計算機アーキテクチャ　89－3
（１９９１．　７．　１９）

階層ビットテーブルとその応用

平木 敬　　（電子技術総合研究所／東京大学）
西田健次　　（電子技術総合研究所）
島田俊夫　　（電子技術総合研究所）

概要

語長１ビットで構成されるビットテーブルは、計算機構成上の多くの局面で重要な地位を占める。しかしながら、１ビットメモリで実現した単純ビットテーブルでは一連続のビット列に対する操作に、長さオーダーの時間が必要であり、その応用範囲が大きく限定されてきた。本論文では、ビットテーブルをサイズに対して対数的な階層数を持つ構造で実現することにより、１以上の長さを持つ連続なビット列に対するセット・リセット、ビットテスト、ビット走査等の操作が高速になることを示し、計算機における応用例について考察する。

A Hierarchical Bit Table and Its Application

Kei Hiraki
Electrotechnical Laboratory†and University of Tokyo‡

Kenji Nishida
Toshio Shimada
Electrotechnical Laboratory

†1-1-4, Umezono, Tsukuba-shi, Ibaraki 305 Japan
‡7-3-1, Hongo, Bunkyo-ku, Tokyo 112 Japan

Abstract

A bit-table is a simplest memory system with one-bit data width. A bit table is an important component of a computing system. But a simple bit-table requires a linear time to the length of the target bit-string when it performs operations on a bit-string. In this paper, a new algorigm and a hardware to accelrate bit-string operations such as set/reset on a bit-string, bit-sequence test, and a bit search.

1 はじめに

本論文はビットテーブルにおけるビット列操作の高速化を実現する方式を提案するものである。メモリとして語長が1ビットであるビットテーブルは、最も単純なメモリであり、計算機アーキテクチャ、ソフトウェアを通じて広く使われている。しかしながら、ビットテーブルを単純に1ビットのメモリで実現した場合には、1以上の長さを持ったビット列に対する操作に、長さに比例したオーダーの時間が必要である。従って、ビット列操作を多用するデータ駆動計算機または類似の細粒度並列処理装置においては、ビット列操作に要する時間は主要なオーバーヘッドの一つである。

本論文では、ビットテーブルの連続する区間を（1）全て0または1に設定する操作（ビットセット、ビットリセット）、（2）区間内の全ビットが0（または1）であることのビット列テスト、（3）始点を指定して、線形的に走査し、最初に発見した1（または0）の位置を求める操作（ビット走査）を高速化するアルゴリズムを提案し、それを実現するハードウェアを示す。従来の単純ビットテーブルを用いると、これらの操作は全て操作の区間長 N に比例した処理時間を必要とした。本論文で提案する階層ビットテーブルをもちいると、これらの操作を $O(1)$ または $O(\log N)$ オーダーの時間で実現可能であり従来ハードウェア要素としてビットテーブルを用いることが不可能であるか、非能率的な実現方法しか無かった応用場面に用いることが可能となった。

本論文では、第2節で最も簡素であるが制限の強い0方向非対称階層ビットテーブルについて述べ、第3節において順次制限をゆるめる。次に第4節で階層ビットテーブルの動作速度を示し、最後に第5節において計算機、特にデータ駆動計算機への適用を述べる。

2 単純な階層ビットテーブル

2.1 階層ビットテーブルの構成

まず、単純な構成のビットテーブルについてのべる。各階層のメモリ読み出し幅を J、階層ビットテーブルの階層数を K、ビットテーブルの大きさを $N = J^K$ とする。ただし、説明の単純化のために J は2のベキ乗である、すなわち $J = 2^M$ であるとする。

ビットテーブルはメモリ読み出し幅である J ビットごとにブロック化されていて、この J ビットが一つ上の階層のメモリの1ビットに対応している。一つ上の階層のメモリも同様に J ビットごとにブロック化されていて、更に上の階層のメモリ1ビットに対応している。この関係を順次繰り返して、J ビットのメモリである最上階層まで構成する。

最も簡単な非対称階層ビットテーブルでは、1個のビットテーブルに1個の階層構造メモリが、対称階層ビットテーブルでは2個の階層構造メモリが付随する。なお、非対称階層ビットテーブルとは、0方向と1方向の操作が非対称に定義されているビットテーブルであり、対称階層ビットテーブルとは全操作が0方向と1方向について対称なものである。この詳細は後に示す。ここから暫くの間、0方向の非対称階層ビットテーブルについて扱う。非対称階層ビットテーブルのメモリ構成を図1に示す。図1から明らかな様に、最下層のメモリはビットテーブルのアドレスに従ってアクセスされ、上層のメモリは、順次ビットテーブルのアドレスから下位のビットを取り去ったアドレスでアクセスされる。

ビットテーブルに対してアドレスを指定すると、階層ビットテーブルにおいては、K組の読み出し幅 J ビットのメモリが（同時に）読み出される。第k階層のビット値は、最下層からk－1階層までのトリー状のビットの状態を代表した値である。0方向非対称階層ビットテーブルでは、第k階層の値が0であれば、第0階層から第k－1階層までの値に関わらずアクセスしたビットテーブルの値は0であると解釈する。すなわち、そのビット以下のトリーに属する全てのビットが0であることを表している。

第k階層の値が1である場合には、アクセスしたビットの値は第0階層から第k－1階層までの値で同様に決定される。勿論、第1階層から第K－1階層までが全て1である場合には、通常のビットテーブルのように第0階層の値がビットテーブルの値である。すなわち、第 i 階層の読み出し値を $b_i, i = 0,1...K$ とすると、

$$B = \prod_{i=0}^{K-1} b_i$$ と表される。

2.2 単純な階層ビットテーブルの操作

階層ビットテーブルとして最も単純な0方向非対称階層ビットテーブルに対する主な操作について述べる。ビットテーブルに対する基本操作として、階層的でない普通のビットテーブルでは、（1）ビットテスト（読み出し）、（2）ビットセット（1の書き込み）、（3）ビットリセット（0の書き込み）が定義されている。0方向非対称階層ビットテーブルではこれらの操作は前節で述べた条件を満たすために、次のように定義される。

2.2.1 ビットテスト

前節にも示したように、まずアドレスの指し示す全階層のメモリ値を $b_i, i = 0,1...K$ とすると、読み出したビットテーブルの値は

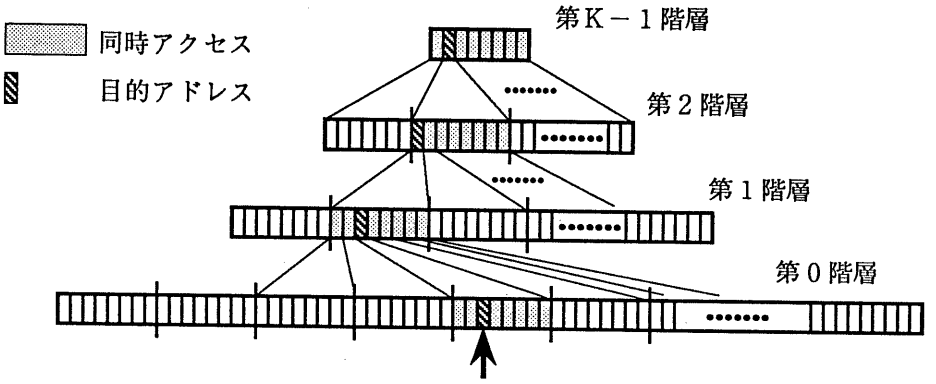

図 1: 階層ビットテーブルのメモリ構成

$B = \prod_{i=0}^{K-1} b_i$ である。

2.2.2 ビットセット

アドレス A で指定されるビットを 1 にセットする操作は、まず、アドレスで指定された全階層のメモリ値 b_i を読み出す。次に、それらアドレスで指定された全階層のメモリを 1 にセットする。もし $b_i, i = 1...K-1$ 全てがすでに 1 であった場合にはこれでビットセット動作を終る。さもなくば、$b_i = 0$ である最大の i 以下の各階層の現在アクセスしているブロックを、アドレスで指定された場所以外全て 0 にクリアする必要がある。図 2は、ビットセット時における各階層のブロックの様子を示すものである。

これらの動作をより明確に示そう。
$wb_{i,j}, i = 0...K-1, j = 0...J-1$ を階層ビットテーブルに対する書き込みデータ、$rb_{i,j}$ を階層ビットテーブルからの読み出しデータとする。階層 i におけるブロック内アドレス $a_i, 0 \leq a_i < J$ は

$$a_i = \frac{A}{J^i} \bmod J$$

と表される。従って、各階層の読み出しビット値は $b_i = rb_{i,a_i}$ であり、各階層の書き込み値は

$$wb_{i,j} = \begin{cases} 1 & \text{if } j = a_i \\ 0 & \text{if } j \neq a_i \text{ and } \prod_{k=i+1}^{K-1} \neq 1 \\ rb_{i,j} & \text{if } j \neq a_i \text{ and } \prod_{k=i+1}^{K-1} = 1 \end{cases}$$

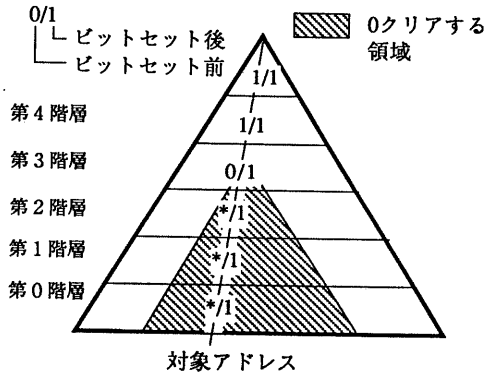

図 2: 階層ビットテーブルにおけるビットセット

2.2.3 ビットリセット

0 方向非対称階層ビットテーブルのリセットは、セット操作と異なり 1 ビットを対象とするだけでなく、複数のビットで構成されるビット列を対象とすることが 1 回のメモリ操作により可能である。

説明を簡単にするため、クリアされるビット列の先頭番地と列の大きさに、2 を基数とするバディシステムと同じ制限が付加されている場合に限定する。すなわち、ビッ

ト列の大きさは2の巾であり（$size = 2^N$）、ビット列の先頭アドレスAは、そのサイズの倍数である（$A = m2^N$）。

まず、動作の概要を述べる。クリア操作は、（1）ビット列の大きさに対応した階層のメモリを0にクリアし、（2）クリアした結果、更に上階層のビットをクリアする必要があるかどうか検査し、必要な場合、上階層もクリアすることにより実現される。更に上階層のビットをクリアすることは、ビットをクリアする階層のクリア操作の対象以外のビットが全て0であることにより判断される。ただし、後述のビット列に対する0テストが不必要である場合には、手順（1）である、ビット列の大きさに対応する階層のメモリクリアだけで機能を実現できる。

この手順をより詳しくのべる。クリアする領域のサイズを$size$、先頭アドレスをAとする。また、この領域のサイズが属する階層sは、$s = \lfloor \log_J size \rfloor$である。

まず、第s階層のメモリについて書き込み値を

$$wb_{s,j} = \begin{cases} 0 & \text{if } j \neq a_i \text{ and } \prod_{k=i+1}^{K-1} \neq 1 \\ rb_{s,j} & \text{if } j \neq a_i \text{ and } \prod_{k=i+1}^{K-1} = 1 \end{cases}$$

とする。次に、wbについて$\prod_{j=0}^{J-1} wb_{s,j} = 0$の場合には、$s+1$階層について、

$$wb_{i,j} = \begin{cases} 0 & \text{if } j = a_i \\ rb_{i,j} & \text{if } j \neq a_i \end{cases}$$

の値を書き込む。以下同様、各階層の書き込みビットが全て0で無くなるまで繰り返す。

これらの操作は、後に述べるビット列に対するテストまたはビット走査を行なうために必要な操作である。したがって、ビット列に対するテストまたはビット走査が必要ない場合には、単純に第s階層のメモリを書き込むだけで終了する。

更に、階層ビットテーブルでは、単純ビットテーブルでは基本操作として実現不可能である次の2操作を基本操作として持つ。

2.2.4 ビット列に対するテスト

ビットテーブルに対するビットクリアが前節前半の方法で実現されているばあいには、前節と同じビット列のサイズと先頭アドレスに関する制限の下に、ビット列の全要素が全て0であることのテストを行なうことが可能である。

テストは、ビット列のサイズ$size$から試験する階層sを、$s = \lfloor \log_J size \rfloor$としてもとめ、その階層の対応するメモリビットが0であることから、当該ビット列が全て0であることを検出する。

2.2.5 ビット走査

ビット走査とは、起点のアドレスをあたえることにより、そこから線形に走査し最初に出会った1のアドレスを返す操作である。走査がアドレス上昇方向の場合をのべると。起点アドレスをA_0とすると、まず第$K-1$階層のメモリをアクセスする。なお、第$K-1$階層メモリは無番地である。この階層を$f_{K-1} = \frac{A_0}{J^{K-1}}$ビット目からアドレスの上昇方向に走査し、最初の1であるビット位置をj_{K-1}とする。もし、1であるビットが発見できなかった場合には、走査は失敗する。

次に第$K-2$階層を$A_1 = j_{K-1}J^{K-2}$をアドレスにしてよみだし、起点ビット位置

$$f_{K-2} = \begin{cases} 0 & \text{if } f_{K-1} \neq j_{K-1} \\ \frac{A_0}{J^{K-2}} \bmod J & \text{if } f_{K-1} \neq j_{K-1} \end{cases}$$

から最初の1を走査する。同様に繰り返して、A_{K-1}まで求める。これが、ビット走査の操作結果である。

2.3 単純な階層ビットテーブルの実現

0方向非対称階層ビットテーブルの基本ハードウェア構成を図3に示ように、階層数Kと等しい個数の、読み出し幅Jビットメモリと、各々の階層メモリに附属する階層内論理回路、および各階層間を接続する階層間論理回路で構成される。勿論、各階層のメモリにはパリティ、ハミングコード等のエラー検出／処理機能を付加してもさしつかえないが、以下の考察では考慮しない。

階層内論理回路は、読み出した各階層のメモリから、階層間論理回路で用いる値を生成すると共に、階層間論理回路からの値を基に、階層メモリへの書き込みデータを作成する機能をはたす。図4は階層内論理回路のブロック図である。選択器はビットテスト等に用い、0試験回路は、指定された領域以外が全て0であることを検出し、ビットリセット等に用いる。ビット探索器は、与えられた位置から最も近い1を検出するプライオリティエンコーダであり、ビット操作の実現にもちいる。また、書き込みビット生成器は、階層間論理回路の出力により、アドレスで指し示される位置以外に0を補うことにより、階層メモリへの書き込みデータを生成する。階層内論理回路に要するハードウェア量は、メモリアクセス幅Jに対して$O(logJ)$のオーダーである。Jが8の場合を例とすると、約100ゲート（2入力換算）が必要である。

階層間論理回路は、各階層から得られる読み出したビットの値、指定された領域以外が0であること、ビット探索

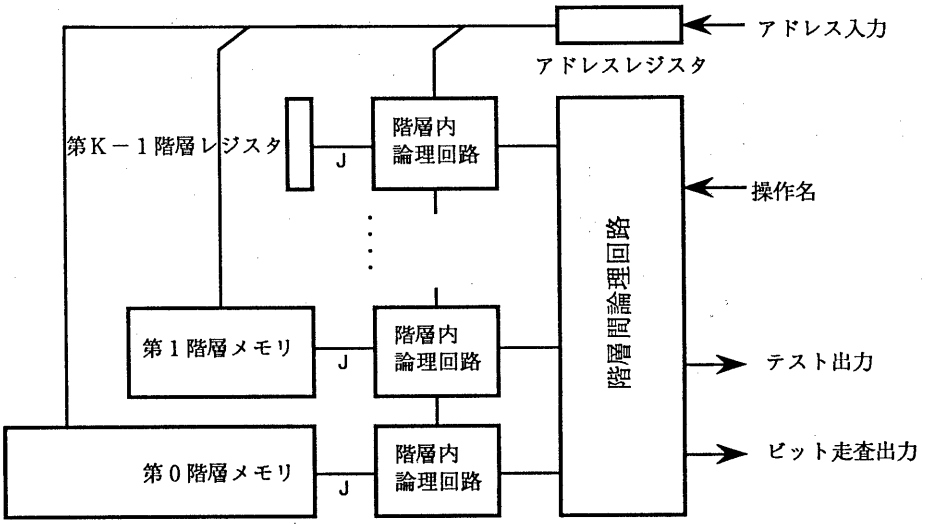

図 3: 階層ビットテーブルのハードウェア構成

器の解を基にビットテーブルとしてのテスト値を求め、各階層への書き込みデータ生成の指示を前期アルゴリズムに従って生成する。また、ビット走査時におけるシーケンス制御の実現も担当する。階層間論理回路のハードウェア量は階層数 K に対して $O(logK)$ のオーダーであり、$K = \log_J N$ であることから、表の大きさ N に対して $O(log^2 N)$ のオーダーである。$K = 8$ （$J = 8$ の場合、１６Ｍビットの表である）の場合、約８０ビットで構成される。８個の階層内論理回路と合計して、１６Ｍビットの表を実現するために約１Ｋゲートの論理回路が必要である。

階層ビットテーブルに必要なメモリ量 M は、ビットテーブルの大きさを N として：

$M = \sum_{i=0}^{\log_J N} \frac{N}{J^i} \simeq N\frac{J+1}{J}$ である。例えば J が８の場合には１４％のメモリ量増加にあたる。

3 一般的な階層ビットテーブルとその操作

これまでに述べてきた階層ビットテーブルは、ビット値に対し操作が非対称であるとともに、操作の対象となるビット列のサイズ、先頭アドレスに対して強い制限が課せられていた。本節ではこれらの制限を緩和する。

０方向非対称階層ビットテーブルでは、１以上の大きさを持ったビット列に対するビットクリア、ビット列テスト、ビット走査の諸操作の対象は、０が連続するビット列に限定されていた。これらの操作を１が連続するビット列に対して拡張し操作の対称性を得るためには、第０階層のメモリを共有する２組の第１から第 $K-1$ 階層までのメモリを用いることが必要である。２組の上部階層に対し、０方向を担当する側は、上記と同じ論理方式を用い、１方向を担当する側は、上記と共役な論理方式を用いる。

ビットテスト操作では、第０階層の値を b_0 、０方向の各階層の値を、$b0_k, k = 1...K-1$ 、１方向の各階層の値を $b1_k, k = 1...K-1$ として、

$B = b_0 \cdot \prod_{i=1}^{K-1} b0_i \cdot \sum_{i=1}^{K-1} b1_i$ と表される。同様にしてビットセット、ビットクリア、ビット列テスト、ビット操作が実現される。

ｐ階層ビットテーブルの０／１方向への対称化を行なっても、全ての演算について操作時間は増大しない。ただし、単純なビットテーブルと比較して余分に必要なメモリ量と、階層化に必要な論理回路の量は２倍となる。先に述べた J が８で８階層の場合、必要なメモリ量の増分が２８％程度であり、論理回路量が約２Ｋゲートである。

ビット列の先頭アドレスとサイズの関係に存する制限をとりはらうことは、目的ビット列を先の制限を満たした複数個の部分列に分解することにより実現される。任意の

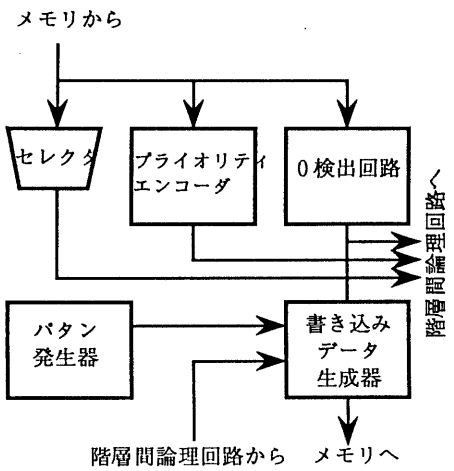

図 4: 階層内論理回路

先頭アドレスで開始する任意のサイズ T のビット列は最大 $2 \cdot \lfloor \log_J T \rfloor + 1$ 個の部分列に分解される。ビットクリア、ビット列テストの操作は、これら部分列に対する操作として実現される。従って、階層メモリ構成、論理回路は前節で扱った制限のある場合と同様で、更にアドレスを分解する回路とシーケンサがつけ加わる。

4 階層ビットテーブルの速度

階層ビットテーブルの速度に関しては、2種類の興味を持つことができる。第一は、ビット列の長さ N 対する速度の振るまいであり、第二は、単純な非階層型ビットテーブルとのハードウェアとしての動作速度比である。

階層ビットテーブルの基本メモリアクセスの単位は1つのメモリ読み出しと、それに引き続く1つのメモリ書き込みである。勿論1ビットに対するメモリテストは1回のメモリ読み出しだけで構成されるが、この場合にはメモリに対する書き込みを無駄にしたと考える。この時間を単位として計った各操作の速度を表1に示す。表1において、ビット列に対する制限つきとは、ビット列のサイズが2の巾乗であり、先頭アドレスがサイズの倍数であることを意味し、制限なしとは、任意のサイズ、任意の先頭アドレスを持つことを意味する。

計算機アーキテクチャの構成要素として、階層ビットテーブルを考える場合には、操作に要する時間のオーダーだけでなく操作に要する時間そのものが非常に重要である。この場合基準となる物は階層化されていない単純ビットテーブルの基本操作に要する時間である。単純ビットテーブルは、読み出し幅1ビット（エラー処理を考慮すると同一内容の2ないし3ビット）のメモリを用いること、または読み出し幅が1ビット以上で、書き込みは読み出しデータに対して1ビットの変更を加えたものを書き込むことにより実現される。前者はアクセス時間、特に書き込みアクセス時間が最も短いため、高速性を要求される応用分野では非常に魅力的であるが、エラー処理を含めて考慮するとメモリの使用量が膨大となり、大規模ビットテーブルを作成するためには現実的な方式でない。後者は、エラー処理に必要な付加部分が同時に呼び出す J ビットに対し一組（または1ビット）しか必要でないので、現実的な方法である。以上の考察から、本節ではメモリからの読み出し幅が1ビット以上である場合の単純ビットテーブルと操作時間を比較・検討する。

ビットテスト 単純ビットテーブルでは、メモリからの出力が語内選択を行うセレクタを経由して操作の結果となる。一方、階層ビットテーブルでは、各階層のメモリからの出力が、同様セレクタを経由した後、全階層を入力とする論理和、又は論理積ゲートを経由して操作の結果となる。メモリのアクセス時間とゲートの遅延時間を考慮すると、この時間差は僅少であると言って差し支えない。

ビットセット、ビットリセット 単純ビットテーブルでは、メモリからの出力に対し、1ビットの変更を行って書き込むことによりビットセットは実現される。階層ビットテーブルでは、同様に1ビットの変更を加える場合と、書き込むデータ全てを変更する場合があり、このいずれであるかの選択は上記ビットテストと同じ回路の出力を用いる。従って、単純ビットテーブルに対するビットセット時間と比べて、上記セレクタ時間と、階層数入力の論理積（和）時間、書き込みデータのセレクタ回路分遅延時間が増大する。ただし、この遅延時間の増大した分は、パイプライン化して吸収すること、または、メモリへの書き込みデータはメモリサイクルの途中で確定すれば良いことを利用すれば外に見えないようにすることが可能である。

ビット列テスト、ビット走査 ビット列テスト、ビット走査は単純ビットテーブルで単位操作として実現不可能であるので、比較の対象とならない。

従って、適切な設計の下には、単純ビットテーブル

操作	単純ビットテーブル	階層ビットテーブル ビット列制限付き	階層ビットテーブル ビット列制限なし
ビットテスト	1	1	1
ビットセット	1	1	1
ビット列クリア	N	1	$2\log_J N+1$
ビット列テスト	N	1	$2\log_J N+1$
ビット列走査	N	$\log_J N$	$\log_J N$

表 1: 階層ビットテーブルの動作速度

とほぼ同じタイミング条件を用いて階層ビットテーブルを用いることが可能である。

5 階層ビットテーブルの計算機における応用

階層ビットテーブルはデータ駆動計算機を実現する際に考案された経緯から、現在までの所データ駆動計算機およびハイブリッド計算機にしか実際には使用されていない。しかしながら、階層ビットテーブル操作の持つ一般性からより広い範囲の有効な応用分野が予想される。本節では、まず、データ駆動計算機における使用形態について述べ、次にその他の応用分野について述べる。

5.1 データ駆動計算機への応用

データ駆動計算機では、変数は主に単一代入的に用いられる。従って、定義された変数が全て使われる場合においても各変数は1回の書き込みに対して1回の初期化が必要である。典型的な例として、1回書き込み、1回読み出しの場合では、初期化は33%のオーバーヘッドとなる。データ駆動計算機ではこの操作が原則的には全ての命令実行にともなうため、メモリ能力を1／3無駄に捨てていることになる。

データ駆動計算機における構造体処理においても同様のオーバーヘッドが存在する。構造体でも要素が単一代入的に用いられるため、非常に多くの構造体の初期化が発生する。更に構造体では、構造体自身が単一代入的に使用されるため、全ての定義されている要素がアクセスされるとは限らず、潜在的により多くのオーバーヘッドが発生する。

特に、スパースな構造体を順次コピーしながら新たな要素を追加して計算を進行させるプログラムでは、一組みのメモリアクセスに対して $\frac{N}{2}$ 要素の初期化が必要であり、初期化速度が全体の計算速度を規定する。このように、データ駆動計算機では、処理装置におけるマッチング操作と、構造体の双方で初期化操作の高速化が重要な課題である。

階層ビットテーブルは、最初にデータ駆動計算機の構造体処理に導入された［1］。要素レベルで読み出しプリミティブと書き込みプリミティブの同期をとる構造体では各語（要素）毎に1ないし2ビットの同期フラグを持つ。同期フラグを階層ビットテーブルで実現することにより、構造体の初期化操作を高速化することが可能になる。メモリ領域に対して頻繁に割り付け／解放操作が行われる場合には、2を基数とするバディシステムは有効なメモリ領域管理方式であり、2を基数とするバディシステムを採用することにより、第2節で述べた0方向非対称階層ビットテーブルを使用することが可能となる。その結果、構造体の初期化操作が（前述の制限の下に）1単位の操作で実現でき、領域の大きさに比例した高速化を達成する。

処理装置に於けるマッチング機能を、ローカルフレームを用いた直接マッピング、命令が相対マッチング位置を指定する間接マッピング方式により実現する場合には、ローカルフレームの初期化、再利用に0方向非対称階層ビットテーブルを用いることが出来る［2］。この場合にはローカルフレームの最大長 L を初期化出来れば十分であるので、階層数を $K = \log_J L$ に制限可能であり、さらにハードウェアが簡略化する。

データ駆動計算機では、初期化の他に処理の終端検出に階層ビットテーブルを用いることが可能である。終端検出としては、プロセス処理の終端検出と構造体処理の終端検出の2種類が存在するが、ここでは構造体処理を例に取る。多数のプロセスが構造体を共有して書き込む場合、または読み出す場合、全ての要素に対する操作が終わっていることを検出することは、構造体を再利用／回収する為に必要な要件である。構造体操作の終端検出は通常カウンタを用いて実現されるが、カウンタが1個の場合は、カウンタが明らかに”ホット”な変数になり、構造体全体の処理を逐字化する。カウンタを多数設ける場合には、カウンタに対するアクセスを分散することは可能となるが、カウンタ処理のオーバーヘッドが増大する。

書き込み操作の終端検出の場合には1方向非対称階層ビットテーブル、読み出し操作の終端検出の場合には0方向非対称階層ビットテーブル、双方を実現する場合には対称階層ビットテーブルを用いる。いずれの場合も、書き込みまたは読み出しに伴うビットセットまたはビットリセット操作と同時に構造体全体に亙るビット列テストを行い、全てのビットが1または0の場合を操作の終端とする。

階層ビットテーブルをカウンタの代替物として用いると、上位階層のセット／リセット操作の頻度は指数関数的に減少するため、下位階層をバンク分けすることにより、構造体処理における並列性を増大できる。

データ駆動計算機に於ける応用例の最後は、構造体の回収に伴う同期未解決要素の回収である［3］。構造体の使用が終了し、回収する場合、書き込み操作により解決していない読み出し操作が残っていると、読み出し要求を発したプロセスが終了できなくなる。従来、未解決の要素は全要素を線形に探索することにより検出した［3］。0方向非対称階層ビットテーブルを同期フラグに用い、ビット走査操作を使用することにより探索に要する時間を短縮することが可能となり、高速化が図れる。

このように、階層ビットテーブルを用いたビット列クリア、ビット列テスト、ビット走査を用いることによりデータ駆動計算機ならびに細粒度の並列処理装置で問題となるオーバーヘッドを短縮することが可能となる。

5.2 データ駆動計算機以外での応用

データ駆動計算機ならびに細粒度の並列処理以外の場面における階層ビットテーブルの応用として、スパースなデータ構造の実現、メモリ領域の設定と検出が可能である。

スパース行列やハッシュ表など構造体の一部要素しかデータの存在しないデータ構造に対して、全要素操作を行うためには、リスト構造を導入するか、線形探索を行う必要がある。リスト構造を用いる場合でも、要素の順序関係を保存するには、二重リンクリストを用いる必要がありメモリ領域、操作時間ともオーバーヘッドが大きい。階層ビットテーブルのビット走査機能を用いることにより、線形時間が必要であった探索が$\log N$時間に短縮可能であり、特に大きい構造体を用いる場合に大きな高速化が可能である。

メモリを領域に分割し、メモリ上の2点が同じ領域に属するかどうかテストすることは、メモリ保護、ベクトルアルゴリズムなど多くの適用範囲を持つ基本操作である。0方向非対称階層ビットテーブルを利用すると、このテストが$\log N$時間、階層メモリの各階層をパイプライン的にアクセスすることにより、単位メモリアクセス時間で実現が可能であり、サイズを常に持ち歩かなくてもメモリ保護、ベクトルの端点検出が実現可能である。

6 おわりに

階層ビットテーブルは、ビット列に対する基本操作を高速化する手法として、最初はデータ駆動計算機を目的として開発された。しかし、その応用範囲はデータ駆動計算機に止まらず一般の細粒度並列処理および逐次処理など広い範囲におよび、今後のアルゴリズム開発が期待される。更に、本論文で述べた階層的メモリ構造はビット列操作の実現だけでなく、ベクトルに対する局所操作と大局的操作が混在する場面で有用であることが予測される。多世代にわたるベクトルのコピー操作の実現はその有力な一例であろう。今後の研究課題である。

謝辞

本研究は、大型プロジェクト「科学技術用高速計算システム」の一環である、科学技術計算用データ駆動計算機ＳＩＧＭＡ－１の研究開発にその源を発する。プロジェクトを推進していただいた方々、特に柏木寛電子技術総合研究所所長、弓場敏嗣情報アーキテクチャ部部長に感謝いたします。また、常日頃ご討論頂く計算機方式研究室のメンバーの皆様に深い感謝の意を表します。

参考文献

[1] Hiraki, K., Nishida, K. and Shimada, T.,"A Hardware Design of the SIGMA-1 - A Data Flow Computer for Scientific Computations," Proc. Int. Conf. Parallel Processing, 1984, pp.851-858.

[2] 平木 敬：プロトタイプハイブリッド・データフロー計算機のアーキテクチャ、ＪＳＰＰ’９０予稿集、１７７－１８５頁、１９９０年。

[3] Sergeant, J. and Kirkham, C. C., "Stored Data Structures on the Manchester Dataflow Machine," Proc. Int. Symp. Computer Architecture, 1986, pp.235-242.