアクセスメモリに簡単な論理操作機能を付加し、記憶しているデータに何らかの操作を行なうことができるようにしたメモリである。機能メモリは、各ワードが簡単な機能を持つ一種の SIMD (Single Instruction Multiple Data stream) 型の並列計算機構として捉えることができる。機能メモリは、通常のメモリと同様、簡単な構成単位が繰り返される構造を持つため、VLSI 化に有利であり、並列度の高い計算機構を実現できる。記憶内容によるデータアクセスが可能である連想メモリ (CAM : Content Addressable Memory) は、機能メモリの一種である。現在、20K ビットの連想メモリ LSI が試作されている [1]。

連想メモリなどの機能メモリでは、全ワードに対する同一の処理を定数時間で行なうことができる。このため、NP 完全や NP 困難な組合せ問題や組合せ最適化問題が、機能メモリを用いたアルゴリズムによって多項式時間で解けることがある。これまでの研究で、機能メモリを用いた並列計算モデルの能力が明らかにされている [2, 3]。また、機能メモリを用いたアーキテクチャやアルゴリズムについても研究されている [4, 5]。

本報告では、二つの問題に対する、機能メモリを用いたメモリ型並列アルゴリズムを提案する。第一は、有向グラフの到達可能性判定問題である。本報告では、この問題を可到達性行列を求める問題として扱っている。$n \times n$ ブール行列の乗算が $O(n^{2.376})$ 時間で可能であることから、この問題に対して、節点数を n として $O(n^{2.376} \log n)$ の逐次アルゴリズムが考えられる。本報告では、計算時間 $O(n \log n)$ のメモリ型並列アルゴリズムを提案する。

第二は、代表的な NP 完全問題である、和積標準形論理式の充足可能性判定問題である。この問題については、入力の大きさに対して線形計算時間のメモリ型並列アルゴリズムが知られている [2]。しかし、このアルゴリズムは、現実の機能メモリを考えた場合、その容量を超える問題に対応することができない。そこで、限定されたワード数の機能メモリを用いて大きな規模の問題を解くアルゴリズムを考えることが必要である。本報告では、メモリ型並列アルゴリズムを逐次アルゴリズムと組み合わせて、機能メモリの使用ワード数を定数に限定したアルゴリズムを提案する。

以下、第 2 章では準備として機能メモリ及び本報告で用いる計算モデルについて述べる。第 3 章、第 4 章ではそれぞれの問題に対するメモリ型並列アルゴリズムを提案し、その評価を行なう。

2 機能メモリを用いた並列計算モデル

本報告で考える計算モデルは、図 1 に示すように、通常のランダムアクセス機械 (RAM)[6] に機能メモリを付け加えたものである。

RAM は、有限制御部と、任意の自然数を記憶できる無限個のレジスタからなる。レジスタには、累算器、入力レジスタ、出力レジスタと呼ばれる特別なレジスタが含まれる。累算器は RAM 上の全ての演算を行なうレジスタである。入力及び出力レジスタは一般に複数個であってよく、それぞれ問題のインスタンスの入力と、解の出力に用いられる。

ここで用いる機能メモリは、アドレスによるデータのアクセスの他に、部分一致検索、一致ワードに対する同時書き込み、及び一致ワードの読み出しの機能を持つ。機能メモリは、任意の自然数を記憶できる無限個のワード m_i $(i \in N = \{0, 1, 2, \ldots\})$、各ワードに対して 1 ビットの検索結果フラグ f_i $(i \in N)$、1 ビットの一致ワード有無フラグ F からなる。各ワード m_i には通常のランダムアクセスメモリと同様にアドレ

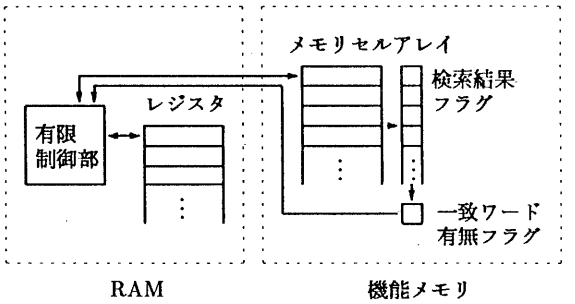

図 1: 機能メモリを用いた計算モデル

スによるアクセスが可能である。

以下では、自然数と、その2進表現によるビット列を同一視する。つまり、ビット列 $a_0a_1\ldots a_j\ldots$ を自然数 $\sum_{j=0}^{\infty} a_j 2^j$ と同じものと考える。また、$\{j \mid a_j = 1\}$ (値が 1 であるビット位置の集合) の要素を $\langle\ \rangle$ で括って対応するビット列を略記する。ビット列に対する論理演算 $\vee, \wedge, \bar{}$ はそれぞれビット毎の $\{0,1\}$ 上の論理和、論理積、否定である。

本報告で考える計算モデルの命令セットは、通常の RAM の命令セットに、機能メモリを用いた次の命令を加えたものである。以下で、a はアドレス、d は検索 / 書き込み / 読み出しデータ、m はマスクデータであり、それぞれ任意のビット列である。

部分一致検索 FMSEARCH d, m, op

マスクデータ m が 0 である全てのビット位置において、ワードの内容 m_i と検索データ d とが一致するならば検索結果を 1 、そうでなければ 0 とする。この値と検索結果フラグ f_i の内容に演算 op を行なった結果を、新たに検索結果フラグの内容とする。全てのワードの検索結果フラグの内容の論理和を、一致ワード有無フラグ F に記憶する。op は任意の二項論理演算であるが、本報告では、$THRU_S(s,f) = s$, $NOT_S(s,f) = \bar{s}$, $NOT_F(s,f) = \bar{f}$, $AND(s,f) = s \wedge f$, $OR(s,f) = s \vee f$ の 5 つを使用する。s は検索結果、f は検索結果フラグの内容である。

一致ワード同時書き込み FMWRITE d, m

検索結果フラグ f_i の内容が 1 であるワードの、マスクデータ m が 0 であるビット位置に、書き込みデータ d の、そのビット位置の値を書き込む。

一致ワード読みだし FMREAD a, d

検索結果フラグ f_i の内容が 1 であるワードのうち、アドレス i が最小のもののアドレスとデータを読み出す。

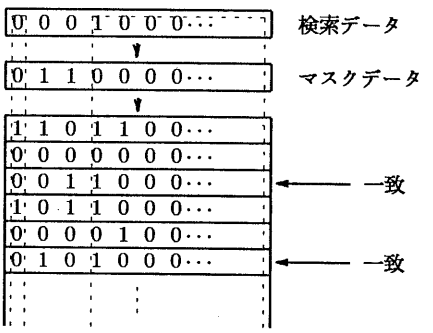

図 2: 部分一致検索の例

これらの命令の 1 命令あたりの実行時間は、使用ビット数やワード数によらず一定であるとする。機能メモリのワード、フラグの初期値は全て 0 とする。図 2に部分一致検索の例を示す。

本報告で考える機能メモリは、連想メモリ LSI で実現されている基本的な機能とほぼ同様の機能を持つ。機能メモリを用いた計算モデル、あるいはアーキテクチャとして、CAFRAM[3]、FMPP[5] 等が提案されている。これらは、メモリの一部を ROM 化してアドレスを持たせることにより、アドレスに対する部分一致検索が可能である。k ビットのアドレスで、2^k ビットのワードを指定できることから、入力に対し指数オーダーのワードを並列に操作できる。このような機能メモリは、多くの問題に対して有効に利用できることが示されている [5]。

3 グラフの到達可能性判定問題のメモリ型並列アルゴリズム

3.1 グラフの到達可能性判定問題

有向グラフ $G = (V, E)$ (節点集合 $V = \{1, 2, \ldots, n\}$, 枝集合 $E \subseteq V^2$) において、節点 $p \in V$ から節点 $q \in V$ へ到達可能であるとは、p から q への有向径路が存在することである。つまり、互いに異なる節点の列 $(p = v_0, v_1, \ldots, v_l = q)$ で、$\forall k \in \{0, 1, \ldots, l-1\}$, $(v_k, v_{k+1}) \in E$ を満たすものが存在するということである。

n 節点の有向グラフ G の行列表示の一つである節点隣接行列 A は次のようなものである。

節点隣接行列 $A = (a_{ij})$ $\left(a_{ij} = \begin{cases} 1, & (i,j) \in E \\ 0, & \text{otherwise} \end{cases} \right)$

任意の節点間の到達可能性は次に示す可到達性行列 R で表すことができる。

可到達性行列 $R = (r_{ij})$
$\left(r_{ij} = \begin{cases} 1, & \text{節点 } i \text{ から節点 } j \text{ へ到達可能} \\ 0, & \text{otherwise} \end{cases} \right)$

本章では、グラフの到達可能性判定問題を、与えられた有向グラフの全ての節点の組の到達可能性を判定すること、即ち、可到達性行列を求める問題として扱う。

可到達性行列 R は、

$$R = \sum_{i=0}^{n-1} A^i = (A+I)^{n-1} \qquad (1)$$

として求められる。これは、$A+I$ を基にして自乗を繰り返し、$(A+I)^2, (A+I)^4, \ldots$ を計算していくことにより求められる。従って $\lceil \log(n-1) \rceil$ 回ブール行列の乗算を行なえばよい。

現在、$n \times n$ 行列の乗算の逐次アルゴリズムとして、$O(n^{2.376})$ のものが得られている [7]。これを用いて、$O(n^{2.376} \log n)$ 時間で可到達性行列を求める逐次アルゴリズムが考えられる。

3.2 メモリ型並列アルゴリズム

本節では、グラフの到達可能性判定問題に対するメモリ型並列アルゴリズムを提案する。まず、このアルゴリズムの基本となるブール行列の乗算のメモリ型並列アルゴリズムを図3に示す。

入力は、$n \times n$ 行列 A, B 、出力は $C = AB$ である。行列 A, B はそれぞれ行ベクトル、列ベクトル毎にビット列 $\{a_i = a_{i,0}a_{i,1}\cdots a_{i,n-1}\}$, $\{b^j = b_{0,i}b_{1,i}\cdots b_{n-1,i}\}$ としてRAMの入力レジスタに格納されているとする。行列 C は行ベクトル毎のビット列 $\{c_i = c_{i,0}c_{i,1}\cdots c_{i,n-1}\}$ として出力する。

2: の (1) では、機能メモリに読み込まれている行列 A に b^i を乗算する。まず、b^i が 1 であるビット位置が全て 0 であるワード、つまり、b^i と内積をとると 0 になる A の行を検索する。マスクデータを b^i のビットを反転したもの、検索データを 0 とすることにより、このよ

うな検索が 1 命令で可能である。検索結果の否定をとれば、Ab^i が得られる。(2) でこの結果を機能メモリに書き込む。3: は、第 $n, n+1, \ldots, 2n-1$ ビットに得られた行列 C を第 $0, 1, \ldots, n-1$ ビットに移す操作である。2: 及び 3: の操作は全てワード並列に行なわれる。

このアルゴリズムで必要な機能メモリの領域は $2n$ ビット $\times$ n ワードである。また、計算時間は $O(n)$ である。

次に、このアルゴリズムを基にして、有向グラフの節点隣接行列から可到達性行列を求めるアルゴリズムを考える。節点隣接行列 A と単位行列 I の和 $\hat{A}$ の累乗を計算していけばよいのだが、以上で示したアルゴリズムをそのまま用いると無駄な操作が多いため、さらに次の 3 点を考慮する。

- 図3のアルゴリズムでは、一方の行ベクトルと他方の列ベクトルを入力として与えねばならないので、$\hat{A}$ の累乗を計算するには $\hat{A}$ の行ベクトルと列ベクトルが必要になる。このために、機能メモリに行列を記憶する 2 つの領域を用意する。以後その内容を行列 U, V と書く。こうすることにより、図3のアルゴリズムを用いて一方の行列の転置行列を他方に右から掛ける操作、つまり

  $$U := U\,{}^tV, \quad V := V\,{}^tU \qquad (2)$$

  (${}^tV$, ${}^tU$ は V, U の転置行列) の計算ができる。
- 最初に行列 $\hat{A}$ と単位行列 I をそれぞれ U, V に書き込んで、式 (2) の計算を繰り返せば、$\hat{A}$ の列ベクトルを求めずに R が求められる。

1: 入力レジスタから機能メモリの最初の n ワードに行列 A の行ベクトルを書き込む
2: $i := 0$ から $n-1$ について以下を繰り返す　{ $C = AB$ の計算 }
　(1) FMSEARCH　$0, \overline{b^i}, NOT_S$　{ $c^i = Ab^i$ の計算 }
　(2) FMWRITE　$\overline{0}, \overline{\langle n+i \rangle}$　{ c^i の書き込み }
3: 各ワードの第 $0, 1, \ldots, n-1$ ビットに 0 を書き込む　{ A の消去 }
　$i := 0$ から $n-1$ について以下を繰り返す　{ C の移動 }
　(1) FMSEARCH　$\overline{0}, \overline{\langle n+i \rangle}, THRU_S$　{ c^i の読み出し }
　(2) FMWRITE　$\langle i \rangle, \overline{\langle i, n+i \rangle}$　{ c^i の書き込み }
4: 行列 C を機能メモリから出力レジスタへ読み出す

図 3: ブール行列の乗算

- $(A+I)$ の累乗を計算していくことは、「節点 i へ到達可能な節点を x とすると、x へ到達可能な任意の節点もまた i へ到達可能である」という性質に基づいている。この計算を節点 i について逐次的に行なっても、到達可能性行列は正しく求められる。従って、$(A+I)$ の累乗そのものを求める必要はなく、乗算の結果を一列ずつ求め、元の行列を書き換えながら計算していっても正しい結果が得られる。

図4に、可到達性行列を求めるメモリ型並列アルゴリズムを示す。入力は、節点隣接行列 A 、出力は、可到達性行列 R である。行列 A は行ベクトル毎にビット列 $\{a_i = a_{i,0}a_{i,1}\cdots a_{i,n-1}\}$ として入力レジスタに格納されているとする。行列 R は行ベクトル毎のビット列 $\{r_i = r_{i,0}r_{i,1}\cdots r_{i,n-1}\}$ として出力する。

1: では、行列を読み込んで、$U = \hat{A}^1, V = {}^tI = {}^t\hat{A}^0$ とする。U, V は、それぞれ $m_0, m_1, \ldots, m_{n-1}$、$m_n, m_{n+1}, \ldots, m_{2n-1}$ の第 $0, 1, \ldots, n-1$ ビットである。ここで、U と V の領域を区別するために V のワードの第 n ビットを 1 にする。2: では、3 のアルゴリズムを用いて、$\hat{A}^{n-1}$ が得られるまで交互に乗算を繰り返す。FMWRITE は、実際には元のデータとの論理和を書き込む操作であるが、U, V の対角成分は全て 1 であるので、1 であるビットに 0 を書き込むことはなく、単なる書き込みと等価である。2: の繰り返しの中の一回の操作を式で書くと次のようになる。

$$\begin{cases} v_{ji} := \bigvee_{k=0}^{n-1} (u_{ik} \wedge v_{jk}) \\ u_{ji} := \bigvee_{k=0}^{n-1} (v_{ik} \wedge u_{jk}) \end{cases}$$

この操作は $j \in \{0, 1, \ldots, n-1\}$ についてワード並列に行なわれる。これは、${}^tU$ (あるいは ${}^tV$) の列ベクトルを V (U) に掛けて、その結果を新たに V (U) の列ベクトルとする操作である。このアルゴリズムでは、これを $i = 0, 1, \ldots, n-1$ の順に行ない、行列を更新しながら計算を進めるので、その結果の V (U) は、$V\,{}^tU$ ($U\,{}^tV$) に比べて、余分な 1 が書き込まれる可能性がある。以後簡単のため、この操作と通常のブール行列の乗算との違いを特に断らない。始めに $U := \hat{A}, V := {}^tI$ としておくと、2: では $V := {}^t\hat{A}^1, U := \hat{A}^2, \ldots$ の計算を行なうことになる。

このアルゴリズムで必要な機能メモリの領域は $(n+1)$ ビット × $2n$ ワードである。次に、計算時間の評価を行なう。2: の繰り返しで $\hat{A}$ の $1, 2, 3, 5, 8, 13, \ldots$ 乗が得られる。これは Fibonacci 数列であり、第 i 回の乗算で得られる $\hat{A}$ の指数を p_i とすると、$p_i \geq 2^{i/2}$ である。従って、$\hat{A}$ の $(n-1)$ 乗を得るには行列の乗算を $O(\log n)$ 回繰り返せばよい。1回の行列の乗算の計算時間は $O(n)$ であるので、このアルゴリズムの計算時間は $O(n \log n)$ である。

本節で提案したアルゴリズムでは、RAM 上の行列のデータ構造に機能メモリに読み込むために都合のよ

1: $i := 0$ から $n-1$ について以下を繰り返す　{ $U := \hat{A}$ }
　　$m_i := a_i \vee \langle i \rangle$
　$i := 0$ から $n-1$ について以下を繰り返す　{ $V := I$ }
　　$m_{n+i} := \langle i, n \rangle$
2: U に $\hat{A}^{n-1} = R$ が得られるまで以下を繰り返す
　(1) $i := 0$ から $n-1$ について以下を繰り返す　{ $V := V\,{}^tU$ }
　　i) FMSEARCH $0, \overline{m_i} \vee \langle n \rangle, NOT_S$　{ $V\,{}^tu_i$ の計算 }
　　ii) FMSEARCH $\overline{0}, \overline{\langle n \rangle}, AND$　{ U の検索結果フラグを 0 にする }
　　iii) FMWRITE $\overline{0}, \overline{\langle i \rangle}$　{ 乗算結果の書き込み }
　(2) $i := 0$ から $n-1$ について以下を繰り返す　{ $U := U\,{}^tV$ }
　　i) FMSEARCH $0, \overline{m_{n+i}} \vee \langle n \rangle, NOT_S$　{ $U\,{}^tv_i$ の計算 }
　　ii) FMSEARCH $0, \overline{\langle n \rangle}, AND$　{ V の検索結果フラグを 0 にする }
　　iii) FMWRITE $\overline{0}, \overline{\langle i \rangle}$　{ 乗算結果の書き込み }
3: 得られた行列 R を機能メモリから出力レジスタに読み出す

図 4: 可到達性行列の計算

い形を仮定した。行列をここで用いたデータ構造で与えるための処理に $O(n\log n)$ 以上の計算時間を要する場合、これらのアルゴリズムの計算時間のオーダーはその処理に依存する。

4 論理式の充足可能性判定問題のメモリ型並列アルゴリズム

4.1 論理式の充足可能性判定問題

和項数 t の n 変数和積標準形 (Conjunctive Normal Form) 論理式

$$f(x_0,x_1,\ldots,x_{n-1})=\bigwedge_{i=0}^{t-1}\left(\bigvee_{l\in L_i} l\right)$$

L_i : 和項 i に属するリテラルの集合

$$(L_i\subseteq\{x_0,x_1,\ldots,x_{n-1},\overline{x_0},\overline{x_1},\ldots,\overline{x_{n-1}}\})$$

が、充足可能であるとは、

$$\exists(x_0,x_1,\ldots,x_{n-1})\in\{0,1\}^n,$$
$$f(x_0,x_1,\ldots,x_{n-1})=1$$

であることである。和積標準形論理式の充足可能性問題 (CNF-SAT) は、与えられた和積標準形論理式が充足可能であるか否かを判定する問題である。

CNF-SAT を解く効率的な逐次アルゴリズムは古くから提案されている。その基本的な手法には、全解探索法、後戻り法等がある。全解探索法は、変数への値の割り当てを順に全て探索する方法であり、最も原始的な方法である。後戻り法は、変数に一つずつ値を割り当てていき、充足解を探索する。それ以上値を割り当てていっても充足の可能性がないと分かれば、そこで探索を打ち切り、後戻りをして別の値の割り当てを探索する。探索打ち切りの判断のため、f に対して $(n+1)$ 個の評価関数

$$\{f_k(x_0,x_1,\ldots,x_{k-1})\}\ (k\in\{0,1,\ldots,n\})$$

を考える。ここで、$f_n=f$, $f_0=1$ である。$0<k<n$ に対しては、$f_k(x_0,x_1,\ldots,x_{k-1})=0$ となる $(x_0,x_1,\ldots,x_{k-1})$ の値に対して、

$$\forall x_k\in\{0,1\},\ f_{k+1}(x_1,x_2,\ldots,x_k)=0$$

となるようなものを選ぶ。f が和積標準形である時、変数 $x_0,x_1,\ldots,x_{k-1}$ のリテラルだけからなる f の和項の全ての積を f_k とすれば、これはこの条件を満たし、また f から簡単に求められる。

この他に、これらの手法に実際の問題に対して有効な発見的手法を組み合わせた逐次アルゴリズムが多く考えられ、その性質が研究されている。

機能メモリを用いた高速アルゴリズムの代表的なものとして、CNF-SAT を解くメモリ型並列アルゴリズムが知られている [2]。このアルゴリズムでは、2^n 個のワードを使って、各ワードのアドレスを n 個の変数に対する値の割り当てと見て、全ての値の割り当てについて探索するものである。図5にこれを示す。

2: では、ある和項が 0 になるような値の割り当てに対応するワードを検索する。これは、アドレスが、その和項に正のリテラルが現れる変数に対応するビットが 0、負のリテラルに対応するビットが 1 であるワードを検索することにより行なう。マスクデータは、和項にリテラルとして含まれる変数に対応するビットを 0 とする。これらのビットが、リテラルが正であれば 0、リテラルが負であれば 1 である検索データを与えることにより、この検索が 1 命令で可能である。3: で検索結果フラグが立っていないワードを検索し、f の充足可能性を判定する。

このアルゴリズムで必要な機能メモリの領域は、n ビット $\times\ 2^n$ ワードである。計算時間は $O(2^n+tn)$ である。

1: 各ワードにそのアドレスを書き込む
2: $i:=0$ から $t-1$ について以下を繰り返す
(1) 和項 L_i に対し、検索データ $s=s_0s_1\cdots s_{n-1}$ マスクデータ $m=m_0m_1\cdots m_{n-1}$ をつくる
$s_j:=(x_j\notin L_i)\wedge(\overline{x_j}\in L_i)\quad(j=0,1,\ldots n-1)$ { 検索データ }
$m_j:=(x_j\notin L_i)\wedge(\overline{x_j}\notin L_i)\quad(j=0,1,\ldots n-1)$ { マスクデータ }
(2) $\bigvee_{j=0}^{n-1}((x_j\in L_i)\wedge(\overline{x_j}\in L_i))=0$ ならば { 恒に 1 である和項の除去 }
FMSEARCH s, m, OR { 和項 i が 0 となる変数への値の割り当ての検索 }
3: フラグが立っていないワードがあれば充足可能、なければ充足不能

図 5: 機能メモリによる充足可能性判定

このアルゴリズムでは、各ワードにアドレスの書き込みを行なう部分が計算時間のオーダーを支配している。そこで、[2]では、メモリの一部をROM化してアドレスを持たせることにより、アドレスに対する部分一致検索機能を持たせた機能メモリを考えている。このような機能メモリを用いれば、このアルゴリズムの計算時間は $O(tn)$ となり、入力の大きさに対し線形の計算時間で解くことができる。

4.2 ワード数限定メモリ型並列アルゴリズム

図5のアルゴリズムは、変数の個数 n に対して 2^n ワードの機能メモリを必要とする。このアルゴリズムでは、有限ワードの機能メモリが与えられた場合、そのワード数を超える規模の問題を解くことができない。本節では、限られたワード数の機能メモリを用いて大きな規模の問題を解く、ワード数限定アルゴリズムを考える。以下では機能メモリのワード数を 2^c ワード(c は問題の大きさによらない定数)に限定する。

まず考えられるのは、全解探索法と機能メモリを用いたアルゴリズムを組み合わせることである。前節で述べたメモリ型並列アルゴリズムは、機能メモリの 2^c ワードを使って一度に c 個の変数について全解探索を行なうことができる。これを用いて $x_0, x_1, \ldots, x_{n-c-1}$ の値の全ての組合せ $v_0, v_1, \ldots, v_{n-c-1}$ について、順次、論理関数

$$f_{v_0,v_1,\ldots,v_{n-c-1}}(x_{n-c}, x_{n-c+1}, \ldots, x_{n-1}) = f(v_0, v_1, \ldots, v_{n-c-1}, x_{n-c}, c_{n-c+1}, \ldots, c_{n-1})$$

の充足可能性を機能メモリで判定する。これらに充足可能であるものがある時、またその時に限り、元の論理関数は充足可能である。

このアルゴリズムで必要な機能メモリの領域は c ビット $\times$ 2^c ワードである。論理式の評価回数は最悪で 2^{n-c} 回であるので、計算時間は $O(2^n tn)$ である。機能メモリにより探索空間が 2^n から 2^{n-c} に縮小されたと考えることができる。

逐次アルゴリズムの後戻り法は、現実に解くことを要求される多くの問題に対して、全解探索法に比べ効率的であるとされている。これを、機能メモリを用いたアルゴリズムと組み合わせれば、全解探索法と組み合わせたものよりも効率の良いアルゴリズムが得られると考えられる。これには、前節で述べたメモリ型並列アルゴリズムを、評価関数の充足可能性判定に用いればよい。このアルゴリズムを図6に示す。*level* はすでに値を割り当てられた変数の個数である。

3:では、現在の *level* 個の変数への値の割り当てに対して、次の c 個の変数に対する評価関数の充足可能性判定を行なう。この判定には、前節のメモリ型並列アルゴリズムを利用する。但し、 $level > n - c$ のときは評価関数は f とし、 $(n - level)$ 個の変数に対して充足可能性を判定する。各ワードにはスタックが設けられ、充足可能性が残っている値の割当が、FMWRITE を用いて記憶される。これは、5:,6:で、

1: 各ワードにそのアドレスを書き込む
2: $level := 0$
3: 評価関数 $f_{level+c}(x_0, x_1, \ldots, x_{level+c-1})$ が 1 になる
$(x_{level}, x_{level+1}, \ldots, x_{level+c-1})$ への値の割り当てを調べる 存在しなければ 6: へ
4: $level \geq n - c$ ならば f は充足可能と判定し停止
5: (1) $x_{level}, x_{level+1}, \ldots, x_{level+c-1}$ に充足の可能性がある値の組の一つを割り当てる
(2) 探索を一段階進める ($level := level + c$) 3: へ
6: (1) 後戻り ($level := level - c$)
(2) $level < 0$ ならば f は充足不能と判定し停止
7: (1) $x_{level}, x_{level+1}, \ldots, x_{level+c-1}$ に
充足の可能性がある、まだ探索されていない値の組の一つを割り当てる
そのような値の組がなければ 6: へ
(2) 探索を一段階進める ($level := level + c$) 3: へ

図 6: 定数ワードの機能メモリによる充足可能性判定

FMREAD を用いて値の割り当てに利用される。3: の評価関数が値 1 をとるような変数への値の割り当てが存在し、全ての変数に値が割り当てられていれば、f は充足可能である。また、6: で $level < 0$ となれば、全ての値の割り当てを探索し尽くしたことになり、f は充足不能である。

このアルゴリズムは、n 個の 2 値変数についての論理関数を、$\lceil n/c \rceil$ 個の 2^c 値変数についての論理関数として扱って、充足可能性を判定していると見做すことができる。このアルゴリズムで必要な機能メモリの領域は $(c + \lceil n/c \rceil)$ ビット × 2^c ワードである。また、論理式の評価回数は最悪で

$$\sum_{k=0}^{\lceil n/c \rceil - 1} 2^k = \frac{2^{c \times \lceil n/c \rceil} - 1}{2^c - 1} = O(2^n)$$

であるので、計算時間は $O(2^n tn)$ である。

さらに、本アルゴリズムでは用いていないが、通常の後戻り法で用いられる様々な発見的手法は、メモリ型並列アルゴリズムでも有効であると考えられる。

5 むすび

本報告では、ランダムアクセス機械に機能メモリを付け加えた計算モデルの上で、グラフの到達可能性判定問題、及び論理式の充足可能性判定問題に対するメモリ型並列アルゴリズムを提案した。

有向グラフの到達可能性判定問題に対しては、節点数 n に対して $(n + 1)$ ビット × $2n$ ワードの機能メモリを用いて計算時間 $O(n \log n)$ のアルゴリズムを得た。このアルゴリズムはブール行列の乗算アルゴリズムを基にしている。メモリ型並列アルゴリズムは、本質的に全ての場合について並列に探索するものであるので、逐次アルゴリズムでも多項式時間で解くことができる問題に対してはその並列性を利用しにくいことが多いが、このアルゴリズムは機能メモリのビット、ワード両方向の並列性を有効に利用できる例である。

和積標準形論理式の充足可能性判定問題に対しては、機能メモリのワード数を定数に制限した場合のメモリ型並列アルゴリズムを提案した。このアルゴリズムは、逐次アルゴリズムと、ワード数を限定しない場合のメモリ型並列アルゴリズムとを組み合わせて得られたものである。与えられた問題の規模に応じて、いくらでも大規模な機能メモリを利用できるとするのは現実的でない。そこで、限定されたワード数の機能メモリを用いて大規模な問題を解くアルゴリズムを考えることによって、利用できる機能メモリの規模に応じた並列性を得ることができる。

機能メモリは大きな並列性を実現することのできる計算機構である。今後の VLSI 技術の発展によりさらに大容量の機能メモリが実現可能となると考えられる。そのため、機能メモリを用いて、従来の計算機では現実的な時間で解くことのできなかった組合せ問題や組合せ最適化問題を、高速に解くことができるようになると考えられる。今後、様々な問題に対するメモリ型並列アルゴリズムの研究が重要になってくると思われる。

参考文献

[1] T.Ogura, J.Yamada, S.Yamada, M.Tan-no : A 20-kbit Associative Memory LSI for Artificial Intelligence Machines, IEEE Journal of Solid-state Circuits, 24 – 4 (Aug. 1989), 1014 – 1020.

[2] 高木直史, 武永康彦, 矢島脩三 : メモリ型並列計算モデルとその能力, 情報処理学会論文誌, 31 – 11 (1990), 1565 – 1571.

[3] 武永康彦、高木直史、矢島脩三 : 連想メモリによるメモリ型並列計算モデルとその能力, 信学技報, COMP89 – 118 (1990), 39 – 44.

[4] 大久保雅且, 安浦寛人, 高木直史, 矢島脩三 : 連想メモリを利用したハードウェア向き単一化アルゴリズム, 情報処理学会論文誌, 28 – 9 (1987), 915 – 922.

[5] 安浦寛人, 辻本泰造, 田丸啓吉 : 組合せ問題に対する機能メモリ型並列プロセッサアーキテクチャ, 信学論 (A), J 72 – A – 2 (1989), 222 – 230.

[6] A.V.Aho, J.E.Hopcroft, J.D.Ullman 共著; 野崎昭弘, 野下浩平他共訳 : アルゴリズムの設計と解析 (I , II) (サイエンス社, 1977).

[7] D.Coppersmith, S.Winograd : Matrix Multiplication via Arithmetic Progressions, Proc. 19th Ann. ACM Symposium on Theory of Computing, (1987), 1 – 6.

[8] P.W.Purdom,Jr. : Polynomial-Average-Time Satisfiability Problems, Information Sciences, 41 (1987), 23 – 42.