(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3744896号
(P3744896)
(45)発行日　平成18年2月15日(2006.2.15)　(24)登録日　平成17年12月2日(2005.12.2)

(51)Int.Cl.		F I		
G06F 17/14	(2006.01)	G06F	17/14	A
G06F 17/16	(2006.01)	G06F	17/16	K

請求項の数 3　(全 70 頁)

(21)出願番号	特願2002-533113(P2002-533113)
(86)(22)出願日	平成13年10月5日(2001.10.5)
(65)公表番号	特表2004-511046(P2004-511046A)
(43)公表日	平成16年4月8日(2004.4.8)
(86)国際出願番号	PCT/US2001/042537
(87)国際公開番号	WO2002/029611
(87)国際公開日	平成14年4月11日(2002.4.11)
審査請求日	平成15年6月4日(2003.6.4)
(31)優先権主張番号	09/680,665
(32)優先日	平成12年10月6日(2000.10.6)
(33)優先権主張国	米国(US)
早期審査対象出願	

(73)特許権者　591003943
インテル・コーポレーション
アメリカ合衆国　９５０５２　カリフォルニア州・サンタクララ・ミッション　カレッジ　ブーレバード・２２００
(74)代理人　100091915
弁理士　本城　雅則
(74)代理人　100099106
弁理士　本城　吉子
(72)発明者　イエリン，ダニー
イスラエル国　ラナナ　ハーゼル・ストリート７１
(72)発明者　ライニッシュ，ドロン
イスラエル国　ラマト－ガン　キッシュ・ストリート３
最終頁に続く

(54)【発明の名称】線形変換を実行する際の効率を強化する方法

(57)【特許請求の範囲】

【請求項１】

実数、複素数、または有限の場の上で、少なくとも１つのｎ次元入力ベクトルに対して与えられたｒ×ｎ変換行列によって表わされる線形変換を実行する際の効率を強化する方法において、

（１）前記ｒ×ｎ変換行列を修正し、次数を削減して修正された行列を生成する段階であって、以下の（Ａ）～（Ｄ）の段階を含む段階、

（Ａ）前記ｒ×ｎ変換行列の２つの行について、同じ次元の２つの０でなく、それぞれ他方の積である行ベクトルである等価な行のグループを識別し、もし存在する場合には前記等価な行のグループから保持する行を選択し、前記等価な行のグループ中の前記選択された行を除く１またはそれ以上の前記等価な行を省略し、かつ、前記各省略された行と前記選択された行との間の比をメモリに格納する段階、

（Ｂ）前記ｒ×ｎ変換行列のゼロの列を省略する段階、

（Ｃ）正規化した行列を生成するために、列の要素が指標順である場合に各列のリード要素は各列のゼロでない最初の要素であるリード要素の逆数を残存する列にそれぞれ掛けることにより、前記ｒ×ｎ変換行列の残存する列を正規化する段階、および

（Ｄ）前記正規化した行列中の等しい列のグループを識別し、前記等しい列から保持する列を選択し、前記等しい列のグループ中の前記選択された列を除く１つまたはそれ以上の前記等しい列を省略する段階、

（２）チャンネル上の信号に対応するｎ次元入力ベクトルを受信する段階と、

（３）前記省略されたゼロの列に対応する前記ｎ次元入力ベクトルの要素を省略し、前記ｒ×ｎ変換行列の前記残存する列を正規化する際に用いられた前記リード要素を前記入力ベクトルの要素に掛けることにより前記入力ベクトルを正規化し、かつ識別された前記等しい列のグループ別に正規化された前記入力ベクトルの要素を合計することにより、前記修正された行列に従って、前記ｎ次元入力ベクトルから修正されたベクトルを発生させる段階と、

（４）前記修正された行列に従って前記修正されたベクトルの要素の合計を累積するためのハードウェア手段を用いて、前記修正されたベクトルに前記修正された行列を掛け、前記メモリ中に格納された１つまたはそれ以上の前記比を用いて、前記１つまたはそれ以上の省略された行に対応する前記ｒ次元の出力ベクトルの要素を発生させることにより、ｒ次元の出力ベクトルを獲得する段階と、

から構成されることを特徴とする方法。

【請求項２】

前記ｒ×ｎ変換行列を修正する段階は、前記変換行列をいくつかのサブ行列へ分割する段階を、および、前記ｒ次元の出力ベクトルを獲得する段階は、各サブ行列との積に起因する複数の出力ベクトルを統合する段階を、さらに含むことを特徴とする請求項１記載の方法。

【請求項３】

前記修正された行列をいくつかのサブ行列に分割する段階をさらに含み、出力ベクトルは各サブ行列との積に起因する出力ベクトルに各対応するサブ・ベクトルを加えることにより獲得されることを特徴とする請求項１記載の方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、デジタル信号処理の分野に属する。それは、演算動作回数を削減し、単純化された回路および低消費電力を達成する線形変換を実行するための改善された方法および装置を提供する。

【０００２】

【従来の技術】

線形変換は、信号またはデータ処理に一般に使用される。線形変換は、「入力ベクトル」とみなされるｎ次元のベクトル、「出力ベクトル」とみなされるｒ次元のベクトルから生成する。多くの応用では、入力ベクトルは、処理を要求する所定の信号であるデジタル・サンプルから成る。しかしながら、線形変換への応用はどのような特定の領域にも制限されず、それらは科学技術のすべての分野にとって必要であるとともに、それらの効率的な特性が非常に期待されるものである。

【０００３】

１組のｎ次元ベクトル（実数、複素数、またはスカラの任意の領域からなる）から１組のｒ次元ベクトル（同じスカラ領域にわたる）への一般的な線形変換は、次式によって与えられるｒ×ｎ次数の行列（マトリックス）によって表わされる。

【０００４】

【数１】

$$A=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{r1} & & & a_{m} \end{bmatrix}$$

【０００５】
一般的な入力ｎ次元の列ベクトルは、次式の形式をとる。
【０００６】
【数２】

$$x=\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$$

【０００７】
線形変換は、入力ベクトルｘから、次の方程式によって与えられるベクトルの行列積を通して、次元ｒの出力列ベクトルｙを生成する。
【０００８】
【数３】

$$y=\begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_r \end{bmatrix}=\begin{bmatrix} a_{11} & a_{12} & \cdot & \cdot & \cdot & \cdot & a_{1n} \\ a_{21} & a_{22} & \cdot & \cdot & \cdot & \cdot & a_{2n} \\ \cdot & & & & & & \\ \cdot & & & & & & \\ \cdot & & & & & & \\ a_{r1} & & & & & & a_m \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix}=$$

$$=\begin{bmatrix} a_{11}x_1+ & a_{12}x_2+ & \cdot & \cdot & \cdot & \cdot & +a_{1n}x_n \\ a_{21}x_1+ & a_{22}x_2+ & \cdot & \cdot & \cdot & \cdot & +a_{2n}x_n \\ \cdot & \cdot & & & & & \cdot \\ \cdot & \cdot & & & & & \cdot \\ \cdot & \cdot & & & & & \cdot \\ a_{r1}x_1 & a_{r2}x_2+ & \cdot & \cdot & \cdot & \cdot & +a_m x_n \end{bmatrix}$$

【０００９】
線形変換の直接演算は、ｒ・ｎ回の乗算およびｒ・（ｎ-１）回の加算を要求する。
【００１０】
２進法の行列は、本発明の開発および応用に特に重要である。２進２極行列は、各エントリに±１の値だけを含むものである。今後、それらはＵ行列と称される。Ｕ行列によって表わされる線形変換はＵ変換と称される。上記の表現において、与えられた変換がＵ変換である場合、そのときｒ・（ｎ-１）回の加算／減算がその直接の演算に要求されるであろう。別のタイプの２進・行列は、０と１の値だけを含んでいるものである。それらは０-１行列と称される。０-１行列によって表わされる線形変換は０-１変換と称される。上記の表現では、平均して、ｒ・（ｎ-１）／２回の加算が直接の演算に要求されるであろう。
【００１１】
本テキストにおいて、用語「２進変換」は、上述した２つのタイプの変換、すなわち、Ｕ変換および０-１変換を含む。用語Ｕベクトルとはそのコン

ポーネントに、および同様に±１値を有するベクトルをいい、０と１の値を有するベクトルは０-１ベクトルと称される。

【００１２】

２進変換処理は、入力ベクトルの加算および減算コンポーネントから成る。それらは、加算器と減算器のようなコンポーネントを必要として、電気的な特定用途向け集積回路（ＡＳＩＣ）としてのハードウェアの中で実行される。これらのコンポーネントの使用は高価であるし、また、エネルギを消費し、さらにそれらの構造は貴重なエリアを占有することになる。これらの資源の消費量を低減した２進変換を行なうハードウェアを実現することが、多くの技術分野においてその必要性を増加させている。

【００１３】

デジタル信号処理の様々な側面から、Ｕ変換への広範囲の利用法がある。それは、イメージ処理や無線通信のような多種多様の通信技術を含む。

【００１４】

現在、直接シーケンス（ＤＳ）符号分割多重接続（ＣＤＭＡ）スペクトラム拡散通信システムに世界的な関心が集まっている。ＩＳ－９５［ＴＩＡ／ＥＩＡ／ＩＳ－９５－Ａ、「二重モードの広帯域スペクトラム拡散セルラ・システム用移動局基地局の互換性基準」、１９９６年２月２７日］が発展するＤＳ－ＣＤＭＡシステムの１つの例である。

【００１５】

ＣＤＭＡ通信技術では、複数ユーザのデータが複合信号で送られ、その後送信前に擬似ランダム雑音（ＰＮ）コードが乗じられるが、それはランダム雑音特性（低い相互相関のような）を有するＵシーケンスである。拡散特性は、マルチパス・ノイズ、ジャミングまたは検出を含むノイズに対する送信抵抗を形成する。チャネリング・コードより長いスクランブル・コードも適用される。第二世代（ＩＳ－９５－Ｂ）および第三世代（３Ｇ）の広帯域（ＷＢ）ＣＤＭＡ標準規格のこの送信方法は、受信機がマルチコード検出を行なうことを要求する。これは結合したチャンネルを同時に逆拡散を行う作業で、その各々は既に異なるチャネリング・コードに従って拡散されていたものである。それはＵ変換の応用によってなされ、ここで変換行列を含む拡散コードはアダマール行列の行である。これは、有効なＵ変換技術が資源の消費を低減する計算タスクを実現するために望まれる多くの例のうちの１つである。

【００１６】

特定のタイプにおいて、線形変換の特性を改善するいくつかの既知の手法がある。非常に広い範囲から選ばれたわずかの適切な例がここで述べられる。畳み込みを行い、ＤＳＰ中でフィルタとして使用されるトップリッツ（ＴＯＰＬＩＴＳ）変換は、ｎがドメイン領域の次元とすると、Ｏ（ｎ・ｌｏｇ$_2$（ｎ））の加算および乗算処理を要求する古典的な高速フーリエ変換（ＦＦＴ）アルゴリズムを使用することにより効率的に行なうことができる。対照的に、標準的で、直接的な方法は、Ｏ（ｎ^2）の加算および乗算を要求する。詳細は、［James W. Cooley and John W. Tukey, in algorithm for the machine calculation of complex Fourier series, Mathematics of computation, 19(90): 297-301, April 1965］を参照すること。

【００１７】

ＣＤＭＡ技術を含むデジタル信号処理で使用される特定タイプのＵ変換は、アダマール行列によって表わされるウォルシュ－アダマール変換である。アダマール行列は、以下のように再帰的に定義することができる。

【００１８】

【数４】

$$H_1=[1],\quad H_2=\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix},$$

【００１９】

そして、あらゆる整数ｎ、累乗２に対し：

【００２０】

【数５】

$$H_{2n}=\begin{bmatrix} H_n & H_n \\ H_n & -H_n \end{bmatrix}$$

【００２１】

複雑性を低減しかつエネルギを長持ちさせるこの変換を行なうアルゴリズムは、高速ウォルシュ・アダマール変換（ＦＷＴ）であり、それは、ｎ・ｌｏｇ$_2$（ｎ）に対するｎ×ｎアダマール行列のための加／減算の数を削減する。これは、これらの動作のための最適な節約を提供する。この方法を強調する基本概念は、ＦＦＴ手法に類似する。

【００２２】

Ｕ変換の形式は本発明による別の特徴によって修正され、トップリッツＵ変換の特性に対し効率的な方法および装置を提供する。これらの変換は与えられたＵシーケンスまたは複雑なＵシーケンスを備えた完全または部分畳み込みを表わす。トップリッツＵ変換のワイヤレス通信への応用には、実数および複素擬似ランダム（ＰＮ）あるいは黄金Ｕ変換を備える畳み込みを利用する初期時間同期および探索器を含む。黄金シーケンスは、２つのＰＮシーケンスのＺ$_2$合計によって構成される。

【００２３】

本発明の上記２進の解釈は、発明の別の局面によって、比較的少数の異なるエントリを備えたｒ×ｎ行列によって表わされる線形変換の特性のための効率的な方法および装置へと一般化される。本発明のこの進化した局面を適用することにより、複素Ｕ変換および線形変換は、｛０，１，－１｝－エントリを備える行列によって表わされる。本発明のこの広範囲な好適な実施例は、一般化された消去法（ＧＥＭ：generalized-elimination method）と呼ばれる。

【００２４】

広帯域ＣＤＭＡおよび進化したＤＳＰの将来技術における恐らく他の派生技術は、異なる拡散係数において到来信号の同時分析から利益を得るであろう。これは、マルチコード・チャンネルに道を与え、それによってファックス、通常電話の会話およびネットワークに過度の負担をかけないインターネット・データのような異なるタイプの情報を同時に受信することが可能となるであろう。本発明の別の局面として、この種の線形動作に対する効率的な方法および装置がある。それは、追加のプロセッサを備える発明におけるＵ－２進の修正版を含む。この追加のプロセッサは、Ｕ－２進によって要求される加算の数に関する情報を用い、多様なサブセクション中のＵ－２進をグローバルな追加の低速度で適用する構成（configuration）を発見する。　本発明の別の主用途は、ワイヤレス・マルチメディア・システム、ＧＰＳ衛星を基礎にする位置システム、その他を含む。モバイル通信や他のモバイル、および計算を基礎とするシステムの中味において、線形処理に用いられる現時点の消費を削減することが本質的である。本発明をモバイル電話技術に応用することは、バッテリの寿命を延ばし、回路を削減することができ、また応答時間を短くすることができる。　本発明のこれらや他の特徴および利点は、次の詳細な説明を参考にすることにより、また添付の図面を参照することによりさらに明らかになるであろう。

【００２５】

【実施例】

本発明の方法を解説するにあたり、基本となる２つの新しい事項がある。最初は、等価ベクトル(equivalent vector)である。同じ次元の２つの０でないベクトルは、それらがそれぞれ他方の積である場合、本テキストにおいて等価と呼ばれる。この項は、与えられた行列の２つの行または２つの列に適用される。第２の用語は、与えられた行列中の列のリード要素である。本発明の全体に亘って使用される定義は、前述の行列の各列で最も高い

指標の０でないコンポーネントである。この定義は、与えられた行列の中で一律に使用される所定の列指標次数（column index-order）に依存する。新しい次数の再定義は、新しいリード要素の再定義で生じる。例えば、与えられた行列のリード要素は、どちらの次数を選択することがその目的のために適切であるらしいかに依存して、各列の最下位の０でない要素または最高位の０でない要素である。

０－１行列

本発明の好適な実施例は、０－１の２進の線形変換を行なう効率的な方法を提供する。以後、それはさらに「０－１方法」と称される。次の例は、０－１方法への導入およびその主要な概念の概略である。

【００２６】

例：与えられた０－１の２進　４×１４行列Ａ：

【００２７】

【数６】

$$A=\begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

【００２８】

および１４次元の入力ベクトル、

【００２９】

【数７】

$$x=\begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_{14} \end{bmatrix}$$

【００３０】

４次元の「出力」ベクトルを計算すると仮定して、

【００３１】

【数８】

$$y=\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}=\begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_{14} \end{bmatrix}=A\bullet x$$

【００３２】

本発明の好適な実施例によれば、行列Ａは、等価線(equal line)があるか最初にチェックされる。等しい２行のラインがないので、その手順は次のステップに進む。次に、出力ベクトルｙは、それぞれの入力ベクトル座標である係数を備えた列の合計として表現される。従って、

【００３３】
【数９】

$$y = x_1\begin{bmatrix}0\\1\\0\\1\end{bmatrix} + x_2\begin{bmatrix}1\\0\\0\\1\end{bmatrix} + x_3\begin{bmatrix}1\\0\\1\\1\end{bmatrix} + x_4\begin{bmatrix}0\\1\\1\\1\end{bmatrix} + x_5\begin{bmatrix}1\\0\\1\\0\end{bmatrix} + x_6\begin{bmatrix}1\\1\\1\\1\end{bmatrix} + x_7\begin{bmatrix}0\\1\\0\\1\end{bmatrix} + x_8\begin{bmatrix}0\\0\\0\\0\end{bmatrix} +$$

$$x_9\begin{bmatrix}1\\0\\0\\1\end{bmatrix} + x_{10}\begin{bmatrix}0\\0\\0\\0\end{bmatrix} + x_{11}\begin{bmatrix}0\\1\\1\\1\end{bmatrix} + x_{12}\begin{bmatrix}0\\1\\0\\1\end{bmatrix} + x_{13}\begin{bmatrix}1\\1\\1\\1\end{bmatrix} + x_{14}\begin{bmatrix}1\\0\\1\\1\end{bmatrix}$$

【００３４】
このステージで、０の列は、それらの係数と共に、削除されるであろう。入力ベクトルに関して、本発明の好適な実施例によって行われる第１のステップは、任意に出現する０でない列の係数を集めて合計することである。したがって、６つの項で、次の減少されたベクトル方程式が実現される。
【００３５】
【数１０】

$$y = (x_1 + x_7 + x_{12})\begin{bmatrix}0\\1\\0\\1\end{bmatrix} + (x_2 + x_9)\begin{bmatrix}1\\0\\0\\1\end{bmatrix} + (x_3 + x_{14})\begin{bmatrix}1\\0\\1\\1\end{bmatrix} + (x_4 + x_{11})\begin{bmatrix}0\\1\\1\\1\end{bmatrix} + x_5\begin{bmatrix}1\\0\\1\\0\end{bmatrix} + (x_6 + x_{13})\begin{bmatrix}1\\1\\1\\1\end{bmatrix}$$

【００３６】
この表現は次の定義によって単純化してもよい。
【００３７】
$W_1 = X_1 + X_7 + X_{12}$, $W_2 = X_2 + X_9$, $W_3 = X_3 + X_{14}$, $W_4 = X_4 + X_{11}$, $W_5 = X_5$, $W_6 = X_6 + X_{13}$
これは次式を意味する。
【００３８】
【数１１】

$$y = \begin{bmatrix}y_1\\y_2\\y_3\\y_4\end{bmatrix} = w_1\begin{bmatrix}0\\1\\0\\1\end{bmatrix} + w_2\begin{bmatrix}1\\0\\0\\1\end{bmatrix} + w_3\begin{bmatrix}1\\0\\1\\1\end{bmatrix} + w_4\begin{bmatrix}0\\1\\1\\1\end{bmatrix} + w_5\begin{bmatrix}1\\0\\1\\0\end{bmatrix} + w_6\begin{bmatrix}1\\1\\1\\1\end{bmatrix}$$

【００３９】
これはオリジナルの問題に等しいが、削減された列数を備える。しかしながら、現時点では、この種の削減は使い尽くされている。さらに利点を増すために、そのベクトル方程式は、２つのベクトル方程式における次の等価なセットに分割されるであろう。
【００４０】
【数１２】

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = w_1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} + w_2 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + w_3 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + w_4 \begin{bmatrix} 0 \\ 1 \end{bmatrix} + w_5 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + w_6 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} y_3 \\ y_4 \end{bmatrix} = w_1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} + w_2 \begin{bmatrix} 0 \\ 1 \end{bmatrix} + w_3 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + w_4 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + w_5 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + w_6 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

【００４１】
この好適な実施例の第１歩として、これら２つの方程式の各々は別々に扱われるであろう。したがって、同じゼロでない列の係数は集められ合計される。その結果、このステージで、方法は６つの別々の加算項を備えた２つのベクトル方程式のセットとなる。
【００４２】
【数１３】

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = (w_1 + w_4) \begin{bmatrix} 0 \\ 1 \end{bmatrix} + (w_2 + w_3 + w_5) \begin{bmatrix} 1 \\ 0 \end{bmatrix} + w_6 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} y_3 \\ y_4 \end{bmatrix} = (w_1 + w_2) \begin{bmatrix} 0 \\ 1 \end{bmatrix} + (w_3 + w_4 + w_6) \begin{bmatrix} 1 \\ 1 \end{bmatrix} + w_5 \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

【００４３】
さて、出力ベクトルｙの演算を完了するために、４つの追加の加算のみが要求されることが明らかである。従って、所望の結果は、１６回の加算の合計で実行された。この計算プロセスを行なう従来の先行技術の方法が２８回の加算を要求するであろうことは容易に検証されるであろう。本発明のこの好適な実施例に対する先行する記述は、行列の次元が大きくなるに従い、本方法による効率性の向上がさらに明らかになるであろう。
【００４４】
一般に、本発明の０－１の２進の特徴となる点は、ｎ次元ベクトルｘによる次元ｒ×ｎの２進の０－１行列Ａの積に対する効率的な方法および装置にある。行列は次式によって与えられる。
【００４５】
【数１４】

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{r1} & & & a_{m} \end{bmatrix} \quad \text{where } a_{ij} = 0,1 \quad \text{for all} \quad 1 \le i \le r \ , \ 1 \le j \le n.$$

【００４６】
入力ベクトルは次式によって与えられる。
【００４７】
【数１５】

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$$

【００４８】
入力ベクトルのエントリは、実数または複素数、またはスカラ（例のためのZ_2のような）の任意の分野に属するものである。ゴールは、ベクトルｘと行列Ａの積の結果を求めることである。この結果は次式のように表現されるであろう。
【００４９】
【数１６】

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_r \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdot & \cdot & \cdot & \cdot & a_{1n} \\ a_{21} & a_{22} & \cdot & \cdot & \cdot & \cdot & a_{2n} \\ \cdot & & & & & & \\ \cdot & & & & & & \\ \cdot & & & & & & \\ a_{r1} & & & & & & a_m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$$

【００５０】
記述される手続きは再帰的であり、また、ステップの次のシーケンスまたはそれらの一部は各反復で繰り返されるであろう。本発明の好適な実施例によれば、最初のステップは、等しい行のチェックである。可能な場合、入力ベクトルのあらゆる処理も予備的な準備として始まる前に、このステップが行われるべきである。ｉ－ラインがｊ－ラインと等しいと判明する場合、そのときy_iはy_jと等しいと演繹することができる。したがって、２本の等しい線のうちの１本は省略されるであろう。明瞭にするために、より大きな指標を有するラインは、常に省略される１つであろう。したがって、もしｊ＞ｉならば、ｊ－ラインは省略されるであろう。このステージ（それがy_iと等しいので）で知られているy_jがその適切な場所へベクトルｙに挿入される場合、この初期動作は、発明の実行の最終ステップで修正される。行列Ａの中に等しい２行のラインがなくなるまで、等しい線の省略は継続する。実際上、このステージは多くの場合スキップされる。等しい線の存在の可能性が合理的である場合は常に、それは実行される。$\log_2$（ｒ）＞ｎの場合この条件が等しい線の存在を保証するので、それは常に実行される。
【００５１】
煩わしい表記法を回避するために、このスクリーニング過程に起因する修正された行列は、オリジナルの行列として同じ名前Ａを有し、次元ｒ×ｎの名前も保存されるであろう。次のステージで、合計プロセスは等しい列を除去するために導入される。この手続は、最小のコストの追加で、修正された行列中の列の数を削減するであろう。最初に、ベクトル積ｙ＝Ａ・ｘによる行列は、それぞれのｘ－コンポーネントを乗じたＡ列の合計に分解される。
従って、
【００５２】
【数１７】

$$y=A\bullet x=x_1\begin{bmatrix}a_{11}\\a_{21}\\\cdot\\\cdot\\\cdot\\a_{r1}\end{bmatrix}+x_2\begin{bmatrix}a_{12}\\a_{22}\\\cdot\\\cdot\\\cdot\\a_{r2}\end{bmatrix}+\cdots\cdots\cdots\cdots+x_n\begin{bmatrix}a_{1n}\\a_{2n}\\\cdot\\\cdot\\\cdot\\a_{m}\end{bmatrix}$$

【００５３】

この合計の各要素は、ベクトルｘの対応するスカラ係数を乗じて、行列Ａの行ベクトルから成る。この合計は、ｙ＝Ａ・ｘ＝$x_1v_1+x_2v_2+\cdots\cdots+x_nv_n$として、より簡潔に記述される。

ここで（あらゆるｊに対して）v_jは、Ａのｊ列である。

【００５４】

【数１８】

$$v_j=\begin{bmatrix}a_{1j}\\a_{2j}\\\cdot\\\cdot\\\cdot\\a_{rj}\end{bmatrix}$$

【００５５】

本発明の好適な実施例に従って、ゼロ列は、それらの係数と共にこのステージで省略される。次に、同一の非ゼロ列は一つにまとめられ、その対応するスカラ係数の合計を掛けて、別個の列がそれぞれ共通因子になるように、再整理される。従って、結果としての合計は、繰り返し列を省略することによって非ゼロ列の元のシーケンスから抽出された別個の非ゼロ列$w_1, \ldots\ldots\ldots, w_m$の全てのサブシーケンスからなる。各列$w_j$は、この列と等しいすべての列の対応するオリジナルの係数の合計である係数t_jによって乗じられる。ｍ≦ｎ、および、ギャップ、ｎ-ｍがオリジナルのシーケンス$v_1, v_2, \ldots\ldots\ldots, v_n$中の反復回数であることは明らかである。次の数学的な記述は、上述したプロセスは次の数学的表現によって公式化することができる。

【００５６】

$$y=A\cdot x=\Sigma_{1\leqq j\leqq n}\ x_jv_j$$
$$=\Sigma_{1\leqq k\leqq m}\ \Sigma_{j\ such\ that:vj=wk}\ x_jv_j$$
$$=\Sigma_{1\leqq k\leqq m}\ (\Sigma_{j\ such\ that:vj=wk}\ x_j)\ w_k$$

さて、全ての１≦ｋ≦ｍに対して、定義する：

$$t_k=\Sigma_{j\ such\ that:vj=wk}\ x_j$$

計算上のタスクはここまで、オリジナルの係数$x_1, \ldots\ldots\ldots, x_n$の合計として新しい係数$t_1, \ldots\ldots\ldots, t_m$を算出することである。その代償はｎ－ｍの加算である。積ｙ＝Ａ・ｘは、次式から求められる。

【００５７】

$$y=A\cdot x=\Sigma_{1\leqq k\leqq m}\ t_kw_k$$

この演算の管理状況、求められた合計における参加体(participants)を決定することは、入力ベクトルの到着の前に予備的準備として行列毎に一度なされてもよい。要約すると、Ｂをその列がそれぞれ$w_1, \ldots\ldots\ldots, w_m$であるｒ×ｍ行列にしておき、そして次のようにする。

【００５８】
【数１９】

$$t=\begin{bmatrix} t_1 \\ t_2 \\ \cdot \\ \cdot \\ \cdot \\ t_m \end{bmatrix}$$

【００５９】
その後ｙ＝Ａ・ｘ＝Ｂ・ｔとする。このように、上記プロセスは、ｒ×ｎ行列Ａにｎ－ベクトルｘを乗じるオリジナルの問題を、ｒ×ｍ行列Ｂにｍ－ベクトルｔを乗じる問題へ限定した。これによって可能にされる利益は、ｍがより小さいとき、より大きい。
【００６０】
次のステップは、前のステージの行列Ｂを２つ以上のサブ行列への水平分割である、ここでラインは壊されないままにされる。サブ行列はそれぞれ、削減された列の数を有し、それは、次の反復中で上記の列収集手続の利益を増大させるであろう。行列Ｂのラインは、u_1，u_2，.....，u_rによって表示される。考慮中の積は次のように表現される。
【００６１】
【数２０】

$$y=B\bullet t=\begin{bmatrix} u_1 \cdot t \\ u_2 \cdot t \\ \cdot \\ \cdot \\ \cdot \\ u_r \cdot t \end{bmatrix}$$

$u_1 \cdot t$

【００６２】
ここで各ラインは、スカラ積の中でベクトルｔが乗じられる。この表現から、ｙ＝Ｂ・ｔを計算するタスクは、ｒスカラ積、すなわち u_1・ｔ，u_2・ｔ，.....，u_r・ｔを計算する結合タスクと等価である。
【００６３】
本発明の好適な実施例によれば、これはベクトル積、すなわちB_1・ｔおよびB_2・ｔによって行列を計算することにより行われるである、ここで行列B_1およびB_2の各々は行列Ｂのラインの部分集合を含んでおり、これらの２つの部分集合が相互に解体し、それらの結合はＢ中のすべてのラインの部分集合を含む。通常、第１の反復を除いて、２つのサブ行列は同数あるいはほとんど同数のラインを有する。しかしながら、いくつかの特別な場合における行列Ａの特性に依存して、その分割は、２を越える行列になるであろう。そのようなことは、行列Ａが特別な内部順序（オーダー）を有しておらず、またそのエントリが任意に選ばれるように考慮される場合のケースである。その後、本発明の好適な実施例は、通常次の最初のラインの分割を暗示し、全ての分割したサブ行列を$\log_2$（列の数）＞行の数の状態へ持って来るであろう。その後、次の反復は、ラインを２つのほとんど等しい半分へ分割するであろう。行列が勝手気ままであると考えられる場合は、本質的に最悪のシナリオである。
【００６４】

代わりにあるいはその前に挿入されてもよい別のステップ、すなわち上記言及された水平分割が垂直分割である場合である。本発明の好適な実施例によれば、上記言及した合計、

$$y = \Sigma_{1 \leqq k \leqq m} t_k w_k$$

は、２つに分割される。すなわち、

$$y = \Sigma_{1 \leqq k \leqq p} t_k w_k + \Sigma_{p+1 \leqq k \leqq p} t_k w_k$$

であり、ここでｐは１とｍ－１の間で選ばれる。このように、B_1に対して、その行列の１組の列は、w_1，．．．．．．．．．．，w_pであり、またB_2に対して、その行列の１組の列は、w_{p+1}，．．．．．．．．．．，w_mである。それは、

$y = B_1 \cdot t' + B_2 \cdot t''$を保持する。

【００６５】

ここで、対応するベクトルは、ｔ’＝（t_1，．．．．．．．．．．，t_p）、そしてｔ”＝（t_{p+1}，．．．．．．．．．．，t_m）である。

従って、本発明のこの好適な実施例によって、２つのより低い次元の積、$B_1 \cdot t'$そして$B_2 \cdot t''$は別々に計算され、結局、２つのｒ－ベクトルである結果は共に合計される。Ｂ列、w_1，．．．．．．．．．．，w_mの指標の予備的な再配置は、このステップの有効性を増強することができる。

【００６６】

垂直の分割は、他の手続きにおいてもそれほど頻繁に使用されない。その主な使用は、行の数が本質的に列の数を超過する場合にあり、ＤＳＰのアプリケーションにおいて、その状況はそれほど一般的でない。このステップの基本的な欠点は、上記水平分割にパラレル性を有しないｒスカラ加算の加算のセットから成る２つの積$B_1 \cdot t'$および$B_2 \cdot t''$の結果を合計する必要性にある。上記水平分割の場合のように、行列Ａの特性に依存して、垂直分割は２を越える行列になるかもしれない。

【００６７】

最後に、上記ステップの各々は繰り返される反復に適用され、再帰的なメカニズムを形成する。

【００６８】

次に、境界は、上記方法によって可能にされる削減に提示される。ｒ×ｎ　０－１行列Ａに対し、s^*（ｎ，ｒ）によって表示された最悪の場合の加算回数は、$r < \log_2(n)$において、次の表現によって与えられる。

【００６９】

$s^*(n, r) = n + s^*(r)$

次表は、小さな数の行を備える行列に対する加算の数に境界を与える。

【００７０】

$s^*(2) = -1$	$s^*(n, 2) = n - 1$
$s^*(3) = 2$	$s^*(n, 3) = n + 2$
$s^*(4) = 7$	$s^*(n, 4) = n + 13$
$s^*(5) = 22$	$s^*(n, 5) = n + 49$

最悪の場合に次の境界がある。

【００７１】

$s^*(r) < 2^r + 2^{r/2+2} - r$

ｕ（Ａ）が行列Ａの中の数１’ｎｓである場合、標準（先行技術）方法はｕ（Ａ）－ｒの加算を要求する。平均では（Ａが任意のとき、期待値によって）ｕ（Ａ）＝（ｎ－２）・ｒ／２である。ｒが一定（またはｒ≦ｃ・$\log_2$（ｎ）ここで１＞ｃ＞０）であるとき、ｎが無限大になるにつれs^*（ｎ，ｒ）／ｎが１に接近するので、この方法は、漸近的に（有界であるｒおよび非有界のｎに対して）最適である。

【００７２】

Ａがｒ×ｎ　０－１行列で、行列Ａの次元ｒ×ｎに関する仮定はないとき、積Ａ・ｘを計算するのに本発明によって要求される加算の数は、１＞δ＞０の場合、（１＋δ）ｎ・

ｒ／ｌｏｇ$_2$（ｎ）によって制限され、ｒおよびｎの両方が無限大に近づくと、δは０に近づく。ｒ＞ｎの場合により厳しい境界が（この場合に関して）存在し、それは垂直分割のアプリケーションに起因する。

【００７３】

（１＋δ）ｎ・ｒ／ｌｏｇ$_2$（ｒ），ここで １＞δ＞０

そして、ｒおよびｎの双方が無限大に近づくと、δは０に近づく。

【００７４】

このプロセスの有効性を評価するために、本発明は、管理動作を増加させ、かつ加算を削減することが認められるべきである。しかしながら、加算は、複雑性の主要原因となり、また、本発明の実施による削減により支配的な効果を発揮する。さらに、データ・ベクトルの処理が始まる前に、ほとんどの管理動作は行列毎に一回行われる。このように、示唆された方法は、本質的に電力消費と回路を節約する。与えられた行列がある程度まばらであるとき、上記０－１－２進は、特に効率的な特徴を発揮する。このようなことは、この実施例を適用することにより、本発明の実数の行列の特徴で記述されるように、一般的な実数の行列をベクトルで乗算する計算において分散型の演算と結び付くとき、しばしば遭遇する。

【００７５】

さらに、発明のこの特徴は、代数のコードに適用可能で（参照：［Shu Lin, Daniel J. Costello, Jr. "Error control coding Fundamentals and Applications" Prentice-Hall, Inc., Englewood Cliffs, N.J., 1983］）、符号化および復号化のプロセスにおいて、Z_2行列はZ_2ベクトル（あるいはZ_{2n}ベクトル）で乗じられる。これは、Z_2行列がZ_{2n}ベクトルで乗じられるすべての状況で真実である。最後に、上記の０－１－２進の実施例および次のＵ－２進の実施例は、特性２のフィールドを除いてスカラのすべてのフィールドで容易に交換可能である。個々の特定の場合では、より効率的な実施例が選択されるであろう。

Ｕ行列

本発明の好適な実施例（以後、「Ｕ方法」と称する）であるＵ－２進の記述に必要な予備的な概念は、ベクトルと等価な概念である。１つが他方の積ならば、同じ次元である２つの非ゼロのベクトルが等価と称されるであろう。この実施例の内容において、それらが等しい場合またはそれらのうちの１つが他方の積（－１）である場合、２つのＵベクトルが等価であることに注意すべきである。次の好適な実施例は、詳細において上記のものとかなり類似している。次の文章で、消去(elimination)が等しいベクトルに対してよりむしろ等価な（行または列）ベクトルになされることが主要な相違点である。これは消去速度を促進し、したがって効率を増強する。この実施例は、例によって導入され、その概念の本質を実証する。

例：次の５×１３　Ｕ行列Ａと１３次元の入力ベクトルｘとの積を考慮する。行列は以下のとおり与えられる。

【００７６】

【数２１】

$$A=\begin{bmatrix} -1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & 1 \end{bmatrix}$$

【００７７】

入力ベクトルは次のとおり与えられる。

【００７８】

【数２２】

$$x=\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_{13} \end{bmatrix}$$

【００７９】
積の結果は次式のように記述される。
【００８０】
【数２３】

$$y=\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}=A\bullet x$$

【００８１】
そのプロセスにおける第１歩は、等価なラインをチェックすることである。実際上、それは、入力ベクトルの到着の前に行われる一回の動作である。スキャンは、与えられた行列中の等価なラインの１つの発生を発見することで、ライン２はライン４に等価であり、事実、ライン４はライン２に（－１）を乗じたものに等しい。それは結果として$y_4=-y_2$となる。したがって、y_4とy_2の両方を計算する必要はなく、y_4の演算は省略されるであろう。このように、ライン４は行列Ａから削除される。その結果としての行列Ａ’は、次式で与えられる。
【００８２】
【数２４】

$$A'=\begin{bmatrix} -1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 & 1 & 1 & 1 \\ -1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & 1 \end{bmatrix}$$

【００８３】
対応する出力ベクトルは、次式で与えられる。
【００８４】
【数２５】

$$y' = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_5 \end{bmatrix}$$

【００８５】
それは、ｙ’＝Ａ’・ｘであることを保持する。
【００８６】
これは最初の削減である。次のステップにおいて、積Ａ’・ｘは、それぞれのｘ－コンポーネントを乗じたＡ’列の合計に分解される。したがって、次のようになる。
【００８７】
【数２６】

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_5 \end{bmatrix} = x_1 \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix} + x_2 \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} + x_3 \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} + x_4 \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix} + x_5 \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} + x_6 \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} + x_7$$

$$\begin{bmatrix} -1 \\ 1 \\ -1 \\ -1 \end{bmatrix} + x_8 \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} + x_9 \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} + x_{10} \begin{bmatrix} -1 \\ 1 \\ -1 \\ -1 \end{bmatrix} + x_{11} \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} + x_{12} \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix} + x_{13} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

【００８８】
次に、上記合計中の列ベクトルの各々に対し正規化が実行され、各修正済のベクトルの上部コンポーネントが１となる。このように、ベクトルの上部コンポーネントが（－１）である場合、ベクトルおよびその係数は両方とも（－１）が乗じられる。しかしながら、ベクトルの上部コンポーネントが１である場合、修正は行われない。上記合計の正規化形が次のように導き出される。
【００８９】
【数２７】

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_5 \end{bmatrix} = (-x_1) \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} + x_2 \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} + (-x_3) \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + (-x_4) \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} + x_5 \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} + x_6 \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} + (-x_7) \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$$

$$+ x_8 \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} + x_9 \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} + (-x_{10}) \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} + x_{11} \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} + (-x_{12}) \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} + x_{13} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

【００９０】
この正規化された合計では、各ベクトルの上部コンポーネントは１であり、また、異なるベクトル数が削減される。次のステップは、同じ列の係数を集めて合計することである。したがって、８つの加算の実行で、次の方程式が得られる。

【００９１】
【数２８】

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_5 \end{bmatrix} = ((-x_1)+(-x_4)+x_6+x_9)\begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} + (x_2+(-x_7)+x_8+(-x_{10})+x_{11})\begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} + ((-$$

$$x_3)+x_{13})\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} ++x_5\begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} + (-x_{12})\begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$

【００９２】
さて、新しい係数が定義される。
【００９３】
$w_1 = (-x_1) + (-x_4) + x_6 + x_9$,
$w_2 = x_2 + (-x_7) + x_8 + (-x_{10}) + x_{11}$,
$w_3 = -x_3 + x_{13}$, $w_4 = x_5$, $w_5 = -x_{12}$
従って、上記の方程式は次の形式で書くことができる。
【００９４】
【数２９】

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_5 \end{bmatrix} = w_1\begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} + w_2\begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} + w_3\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + w_4\begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} + w_5\begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$

【００９５】
係数、w_1, w_2, w_3, w_4, w_5はこのステージでプロセッサには知られている。次のステージで、このベクトル方程式は、次の２つの方程式のセットに水平に分割されるであろう。
【００９６】
【数３０】

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = w_1\begin{bmatrix} 1 \\ -1 \end{bmatrix} + w_2\begin{bmatrix} 1 \\ -1 \end{bmatrix} + w_3\begin{bmatrix} 1 \\ 1 \end{bmatrix} + w_4\begin{bmatrix} 1 \\ 1 \end{bmatrix} + w_5\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} y_3 \\ y_5 \end{bmatrix} = w_1\begin{bmatrix} -1 \\ 1 \end{bmatrix} + w_2\begin{bmatrix} 1 \\ 1 \end{bmatrix} + w_3\begin{bmatrix} 1 \\ 1 \end{bmatrix} + w_4\begin{bmatrix} -1 \\ 1 \end{bmatrix} + w_5\begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

【００９７】
さて、これらの方程式の各々は、第１の反復の同じ方法で別々に処理される。それが前の方程式の正規状態を継承したので、上部の方程式は正規化を要求せず、正規化は第２の方程式にのみ必要である。従って、生起する結果としてのセットは次のとおりである。
【００９８】
【数３１】

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = w_1 \begin{bmatrix} 1 \\ -1 \end{bmatrix} + w_2 \begin{bmatrix} 1 \\ -1 \end{bmatrix} + w_3 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + w_4 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + w_5 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} y_3 \\ y_5 \end{bmatrix} = (-w_1) \begin{bmatrix} 1 \\ -1 \end{bmatrix} + w_2 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + w_3 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + (-w_4) \begin{bmatrix} 1 \\ -1 \end{bmatrix} + (-w_5) \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

【００９９】
次のステップで、同じベクトルの係数は集められ合計される。したがって、さらに６つの加算で、その結果は次のとおりである。

【０１００】
【数３２】

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = (w_1 + w_2) \begin{bmatrix} 1 \\ -1 \end{bmatrix} + (w_3 + w_4 + w_5) \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} y_3 \\ y_5 \end{bmatrix} = (-w_1 - w_4) \begin{bmatrix} 1 \\ -1 \end{bmatrix} + (w_2 + w_3 + -w_5) \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

【０１０１】
明らかに、ベクトルｙ’はさらに４つの加算で済むことがわかる。オリジナルの出力ベクトルｙを得るために、$y_4 = -y_2$を思い起こすことが単に必要なだけである。このように、１８回の加算の合計が、このプロセスに必要とされた。従来の先行技術方法が出力ベクトルを計算するために６０回の加算の合計を要求することに認識すべきである。

【０１０２】
本発明のＵ－２進の好適な実施例は、ｒ×ｍのＵ行列Ａとｎ次元の入力ベクトルｘとの積を求めるための効率的な方法である。それは通常０－１の好適な実施例より多くの利益を提供し、より多くのアプリケーションを有する。

【０１０３】
行列Ａは次式で与えられる。

【０１０４】
【数３３】

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{r1} & & & a_{rn} \end{bmatrix} \quad \text{where } a_{ij} = \pm 1 \quad \text{for all} \quad 1 \le i \le r \ ,$$

$$1 \le j \le n.$$

【０１０５】
そして、入力ベクトルは次式のように書かれている。

【０１０６】
【数３４】

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$$

【０１０７】
入力ベクトルのエントリ値は実数または複素数、あるいは２より大きな特性を備えるスカラ領域に属するものであってもよい。目標は、次の積の結果を計算することである。
【０１０８】
ｙ＝Ａ・ｘ
最初のステップは、等価ラインの消去によって与えられた行列を修正することで、そのステップは予備的な準備の一部である。ｉラインがｊラインと等価であると分かると、そのときy_1は$\pm y_j$と等しいと推論することができる。したがって、本発明の最初のステップにおける好適な実施例は、２つの等価なラインのうちの１つを省略することを決めることである。明瞭にするために、方針としては、より大きな指標を有するラインを常に省略すべき１つとすることである。このように、それがｊ＞ｉと推定される場合、、そのとき、ｊラインが省略されるであろう。このステージで知られているy_jがベクトルｙのその場所へ戻される場合、この最初の動作はプロセスの最終ステップで逆にされる。この消去プロセスは、修正済の行列に等価である２行のラインがなくなるまで、継続する。
【０１０９】
等価なラインが実際に存在する可能性がある場合は常に、このステージが実行される。これは、多くの場合Ｕ方法による複数の積の（本発明の別の側面）アプリケーションでの場合である。$\log_2(r) \geqq n$の場合、この条件は等価ラインの存在を保証するので、このステージは常に行なわれる。しかしながら、等価ラインの存在を保証し、かつ検討すべきである他の条件がある。例えば、これは、アダマール行列のサブ行列中にある場合である。不体裁な表記法を回避するために、結果としての修正済の行列は、オリジナルの行列と同じ名前Ａを有し、次元ｒ×ｎの名前もまた保存されるであろう。
【０１１０】
次のステップは等価な列の消去である。これは、加算による最小の犠牲で修正済の行列における水平の次元（つまり列の数）を減少させるであろう。０－１の場合のように、このステップと関係する全ての管理は、データ・ベクトルの到着に先立って予備的な準備として通常各行列当たり１回行われる。最初に、積ｙ＝Ａ・ｘは、対応するｘコンポーネントを乗じたＡ列の合計として表現される。
【０１１１】
【数３５】

Let v_j (for every j) be the j column of A: $v_j = \begin{bmatrix} a_{1j} \\ a_{2j} \\ \cdot \\ \cdot \\ \cdot \\ a_{nj} \end{bmatrix}$ Then:

$$A \bullet x = x_1 v_1 + x_2 v_2 + \cdots\cdots + x_n v_n$$

【０１１２】

次に、それがまだ＋１でない場合、行列Ａにおける各列ベクトルの上部の要素が検査され、＋１値に正規化される。従って、a_{1j}＝＋１の場合、そのとき正規化は必要ではないが、しかしa_{1j}＝－１ならば、そのとき正規化が行われ、その対応するｊ列のベクトルは、その対応する係数x_jと同様に、－１が乗じられる。上記の合計の新しい表現が得られるが、ここで各ベクトルの上部のエントリは＋１に等しくなる。この表現では、等価な列は常に等しい。

【０１１３】

その後、上記の０－１の実施例でのように、同一の列は一つにまとめられ、各列をその対応するスカラ係数の合計として乗じられる共通因子により再整理される。これによって、結果としての合計は、反復する正規化された列をふるいにかけることにより、正規化された列ベクトルのシーケンスから抽出され、個別に正規化された全ての列ベクトルw_1，.....，w_m（反復をせずに）のシーケンスからなる。別個の正規化された列w_jそれぞれは、係数t_jが乗じられ、その係数はこの列と等しいすべての正規化された列の対応する正規化された係数の合計である。ｍ≦ｎで、その差ｎ-ｍは、正規化された列のオリジナルのシーケンスにおける反復数である、すなわち、$a_{11}v_1$,$a_{12}v_2$，.....，$a_{1n}v_n$。

【０１１４】

数学上、そのプロセスは次のとおりである。

【０１１５】

$$A \cdot x = \Sigma_{1 \leqq j \leqq n} x_j v_j$$
$$= \Sigma_{1 \leqq k \leqq m} \Sigma_{j \text{ such that } a1j\ vj=wk} (x_j a_{1j}) a_{1j} v_j$$
$$= \Sigma_{1 \leqq k \leqq m} \Sigma_{j \text{ such that } a1j\ vj=wk} a_{1j} x_j) w_k$$

全ての１≦ｋ≦ｍに対して、定義する：

$$t_k = \Sigma_{j \text{ such that:} vj=wk} a_{1j} x_j$$

本発明のこの部分に含まれる主要な計算タスクは、正規化されたオリジナルの係数の合計として新しい係数t_1，.....，t_mを計算することである。加算による負担はｎ-ｍである。積Ａ・ｘは、次式で与えられる。

【０１１６】

$$A \cdot x = \Sigma_{1 \leqq k \leqq m} t_k w_k$$

列消去プロセスによる利点は、オリジナルの行列Ａの構造に依存する。それは、アダマールまたは周期的なＰＮ行列のサブ行列のようなある種の無線通信において、信号ベクトルの符号化および復号化で一般に使用されるいくつかの行列で重要である。さらに、行列Ａ中のライン数であるｒが行ｎの数に比べて小さいとき、それは重要である。特に、ｒ≦ｌｏｇ（ｎ）、のとき、そのような場合、$m-n \geqq n-2^{r-1} > 0$である。これらの考察は、０－１行列においてもかなり正しい。

【０１１７】

本発明のこの好適な実施例の残りの部分、水平および垂直の分割、反復は、その０－１の対応物と等価である。

【０１１８】

本発明の使用によって可能にされる節約を考察すると、その加算が複雑性の主要部分であることがまず分かる。$r \leqq \log_2(n)$でｒ×ｎ　Ｕ行列Ａに対し、ｓ（ｎ，ｒ）によって表わされる最悪の場合における加算数は、次式で与えられる。

【０１１９】

$$s(n, r) = n + s(r)$$

ここで：

ｓ（１）＝－１	ｓ（ｎ，１）＝ｎ-1
ｓ（２）＝０	ｓ（ｎ，２）＝ｎ
ｓ（３）＝３	ｓ（ｎ，３）＝ｎ＋３
ｓ（４）＝８	ｓ（ｎ，４）＝ｎ＋８
ｓ（５）＝１９	ｓ（ｎ，５）＝ｎ＋１９

ｓ（６）＝３８　　ｓ（ｎ，６）＝ｎ＋３８
ｓ（７）＝７５　　ｓ（ｎ，７）＝ｎ＋７５
ｓ（８）＝１４４　　ｓ（ｎ，８）＝ｎ＋１４４
ｓ（９）＝２８３　　ｓ（ｎ，９）＝ｎ＋２８３
次の境界が常に有効である。

【０１２０】

$s(r) < 2^{r-1} + 2^{r/2+1} - r$

先行技術の従来方法は、（ｎ-１）・ｒ回の加算を要求する。ｎが無限大に行き、ｒが一定のとき、ｓ（ｎ，ｒ）／ｎは１に接近するので、この方法は漸近的に最適である。後で記述される本発明のより複雑な多重積による実施例は、加算の最悪の場合における加算数に正確な境界を要求する。これは次の回帰方程式によって可能になる。

【０１２１】

偶数のｒに対して：

$s(r) = 2^{r-1} + 2s(r/2)$

奇数のｒに対して：

$s(r) = 2^{r-1} + s((r+1)/2) + s((r-1)/2)$

Ａがｒ×ｎ　Ｕ行列で、行列Ａの次元ｒ×ｎに対する制限はないとき、本発明によって求められる積Ａ・ｘを計算するのに必要な加算数は、常に次式によって制限される、すなわち（１＋δ）ｎ・ｒ／ｌｏｇ$_2$（ｎ）、ここで１＞δ＞０、および、ｒおよびｎの両方が無限大に近づくと、δは０に近づく。ｒ＞ｎの場合には、よりきつい境界（この場合に関しての）が存在し、それは垂直の分割を適用することに起因する、すなわち、

（１＋δ）ｎ・ｒ／ｌｏｇ$_2$（ｒ）、ここで１＞δ＞０、

および、ｒおよびｎの両方が無限大に近づくと、δが０に近づく。

【０１２２】

しかしながら、行列Ａに特定のタイプの構造がある場合、積Ａ・ｘを計算するために本発明の好適な実施例によって要求される加算の回数は、劇的に減少するであろう。その最も一般的な条件は本テキストの範囲外であるが、しかし、Ａがｎ×ｎアダマールまたは周期的な擬似ランダム（ＰＮ）行列である場合、いくつかの例が言及されるであろう。しかしながら、単にｎ・ｌｏｇ$_2$（ｎ）、加算およびｎスカラのメモリが要求され、それはこの点で最適である。この見解はさらに０－１行列に適用できる。

産業上の応用例

上述された本発明の好適な実施例を実施することから利益を得るいくつかの技術がある。そのような技術の１つは、画像処理であり、それはＵ行列をベクトルで乗じた積を含むいくつかの手順を達成する。無線通信のＣＤＭＡ、ＩＳ－９５、またより発展した第３世代の広帯域ＣＤＭＡにおいて、Ｕ行列をベクトル乗じた積を使用するいくつかのプロセスがある。これらのプロセスは本発明の使用によって、よりエネルギ使用量、回路類およびさらに実行時間の削減をもたらすであろう。

【０１２３】

ＩＳ－９５－Ｂのマルチコード化の内容で、アダマール－６４からの８行のラインは、そのエントリが信号のサンプルであるベクトルによって乗じられるＵ行列を含む。別の応用は、デスプレッダ(despreader)によって成される隣接検出にある。データ・ベクトルによってわずかのアダマール・ラインからできている行列の積がある。別の応用は検索である。それは相互のスカラ積の演算によって高い相関をもつシーケンスを探索する。相関器のシーケンスは、Ｕシーケンスである擬似ランダム（ＰＮ）シーケンスから抽出され、データ・ベクトルによるそれらのスカラ積が要求された演算となる。最初の調査は、検索者の１つの段階である。

一般化された消去法

本発明の上記２進法の特徴は、下記プロセスの反復によって特徴づけられる、すなわち、ゼロ・ラインおよび等価ライン（一つが他方のスカラ積であることを意味する）が省略され、ゼロ列がそれらに関連した入力ベクトルのスカラ・コンポーネントと共に省略され、

等価列は一まとめにされて合計され、また、これらの２つの特徴が使い尽くされた後にその行列は水平にまたは垂直に分割される。本発明の好適な実施例によれば、このステップの反復シーケンスは、スカラのあらゆる分野に行列とベクトルの積に適用できる。本発明のこのより広い好適な実施例は、一般化された消去法（ＧＥＭ：generalized-elimination method）と称される。

【０１２４】

等価列の合計は、基本的に下記プロセスの反復である。v_1，v_2，．．．．．．．．．，v_nを変換行列Ａの列にし、ｘ＝（x_1，x_2，．．．．．．．．．，x_n）を入力ベクトルとすると、ゴールはＡ・ｘを計算することである。列v_{k+1}，．．．．．．．．．，v_n（２≦ｋ＜ｎに対して）の各々が、ｋ－列のスカラ積、すなわちｋ＜ｊ≦ｎに対して$v_j = z_j v_k$であることを、指標の適切な再配置の後に保持すると仮定する。その後、ｎ次元のベクトルｘは減少したｋ次元のベクトルと置き換えられる。すなわち、

ｘ’（x_1，x_2，．．．．．．．．．，x_{k-1}，ｘ’$_k$）ここで、

ｘ’$_k = x_k + x_{k+1} \cdot z_{k+1} +$，．．．．．．．．．，$+ x_n \cdot z_n$

そして、ｒ×ｎ行列は、その列がv_1，v_2，．．．．．．．．．，v_kである減少したｒ×ｋ行列Ａ’と置き換えられる。その後、Ａ’・ｘ’＝Ａ・ｘを保持する。本発明の好適な実施例に従って、必要ならば指標の変更を含むこのプロセスは、２つの列が等価でなくなるまで、繰り返される。確かに、このプロセスは、単に本発明におけるＵ行列の実施例で行われる正規化プロセスの一般化である。実際上、上記の等価な列の削減のすべては、同時に行われてもよい。このステージが終わると、それは全工程の最初の反復と考えられるであろう。その後、行列は、水平に分割され、各分割において上記列の削減が上記実施例中で行われたように、再帰的な方法で繰り返す。

【０１２５】

Ｕ実施例により、また次の例によって示されるように、等価な列を除去する効率的な方法は、最初に最上部のコンポーネントで各列を分割し、続いて対応する係数に同じ最上部のコンポーネントを乗ずることである。それは各修正された列の最上部のコンポーネントに１を有する結果になる。しかしながら、時々、ＧＥＭを適用する際に、変換行列中の列における最上部のコンポーネントが、この分割を不可能にする０となることがある。このような場合、簡単な解決法は、その最上部の０でないコンポーネントで各非ゼロの列を分割し、それにより、対応する係数に同じ最上部の非ゼロのコンポーネントを掛けることである．その結果、各修正済の列における最上部の非ゼロのコンポーネントに１が存在することになる。２つの列が、等しい場合および等しい場合にのみ、修正済の行列中で等価であるという望ましい結果をもたらすことになる。

【０１２６】

本発明のこの好適な実施例がその効率を増進する場合を示唆するためには、比較的小さな有限集合Ｓ中にそのエントリを有する行列を考慮することである。実際上、このセットＳは、有限集合体、集合体の乗算グループにおける有限のサブグループまたは乗算の下で閉じた集合体の有限部分集合、およびより大きな普遍性の中で、集合体の任意の有限部分集合であってよい。それは、既に議論された２進法のケースを含み、ここでＳ＝｛０，１｝、またはＳ＝｛１，－１｝および他の非常に共通した状況、ここでＳ＝｛０，１，－１｝またはＳ＝｛１，－１，ｊ，－ｊ｝またはＳ＝｛０，１，－１，ｊ，－ｊ｝である。利得はＳのサイズで減少し、その一般的な規則は、ＧＥＭが$\log_{|s|}$（ｎ）の要因だけ加算の数を減少することを維持する、ここでｎは与えられた変換行列中の列の数である。適用可能な場合、ＧＥＭは、効率のために、後で記述されるであろう複雑で一般的な実施例と比較される。

【０１２７】

この方法が動作する方法を示すために、エントリが集合体からくる行列を示す、すなわちU_1＝ｄｅｆ＝｛１，－１，ｊ，－ｊ｝はU_1行列と呼ばれる。このようなタイプの行列は、しばしば実際上現われる。しかしながら、ＧＥＭがあらゆる特定のタイプの行列に制限されないことが再度強調されなければならない。加算の数は、ベクトルによるｒ×ｎ　Ｕ

$_1$行列の積のために、Ｃ＝ｎ＋４$^{r-1}$＋ｏ（４$^{r-1}$）によって制限される。

【０１２８】

例：次の３×１５　Ｕ$_1$行列Ａを１５次元の複素入力ベクトルｘで乗じた積を検討する。行列は次式で与えられる。

【０１２９】

【数３６】

$$A=\begin{bmatrix} 1 & -j & j & -1 & 1 & -j & -1 & j & -1 & j & -1 & 1 & -j & 1 & j \\ -1 & j & -1 & j & 1 & j & 1 & 1 & j & -j & -j & -j & -j & -1 & -j \\ j & 1 & -1 & j & j & j & 1 & -1 & 1 & -1 & -1 & -j & 1 & j & -1 \end{bmatrix}$$

【０１３０】

そして、入力ベクトルは次に与えられる。

【０１３１】

【数３７】

$$x=\begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_{15} \end{bmatrix}$$

【０１３２】

積の結果は次式で書かれる。

【０１３３】

【数３８】

$$y=\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = A \bullet x$$

【０１３４】

この例において行われる加算は、すべて複素加算である。最初のステップは、等価ラインに対するチェックである。等価なラインは見つからない。次のステップで、積Ａ・ｘは、それぞれのｘコンポーネントを乗じたＡ列の合計に分解される。

【０１３５】

【数３９】

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = x_1\begin{bmatrix} 1 \\ -1 \\ j \end{bmatrix} + x_2\begin{bmatrix} -j \\ j \\ 1 \end{bmatrix} + x_3\begin{bmatrix} j \\ -1 \\ -1 \end{bmatrix} + x_4\begin{bmatrix} -1 \\ j \\ j \end{bmatrix} + x_5\begin{bmatrix} 1 \\ 1 \\ j \end{bmatrix} + x_6\begin{bmatrix} -j \\ j \\ j \end{bmatrix} + x_7\begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} +$$

$$x_8\begin{bmatrix} j \\ 1 \\ -1 \end{bmatrix} + x_9\begin{bmatrix} -1 \\ j \\ 1 \end{bmatrix} + x_{10}\begin{bmatrix} j \\ -j \\ -1 \end{bmatrix} + x_{11}\begin{bmatrix} -1 \\ -j \\ -1 \end{bmatrix} + x_{12}\begin{bmatrix} 1 \\ -j \\ -j \end{bmatrix} + x_{13}\begin{bmatrix} -j \\ -j \\ 1 \end{bmatrix} + x_{14}\begin{bmatrix} 1 \\ -1 \\ j \end{bmatrix} + x_{15}\begin{bmatrix} j \\ -j \\ -1 \end{bmatrix}$$

【０１３６】

次に、上記合計中の各列ベクトルに正規化が行われる。それは、各ベクトルにその上部コ

ンポーネントの逆数を掛けることにより行われる、したがって各係数に対応するベクトルの（同じ）上部コンポーネントを掛けることにより行われる。その結果、各修正済のベクトルの上部コンポーネントは１となる。ベクトルの上部コンポーネントが既に１である場合、修正は必要ではない。上記合計の正規化フォームが次に導き出される。

【０１３７】

【数４０】

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = x_1 \begin{bmatrix} 1 \\ -1 \\ j \end{bmatrix} + (-j)\cdot x_2 \begin{bmatrix} 1 \\ -1 \\ j \end{bmatrix} + j\cdot x_3 \begin{bmatrix} 1 \\ j \\ j \end{bmatrix} + (-1)x_4 \begin{bmatrix} 1 \\ -j \\ -j \end{bmatrix} + x_5 \begin{bmatrix} 1 \\ 1 \\ j \end{bmatrix} + (-j)x_6 \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} + (-1)x_7 \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} +$$

$$+j\cdot x_8 \begin{bmatrix} 1 \\ -j \\ j \end{bmatrix} + (-1)x_9 \begin{bmatrix} 1 \\ -j \\ -1 \end{bmatrix} + j\cdot x_{10} \begin{bmatrix} 1 \\ -1 \\ j \end{bmatrix} + (-1)x_{11} \begin{bmatrix} 1 \\ j \\ 1 \end{bmatrix} + x_{12} \begin{bmatrix} 1 \\ -j \\ -j \end{bmatrix} + (-j)x_{13} \begin{bmatrix} 1 \\ 1 \\ j \end{bmatrix} + x_{14} \begin{bmatrix} 1 \\ -1 \\ j \end{bmatrix} + j\cdot x_{15} \begin{bmatrix} 1 \\ -1 \\ j \end{bmatrix}$$

【０１３８】

この正規化された合計では、各ベクトルの上部コンポーネントは１であり、また、列ｎの数が、行ｒの数（要求：$n > 4^{r-1}$）に比べて十分に大きいとき（一般的なセッティングで）、異なるベクトルの数は実質的に削減される。次のステップは、同じ列の係数を集めて合計することである。したがって、７回の加算を負担することで、次の削減が得られる。

【０１３９】

【数４１】

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = (x_1 + (-j)\cdot x_2 + j\cdot x_{10} + x_{14} + j\cdot x_{15}) \begin{bmatrix} 1 \\ -1 \\ j \end{bmatrix} + j\cdot x_3 \begin{bmatrix} 1 \\ j \\ j \end{bmatrix} + ((-1)x_4 + x_{12}) \begin{bmatrix} 1 \\ -j \\ -j \end{bmatrix}$$

$$+ (x_5 + (-j)x_{13}) \begin{bmatrix} 1 \\ 1 \\ j \end{bmatrix} + ((-j)x_6 + (-1)x_7 \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}) + j\cdot x_8 \begin{bmatrix} 1 \\ -j \\ j \end{bmatrix} + (-1)x_9 \begin{bmatrix} 1 \\ -j \\ -1 \end{bmatrix} + (-1)x_{11} \begin{bmatrix} 1 \\ j \\ 1 \end{bmatrix}$$

【０１４０】

さて、新しい係数が定義される。

【０１４１】

$w_1 = x_1 + (-j)\cdot x_2 + j\cdot x_{10} + x_{14} + j\cdot x_{15}$,
$w_2 = j\cdot x_3$, $w_3 = (-1)x_4 + x_{12}$,
$w_4 = x_5 + (-j)x_{13}$, $w_5 = (-j)x_6 + (-1)x_7$,
$w_6 = j\cdot x_8$, $w_7 = (-1)x_9$, $w_8 = (-1)x_{11}$

従って、上記の方程式は次の形式で書くことができる。

【０１４２】

【数４２】

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = w_1 \begin{bmatrix} 1 \\ -1 \\ j \end{bmatrix} + w_2 \begin{bmatrix} 1 \\ j \\ j \end{bmatrix} + w_3 \begin{bmatrix} 1 \\ -j \\ -j \end{bmatrix} + w_4 \begin{bmatrix} 1 \\ 1 \\ j \end{bmatrix} + w_5 \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} + w_6 \begin{bmatrix} 1 \\ -j \\ j \end{bmatrix} + w_7 \begin{bmatrix} 1 \\ -j \\ -1 \end{bmatrix} + w_8 \begin{bmatrix} 1 \\ j \\ 1 \end{bmatrix}$$

【０１４３】
係数：w_1，．．．．．．．．．．，w_8は、このステージでプロセッサに知られている。
【０１４４】
次に、このベクトル方程式は水平に分割され、２つの方程式からなる次の等価な組になる。
【０１４５】
【数４３】

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = w_1 \begin{bmatrix} 1 \\ -1 \end{bmatrix} + w_2 \begin{bmatrix} 1 \\ j \end{bmatrix} + w_3 \begin{bmatrix} 1 \\ -j \end{bmatrix} + w_4 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + w_5 \begin{bmatrix} 1 \\ -1 \end{bmatrix} + w_6 \begin{bmatrix} 1 \\ -j \end{bmatrix} + w_7 \begin{bmatrix} 1 \\ -j \end{bmatrix} + w_8 \begin{bmatrix} 1 \\ j \end{bmatrix}$$

$$[y_3] = w_1[j] + w_2[j] + w_3[-j] + w_4[j] + w_5[-1] + w_6[j] + w_7[-1] + w_8[1]$$

【０１４６】
一般に、両方はベクトルの方程式である。しかしながら、小規模な本例において、第２の方程式はスカラ方程式に退化している。さて規則に従えば、これらの方程式の各々は、最初の反復と同じ方法で別々に処理されるであろう。上の方程式は前の方程式の基底状態を継承したので、上の方程式は正規化を要求せず、それ故、いかなる非退化の応用においても、正規化は第２の垂直の方程式にのみ必要となる。従って、上のベクトル方程式に対して生じる減少は次のとおりである。
【０１４７】
【数４４】

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = (w_1+w_5) \begin{bmatrix} 1 \\ -1 \end{bmatrix} + (w_2+w_8) \begin{bmatrix} 1 \\ j \end{bmatrix} + (w_3 + w_6 + w_7) \begin{bmatrix} 1 \\ -j \end{bmatrix} + w_4 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

【０１４８】
このステップに対し、さらに４回の加算が必要になる。その演算は、両方の方程式に対しさらに１３回の加算を行って、直接の方法で実行される。その結果、この例で本発明の好適な実施例は、合計２４回の加算が使用される。力任せの演算では、４２回の加算を要求していたであろう。より大きな規模では、その節約はより本質的なものとなる。
トップリッツ行列
本発明のこの好適な実施例を紹介するために、例が提示される。

【０１４９】
例：長さ８のシーケンスが与えられると仮定する、すなわち、Ｕ＝（１，１，－１，－１，１，－１，１，－１）およびそれはデータ・ベクトルｘ＝（x_1，x_2，.....，x_{10}）の３つの連続する仮説をチェックすることが望まれる。その目的は最大の相関の検索である。次の３つの合計を計算するここことが必要である。
【０１５０】
$y_1=1\cdot x_1+1\cdot x_2+(-1)\cdot x_3+(-1)\cdot x_4+1\cdot x_5+(-1)\cdot x_6+1\cdot x_7+(-1)\cdot x_8$
$y_2=1\cdot x_2+1\cdot x_3+(-1)\cdot x_4+(-1)\cdot x_5+1\cdot x_6+(-1)\cdot x_7+1\cdot x_8+(-1)\cdot x_9$
$y_3=1\cdot x_3+1\cdot x_4+(-1)\cdot x_5+(-1)\cdot x_6+1\cdot x_7+(-1)\cdot x_8+1\cdot x_9+(-1)\cdot x_{10}$
これは、次のトップリッツ行列の積によって表わされる。
【０１５１】
【数４５】

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & 0 & 0 \\ 0 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & 0 \\ 0 & 0 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_{10} \end{bmatrix}$$

$$= x_1\begin{bmatrix}1\\0\\0\end{bmatrix} + x_2\begin{bmatrix}1\\1\\0\end{bmatrix} + x_3\begin{bmatrix}-1\\1\\1\end{bmatrix} + x_4\begin{bmatrix}-1\\-1\\1\end{bmatrix} + x_5\begin{bmatrix}1\\-1\\-1\end{bmatrix} + x_6\begin{bmatrix}-1\\1\\-1\end{bmatrix} + x_7\begin{bmatrix}1\\-1\\1\end{bmatrix} +$$

$$x_8\begin{bmatrix}-1\\1\\-1\end{bmatrix} + x_9\begin{bmatrix}0\\-1\\1\end{bmatrix} + x_{10}\begin{bmatrix}0\\0\\-1\end{bmatrix}$$

【０１５２】
次のステップは、次式のように相補するベクトルを集めることである。
【０１５３】
【数４６】

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \left(x_1\begin{bmatrix}1\\0\\0\end{bmatrix} + x_9\begin{bmatrix}0\\-1\\1\end{bmatrix}\right) \left(x_2\begin{bmatrix}1\\1\\0\end{bmatrix} + x_{10}\begin{bmatrix}0\\0\\-1\end{bmatrix}\right) + x_3\begin{bmatrix}-1\\1\\1\end{bmatrix} + x_4\begin{bmatrix}-1\\-1\\1\end{bmatrix} +$$

$$x_5\begin{bmatrix}1\\-1\\-1\end{bmatrix} + x_6\begin{bmatrix}-1\\1\\-1\end{bmatrix} + x_7\begin{bmatrix}1\\-1\\1\end{bmatrix} + x_8\begin{bmatrix}-1\\1\\-1\end{bmatrix}$$

【０１５４】
この段階で、各ブラケット内部の項を検討し、かつ補助定理を利用し、ｘ，ｙがスカラで、ｖ，ｕがベクトルである場合、その場合、
ｘｖ＋ｙｕ＝（1/2）（ｘ＋ｙ）（ｖ＋ｕ）＋（1/2）（ｘ－ｙ）（ｖ－ｕ）から、従って、次式となる。
【０１５５】
【数４７】

$$x_1\begin{bmatrix}1\\0\\0\end{bmatrix}+x_9\begin{bmatrix}0\\-1\\1\end{bmatrix}=\frac{1}{2}(x_1+x_9)\left(\begin{bmatrix}1\\0\\0\end{bmatrix}+\begin{bmatrix}0\\-1\\1\end{bmatrix}\right)+\frac{1}{2}(x_1-x_9)\left(\begin{bmatrix}1\\0\\0\end{bmatrix}-\begin{bmatrix}0\\-1\\1\end{bmatrix}\right)$$

$$=\frac{1}{2}(x_1+x_9)\begin{bmatrix}1\\-1\\1\end{bmatrix}+\frac{1}{2}(x_1-x_9)\begin{bmatrix}1\\1\\-1\end{bmatrix}$$

【０１５６】
同様に、
【０１５７】
【数４８】

$$x_2\begin{bmatrix}1\\1\\0\end{bmatrix}+x_{10}\begin{bmatrix}0\\0\\-1\end{bmatrix}=\frac{1}{2}(x_2+x_{10})\begin{bmatrix}1\\1\\-1\end{bmatrix}+\frac{1}{2}(x_2-x_{10})\begin{bmatrix}1\\1\\1\end{bmatrix}$$

【０１５８】
従って、４つの加算の負担で、結果は次のとおりである。
【０１５９】
【数４９】

$$\begin{bmatrix}y_1\\y_2\\y_3\end{bmatrix}=\frac{1}{2}(x_1+x_9)\begin{bmatrix}1\\-1\\1\end{bmatrix}+\frac{1}{2}(x_1-x_9)\begin{bmatrix}1\\1\\-1\end{bmatrix}+\frac{1}{2}(x_2+x_{10})\begin{bmatrix}1\\1\\-1\end{bmatrix}+\frac{1}{2}($$

$$x_2-x_{10})\begin{bmatrix}1\\1\\1\end{bmatrix}+x_3\begin{bmatrix}-1\\1\\1\end{bmatrix}+x_4\begin{bmatrix}-1\\-1\\1\end{bmatrix}+x_5\begin{bmatrix}1\\-1\\-1\end{bmatrix}+x_6\begin{bmatrix}-1\\1\\-1\end{bmatrix}+x_7\begin{bmatrix}1\\-1\\1\end{bmatrix}+x_8\begin{bmatrix}-1\\1\\-1\end{bmatrix}=$$

【０１６０】
しかし、今、上記Ｕ２進法が適用される。すなわち、次のステップは、各列の上部コンポーネントが１に等しくなるようにする正規化である。
【０１６１】
【数５０】

$$\begin{bmatrix}y_1\\y_2\\y_3\end{bmatrix}=\frac{1}{2}(x_1+x_9)\begin{bmatrix}1\\-1\\1\end{bmatrix}+\frac{1}{2}(x_1-x_9)\begin{bmatrix}1\\1\\-1\end{bmatrix}+\frac{1}{2}(x_2+x_{10})\begin{bmatrix}1\\1\\-1\end{bmatrix}+\frac{1}{2}(x_2-$$

$$x_{10})\begin{bmatrix}1\\1\\1\end{bmatrix}+(-x_3)\begin{bmatrix}1\\-1\\-1\end{bmatrix}+(-x_4)\begin{bmatrix}1\\1\\-1\end{bmatrix}+x_5\begin{bmatrix}1\\-1\\-1\end{bmatrix}+(-x_6)\begin{bmatrix}1\\-1\\1\end{bmatrix}+x_7\begin{bmatrix}1\\-1\\1\end{bmatrix}+(-x_8)\begin{bmatrix}1\\-1\\1\end{bmatrix}$$

【０１６２】
次のステップは、共通の列の係数を集めて合計することである。すなわち、６つの追加の

加算による負担で、次の等式となる。

【０１６３】

【数５１】

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \left(\frac{1}{2}(x_1+x_9)+(-x_6)+x_7+(-x_8)\right)\begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} + \left(\frac{1}{2}(x_1-x_9)+\frac{1}{2}(x_2+x_{10})+(-x_4)\right)\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} + \frac{1}{2}(x_2-x_{10})\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + \left((-x_3)+x_5\right)\begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}$$

【０１６４】

したがって、便宜上次のように定義すれば、

$w_1=(1/2)(x_1+x_9)+(-x_6)+x_7+(-x_8)$

$w_2=(1/2)(x_1-x_9)+(1/2)(x_2+x_{10})+(-x_4)$

$w_3=(1/2)(x_2-x_{10})$，　$w_4=(-x_3)+x_5$

そのとき、その等式は次のように書かれる。

【０１６５】

【数５２】

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = w_1\begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} + w_2\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} + w_3\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + w_4\begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}$$

【０１６６】

次のステージでは、このベクトル方程式は、次の２つの方程式に水平に分割される。

【０１６７】

【数５３】

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = w_1\begin{bmatrix} 1 \\ -1 \end{bmatrix} + w_2\begin{bmatrix} 1 \\ 1 \end{bmatrix} + w_3\begin{bmatrix} 1 \\ 1 \end{bmatrix} + w_4\begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$y_3 = w_1 + (-w_2) + w_3 + (-w_4)$$

【０１６８】

最初の方程式中の同一の列を集めると、さらに２回の加算で次式が得られる。

【０１６９】

【数５４】

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = (w_1+w_4)\begin{bmatrix} 1 \\ -1 \end{bmatrix} + (w_2+w_3)\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

【０１７０】

さて、所望の出力ベクトルｙは、５回の追加の加算ですむことが分かる。したがって、本発明を具体化する方法は、この演算を行なうために１７回の加算を要求する。力任せの方法では、２１回の加算を要求するであろう。この小規模の例によって得られた利点は、あ

まり大きなものではない、しかし、大規模な応用において、その利得は上記「正方形」（非トップリッツ）Ｕ方法の利得に匹敵することを理解できるであろう。

【０１７１】

次に、本発明の特徴における一般的な構成を記述する。Ｕシーケンス、

$u_1, u_2, \ldots\ldots\ldots, u_n$

および、実数か複素数のスカラのデータのシーケンス、

$x_1, x_2, \ldots\ldots\ldots, x_{n+m-1}$

が与えられると仮定する。

【０１７２】

データ・シーケンスの要素は、実数か複素数または、２より大きな特性のスカラの任意の集合体に属していてもよい。次の合計を計算することが所望されると仮定する。

【０１７３】

$y_1 = u_1 \cdot x_1 + u_2 \cdot x_2 + \ldots\ldots\ldots + u_n \cdot x_n$

$y_2 = u_1 \cdot x_2 + u_2 \cdot x_3 + \ldots\ldots\ldots + u_n \cdot x_{n+1}$

・

・

・

・

$y_m = u_1 \cdot x_m + u_2 \cdot x_{m+1} + \ldots\ldots\ldots + u_n \cdot x_{n+m-1}$

$m < \log_2(n)$ならば、$r = m$にする一方、$m \geqq \log_2(n)$ならば、本発明の好適な実施例は、ｍ合計のセットをｒの連続する合計のブロックへ分割することにより開始し、ここで$r < \log_2(n)$である。その方法が第１のブロック上で示されてもよいように、すべてのブロックは同一の方法で扱われる。トップリッツ行列とベクトルの積は、最初のｒ合計を表わす。そのとき、ｒ×（ｎ＋ｒ－１）トップリッツ行列は、

【０１７４】

【数５５】

$$A=\begin{bmatrix} u_1 & u_2 & u_3 & u_4 & \cdots & u_n & 0 & \cdots & 0 & 0 \\ 0 & u_1 & u_2 & u_3 & \cdots & u_n & 0 & \cdots & 0 & 0 \\ 0 & 0 & u_1 & u_2 & \cdots & u_n & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & u_1 & \cdots & u_n & 0 & \cdots & 0 & 0 \\ \vdots & & & & \ddots & & & & \vdots & \vdots \\ \vdots & & & & & \ddots & & & 0 & 0 \\ \vdots & & & & & & \ddots & & 0 & 0 \\ 0 & 0 & \cdots & \cdots & 0 & u_1 & u_2 & \cdots & u_n & 0 \\ 0 & 0 & \cdots & \cdots & 0 & 0 & u_1 & \cdots & u_{n-1} & u_n \end{bmatrix}$$

【０１７５】

そして、その（ｎ＋ｒ－１）次元のベクトルは、

【０１７６】

【数５６】

$$x=\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n+r-1} \end{bmatrix}$$

【０１７７】
それから、最初のｒ合計は、ベクトルから与えられ、
【０１７８】
【数５７】

$$y=\begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_r \end{bmatrix}$$

【０１７９】
となる。ここで、ｙ＝Ａ・ｘである。
【０１８０】
本発明の好適な実施例について、その強調する概念は、上記Ｕ方法が適用可能になるように、与えられた問題を再編成することである。それらの順序に従って、v_1，v_2，.....，v_{n+r-1}を行列Ａの列にする。次の全ての「中間」列ベクトルがＵベクトルであると考える。
【０１８１】
【数５８】

$$v_r=\begin{bmatrix} u_r \\ u_{r-1} \\ \cdot \\ \cdot \\ \cdot \\ u_1 \end{bmatrix},\ v_{r+1}=\begin{bmatrix} u_{r+1} \\ u_r \\ \cdot \\ \cdot \\ \cdot \\ u_2 \end{bmatrix},\ldots\ldots\ldots\ldots\ldots\ldots,v_n=\begin{bmatrix} u_n \\ u_{n-1} \\ \cdot \\ \cdot \\ \cdot \\ u_{n-r+1} \end{bmatrix}$$

【０１８２】
さらに「サイド」列ベクトルの「マッチング」ペアにおける次の合計および減算が、Ｕベクトルであることに注意すること。
【０１８３】
【数５９】

$$v_1 + v_{n+1} = \begin{bmatrix} u_1 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ u_n \\ \cdot \\ \cdot \\ \cdot \\ u_{n-r+2} \end{bmatrix} = \begin{bmatrix} u_1 \\ u_n \\ u_{n-1} \\ \cdot \\ \cdot \\ u_{n-r+2} \end{bmatrix}, \quad v_1 - v_{n+1} = \begin{bmatrix} u_1 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ u_n \\ \cdot \\ \cdot \\ \cdot \\ u_{n-r+2} \end{bmatrix} = \begin{bmatrix} u_1 \\ -u_n \\ -u_{n-1} \\ \cdot \\ \cdot \\ -u_{n-r+2} \end{bmatrix}$$

$$v_2 + v_{n+2} = \begin{bmatrix} u_1 \\ u_2 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ u_n \\ \cdot \\ \cdot \\ u_{n-r+3} \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_n \\ \cdot \\ \cdot \\ u_{n-r+3} \end{bmatrix}, \quad v_2 - v_{n+2} = \begin{bmatrix} u_1 \\ u_2 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ u_n \\ \cdot \\ \cdot \\ u_{n-r+3} \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ -u_n \\ \cdot \\ \cdot \\ -u_{n-r+3} \end{bmatrix}$$

$$v_{r-1} + v_{n+r-1} = \begin{bmatrix} u_{r-1} \\ u_{r-2} \\ \cdot \\ \cdot \\ u_1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ 0 \\ u_n \end{bmatrix} = \begin{bmatrix} u_{r-1} \\ u_{r-2} \\ \cdot \\ \cdot \\ u_1 \cdot \\ u_n \end{bmatrix}, \quad v_{r-1} - v_{n+r-1} = \begin{bmatrix} u_{r-1} \\ u_{r-2} \\ \cdot \\ \cdot \\ u_1 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ 0 \\ u_n \end{bmatrix} = \begin{bmatrix} u_{r-1} \\ u_{r-2} \\ \cdot \\ \cdot \\ u_1 \cdot \\ -u_n \end{bmatrix}$$

【０１８４】

必要な予備的な準備を行った後に、その方法が導入される。上記に従って、再構成する。

【０１８５】

$A \cdot x = \Sigma_{1 \leqq j \leqq n+r-1} x_j v_j$

$= \Sigma_{1 \leqq j \leqq r-1} x_j v_j + x_{n+j} v_{n+j} + \Sigma_{r \leqq j \leqq n} x_j v_j$

次に、その発明は、ｘ，ｙがスカラであり、ｖ，ｕがベクトルであるという規則を使用する。

【０１８６】

$xv+yu=(1/2)(x+y)(v+u)+(1/2)(x-y)(v-u)$

したがって、

$A \cdot x = \delta_{1 \leqq j \leqq r-1} (1/2)(x_j + x_{n+j})(v_j + v_{n+j}) + (1/2)(x_j - x_{n+j})(v_j - v_{n+j}) + \delta_{r \leqq j \leqq n} x_j v_j$

このプロセスは２ｒ－２加算の負担であり、達成された形式は実際ｒ×（ｎ＋ｒ－１）Ｕ行列とｎ＋ｒ－１ベクトルの積である。これは、上記すべてのベクトル、

$v_r, v_{r+1}, \ldots\ldots\ldots, v_n,\ v_1 + v_{n+1}, \ldots\ldots\ldots, v_{r-1} + v_{n+r-1},$

$v_1 - v_{n+1}, \ldots\ldots\ldots, v_{r-1} - v_{n+r-1}$

がＵベクトルであるので、そのとおりである。したがって、演算の残りは、Ｕ方法によって行われてもよい。Ｕ形式へのこの修正のすべての局面は、実際の合計／減算を除く、すなわち$x_j \pm x_{n+j}$，は入力ベクトルの到来に先立つ「一回のジョブ」として行われる。

【０１８７】

最悪の場合での加算数は、次の式から与えられる

$s_t(n, r) = s(n+r-1, r) + 2r - 2 = n + 3r - 3 + s(r)$

一般的な設定で、ｍは計算され、$m > log_2(n)$で、最悪の場合での加算数は$(1+\delta)(n + 3\log_2(n)) \cdot m / \log_2(n)$によって制限され、ここで$1 > \delta > 0$において、ｍおよびｎ両方が無限大に近づくとδは０に近づく。

【０１８８】

上記方法への本発明の別の好適な実施例によれば、ＧＥＭのコンポーネントは、統合されてもよい。そのような場合、いくつかの「サイド」列はサイド列を結合する最初のステー

ジでは未処理加工のまま残る。

（０、１、－１）行列

トップリッツ行列は、より一般的なクラスの行列の一部である、すなわちそれらの行列のエントリは０、１または－１である。そのような行列は、ここでは（０，１，－１）行列と称する。上記トップリッツ方法と関係する概念のより広いバージョンが、ここで展開されるであろう。Ａが次元ｒ×ｎの（０，１，－１）行列で、ｘがｎ次元の入力ベクトルであり、積Ａ・ｘを計算することが所望されると仮定する。この積は、対応するｘコンポーネントを乗じたＡ列の合計として表現される。したがって、v_j（すべてのｊに対して）がＡのｊ列であることを示すと、そのとき、次式が示される。

【０１８９】

$$A \cdot x = x_1 v_1 + x_2 v_2 + \cdots\cdots\cdots + x_n v_n$$

いくらか、あるいは恐らくＡのすべての列は０－エントリを含む。表記を簡単にするために、指標は非常によく整えられ（もちろん、方法的にプロセッサにタスクを課さずに）、最初のｋ列v_1，v_2，……，v_kの各々は０のエントリを含みかつ残りのすべてのｎ-ｋ列（いずれにあっても）v_{k+1}，v_{k+2}，……，v_nはＵベクトル（すなわち０のエントリのない）であると仮定される。「混合」ベクトル、v_1，v_2，……，v_kの各々は、平均２つのＵベクトルの平均である。すなわち、各$1 \leqq j \leqq k$に対して、２つのＵベクトルu_j，w_j，がv_j，$=(1/2)(u_j+w_j)$のように存在する。本発明の好適な実施例は、ベクトルu_1，w_1，……，u_k，w_k，を行列毎に一度の準備として認識する。したがって上記のによって次式となる。

【０１９０】

$$A \cdot x=(1/2)x_1(u_1+w_1)+(1/2)x_2(u_2+w_2)+\cdots\cdots\cdots$$
$$+(1/2)x_k(u_k+w_k)+x_{k+1}v_{k+1}+\cdots\cdots\cdots+x_n v_n$$

しかしながら、ブラケットを開くと、ｒ×（ｎ＋ｋ）Ｕ行列と（ｎ＋ｋ）ベクトルの積の表現を発見する。本発明の好適な実施例によれば、このタスクは、本発明の上記のＵ特徴によって行われるであろう。本発明の上記トップリッツ特徴はこのより広い特徴の特別な場合である。

産業上の応用の例

サーチャ(searcher)は、通常トップリッツ行列とデータ・ベクトルの積によって表わされる。これは、ＣＤＭＡおよび広帯域ＣＤＭＡの両方にみられる。

【０１９１】

実数行列

本発明の別の好適な実施例によれば、計算の動作回数は分散型の算術の使用によってあらゆる実数の線形変換に対して削減される。以下の説明において、それは「実数マトリクス法」と呼ばれる。線形変換は行列によって表現され、そのエントリは整数である必要はなく、２進数の定数である実数である。行列は、２進の係数を備えた２進の行列の合計に分解されるであろう。次のステージでは、本発明の２進の実施例が適用されるであろう。この方法を導入するためには、次の例を考慮すべきである。

例：次の３×８のＵ行列Ａを検討するが、整数のエントリを有し、（この例の単純化のために）１０進数で書かれる。

【０１９２】

【数６０】

$$A=\begin{bmatrix} 2 & 1 & 5 & 4 & 3 & 0 & 4 & 5 \\ 5 & 0 & 4 & 1 & 4 & 7 & 1 & 2 \\ 2 & 7 & 3 & 1 & 4 & 5 & 0 & 3 \end{bmatrix}$$

【０１９３】

入力が８次元のベクトルは、次のように与えられる。

【０１９４】

【数６１】

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix}$$

【０１９５】
３次元の出力ベクトルを計算することが所望される。
【０１９６】
【数６２】

$$y=\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}=A\bullet x.$$

【０１９７】
プロセッサは通常２進を基本に動作するので、行列Ａはこの基本により表わされ、したがって、次のように与えられる。
【０１９８】
【数６３】

$$A=\begin{bmatrix} 010 & 001 & 101 & 100 & 011 & 000 & 100 & 101 \\ 101 & 000 & 100 & 001 & 100 & 111 & 001 & 010 \\ 010 & 111 & 011 & 001 & 100 & 101 & 000 & 011 \end{bmatrix}$$

【０１９９】
この表現は、その行列を３つの２進・行列の合計として表現するために、分散型算術を使用する可能性を示唆する。ここで、次式が示される。
Ａ＝2^{0}・Ａ［０］＋2^{1}・Ａ［１］＋2^{2}・Ａ［２］
【０２００】
【数６４】

$$A[0] = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$A[1] = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A[2] = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

【０２０１】
本発明のこの実施例によって行われる最初のステップは、その特徴に従って、Ａにおけるエントリのビットからなる２進・行列Ａ［０］，Ａ［１］，Ａ［２］を構成することである。検討中の好適な実施例は、同一の結果になるであろう。
【０２０２】
$$A \cdot x = 2^0 \cdot A[0] \cdot x + 2^1 \cdot A[1] \cdot x + 2^2 \cdot A[2] \cdot x$$
次のステップは、Ａ［０］，Ａ［１］，Ａ［２］を水平に結び付けることによって形成される２４×３の２進・行列A^*を構成することである。
【０２０３】
【数６５】

$$\mathbf{A^*} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

【０２０４】
そして、また上記の合計から推定された対応する２進の重み付けを各々有する３つのベクトルｘのレプリカから構成される２４次元の列ベクトルx^*を形成することである。これは、入力ベクトルの到着が始まる前に、なされる。
【０２０５】
【数６６】

$$x^*=\begin{bmatrix} 2^0x_1 \\ \cdot \\ \cdot \\ 2^0x_8 \\ 2^1x_1 \\ \cdot \\ \cdot \\ 2^1x_8 \\ 2^2x_1 \\ \cdot \\ \cdot \\ 2^2x_8 \end{bmatrix}$$

【０２０６】
2^1で乗ずることは、単に指標のシフト動作だけですむことに注意すること。この実施例の鍵は、次式を主張することである。

【０２０７】
$y=A\cdot x=A^*\cdot x^*$
したがって、次にすることは、０－１方法による$A^*\cdot X^*$の演算を行なうことである。すなわち、次の作業は、新しい係数を作成し、それによって、行列のサイズを削減して、共通の非ゼロ列の係数を集めて合計することである。すなわち、それは次式を維持する。

【０２０８】
【数６７】

$$y=w_1\begin{bmatrix}1\\0\\0\end{bmatrix}+w_2\begin{bmatrix}0\\1\\0\end{bmatrix}+w_3\begin{bmatrix}1\\1\\0\end{bmatrix}+w_4\begin{bmatrix}0\\0\\1\end{bmatrix}+w_5\begin{bmatrix}1\\0\\1\end{bmatrix}+w_6\begin{bmatrix}0\\1\\1\end{bmatrix}$$

【０２０９】
ここで
$w_1=2^2\cdot x_7+2^2\cdot x_8+2^1\cdot x_4+2^1\cdot x_5+2^0\cdot x_5$
$w_2=2^2\cdot x_1+2^1\cdot x_6+2^0\cdot x_1+2^0\cdot x_7$
$w_3=2^2\cdot x_3$
$w_4=2^2\cdot x_2+2^1\cdot x_2+2^1\cdot x_3$
$w_5=2^1\cdot x_5+2^0\cdot x_2+2^0\cdot x_1+2^0\cdot x_3+2^0\cdot x_8$
$w_6=2^2\cdot x_1+2^1\cdot x_6+2^0\cdot x_1+2^0\cdot x_7$
これは１６回の加算で行われる。次に、生起する行列の行分割が行われる。すなわち、

【０２１０】
【数６８】

$$\begin{bmatrix}y_1\\y_2\end{bmatrix}=w_1\begin{bmatrix}1\\0\end{bmatrix}+w_2\begin{bmatrix}0\\1\end{bmatrix}+w_3\begin{bmatrix}1\\1\end{bmatrix}+w_5\begin{bmatrix}1\\0\end{bmatrix}+w_6\begin{bmatrix}0\\1\end{bmatrix}$$

【０２１１】
であり、そして、
$y_3=w_1+w_3+w_5$である。

【０２１２】
さて、第１の方程式中における共通の列の係数を合計すると、次式を導く。
【０２１３】
【数６９】

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = (w_2 + w_5)\begin{bmatrix} 1 \\ 0 \end{bmatrix} + (w_2 + w_6)\begin{bmatrix} 0 \\ 1 \end{bmatrix} + w_3\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

【０２１４】
このステップは２つの加算を要する。残りはさらに２つの加算でなされる。この例における変換を計算するために、トータル２０回の加算動作が必要とされる。従来方式を使用すれば、２９回の加算に相当する動作を要求していたであろうが、ここで意味する「相当」とは、乗算動作を含む加算も考慮に入れる。
【０２１５】
明らかに、これは顕著な削減の例ではない。それは、本発明のこの実施例についての概念を容易な方法で紹介することである。しかしながら、行列（各エントリの次元および桁数）のパラメータが大きいとき、実質的な削減は本発明によって実現される。
【０２１６】
一般に、検討中の本発明の好適な実施例は、実数，無制限の線形変換の効率的な演算に関係する。変換を表現するｒ×ｍ行列Ａは、次式で書かれる。
【０２１７】
【数７０】

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{r1} & & & a_m \end{bmatrix}$$

【０２１８】
ここで、行列のエントリは実数である。Ａ・ｘを計算することが所望され、ここでｘは次のように表現される実数か複素数のスカラのエントリからなるｎ次元のベクトルである。
【０２１９】
【数７１】

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

【０２２０】
行列Ａのエントリは、そのポイントの前後で、固定されたデジット数の２進数で書かれていると仮定する。本発明の２つの好適な実施例が、変換行列の構造に依存する中で、その

最適な解決および選択として示される。最初は、０－１分解と呼ばれ、２番目はＵ分解と称される。

【０２２１】

行列Ａのエントリが2^{m1}によって制限され、適切な正の整数m_1およびm_2のためのポイントを越えるm_2デジットの精度を有すると仮定する。プロセッサが遭遇したすべてのスカラはそのポイントの前後に有限なデジット数を有するので、この仮定は本発明の範囲を制限するものではない。$m=m_1+m_2+1$とする。分散演算は、積Ａ・ｘを上記の例によって扱われたタイプのｍ２進積の合計へ分解するために適用される。

【０２２２】

最初に、その発明は、行列のエントリが負ではなく、またその分解が０－１－ベースである場合に記述される。2^{m1}によって境界がなされ、そのポイントを越えるm_2デジットを有する実数ｔは、次の方法で表わされる。

【０２２３】

$$t=\Sigma_{-m2\leqq k\leqq m1}\ t_k\cdot 2^k$$

ここで、t_kはそれぞれ０または１である。これは標準の２進数の表現である。典型的な分散演算の議論によって、Ａは、次の方法でｍの合計　ｒ×ｎの０－１－２進・行列に分解することができる。

【０２２４】

$$A=\Sigma_{-m2\leqq k\leqq m1}\ 2^k\cdot A[k]$$

この合計では、Ａ［m_1］は最上位ビット（ＭＳＢ）の０－１行列であり、Ａ［$-m_2$］は最下位ビット（ＬＳＢ）の０－１行列である。一般に、行列Ａ［ｋ］の各々は０－１行列であり、そのエントリはＡにおけるエントリのｋ番目のビットで構成される、ここで各ビットはその対応するエントリに置かれる。分配規則によって、次のとおりとなる。

【０２２５】

$$A\cdot x=\Sigma_{-m2\leqq k\leqq m1}\ 2^k\cdot A[k]\cdot x$$

これは本発明の現在の実施例に対する基礎となる。

【０２２６】

次に、A^*は、それらが合計に現われる上記合計の２進・行列の水平上のラインによって順番に構成されたｒ×（ｍ・ｎ）の０－１行列であるとする。そのとき、次式となる。

【０２２７】

$A^*=[A[-m_2],\ .....,\ A[0],\ .....,\ A[m_1]]$

入力ベクトルの到着が始まる前に、これは所定の行列のために１回行われるであるであろう。

【０２２８】

さらに、ｍ・ｎ次元の列ベクトルx^*が次式のように与えられる。

【０２２９】

【数７２】

$$x^*=\begin{bmatrix}2^{-m_2}x\\2^{-m_2+1}x\\\vdots\\2^{0}x\\\vdots\\2^{m_1}x\end{bmatrix}$$

【０２３０】
検討中の実施例を強調する重要な点は次のことである。
【０２３１】
$A \cdot x = A^{*} \cdot x^{*}$
後者は、ｒ×（ｍ・ｎ）の０－１行列を、元のベクトルのシフトしたｍ個のレプリカから構成される（ｍ＋１）・ｎ次元のベクトルで乗じた積である。
積$A^{*} \cdot x^{*}$の演算は、本発明の０－１－２進の実施例でなされる。２の整数累乗による乗算がそれぞれビットをシフトすることにより実行されるので、ほとんど、追加の複雑性は加わらない。
【０２３２】
本発明のこの実施例は、積の複雑性を著しく減少させることができる。
【０２３３】
$l = \log_2(m \cdot n) = \log_2(m) + \log_2(n)$
であるから、Ｃ（Ａ）は本発明によって$A^{*} \cdot x^{*}$を計算するのに必要な加算の数である。
【０２３４】
もし、$l \geqq r$ ならば、そのとき
$C(A) < m \cdot n + 2^{r} + 2^{r/2+1} - r$
特に、この場合その境界は次のとおりとなる。
【０２３５】
$C(A) < 2m \cdot n$
$l \geqq r$の仮定をはずすと、それは次のようになる。
【０２３６】
$C(A) < (1+\delta) \cdot m \cdot r \cdot n / l$
ここで$1 > \delta > 0$、およびｍ・ｎおよびｒが無限大に行くにつれ、δは０へ接近する。
【０２３７】
積動作を含む加算が考慮に入れられる場合、積Ａ・ｘを計算する従来の（力まかせによる）先行技術の方法は、平均して、ｎ・ｒ・ｍ／２回の加算に相当する回数を要求する。
【０２３８】
いくつかの場合、特に、ｒ＝１あるいはｒがやや小さいとき、上述した実施例とは別のやり方が所望され、それはさらなる削減を可能にする。ケースｒ＝１では、問題は２つのベクトル間におけるスカラ積の問題に形をかえることに注意すること。これはそれ自体、科学技術にやや共通した演算であり、また、その効率的な実行は多くの場合役に立つ。この変形によれば、行列A^{**}は、行列$A[-m_2]$，$A[-m_2]$，.....，$A[0]$，.....，$A[m_1]$を垂直のシーケンスにつなぐことにより形成される。したがって、それは、２進乗算のためのシフトを用いることで、所望の結果が次に合計を実行することにより得られる。次のものから与えられたｒ・（ｍ＋１）×ｎの０－１行列である。
【０２３９】
【数７３】

$$A^{**} = \begin{bmatrix} A[-m_2] \\ \cdot \\ \cdot \\ \cdot \\ A[0] \\ \cdot \\ \cdot \\ \cdot \\ A[m_1] \end{bmatrix}$$

【０２４０】
次に、A^{**}・ｘの演算は０－１－行列のために本発明の好適な実施例によって行われる。積A^{**}・ｘは積をすべて含んでいる。
【０２４１】
Ａ［－m_2］・ｘ，．．．．．．．．．，Ａ［０］・ｘ，．．．．．．．．．，Ａ［m_1］・ｘ
従って、シフトを２進乗算のために使用することで、所要の結果はその合計を行なうことにより達成される。
【０２４２】
$y = \Sigma_{-m2 \leqq k \leqq m1} \ 2^k \cdot A[k] \cdot x$
これは、負でないエントリで行列の一部を終了する。
【０２４３】
行列Ａが両符号のエントリを有する実数のとき、行列Ａは負でないエントリを備えた２つの行列の減算として表現することができる、すなわち$A = A_1 - A_2$である。この表現に続いて、議論中の本発明の特徴は、まず積$y_1 = A_1 \cdot x$および$y_2 = A_2 \cdot x$の各々を上記方法により別々にまたは結合した形式で計算することにより$y = A \cdot x$の計算を行なう。その後、最終ステップである減算、すなわち$y = y_1 - y_2$を実行する。
【０２４４】
実数行列に関しての本発明の好適な実施例における上記０－１－２進オプションは、２進・行列を分解するとき、すなわち、Ａ［－m_2］，Ａ［－m_2＋１］，．．．．．．．．．，Ａ［０］，．．．．．．．．．，Ａ［m_1］がややばらばらであるとき、特に効率的である。これは、Ａのエントリが様々なサイズをもちポイントを越えて非同一の２進デジット数を有している結果である。そのような場合、上記言及した一定のデジタル形式に対し必要な０を挿入することにより、より高いレベルのまばらを引き起こす。
【０２４５】
Ｕ－２進の分散演算に基づいて、本発明の好適な実施例の別の形式が次に記述される。本発明のこの形式は、両方の符号のエントリを有しおよびより高速のＵ方法に基づく行列により適応した利点を有する。実際、行列のエントリのサイズにある統一性と精度があるとき、上記の０－１バージョンはより効率的である。下記方法の主な特徴は、０－１方法に類似する。
【０２４６】
次の検討は、本発明の説明を進めるために必要である。2^{m1}によって境界付けされ、そのポイントを越えるm_2個のデジットを有する実数ｔは、次の方法でＵ２進合計として表わすことができる。
【０２４７】
$t = \Sigma_{-m2-1 \leqq k \leqq m1-1} \ s_k \cdot 2^k + s_{m1} \cdot (2^{m1} - 2^{-m2-1})$
ここで、$-m_2 - 1 \leqq k \leqq m_1 - 1$に対する全ての$s_k$は、±１であり、そしてｔが負でないとき、$s_{m1} = 1$であり、ｔが負のとき、$s_{m1} = -1$である。したがって、両方の符号

のエントリを備えたｒ×ｎの実数の行列Ａは、次の方法でＵ行列の合計に分解することができる。

【０２４８】

$A = \Sigma_{-m2-1 \leqq k \leqq m1-1} 2^{k} \cdot A[k] + (2^{m1} - 2^{-m2-1}) \cdot A[m_1]$

ここで行列Ａ［ｋ］の各々はｒ×ｎのＵ行列である。

【０２４９】

A^{*}がｒ×（（ｍ＋１）・ｎ）のＵ行列であるとする。

【０２５０】

$A^{*}=[A[-m_2-1],A[-m_2],.....,A[0],.....,A[-m_1-1],A[-m_1]]$

この行列は、入力してくるデータ・ベクトルの到着前に、各行列について１度のタスクとして構成される。さらに、（ｍ＋１）・ｎ次元の列ベクトルx^{*}は次のように定義される。

【０２５１】

【数７４】

$$x^{*}=\begin{bmatrix} 2^{-m_2-1}x \\ 2^{-m_2}x \\ \cdot \\ \cdot \\ 2^{0}x \\ \cdot \\ \cdot \\ 2^{m_1-1}x \\ (2^{m_1}-2^{-m_2-1})\cdot x \end{bmatrix}$$

【０２５２】

それは、$A \cdot x = A^{*} \cdot x^{*}$を保持する。これは、ｒ×（（ｍ＋１）・ｎ）のＵ行列を（ｍ＋１）・ｎ次元のベクトルで乗じた積である。この積の演算は、本発明のＵ行列実施例の応用よって行われる。

【０２５３】

０－１－２進分解に関係する上記の方法でのように、さらにここに、ｒ＝ｌまたは小さなｒのための垂直バージョンがある。それは上記記述したものと完全に類似しており、詳細を繰り返す理由はない。

【０２５４】

ｌ＝ｌｏｇ（（ｍ＋１）・ｎ）＝ｌｏｇ（ｍ＋１）＋ｌｏｇ（ｎ）とする。それはｌ≧ｒに対し、上記の方法を行なうのに必要な加算の数は、

$C(A) < (m+1) \cdot n + 2^{r-1} + 2^{r/2+1} - r$

によって制限される。

【０２５５】

上記の説明のように、Ｃ（Ａ）は上記Ｕ実施例によってＡ・ｘを計算するために必要な加算の数であると定義される。より広い一般法則では、それは次のようになる。

【０２５６】

$C(A) < (1+\delta) \cdot (m+1) \cdot r \cdot n / l$

ここで、１＞δ＞０、（ｍ＋１）・ｎおよびｒが無限大に近づくにつれ、δは０に近づく。

産業上の応用例

線形変換は、あらゆる分野の技術と科学に共通して使用される。本発明の実数行列の面を通信技術に応用することは、マルチユーザの検出器（ＭＵＤ）行列（例えば非相関器ま

たは最小平均二乗誤差（ＭＭＳＥ）行列）をｄｅｓｐｒｅａｄｅｒの出力ベクトルで乗じた積を含む。さらに、それは最小二乗法の計算に適用可能である。有限インパルス応答（ＦＩＲ）フィルタでは、離散フーリエ変換（ＤＦＴ）が完全にまたは部分的に計算される。特に、部分ＤＦＴのＦＩＲフィルタおよびＦＦＴが効率的でない中小規模のフィルタは、本発明のこの特徴が適用可能な例である。離散コサイン変換ＤＣＴは、別のタイプの線形変換で、その演算は本発明によって改善され得る。これは、特に部分的に計算されるだけであるとか、そのサイズがあまり大きくない場合に特に効果的で、その結果、より高次の高速アルゴリズムにはあまり効率的ではない。

【０２５７】

ＦＩＲを使用する処理回路のようないくつかのデジタル信号処理アプリケーションでは、２つの比較的長いベクトルの相関が要求される。そのベクトルの１つは、ＦＩＲフィルタのタップを表わすことがあり、それはフィルタされるべき入力を表わす第２のベクトル上で動作する。部分的な畳み込みから成るフィルタ動作は、トップリッツ行列をベクトルで乗じた積によって表わされる。これは、本発明の実数行列の面から効率的に行われる。

複素行列

例：次の合計を計算することを目的と仮定する。

【０２５８】

$y_1=(1+j)\cdot x_1+(1-j)\cdot x_2+(-1-j)\cdot x_3+(-1+j)\cdot x_4+(1-j)\cdot x_5+(-1+j)\cdot x_6$
$y_2=(-1+j)\cdot x_1+(1+j)\cdot x_2+(1+j)\cdot x_3+(-1-j)\cdot x_4+(-1-j)\cdot x_5+(1-j)\cdot x_6$
$y_3=(1-j)\cdot x_1+(-1-j)\cdot x_2+(-1-j)\cdot x_3+(1+j)\cdot x_4+(-1+j)\cdot x_5+(1+j)\cdot x_6$

ここで、入力スカラx_1，x_2，.....，x_6は複素数である。

【０２５９】

従来の先行技術からのアプローチでは、６６回の実数加算を要求するであろう。これは、ソート（±１±ｊ）のファクタを複素数で乗じた積は２回の実数加算を要求し、２つの複素数の加算は２回の実数加算を必要とすることを考慮に入れると、理解される。

【０２６０】

この演算を行なうための本発明の好適な実施例の２つの主要な選択が示される。相回転プラスＧＥＭと呼ばれる本発明の第１の好適な実施例で、、それは次の事実を使用する。

【０２６１】

$(1/2)(1+j)\cdot(1+j)=j$
$(1/2)(1+j)\cdot(1-j)=1$
$(1/2)(1+j)\cdot(-1+j)=-1$
$(1/2)(1+j)\cdot(-1-j)=-j$

従って、上記の合計すべてに(1/2)（１＋ｊ）を掛けることによって、ここで相回転と呼ばれる行為を行い、および我々はセット｛１，－１，ｊ，－ｊ｝からの係数を得る。

【０２６２】

$(1/2)(1+j)\cdot y_1=j\cdot x_1+1\cdot x_2+(-j)\cdot x_3+(-1)\cdot x_4+1\cdot x_5+(-1)\cdot x_6$
$(1/2)(1+j)\cdot y_2=(-1)\cdot x_1+j\cdot x_2+j\cdot x_3+(-j)\cdot x_4+(-j)\cdot x_5+1\cdot x_6$
$(1/2)(1+j)\cdot y_3=1\cdot x_1+(-j)\cdot x_2+(-j)\cdot x_3+j\cdot x_4+(-1)\cdot x_5+j\cdot x_6$

本発明の好適な実施例によれば、これらの合計はＧＥＭによって計算される。この例の小規模なサイズにより、従来の方法と比較して、利得はこの場合欄外であるが、しかし、それはより大きな次元で本質的なものとなる。この例のように、次元が小さい時、他の従来の演算方式が相回転ステップの後に適用される。最後に、各合計の結果は所要の結果を得るために（１－ｊ）が乗じられる。ｊまたは（－１）は、適度な量の時間とエネルギを要求する「組織的な」動作である。好適な実施例の第２のオプションは、複素Ｕ方法と呼ばれる。それは、次の方法で各係数のブラケットを開くことにより、合計を求める。

【０２６３】

$y_1=x_1+(jx_1)+x_2-(jx_2)-x_3-(jx_3)-x_4+(jx_4)+x_5-(jx_5)-x_6+(jx_6)$
$y_2=-x_1+(jx_1)+x_2+(jx_2)+x_3+(jx_3)-x_4-(jx_4)-x_5-(jx_5)+x_6-(jx_6)y_2$
$y_3=x_1-(jx_1)-x_2-(jx_2)-x_3-(jx_3)+x_4+(jx_4)-x_5+(jx_5)+x_6+(jx_6)$

残りはU方法の適用により行われる。ｓ（１２，３）＝１２＋３＝１５であるから、複素加算の回数である場合、これはｓ（ｎ，ｒ）の上記テーブルから高々３０回の実数加算を要求する。

【０２６４】

上記の例によって基本原理を示したが、今一般的な場合の詳細な説明を与えることは適切であろう。複素行列に関しての本発明の好適な実施例を示すために、係数として使用されるセットを考慮する、すなわち、U_1＝｛１，１，ｊ，－ｊ｝およびU_2＝｛１＋ｊ，１－ｊ，－１＋ｊ，－１－ｊ｝。U_1ベクトルまたはU_1行列はU_1の中にそれらのエントリを有する。同様に、U_2ベクトルまたはU_2行列はU_2の中にそれらのエントリを有する。このような行列およびベクトルは、ワイヤレス応用において一般的である。続くものでは、U_2の番号を複素数で乗じた積は２回の実数加算を要求し、その一方でU_1番号を複素数で乗じた積は比較的少量の複雑性を含んでいることを考慮に入れるべきである。

【０２６５】

解決される第１の計算上の問題は、次のとおりである、すなわち、ｒ×ｎのU_2行列Ａおよびｎ次元の複素列の入力ベクトルｘが与えられ、積ｙ＝Ａ・ｘを計算するこが所望されると仮定する。確かにスカラ積であるケースｒ＝１の場合が、含まれる。この演算への２つの主要なアプローチが示されるであろう。適切なとき、各々が所望されるであろう。そのデータ・ベクトルが実数のとき、同じプロセスが、少しの修正で、適用可能である。

【０２６６】

最初に、本発明の好適な実施例である相回転プラスＧＥＭが導入される。

【０２６７】

Ｂ＝(1/2)（１＋ｊ）・Ａ　および　ｚ＝(1/2)（１＋ｊ）ｙ

であるとし、そのとき、Ｂはｒ×ｎのU_1－行列であり、そして、ｚ＝Ｂ・ｘである。次に、積ｚ＝Ｂ・ｘはＧＥＭによって計算される。一旦ｚが計算されれば、出力ベクトルｙは積ｙ＝（１－ｊ）・ｚによって求められる 。最初の位相回転ステップに起因する従来方法に対する利得は、２ｒ・（ｎ-１）加算の節約である。この利得はｒ＝１の場合、すなわちスカラ積の場合でさえ存在する。一層の利得はＧＥＭのアプリケーションに起因する。

【０２６８】

第２の本発明の好適な実施例は、Ｕ複素方法と呼ばれる。第１の事項は合計としてＡ、すなわちＡ＝A_1＋ｊA_2を表わすことである、ここで、A_1およびA_2はU行列である。次に、同一を考慮すること、すなわちＡ・ｘ＝A_1・ｘ＋ｊA_2・ｘを検討することである。この同一は、２ｎ複素次元の列ベクトル

【０２６９】

【数７５】

$$x^{*}=\begin{bmatrix} x \\ jx \end{bmatrix}$$

【０２７０】

を備えるｒ×２ｎのU行列A^{*}＝［A_1，A_2］の積によってA^{*}ｘを計算することができることを意味する。これは次の同一、すなわちＡ・ｘ＝A^{*}・x^{*}の中で表現される。今、積A^{*}・x^{*}はU方法によって計算されるであろう。この本発明の好適な実施例中の別の変形があり、それはｒが小さいとき合理的であり、

【０２７１】

【数７６】

$$A^{**}=\begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$

【０２７２】
とすると、これは２ｒ×ｎのU行列である。その後、積、すなわちA^{**}・ｘを計算するためにU方法を適用する。これは実際両方の積$y_1=A_1$・ｘおよび$y_2=A_2$・ｘを計算する。そのプロセスは合計、すなわち$y=y_1+y_2$で終了する。
【０２７３】
上記の問題に関する変化が、ＣＤＭＡでのＰＮ相関器のトップリッツ行列表現を含むいくつかの応用で発生するかもしれない。この設定では、行列はまた０のエントリを有してもよい。従って、$U'_1=\{0, 1, -1, j, -j\}$および$U'_2=\{0, 1+j, 1-j, -1+j, -1-j\}$とする。$U'_1$ベクトルまたは$U'_1$行列は$U'_1$中にそれらのエントリを有する。同様に、$U'_2$ベクトルまたは$U'_2$行列は$U'_2$中にそれらのエントリを有する。ｒ×ｎの$U'_2$行列Ａが与えられ、ｎ次元の複素列入力ベクトルｘが与えられる。ゴールは積$y=A\cdot x$を計算することである。確かにスカラ積であるｒ＝１のケースの場合含まれる。
【０２７４】
本発明の好適な実施例である相回転プラスＧＥＭが最初に議論されるであろう。
【０２７５】
$B=(1/2)(1+j)\cdot A$　および　$z=(1/2)(1+j)y$
とすると、そのとき、Ｂはｒ×ｎのU'_1行列および$z=B\cdot x$となる。次に、積$z=B\cdot x$がＧＥＭの応用によって計算され、または次元が低いとき、より従来の方式によって計算されるであろう。最後に、一旦ｚが計算されれば、出力ベクトルｙは、積、すなわち$y=(1-j)\cdot z$によって求められるであろう。
【０２７６】
別の本発明の好適な実施例によれば、Ａが、まず合計、すなわち、$A=A_1+jA_2$として表わされる、ここでA_1とA_2は（０，１，－１）行列、可能ならトップリッツである。そのとき等式、すなわち、$A_1\cdot x+jA_2\cdot x$積$A\cdot x$によって、$A\cdot x=$は、ｒ×２ｎの（０，１，－１）行列$A^*=[A_1, A_2]$を２ｎ複素次元の列ベクトル、すなわち
【０２７７】
【数７７】

$$x^*=\begin{bmatrix} x \\ jx \end{bmatrix}$$

【０２７８】
で乗じた積によって計算され得る。結局、積$A^*\cdot x^*$は、トップリッツによって、または発明のより一般的な（０，１，－１）の特徴によって計算されるであろう。
【０２７９】
ｒが小さいとき、その非トップリッツのカウンターパートに反映して、本発明の別の（選択的な）好適な実施例は、効率的であり、
【０２８０】
【数７８】

$$A^{**}=\begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$

【０２８１】
とすると、これは２ｒ×ｎ（０，１，－１）行列となる。次に、積、すなわちA^{**}・ｘを計算するために、本発明の（０，１，－１）の特徴を適用する。これは実際に積$y_1=A_1$・ｘおよび$y_2=A_2$・ｘを計算する。最後に、このプロセスは、合計、すなわち$y=y_1+j\cdot y_2$で終わる。
【０２８２】

さて、複素数の章は、一般的な複素ｒ×ｎ行列であるＡ∈C^{rxn}を実数または複素数のｎ次元ベクトルｘで乗じた積を計算する方法で終了するであろう。本発明のある好適な実施例によれば、最初にＡを合計、すなわちＡ＝A_1＋ｊA_2として表わす、ここでA_1とA_2は実数の行列である。次に等式、Ａ・ｘ＝A_1・ｘ＋ｊA_2・ｘによって、Ａ・ｘ＝A^*・x^*であるから、Ａ・ｘがｒ×２ｎの実数行列A^*＝［A_1，A_2］を２ｎ次元の列ベクトル

【０２８３】

【数７９】

$$x^*=\begin{bmatrix} x \\ jx \end{bmatrix}$$

【０２８４】

で乗じた積によって計算することができる。最後に、積A^*・x^*は実数の行列法によって計算されるであろう。

【０２８５】

本発明の別の（選択的な）好適な実施例に従って、

【０２８６】

【数８０】

$$A^{**}=\begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$

【０２８７】

とする。これは２ｒ×ｎの実数の行列である。次に、積、すなわち、A^{**}・ｘを計算するために実数の行列法を適用する。これは積y_1＝A_1・ｘおよびy_2＝A_2・ｘを計算することである。最後に、そのプロセスは、合計、すなわちｙ＝y_1＋ｊ・y_2で終わる。

【０２８８】

最後に、U_2係数を備えたトップリッツ行列をベクトルで乗じた積は、上記のトップリッツ技法の応用によって行われる。

トップリッツ行列、（０，１，－１）行列および複素行列技法の産業上の応用例

ＩＳ－９５　サーチャ(searcher)：

ＩＳ－９５　ＣＤＭＡシステムによって、小さな地理的な場所における多くの異なる基地局が、データをモバイル受信機に送信するために、スペクトラムの同じセグメントを同時に使用することができる。異なる基地局からのデータを区別するための方法は、送信データを拡散するために使用されるＰＮシーケンスによるものである。各基地局は、ＰＮシーケンスの異なった過程で送信する。モバイル受信機におけるサーチャ機構の作業は、それらのＰＮシーケンスと調整することにより周囲の基地局によって送信された異なるパイロット信号を識別することである。それはまた、同じ基地局から到着するいくつかの多重経路伝播の信号（エコーを意味する）を区別するために適用される。同様のプロセスは、初期の同期手順に適用される。

【０２８９】

サーチャは、ローカルに生成されたＰＮシーケンスをもって、各仮説(hypothese)に関して、受信信号の偏相関(partial correlation)によって、多くの仮説をテストすることを要求される。その後、シーケンスは各仮説のためにシフトされ、また、相関が信号エレメント（チップ）の固定番号に対して実行される。通常、サーチャは、与えられたウィンドウ内で仮説のすべてを検索することを要求され、一回毎に、シーケンスが１ずつシフトされる。サーチャは、行が上記言及したシフトされるＰＮシーケンスからなる行列Ａの構築により、ＤＳ－ＣＤＭＡシステムで実行され得る。その後、その検索結果はベクトルｙ＝Ａ・ｘの中で与えられるが、ここでｘは単一のチップ期間でサンプリングされた到来信号を示すベクトルである。本発明の好適な実施例によれば、効率的な線形変換のための上記

言及した創作性のあるアルゴリズムは、資源の消費を削減したベクトルｙを得るために実行される。本発明の多くの応用は、さらに、広帯域ＣＤＭＡの提案された標準規格のサーチャに有用である。

多重（マルチ）積

本発明の別の好適な実施例は、ベクトル積によるＵ行列の部分和が望まれる状況と関係する。異なる速度（拡散因子：spreading factor）を有するいくつかのコードが同時にテストされるとき、これはＣＤＭＡ通信への応用において生じる。この実施例の研究は、既に本発明の前状況における本質的な事項を実行した者にとってより有益である。それは、豊富な詳細がなくてもこのやや複雑な方法についての最初の概念を与える次の実例で紹介されるであろう。しかしながら、合理的なサイズの例が、この実施例のすべての状況について記述することはできない。本発明の発明の要約、項目６をさらに参照してもよい。

【０２９０】

例

次の５×８のＵ行列

【０２９１】

【数８１】

$$A = \begin{bmatrix} 1 & 1 & -1 & 1 & 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 \\ -1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 \end{bmatrix}$$

【０２９２】

および、８次元の入力ベクトルを検討する。

【０２９３】

【数８２】

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix}$$

【０２９４】

ライン１と２の拡散因子は２、およびライン３と４の拡散因子は４、およびライン５の拡散因子は８であるとする（多重積の用語において）。最初の２行では２つ毎の連続したエレメントが合計され、第３および第４行目においては各４つの連続したエレメントが合計され、第５行目では、全ラインが合計される。規約は、拡散因子は減少することはなく、すなわち、各ラインの拡散因子は前のラインの拡散因子に等しいかそれより大きいということである。この用語は後に正確に定義されるであろう。

【０２９５】

次の合計が計算されることが上記から理解されるであろう。

【０２９６】

$x_1 + x_2$，　$-x_3 + x_4$，　$x_5 - x_6$，　$-x_7 - x_8$

$-x_1+x_2$, $-x_3-x_4$, x_5+x_6, $-x_7+x_8$
$x_1-x_2+x_3+x_4$, $x_5-x_6+x_7-x_8$
$x_1+x_2-x_3+x_4$, $-x_5-x_6+x_7-x_8$
$-x_1-x_2+x_3+x_4-x_5+x_6-x_7-x_8$
最初に４つの４×２のU行列への分割があり、そこで水平の次元が最低の拡散因子に反映する。その後、この新しい側面から加算がどのように削減されるかを示すために、極めて基礎的なレベルで、U方法が適用される。このように、同値のラインを消去することに適用して、２つの加算だけが各ラインのすべての最初の合計を計算するために要求される。

【０２９７】

x_1+x_2
$-x_1+x_2$
x_1-x_2
x_1+x_2
$-x_1-x_2$
同様に、２つの加算のみが各ラインの第２の合計に必要となる。

【０２９８】

$-x_3+x_4$
$-x_3-x_4$
x_3+x_4
$-x_3+x_4$
x_3+x_4
など。従って、合計８回の加算がこの部分のために必要である。さらに４回の加算がライン３および４の求められた合計を計算するために必要であり、さらに４回の追加の加算がライン５の求められた合計を計算するために要求される。従って、合計１６回の加算が本発明の好適な実施例を適用することにより要求された。先行技術である従来の力まかせの方法では、同じ作業をするために、２８回の加算を要求していたであろう。

【０２９９】

本発明の現在の状況における環境はU行列を含み、それは上記の実例でのように各ラインで、等しい間隔のサブ合計が必要な場合、大きな次元であってもよい。入力ベクトルは実数かまたは複素数である。その行列は、ラインによっていくつかのサブ行列へサブ分割され、各ラインは上記の実例中に別々にかつ独立して使用される方法によって計算される。加算を削減しそれにより複雑性を低減する点から、ほぼ最良のサブ分割を見つける方法がこの実施例へ統合される。そのためのツールは、ダイナミックなプログラミングに基づく追加のプロセッサまたは装置であり、それはテーブル、境界およびU方法の回帰的な方程式ｓ（ｎ，ｒ）を使用して様々なサブ分割を分析する。非常に正確な構成がこの実施例の開発に必要である。いくつかの新しい定義は予備資料として必要であろう。

【０３００】

ｖ＝（v_1，v_2，．．．．．．．．．，v_n）にベクトルにすると、ｐはｎを分割する正の整数である（要するに：ｐ｜ｎ）。ｖ［ｐ］を、長さｐのセクションへｖを細分することにより形成される、ベクトルのベクトルであると定義する。したがって、

$v[p]=((v_1,v_2,\ldots\ldots\ldots,v_p),(v_{p+1},v_{p+2},\ldots\ldots\ldots,v_{2p}),\ldots\ldots\ldots,$
$(v_{n-p+1},v_{n-p+2},\ldots\ldots\ldots,v_n))$

セクションを次のように表示する。

【０３０１】

$v[p,k]=(v_{(k-1)p},v_2,\ldots\ldots\ldots,v_{kp})$ $1\leqq k\leqq n/p$に対して

文中の整数ｐは拡散因子(spreading factor)と呼ばれる。マルチベクトルは行列の構造と同種の構造である認識する。次の項目、すなわちマルチベクトルのマルチスカラ積は、行列積と通常のスカラ積との間のクロス（中間物）である。２つのｎ次元ベクトルをｖ＝（v_1，v_2，．．．．．．．．．，v_n）およびｗ＝（w_1，w_2，．．．．．．．．．，w_n）とすると、ｖおよびｗのｐ－マルチスカラ積は次のように定義される。

【０３０２】
v・w[p]=(v[p,1]・w[p,1],v[p,2]・w[p,2],.....,v[p,n/p]・w[p,n/p])
ここで、内部積(internal products)、ｖ［ｐ，１］・ｗ［ｐ，１］，ｖ［ｐ，２］・ｗ［ｐ，２］，.....は、通常のスカラ積である。この積の結果はｎ／ｐ次元のベクトルであることに注意すること。
【０３０３】
ＡをラインA_1，.....，A_rを備えたｒ×ｎ行列であり、ｐ＝（p_1，p_2，.....，p_r）を正の整数ベクトルであるとする。すべての１≦ｉ≦ｒに対し、p_iがｎを分割するならば、ｐはｎを分割すると言えるであろう。これはｐ｜ｎと表示される。今そのｐ｜ｎを仮定する。ｎ次元のベクトルｘをとり、ベクトルのベクトルである、Ａ掛けるｘのｐ－マルチ積をＡ・ｘ［ｐ］と表示して、次のように定義する。
【０３０４】
A・x[p]=(A_1・x[p_1],A_2・x[p_2],.....,A_r・x[p_r])
本発明の最新の実施例は、今後記述される構成中の積の演算を改善するであろう。
マルチ積システム
マルチ積システム（または短くしてＭＰシステム）は、パラメータとして整数ｎ，ｒおよび正の整数ベクトルｐ＝（p_1，p_2，.....，p_r）を含む設定である、すなわち、
$p_1|p_2$,$p_2|p_3$,$p_3|p_4$,.....,$p_{r-1}|p_r$,および$p_r|n$
整数p_1，p_2，.....，p_rは、システムの拡散因子と呼ばれる。ＭＰシステムのパラメータは次の方法で書かれるであろう。
【０３０５】
Ｐ＝（ｒ，ｎ，ｐ）
これらのパラメータに今ｒ×ｎのＵ行列Ａおよびｎ次元の実数のベクトルｘを付す。目標は効率的に積Ａ・ｘ［ｐ］を計算することである。全ＭＰシステムは次の方法で書かれるであろう。
【０３０６】
Ｓ＝（ｒ，ｎ，ｐ，Ａ，ｘ）
さらに、整数p_1，p_2，.....，p_r，ｎがすべて２の累乗であるとき、そのとき、そのシステムは２進多重積システムまたは短くして、ＢＭＰシステムと呼ばれる。
Ｍ－１方法
本発明のこの特徴は、ＭＰシステムへのＵ方法のストレートは適応である。それは上記の実例によって表わされる。実際には、後で記述されるほぼ最適のサブ区分の後に、それは、ＭＰシステムのサブシステムに通常適用されるであろう。
【０３０７】
ＭＰシステムＳ＝（ｒ，ｎ，ｐ，Ａ，ｘ）を与える。この方法は、最小の拡散因子p_1に関してサブ行列への行列の水平分割に基づき、サブ行列はそれぞれ幅p_1である。それは、行列ＡのＵ係数を備えたベクトルｘの最初のp_1実数の合計を計算することにより始まる。これは、Ｕ－２進方法によってすべてのラインで同時に行われる。その後、それは次のp_1列へ進み、同じプロセスを行なう。すべてのｎ変数が使い尽くされるまで、それはこのように継続する。次に、p_i＞p_1であるラインの各々へ進み、ありふれた方法で合計プロセスを完了する。
【０３０８】
各サブ行列上のＵ方法の最初のステップは、他のラインを生のまたは反転のコピーであるラインを走査することである。例えば、p_1＝２の場合、そのとき、高々２つの加算がｒにかかわらず各セクションで必要とされる。一般に、わずか2^{p1-1}のラインが、あらゆるセクション中のＵ－２進方法によって考慮される。さらに、このテキストによって意図する１つのアプリケーションは、Ａがアダマールである場合である。この場合、各セクション中にわずかp_1個の非等価ラインがある。したがって、どれだけの非等価ラインが各サブ行列に現われることがあるかについての境界を含む別のソースが挿入される。それは、近い将来行列のタイプの結果であるｚ（テーブルとして格納される）によって表示される

機能に含まれるであろう。そのパラメータはp_1とｒであり、それはｚ（p_1，ｒ）によって書かれる。例えば、Ａがアダマールであるとき、ｚ（４，６）＝４，ｚ（８，５）＝５であり、Ａが一般的なＵ行列であるとき、ｚ（４，４０）＝８である。それは常にｚ（p_1，ｒ）≦ｍｉｎ｛2^{p1-1}，ｒ｝を維持し、Ａがアダマールである場合、ｚ（p_1，ｒ）≦ｍｉｎ｛p_1，ｒ｝である。

【０３０９】

Ｍ－ｌ方法の複雑性の演算は、上記のＵ－２進方法の記述で現われるテーブルおよびｓ（ｒ）の回帰方程式に基づくであろう。すべての正の整数ｙに対して、ｓ’（ｙ）＝ｓ（ｙ）＋ｙとする。さらにｓ’（ｙ）＜$2^{y-1}+2^{y/2}+1$であることを想起し、そしてこの不等式はややきつい。これは、次の方程式中の複雑性の程度に直覚的な視点を与える。Ｍ－ｌ方法によって行われる加算の数は、次の式で境界付けられる。

$C(n,r,\underline{p},z)=(n/p_1)(p_1+s(z(p_1,r)))+(n/p_2)(p_2/p_1-1)+(n/p_3)(p_3/p_1-1)$
$+.....+(n/p_r)(p_r/p_1-1)$
$=n\cdot(1+(s(z(p_1,r))+r-1)/p_1-(1/p_2+1/p_3+.....+(1/p_r))$
$=n\cdot(1+(s'(z(p_1,r))/p_1-(1/p_1+1/p_2+1/p_3+.....+(1/p_r))$

いくつかのつまらない点があるが、しかし、この方程式の重要な例に注意を要する。最初は、行列が１ラインを有する場合である、すなわち、ｒ＝１のとき、

Ｃ（ｎ，ｒ，ｐ）＝ｎ－ｎ／p_1

第２は、一定の拡散因子の場合である、すなわち、$p_1=p_2=.....=p_r$のとき、

Ｃ（ｎ，ｒ，$\underline{p}$，ｚ）＝ｎ・（１＋ｓ（ｚ（p_1，ｒ））／p_1）

明らかに、Ｍ－１方法は、ｒがp_1に比べて大きいとき、決して効率的でない。より賢い方法のための踏み石がそれを基礎として開発されるであろう。このより強力な方法は、行列を水平に細分しＭ－ｌ方法を各サブ行列に別々に適用することによりうまくいく。より少ない計算でより少ない加算総数をもたらすサブ区分の構造を見つけるために、上記方程式の次のより短いバージョンを有することが有用であろう。そこで、次のように定義する。

【０３１０】

C^*（ｎ，ｒ，$\underline{p}$，ｚ）＝１＋ｓ’（ｚ（p_1，ｒ））／p_1

マルチシステムのサブ区分およびマルチベクトルについての一般化された概念

次に、構成は、行列を水平ライン上に細分するように命じるために形成される。この命令は、細分するベクトル$\underline{r}$によって表わされるであろう。その結果は、オリジナルの問題を同じ幅ｎの少数のサブ問題に分解し、各々は上記のＭ－ｌ方法によって解決されるであろう。サブ区分は、効率を最大限にするために後で発明性のあるアルゴリズムに添付されるであろう。拡散因子のｒ次元の整数ベクトル、$\underline{p}$＝（p_1，p_2，.....，p_r）、および$r\times n$のＵ行列を次式とする。

【０３１１】

【数８３】

$$A=\begin{bmatrix} a_{11} & a_{12} & \cdot & \cdot & \cdot & \cdot & a_{1n} \\ a_{21} & a_{22} & \cdot & \cdot & \cdot & \cdot & a_{2n} \\ \cdot & & & & & & \\ \cdot & & & & & & \\ \cdot & & & & & & \\ a_{r1} & & & & & & a_{rn} \end{bmatrix}$$

【０３１２】

そして、整数ｋ，ｍを、１≦ｋ≦ｍ≦ｒとする。ベクトルｐのセクションを最初に定義する。

【０３１３】

p（k，m）＝（p_k，.....，p_m）
それは、単にkからmの指標を備えたコンポーネントをとる。行列Aのセクションを次のように定義する。
【０３１４】
【数８４】

$$A(k,m)=\begin{bmatrix} a_{k1} & a_{k2} & \cdot & \cdot & \cdot & \cdot & a_{kn} \\ a_{k+1,1} & a_{k+1,2} & \cdot & \cdot & \cdot & \cdot & a_{k+1,n} \\ \cdot & & & & & & \\ \cdot & & & & & & \\ \cdot & & & & & & \\ a_{m1} & & & & & & a_{mm} \end{bmatrix}$$

【０３１５】
同様に、それはkからmの指標を備えたラインを有することを意味する。
次に、整数ベクトルr＝（r（1），.....，r（t＋1））が、
k＝r（1）＜r（2）＜.....＜r（t）＜r（t＋1）＝m＋1
を満たすことを考慮し、それはセクションに分割する道具となる。まず、rは、次の方法で、pのサブ区分をベクトルp［r］のベクトルへ作成するツールである。
$p[r]=(p_{r(1)},.....,p_{r(2)-1}),(p_{r(2)},.....,p_{r(3)-1}),.....,(p_{r(t)},p_2,.....,p_{r(t+1)-1}))$
サブ・ベクトルは次のように表示される。
【０３１６】
p［r，1］＝（$p_{r(1)}$，.....，$p_{r(2)-1}$）
p［r，2］＝（$p_{r(2)}$，.....，$p_{r(3)-1}$）
・
・
・
p［r，t］＝（$p_{r(t)}$，.....，$p_{r(t+1)-1}$）
行列Aの行は、次の方法で細分するベクトルrに従って同様に細分される。
すべての整数1≦q≦tに対して、
A［r，q］＝（a_{ij}：r（q）≦i＜r（q＋1），1≦j≦n）
与えられたサブ区分のためのM方法
このセクションは、メカニズムの形成に主要なステップがあり、それはこの実施例で中心となる低い複雑性のサブ区分であるメカニズムを見つける。行列におけるラインのサブ区分を与え、各サブ行列上でM－l方法を別々に行なう。目標は加算の総数を評価し、その結果より少数の加算のサブ区分が次のステージで見つかるであろう。MPシステムS＝（r，n，p＝（p_1，p_2，.....，p_r），A，x，z）およびサブ区分ベクトルr＝（r（1），.....，r（t））を固定する、ここで1≦k＝r（1）＜r（2）＜.....＜r（t）＜r（t＋1）＝m＋1≦r＋1である。
1≦q≦tに対して、S_qMPサブシステムは次式によって与えられる。
【０３１７】
$S_q=(r(q+1)-r,n,p[r,q],A[r,q],x,z)$
すべてのサブシステムの加算総数は次のとおりである。
【０３１８】
$C(n,r,r,p,z)=\Sigma_{1\leqq q\leqq t}\ C(n,r(q+1)-r(q),p[r,q],z)$
$=n\cdot(\Sigma_{1\leqq q\leqq t}\ s'(z(p_{r(q)},(r(q+1)-r(q)))/p_{r(q)}-(1/p_k+.....+1/p_m)+t)$
次の目標は効率的な方法を開発することで、それはこの事項を最小にするサブ区分を見つける。各ステージで反復する付加的なサブ事項1／p_k＋.....＋1／p_mおよびn

要因を見積もらずに、計算を行なうほうが効率的であろう。
したがって、次のように定義する。
【０３１９】
$C^*(n,r,\underline{r},p,z)=\Sigma_{1\leqq q\leqq t}\ s'(z(p_{r(q)},(r(q+1)-r(q)))/p_{r(q)}+t$
ほぼ最適なサブ区分
このステージでは、上記の項Ｃ（ｎ，ｒ，ｒ，ｐ，ｚ）によって与えられる加算量を最小限にするサブ区分の特性を検討する。最初に、加算数に対する抽象的な公式を与えるが、ここでは、サブ区分がオリジナルの問題の与えられた部分区間上で拘束される最小の最悪のケースを有する。
【０３２０】
そのとき、１≦ｋ≦ｍ≦ｎであるとして、ＭＰシステムＳ＝（ｒ，ｎ，ｐ＝（p_1，p_2，.....，p_r），Ａ，ｘ，ｚ）および整数ｋ，ｍを検討する。次式を定義する。
【０３２１】
h(k,m)=min {C(n,m-k+1,r,p(k,m),z):for r=(r(1),.....,r(t+1))
ここで k=r(1)<r(2)<.....<r(t)<r(t+1)=m+1}
$h^*(k,m)$=min {C^*(n,m-k+1,r,p(k,m),z):for r=(r(1),.....,r(t+1))
ここで k=r(1)<r(2)<.....<r(t)<r(t+1)=m+1}
それは次式を保持する。
【０３２２】
$h(k,m)=n\cdot(h^*(k,m)+(1/p_k+.....+1/p_m))$
次の循環公式を保持する。
【０３２３】
h(k,m)=min {C(n,m-k+1,p(k,m),z),min {h(k,q-1)+h(q,m):
for all k<q≦m} }
ここで　min(empty set)=infinity
$h^*(k,m)$=min {C^*(n,m-k+1,p(k,m),z) min {$h^*(k,q-1)+h^*(q,m)$]:
for all k<q≦m} }
さて、これらの表現は次のダイナミックなコードによって使用されるが、そのサブ構造はｋとｍの間のインターバルにある。このコードh^*の複雑性を低減することは、最良のサブ構造を発見するためにルート上のｈを交換する。これは最終結果に影響はない。最適のサブ区分の第１歩を見つけるために、我々はすべてのｋおよびｍに対する、式ｈ（ｋ，ｑ－１）＋ｈ（ｑ，ｋ）を最小にするｑを計算する、ここで、１≦ｋ≦ｍ≦ｎであり、それはしきh^*（ｋ，ｑ－１）＋*ｈ（ｑ，ｋ）を最小にするのと同じｑである。従って、次を定義する。
【０３２４】
ｑ（ｋ，ｍ）＝ｋ　　ここでh^*（ｋ，ｍ）＝C^*（ｎ，ｒ，ｐ（ｋ，ｍ），ｚ）
そうでなければ、
ｑ（ｋ，ｍ）＝最小のｑ，ここで、ｋ＜ｑ≦ｍおよびh^*（ｋ，ｍ）＝h^*（ｋ，ｑ－１）＋h^*（ｑ，ｋ）
今、最適なサブ区分コードを形成することができる。
最適なサブ区分を見つけるダイナミックなプログラムの創造的なコード
次のコードは、データとしてＭＰシステムＰ＝（ｒ，ｎ，ｐ，ｚ）のパラメータを受け取り、出力としてＭ－方法が最適に達成するサブ区分ｒを生成する。加えて、それは、さらに最適なＭ－方法の複雑性(complexity)を返す。
【０３２５】
それを効率的に実行するために、ｓ’（ｒ）のテーブルを前もって計算し格納する必要がある。これは、次の回帰公式を使用して、このテーブルを計算する、高速コードによって行われる。
【０３２６】
$s'(r)=2^{r-1}+s'(r(1))+s'(r(2))$

最適なサブ区分テーブル（ｎ，ｒ，ｐ，ｚ）

```
for b going from 0 to r-1 do
      for k going from 1 to r-b do
      m:=k+b
      h*(k, m) <----- C*(n,m-k+1, p(k, m), z)
      q(k, m) <----- k
          for q going from k+1 to m do
          d <----- h*(k, q-1)+h*(q, m)
              if d < h* (k, m) then
              h*(k, m)<-----d and
              q(k, m) <-----q
return the tables h* and q.
```

今残されていることは、この手続きによって構築されたテーブルｑ（ｋ，ｍ）から最適なサブ区分ベクトルを得ることである。これは最適のベクトルｒのコンポーネントを含むセットＲを作成する次のコードによって行われる。

最適サブ区分ベクトル（ｎ，ｒ，ｐ）

```
Set: R:=empty set and k:=1 m:=r
      Find Vector (k,m):
      if q(k,m) > k then
      put q:=q(k,m)
      add q to the set R
      Find Vector (q,m)
      Find Vector (k,q-1)
Return the set R.
```

さて、最適のサブ区分ｒが見つかるので、Ｍ方法はこのサブ区分上で走るであろう。

・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・

インプリメンテーション

本発明の次の好ましい実施例は、上記方法の詳細な実行となる創造性のあるアルゴリズムを提供する。それらは、上記方法を行なうのに必要なメモリとエネルギ資源の削減を可能にする。それらは、上記方法を増強し、装置を構築するのに必要な命令を提供するのに２つのゴールを有する。これらを実行する際に強調すべき１つの仮定は、変換行列が総括的で、エントリの与えられたセットを有する行列のセットから任意に選ばれるように考慮されることができるということである。もう１つは、行の数が列の数と比較したとき、十分に少ないということであり、それにより上述した基本の境界

行の数＜ｌｏｇ（列の数）

は、満たされる。従って、等価ラインをチェックするような他の状況において適切であるステップのいくつかは、ここでは余分である。

【０３２７】

後述のテキストに記述されたマッピングは、１つの場所から次の場所へデータのフローに手順を開く。各場所は、所定の反復で、行列の列に対応する２進のアドレスで割り当てられる。Ｕ行列の場合には、列における（－１）のコンポーネントは、アドレスにおける１に対応し、同様に列における１のコンポーネントはアドレスにおける０に対応する。同様の対応が、U_1行列に対して定義される。

【０３２８】

説明されるインプリメンテーションは、到来するx_j信号が、それらの対応する列によって決定された宛先を前もってセットするために加えられる第１ステップを含み、ここで各x_jは記号および／または２の累乗が乗じられる。進行する反復中において、列が本発明に従って分割される場合は常に、２つの各の分割に割り当てられたアドレスは、その所定の列アドレスへより小さいか等しい。これは、メモリのある場所に格納されたすべての情報を、処理され失われる前にその宛先へ送ることができる。加えて、それが所定の列を分

割した半分の１つが０であると考えれば、この列を表わすアドレスは次の反復に使用されるであろう。これらの特性は、データの移動を最小限にするために設定されるデータ・フロー・マップの斬新な構築によって遂行される。各反復で、アドレスの最大の数は手をつけないままで、その現在の内容が次の反復によって使用することができるものをすべて含める。

【０３２９】

次のインプリメンテーションのメカニズムは、上記において記述された本発明の好適な実施例であるＧＥＭを考慮してより一層理解される。数１を含む数の有限集合Ｓが考慮され、この文章中では、それがそのコンポーネントが集合Ｓに属し、正規化されたｒ次元の列ベクトルのすべての可能なゼロでない構成から成る場合、ｒラインの行列は完全なＳ－ｒ－行列と名付けられ、その構成がそれぞれ一度正確に現われ、その正規化規約は最低のゼロでない要素が１であるべきということである。

次のより低い次元の例を観察する。

行列

【０３３０】

【数８５】

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

【０３３１】

は、完全な｛０，１｝－２－行列である。

行列

【０３３２】

【数８６】

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$$

【０３３３】

は、完全なＵ－２－行列である。

行列

【０３３４】

【数８７】

$$\begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

【０３３５】

は、完全なＵ－３－行列である。

行列

【０３３６】

【数８８】

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

【０３３７】

は、完全な｛０，１｝－３－行列である。
行列
【０３３８】
【数８９】

$$\begin{bmatrix} 1 & -1 & j & -j \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

【０３３９】
は、完全な｛１，－１，ｊ，－ｊ｝－２－行列である。
【０３４０】
次のインプリメンテーションは、列の数に対して行の数が十分に少ないという仮定に基づき、その結果すべての可能な構成の大部分は、変換行列の列に現われる。従って、すべての進行中のインプリメンテーションで作成された最初の反復は、演算を完全なＳ－ｒ－行列と修正済のベクトルとの積にする。従って、さら前進した反復は、より低い次元の完全なＳ行列とそれぞれの修正済のベクトルとの積を計算する。
【０３４１】
列が上記例の行列に現われる規律正しい方法は、これらのインプリメンテーションのアドレス・セッティングに別の特徴を反映する。それは、列を｜Ｓ｜－ベースのアドレスに翻訳する一貫した規則に基づいたアドレスであり、ここで、ＭＳＢがベクトルのボトム・コンポーネントであり、ＬＳＢがトップである。
【０３４２】
基本的な結果は、第１の反復の後に、一定の（そしてむしろ明白な）アドレス番号付けをする唯一の一定の完全な変換行列があるということである。この一定の行列は、Ｓおよびｒのみに依存し、初期変換行列には依存しない。これは、残りのプロセスの中で等質性に通じるが、その結果最初のステージ後に一般的かつ特定の変換行列から独立したデータ・フロー構造の創出を可能にする。これは、ハードウェア・インプリメンテーションに、およびこの章の主要な目標であるが、本発明に基づいた装置を構成する際に非常に役立つ。
【０３４３】
これらのインプリメンテーションの追加・個別的な側面は、要求された読取りおよび書込みメモリの割当て、および適切な変換行列のための加算数の削減である。従って、そのプロセスの実行のために必要とされるエネルギ、および装置のコストが削減される。いくつかの入力ｘベクトルと同じ初期変換行列を処理するとき、次の各コードの効率が強調されることが認められるべきである。
【０３４４】
　最後に、適切な展望を持つために、次のインプリメンテーションは、それらが本発明の多くの可能な効率的なインプリメンテーションの内のわずかな例であることを理解して読まれるべきである。

例１：０－１－２進の行列

　次のステップのシーケンスは、本発明の０－１－行列の局面からみたインプリメンテーションについて記述する。データは、ｒ×ｎの０－１－行列　Ａ＝（a_{ij}：０≦ｉ＜ｒ，０≦ｊ＜ｎ）、および実数または複素数入力ベクトルｘ＝（x_0，．．．．．．．．．，x_{n-1}）から成る。行列Ａの列は、v_0，．．．．．．．．．，v_{n-1}によって表示される。このステップのシーケンスは、積ｙ＝（y_0，．．．．．．．．．，y_{r-1}）^{T}＝Ａ・ｘを計算する。すべての与えられたｋ－反復で、各場所は、ベクトルｘに依存して、１つの実数または複素数を含んでいる。割り付けられた読取りおよび書込みメモリは、各反復でプロセスに関係する列を表わして、１から2^r－１までラベル付けされた2^r－１のアドレスを含む。次の定義は、本発明の好適な実施例の記述の一部である。
【０３４５】
定義

１）全てのk≧0，
j≧1に対して、
【０３４６】
【数９０】

$$l(k,j) = \lceil (j-1)\cdot(r/2^k) \rceil$$

【０３４７】
を定義する。
２）全ての０≦ｍ＜ｒに対して、$Y_m = 2^m$を定義する。これらのアドレスは、Y_0がy_0のアドレスであり、Y_1がy_1のアドレスとなる場合のように、プロセスの終わりに、出力ベクトルｙ＝（y_0，．．．．．．．．．，y_{r-1}）^{T}のコンポーネントを含むであろう。
３）ｋ≧０およびｊ≧１に対して、以下のように各場所から次の場所へのデータの移動を制御する関数F_{kj}，G_{kj}を定義する。まず、ｌ＝ｌ（ｋ，ｊ），ｍ＝ｌ（ｋ＋ｌ，２ｊ），ｈ＝ｌ（ｋ，ｊ＋１）－１を置き、各整数ｖ≧０に対して、
【０３４８】
【数９１】

$$F_{k,j}(v) = 2^m \cdot \lfloor v/2^m \rfloor$$

$$G_{k,j}(v) = v \bmod 2^m$$

【０３４９】
を定義する。
４）すべてのベクトルｖ＝（v_0，．．．．．．．．．，v_{r-1}）∈[０，１]^{r}に対して、
σ（ｖ）＝$\Sigma_{0 \leqq j < r} 2^j v_j$
を定義する。
【０３５０】
コード
１．初期化：１から２^r－１まですべてのアドレスに０を入れること。
２．第１段階：ｊが０からｎ-１へ進み
もし$v_j \neq 0$ なら、そのときx_j をアドレスv=σ(v_j)に加える
３．主要部分：
【０３５１】
【数９２】

for k going from 0 to $\lceil \log(r) \rceil - 1$ do
for j going from 1 to 2^k do
if $l(k,j+1)-1 > l(k,j)$ then
for p going from 1 to $2^{l(k,j+1)-l(k,j)}-1$ do
put $v = 2^{l(k,j)} \cdot p$
for (source) address v do
if $F_{k,j}(v) \neq 0$ and $F_{k,j}(v) \neq v$ then

【０３５２】
そのとき、（ソース）アドレスｖ中に存在する値を（宛先）アドレスG_{kj}（ｖ）中の値へ加え、さらに、（ソース・アドレスｖの）この値を（宛先）アドレスF_{kj}（ｖ）中の値に加える。
４．出力の獲得：
このステージでは、すべてのアドレスY_1は、すべての０≦ｉ＜ｒに対して、出力コン

ポーネントy_iの値を含む。

【０３５３】

複雑性

現在の用語における基礎的ステップは、ソースと呼ばれるメモリの１つの場所から数を読み、宛先と呼ばれるメモリの別の場所に置かれた数にそれを加えることである。

【０３５４】

上記コードに基づく装置は、Ａ・ｘの上記演算の主要部分を計算するための次の数の基礎的ステップを要求するだろう。

【０３５５】

$C^{0,1}_{r} \equiv 2^{r+1}-2r-2$

この数はｒにのみ依存する。

【０３５６】

Ａ・ｘの全演算は、次の合計の基礎的ステップ数を要求する。

【０３５７】

$C^{0,1}_{n,r} \equiv n+C^{0,1}_{r}$

チャート１：０－１－２進・行列

図１は、上記コードの主要部分を実行する装置によって使用されたメモリの内容を更新する動作を概略的に示すが、０－１－２進・行列は４つの行からなり、本発明の０－１－２進の特徴を好ましい手法でインプリメントする。各反復における各行列の列は、メモリ中のアドレスによって２進形式で表わされる。ボトム（０または１）コンポーネント（水平表示での最も右手のコンポーネント）はＭＳＢであり、また、トップ（０または１）コンポーネント（水平表示における最も左手のコンポーネント）はＬＳＢである。アドレス$Y_m=2^m$，０・ｍ・３は、Y_0がy_0のアドレスであり、Y_1がy_1のアドレスになる場合のように、そのプロセスの終わりに出力ベクトル$y=(y_0, \ldots\ldots\ldots, y_{r-1})^T$のコンポーネントを含む。矢印は、単一アドレスの内容をとり、別のアドレスの内容に加えられるためにそれを送る行為を示す。上記のコードによって示されるように、これは、反復の順序、および各反復に関係するアドレスの（増加）順序に従って行われる。

【０３５８】

例２：Ｕ行列

次のステップのシーケンスは、本発明のＵ行列の面からのインプリメンテーションについて記述する。そのデータは、$r \times n$Ｕ行列$A=(a_{ij}: 0 \leqq i < r, 0 \leqq j < n)$および入力の実数ベクトル$x=(x_0, \ldots\ldots\ldots, x_{n-1})$から成る。行列Ａの列は、$w, \ldots\ldots\ldots, w_{n-1}$によって表示されるであろう。このステップのシーケンスは、積$y=(y_0, \ldots\ldots\ldots, y_{r-1})^T=A \cdot x$を計算する。すべての与えられた反復では、各場所はベクトルｘに依存して、１つの実数または複素数を含む。割り当てられた読取りおよび書込みメモリは、０から$2^{r-1}+r-2$までラベル付けされた$2^{r-1}+r-1$個のアドレスを含む。次の定義および前例の定義は、本発明の好適な実施例の記述に必要である。

定義

１）各ｒ次元のＵベクトルに対し、$u=(u_0, \ldots\ldots\ldots, u_{r-1})$は：

$Sign(u)=u_{r-1}$

$h(u)=(u_0 \cdot u_{r-1}, \ldots\ldots\ldots, u_{r-1} \cdot u_{r-1})$

を定義する。

２）バイポーラ２進セット、$U=\{1, -1\}$およびロジック・ビットの２進セット、$B=(0, 1)$の間の一致を次式によって定義する。

【０３５９】

$(-1)'=1$

$1'=0$

従って、ｒ次元のＵベクトルに対して、$u=(u_0, \ldots\ldots\ldots, u_{r-1})$は、

$\pi(u)=\Sigma_{0 \leqq j < r} 2^j u'_j$

を定義する。

3）$Y_0=0$、およびすべての$1 \leqq m \leqq r-1$に対して$Y_m=2^{r-1}+m-1$を定義する。これらのアドレスは、Y_0がアドレスであり、Y_1がy_1のアドレスとなる場合のように、プロセスの終わりに出力ベクトル$y=(y_0, \ldots\ldots\ldots, y_{r-1})^T$のコンポーネントを含むであろう。

4）すべての$k \geqq 0$および$j \geqq 1$に対し、マップF_{kj}、G_{kj}、$Sign_{kj}$は、以下のように定義される。

$l=l(k, j)$，$m=l(k+1, 2j)$、$h=l(k, j+1)-1$とする。あらゆる整数$v \geqq 0$に対し、

【0360】

【数93】

$$Sign_{kj}(v) = 1 - 2\cdot\lfloor (v \bmod 2^m)/2^{m-1} \rfloor$$

$$F_{kj}(v) = 2^m\cdot\lfloor v/2^m \rfloor$$

$$v \bmod 2^m \qquad if \qquad Sign_{kj}(v) = 1$$

$$G_{kj}(v) = 2^m - 2^l - (v \bmod 2^m) \qquad if \qquad Sign_{kj}(v) = -1$$

【0361】

を定義する。

【0362】

コード

1．初期化：0から$2^{r-1}+r-2$までのあらゆるアドレスに0を入れる。

2．第1段階：jが0からn-1まで進行することに対し、$Sign(w_j)\cdot x_j$をアドレスπ（$h(w_j)$）に加える。

3．主要部分：

【0363】

【数94】

for k going from 0 to $\lceil log(r) \rceil - 1$ do

for j going from 1 to 2^k do

if $l(k,j+1)-1 > l(k,j)$ then

【0364】

1）（ソース）アドレス$y_{l(kj)}$に存在する値を（宛先）アドレス$Y_{l(k+1,2j)}$中の値への加える。

2）for p going from 1
to $2^{l(kj+1)-l(kj)-1}-1$ do
put $u=2^{l(kj)}\cdot p$
and for source address u do:
if $G_{kj}(u)=u$
then

ソース・アドレスuに存在する値を終点アドレス$Y_{l(k+1,2j)}$中の値へ加える。

もし、$F_{kj}(u)=u$の場合、そのとき

（ソース）アドレスuに存在する値を（宛先）アドレス$Y_{l(kj)}$中の値へ加える。

もし、$G_{kj}(u)=0$の場合、そのとき

（－1）で乗じられた（ソース）アドレスuに存在する値を（宛先）アドレス$Y_{l(kj)}$中の値に加えて、また（アドレスuの）この値をアドレス$F_{kj}(u)$中の値にさらに加える。

ほかに、$Sign_{kj}(u)$を掛けたソース・アドレスuに存在する値を終点アドレス$G_{kj}(u)$中の値に加えて、また（アドレスuの）この値をアドレス$F_{kj}(u)$中の値にさら

に加える。
４．出力の獲得：
あらゆるアドレスY_iは、すべての$0 \leqq i < r$に対して、y_iの値を含む。
【０３６５】
複雑性
ｉ）上記インプリメンテーションを備えたＵ設定中の基礎的ステップは、ソースと呼ばれるメモリの１つの場所から数を読み出し、１または－１である符号をそれに掛けて、次に、その結果を宛先と呼ばれるメモリの別の場所に置かれた数に加える。
ii）複雑性は次の事項で公式化される。
【０３６６】
【数９５】

$$\text{For} \quad 0 \leqq k \leqq \lceil \log(r) \rceil - 1 \quad \text{and for} \quad 1 \leqq j \leqq 2^k \quad \text{let:}$$

$$u_{k,j} = 2^{l(k,j+1)-l(k,j)-1}$$

$$u_k = \sum_{1 \leqq j \leqq 2^k} u_{k,j} = \sum_{1 \leqq j \leqq 2^k} 2^{l(k,j+1)-l(k,j)-1}.$$

$$\text{For} \quad k = \lceil \log(r) \rceil \quad \text{and for} \quad 1 \leqq j \leqq 2^k \quad \text{let:}$$

$$u_{k,j} = \lfloor 2^{l(k,j+1)-l(k,j)-1} \rfloor,$$

$$\text{then:}$$

$$u_k = \sum_{1 \leqq j \leqq 2^k} u_{k,j} = r.$$

【０３６７】
iii）上記コードに基づく装置は、Ａ・ｘの演算を完了するために次の数の基礎的ステップを要求するであろう。
（１）最初のステージに対して、ｎ個の基礎的ステップが必要である。
（２）すべての
【０３６８】
【数９６】

$$0 \leqq k \leqq \lceil \log(r) \rceil - 1 \quad \text{and} \quad 1 \leqq j \leqq 2^k$$

【０３６９】
に対しては、$2 \cdot u_{k,j} - u_{k+1,2j} - u_{k+1,2j-1} + 1$
個の基礎的ステップが（ｋ，ｊ）ステップで行われる。
（３）したがって、すべての
【０３７０】
【数９７】

$$0 \leqq k \leqq \lceil \log(r) \rceil - 1$$

【０３７１】
に対して、主要部分のｋ－反復は、次の数の基礎的ステップを要求する。
【０３７２】
$2^k + 2 \cdot u_k - u_{k+1}$
（４）Ａ・ｘの上記演算の主要部分は、このように次の個数の基礎的ステップを要求する。

【０３７３】
【数９８】

$$C^{u}_{r} \equiv 2u_0+u_1+u_2+.....+u_{\lceil \log(r) \rceil -1} + 2^{\lceil \log(r) \rceil} -1 - r.$$

【０３７４】
この数はｒにのみ依存する。
（５）Ａ・ｘの全演算は、このように次の合計の基礎的ステップ数を要求する。
【０３７５】
【数９９】

$$C^{u}_{n,r} \equiv n + C^{u}_{r}$$

【０３７６】
チャート２：Ｕ行列
図２は、上記コードの主要部分を実行する装置によって使用されたメモリの内容を更新する動作を概略的に示すが、Ｕ-２進・行列は４つの行からなり、本発明のＵ-２進の特徴を好ましい手法でインプリメントする。各反復における各行列の列は、メモリ中の２進アドレスによって表わされ、列中の（－１）コンポーネントはアドレスにおける１に対応し、列中の１のコンポーネントはアドレスにおける０に対応する。２進の知識が適用され、ボトム・コンポーネント（水平表示での最も右手のコンポーネント）はＭＳＢであり、また、トップ・コンポーネント（水平表示における最も左手のコンポーネント）はＬＳＢである。特別のアドレス$Y_0=0$，$Y_1=8$，$Y_2=9$，$Y_3=10$は、プロセスの終わりに出力ベクトル$y=(y_0, ..., y_3)^T$のコンポーネントを含み、ここで、Y_0はy_0のアドレスであり、Y_1はy_1のアドレスなどのようになるであろう。矢印は、１または－１である符号を掛けた１つのアドレスの内容をとり、別のアドレスの内容に加えられるためにそれを送る行為を示す。１つの矢印は符号が１であることを示し、そして２つの矢印は符号が－１であることを示す。これは、反復の順序、および各反復に関係するアドレスの（増加）順に従って行われる。
【０３７７】
例３：U_1行列
ステップの次のシーケンスは、変換行列のエントリがセット$U_1 \equiv \{1, -1, j, -j\}$に属するときに、ＧＥＭ方法のインプリメンテーションについて記述する。これは、アプリケーションにしばしば現われるＧＥＭのサブケースの１つである。データは、ｒ×ｎのU_1行列$A=(a_{ij}: 0 \leqq i < r, 0 \leqq j < n)$および入力複素ベクトル$x=(x_0, ..., x_{n-1})$から成る。行列の列は、$w_0, ..., w_{n-1}$によって表示される。このステップのシーケンスは積$y=(y_0, ..., y_{r-1})^T = A \cdot x$を計算する。あらゆる与えられた反復で、各場所はベクトルｘに依存して、１つの実数または複素数を含む。割り付けられた読取りおよび書込みメモリは、０から$4^{r-1}+r-2$までラベル付けされた$4^{r-1}+r-1$個のアドレスを含む。この例は、次の定義と同様に、前例においてセットされた定義を使用する。
【０３７８】
定義
１）次のものから与えられたセットU_1とセット｛０，１，２，３｝との間の一致（および逆の一致）を定義する。
【０３７９】
$1' = 0$, $0^* = 1$
$(-1)' = 1$, $1^* = -1$,
$j' = 2$, $2^* = j$,
$(-j)' = 3$, $3^* = -j$

従って、ｒ次元のU_1ベクトル、ｕ＝（u_0，．．．．．．．．．，u_{r-1}）に対し、次式を定義する。

【０３８０】

$\alpha(u) = \Sigma_{0 \leqq j < r} 4^j u'_j$

２）各ｒ次元のU_1ベクトル、ｕ＝（u_0，．．．．．．．．．，u_{r-1}）に対し、次式を定義する。

【０３８１】

$Sign(u) = u_{r-1}$

$g(u) = (u_0 \cdot (u_{r-1})^{-1}, \ldots\ldots\ldots, u_{r-1} \cdot (u_{r-1})^{-1})$

３）$Y_0 = 0$およびすべての$1 \leqq m < r$に対し、$Y_m = 4^{r-1} + m - 1$を定義する。これらは出力ベクトルのコンポーネントのアドレスになるであろう。

４）すべての$k \geqq 0$、$j \geqq 1$に対し、マップF_{kj}、G_{kj}が次のように定義される。

【０３８２】

$l = l(k, j)$，$m = l(k+1, 2j)$，

$h = l(k, j+1) - 1$とし、

$v = \Sigma_{0 \leqq j < r-2} 4^j \cdot v_j$によって、４つの基礎の中で表わされた、整数$v \neq 0$をとり、次式を定義する。

【０３８３】

【数１００】

$$Sign_{kj}(v) = (v_{m-1})^* = (\lfloor (v \bmod 4^m)/4^{m-1} \rfloor)^*$$

$$F_{kj}(v) = 4^m \cdot \lfloor v/4^m \rfloor$$

$$G_{kj}(v) = \sum_{l \leqq j' < m-1} 4^{j'} \cdot ((v_{j'})^* \cdot ((v_{m-1})^*)^{-1})'$$

【０３８４】

コード

１．初期化：０から$4^{r-1} + r - 2$までのあらゆるアドレスに０を入れる。

２．第１段階：　for j going from 0 to n-1 add $Sign(w_j) \cdot x_j$ to the address $\alpha(g(w_j))$

３．主要部分：

【０３８５】

【数１０１】

for k going from 0 to $\lceil \log(r) \rceil - 1$ do

　　for j going from 1 to 2^k do

　　if $l(k,j+1) - 1 > l(k,j)$ then:

【０３８６】

１）アドレス$Y_{l(k,j)}$に存在する値をアドレス$Y_{l(k+1,2j)}$中の値に加える。

２）for p going from 1 to $4^{l(kj+1)-l(kj)-1} - 1$ do

　put $u = 4^{l(kj)} \cdot p$

and for source address u do:すなわち

　$G_{kj}(u) = u$の場合

（ソース）アドレスｕに存在する値を（宛先）アドレス$Y_{l(k+1,2j)}$中の値へ加える。

ほかに、$F_{kj}(u) = u$の場合、（ソース）アドレスｕに存在する値を（宛先）アドレス$Y_{l(k,j)}$中の値へ加える。

ほかに、G_{kj}（ｕ）＝０の場合、$Sign_{kj}$（ｕ）を掛けた（ソース）アドレスｕに存在する値を（宛先）アドレス$Y_{l(k,j)}$中の値に加えて、さらにこの値（アドレスｕの）をアドレスF_{kj}（ｕ）中の値に加える。
ほかに、$Sign_{kj}$（ｕ）を掛けた（ソース）アドレスｕに存在する値を（宛先）アドレスG_{kj}（ｕ）中の値に加えて、この値（アドレスｕの）をアドレスF_{kj}（ｕ）中の値に加える。
４．出力の獲得：
あらゆるアドレスY_iは、すべての$0 \leqq i < r$に対して、y_iの値を含む。
【０３８７】
複雑性
i）上記インプリメンテーションを備えたU_1設定中の基礎的ステップは、ソースと呼ばれるメモリの１つの場所から複素数を読み出し、１または－１またはｊまたは－ｊである複素符号をそれに掛けて、次に、その結果を宛先と呼ばれるメモリの別の場所に置かれた複素数に加えることを意味する。
ii）複雑性は次の事項によって形成される。
【０３８８】
【数１０２】

$$0 \leqq k \leqq \lceil \log(r) \rceil - 1 \text{ および } 1 \leqq j \leqq 2^k \text{ に対して}$$

$$w_{k,j} = 4^{l(k,j+1)-l(k,j)-1}$$

$$w_k = \sum_{1 \leqq j \leqq 2^k} w_{kj} = \sum_{1 \leqq j \leqq 2^k} 4^{l(k,j+1)-l(k,j)-1}$$

$$k = \lceil \log(r) \rceil \text{ および } 1 \leqq j \leqq 2^k \text{ に対して}$$

$$w_{kj} = \lfloor 4^{l(k,j+1)-l(k,j)-1} \rfloor$$

$$w_k = \sum_{1 \leqq j \leqq 2^k} w_{kj} = r.$$

【０３８９】
iii）上記のコードに基づいた装置は、Ａ・ｘの演算を完了するために次の個数の基礎的ステップを要求するであろう。
（１）最初のステージに対して、ｎ個の基礎的ステップが必要である。
（２）すべての
【０３９０】
【数１０３】

$$0 \leqq k \leqq \lceil \log(r) \rceil - 1 \text{ and } 1 \leqq j \leqq 2^k$$

【０３９１】
に対しては、$2 \cdot w_{k,j} - w_{k+1,2j} - w_{k+1,2j-1} + 1$個の基礎的ステップの合計が（ｋ，ｊ）ステップで行われる。
（３）したがって、すべての
【０３９２】
【数１０４】

$$0 \leqq k \leqq \lceil \log(r) \rceil - 1$$

【０３９３】
に対して、主要部分のｋ－反復は、次の数の基礎的ステップを要求する。
【０３９４】

$2^{k}+2\cdot w_{k}-w_{k+1}$
（４）Ａ・ｘの上記演算の主要部分は、このように次の個数の基礎的ステップを要求する。
【０３９５】
【数１０５】

$$C^{U_1}_{r} \equiv +2w_0+w_1+w_2+\ldots\ldots+w_{\lceil \log(r) \rceil -1}+2^{\lceil \log(r) \rceil}-1-r.$$

【０３９６】
この数はｒにのみ依存する。
（５）Ａ・ｘの全演算は、このように次の合計の基礎的ステップ数を要求する。
【０３９７】
【数１０６】

$$C^{U_1}_{n,r} \equiv n+C^{U_1}_{r}$$

【０３９８】

例４：トップリッツ　Ｕ行列

ステップの次のシーケンスは、Ｕ係数を備えた本発明のトップリッツ行列の面からインプリメンテーションについて記述する。データは、Ｕシーケンス$t_0, \ldots\ldots\ldots, t_{n-1}$、および入力実数または複素ベクトル$x=(x_0, \ldots\ldots\ldots, x_{n+r-2})$から成る。Ｕシーケンスから、$r\times(n+r-1)$のトップリッツ行列$A\equiv(a_{ij}\equiv t_{i-j}: 0\leqq i<r, 0\leqq j\leqq n+r-2)$が形成され、ここで、すべての$k<0$、または、$k\geqq n$に対して、$t_k\equiv 0$である。これらのステップは、積$y=(y_0, \ldots\ldots\ldots, y_{n-1})=A\cdot x$を計算する。最初のステージのみがＵ行列例のステージと異なり、したがって、このステージだけを紹介することが必要である。前例のインプリメンテーションでリストされた定義は、すべてここに適用可能である。
【０３９９】
コード
１．初期化：０から$2^{r-1}+r-2$までのあらゆるアドレスに０を入れる。
２．第１段階：
１）for j going from 0 to r-2 add $(1/2)\cdot t_{n-r+j+1}\cdot x_j$ to the address
$\pi(h(t_j,t_{j-1},\ldots\ldots\ldots,t_1,t_0, t_{n-1},t_{n-2},\ldots\ldots\ldots, t_{n-r+j+1}))$
and also add $-(1/2)\cdot t_{n-r+j+1}\cdot x_j$ to the address
$\pi(h(t_j,t_{j-1},\ldots\ldots\ldots,t_1,t_0, -t_{n-1},-t_{n-2},\ldots\ldots\ldots, -t_{n-r+j+1}))$
２）for j going from r-1 to n-1 add $(1/2)\cdot t_{j-r+1}\cdot x_j$ to the address
$\pi(h(t_j,t_{j-1},\ldots\ldots\ldots,t_{j-r+1}))$
３）for j going from n to n+r-2 add $(1/2)\cdot t_{j-r+1}x_j$ to the address
$\pi(h(t_{j-n},t_{j-n-1},\ldots\ldots\ldots,t_1,t_0, t_{n-1},t_{n-2},\ldots\ldots\ldots, t_{j-r+1}))$
and also add $(1/2)\cdot t_{j-r+1}\cdot x_j$ to the address
$\pi(h(t_{j-n},t_{j-n-1},\ldots\ldots\ldots,t_1,t_0, -t_{n-1},-t_{n-2},\ldots\ldots\ldots, -t_{j-r+1}))$
３．主要部分：アルゴリズムは、Ｕ行列インプリメントのアルゴリズム中の主要部分のように進行し、その出力は同じアドレスに格納される。
【０４００】
複雑性
ｉ）上記トップリッツのインプリメンテーションでの基礎的ステップは、Ｕインプリメンテーションと同じである、すなわち、ソースと呼ばれるメモリの１つの場所から数を読み出し、１または－１である符号をそれに掛けて、次に、その結果を宛先と呼ばれるメモリの別の場所に置かれた数に加える。
ii）上記コードに基づく装置は、Ａ・ｘの演算を完了するために次の数の基礎的ステップ

を要求するであろう。
（１）最初のステージに対して、
２・（ｒ－１）＋ｎ－ｒ＋１＋２・（ｒ－１）＝ｎ＋３ｒ－３
個の基礎的ステップが必要である。
（２）Ａ・ｘのトップリッツ演算の主要部分は、ｒラインを備えたＵコードの主要部分のインプリメンテーションによって行われる。したがって、それはC^{u}_{r}個の基礎的ステップを必要とする。
（３）Ａ・ｘのトップリッツ演算全体は、このように次の回数の基礎的ステップを要求する。
【０４０１】

$$C^{T}_{n,r} \equiv n+3r-3+C^{u}_{r}$$

チャート３：トップリッツ行列

図３は、到来データを装置に使用される適切なメモリ位置へ送る動作を概略的に示すが、その装置は４つの行からなるトップリッツ行列を有して、上記コードの初期部分を実行する。この最初の部分を除いて、他のあらゆる面は、Ｕ行列インプリメンテーションおよび装置の面と同一であり、その記述がここにあてはまる。ここでまた、１つの矢印は符号が１であることを示し、そして２つの矢印は符号が－１であることを示す。
【０４０２】

例５：トップリッツ　U_1行列

本発明の次の好適な実施例は、U_1係数を備えた本発明のトップリッツ行列の面からのインプリメンテーションである。データは、U_1シーケンス$t_0, \ldots\ldots\ldots, t_{n-1}$、および入力複素ベクトル$x=(x_0, \ldots\ldots\ldots, x_{n+r-2})$から成る。$U_1$シーケンスから、ｒ×（ｎ＋ｒ－１）のトップリッツ行列$A\equiv(a_{ij}\equiv t_{i-j} : 0\leqq i<r,\ 0\leqq j\leqq n+r-2)$が形成され、ここで、すべての$k<0$、または、$k\geqq n$に対して、$t_k\equiv 0$である。以下にリストされたステップのシーケンスは、積$y=(y_0, \ldots\ldots\ldots, y_{n-1})=A\cdot x$を計算する。最初のステージのみが$U_1$行列例のステージと異なり、したがって、このステージだけを紹介することが必要である。前例でリストされたすべての定義は、ここに適用可能である。
【０４０３】
コード
１．初期化：０から$4^{r-1}+r-2$までのあらゆるアドレスに０を入れる。
２．第１段階：
１）for j going from 0 to r-2 add $(1/2)\cdot t_{nr+j+1}\cdot x_j$ to the address
$\alpha(g(t_j,t_{j-1},\ldots\ldots\ldots,t_1,t_0,\ t_{n-1},t_{n-2},\ldots\ldots\ldots,\ t_{n-r+j+1}))$
and also add $-(1/2)\cdot t_{nr+j+1}\cdot x_j$ to the address
$\alpha(g(t_j,t_{j-1},\ldots\ldots\ldots,t_1,t_0,\ -t_{n-1},-t_{n-2},\ldots\ldots\ldots,\ -t_{n-r+j+1}))$
２）for j going from r-1 to n-1 add $t_{j-r+1}\cdot x_j$ to the address
$\alpha(g(t_j,t_{j-1},\ldots\ldots\ldots,t_{j-r+1}))$
３）for j going from n to n+r-2 add $(1/2)\cdot t_{j-r+1}x_j$ to the address
$\alpha(g(t_{j-n},t_{j-n-1},\ldots\ldots\ldots,t_1,t_0,\ t_{n-1},t_{n-2},\ldots\ldots\ldots,\ t_{j-r+1}))$
and also add $(1/2)\cdot t_{j-r+1}x_j$ to the address
$\alpha(g(t_{j-n},t_{j-n-1},\ldots\ldots\ldots,t_1,t_0,\ -t_{n-1},-t_{n-2},\ldots\ldots\ldots,\ -t_{j-r+1}))$
and also add $-(1/2)\cdot t_j x_j$ to the address
$\alpha(g(t_{j-n},t_{j-n-1},\ldots\ldots\ldots,t_1,t_0,\ -t_n,-t_{n-1},\ldots\ldots\ldots,\ -t_{j-r+2}))$
３．主要部分：U_1行列アルゴリズムの主要部分として進行し、そのデータは同じ方法で受け取られる。
【０４０４】
複雑性
ｉ）上記トップリッツのインプリメンテーションでの基礎的ステップは、U_1インプリメンテーションと同じである、すなわち、ソースと呼ばれるメモリの１つの場所から複素数

を読み出し、１または－１またはｊまたは－ｊである複素符号をそれに掛けて、次に、その結果を宛先と呼ばれるメモリの別の場所に置かれた複素数に加える。
ii）上記コードに基づく装置は、Ａ・ｘの演算を完了するために次の数の基礎的ステップを要求するであろう。
（１）第１段階に対して、
２・（ｒ－１）＋ｎ－ｒ＋１＋２・（ｒ－１）＝ｎ＋３ｒ－３
個の基礎的ステップが必要である。
（２）Ａ・ｘにおけるトップリッツ演算の主要部分は、ｒラインを備えたU_1コードの主要部分のインプリメンテーションによって行われる。したがって、それは$C^{U1}{}_r$回の基礎的ステップを必要とする。
（３）Ａ・ｘのトップリッツ演算全体は、このように次の回数の基礎的ステップを要求する。
【０４０５】

$$C^{T1}{}_{n,r} \equiv n+3r-3+C^{U1}{}_r$$

例６：実数の行列、２進０－１の表現
本発明の次の好適な実施例は、行列エントリの２進０－１表現を備えた本発明の実数行列の面からみたインプリメンテーションである。データは、負でないエントリを備えたｒ×ｎ実数行列Ａ＝（a_{ij}：$0 \leqq i < r$，$0 \leqq j < n$）、および入力の実数または複素ベクトルｘ＝（x_0，．．．．．．．．．，x_{n-1}）から成る。アルゴリズムはｙ＝（y_0，．．．．．．．．．，y_{n-1}）＝Ａ・ｘを計算する。
【０４０６】
行列のエントリが、すべての$0 \leqq i < r$，$0 \leqq j < n$に対して、次の０－１の２進の表現で与えられると仮定される。
【０４０７】

$$a_{ij} = \Sigma_{-m2 \leqq k \leqq m1} t_{ijk} \cdot 2^k$$

ここで、すべての$-m_2 \leqq k \leqq m_1$に対し$t_{ijk} \in \{0, 1\}$
第１段階だけが、０－１－行列のインプリメンテーションのステージと異なっており、このステージについてだけ記述される必要がある。０－１－行列のインプリメンテーションのセクションにリストされた定義はすべて、ここで適用可能である。さらに次のｒ次元の｛０，１｝ベクトルをすべての$0 \leqq j < m$、$-m_2 \leqq k \leqq m_1$に対して定義する。
【０４０８】

$$v_{jk} = (t_{1jk}, t_{2jk}, \ldots\ldots\ldots, t_{rjk})$$

１．初期化：０から2^r-1までのあらゆるアドレスに０を入れる。
２．第１段階：

```
for a counter j going from 0 to n-1 do
  for k going from -m2 to m1 do
      add 2^k・x_j to the value address σ(v_jk)
```

３．主要部分：
｛０，１｝行列のアルゴリズムのｒ行に対する主要部分として進行し、そのデータはその演算処理の最後に同じアドレスに格納される。
【０４０９】
複雑性
ｉ）上記インプリメンテーションでの基礎的ステップは、０－１－２進のインプリメンテーションと同じである、すなわち、ソースと呼ばれるメモリの１つの場所から数を読み出し、それを宛先と呼ばれるメモリの別の場所に置かれた数に加える。
ii）上記コードに基づく装置は、Ａ・ｘの演算を完了するために次の数の基礎的ステップを要求するであろう。
（１）第１段階に対して、
$(m_1+m_2+1) \cdot n$
個の基礎的ステップが必要である。

（２）コードの主要部分は、ｒラインを備えた０－１の２進コードの主要部分によって行われる。すなわち、それは$C^{0,1}{}_r$個の基礎的ステップを必要とする。
（３）Ａ・ｘの演算全体は、このように次の回数の基礎的ステップを要求する。
【０４１０】
$$C^{Re-0-1} \equiv (m_1 + m_2 + 1) \cdot n + C^{0,1}{}_r$$
例７：実数の行列、２進のＵ表現
本発明の次の好適な実施例は、行列エントリの２進Ｕ表現を備えた本発明の実数行列の面からみたインプリメンテーションである。データは、ｒ×ｎ実数行列$A = (a_{ij} : 0 \leqq i < r, 0 \leqq j < n)$、および入力の実数または複素ベクトル$x = (x_0, \ldots\ldots\ldots, x_{n-1})$から成る。そのアルゴリズムは$y = (y_0, \ldots\ldots\ldots, y_{n-1}) = A \cdot x$を計算する。
【０４１１】
行列のエントリは、すべての$0 \leqq i < r$, $0 \leqq j < n$に対して、次のＵ表現で与えられると仮定される。
【０４１２】
$$a_{ij} = \Sigma_{-m2-1 \leqq k \leqq m1-1} t_{ijk} \cdot 2^k + t_{ijm1} \cdot (2^{m1} - 2^{-m2-1})$$
ここで、すべての$-m_2 - 1 \leqq k \leqq m_1$に対して、$t_{ijk} \in U$
この表現の存在は、本発明の実数行列の面から既に言及されている。第１段階だけが、Ｕ行列のインプリメンテーションのステージと異なっており、このステージについてだけ記述される必要がある。Ｕ行列のインプリメンテーションのセクションにリストされた定義はすべて、ここで適用可能である。さらに次のｒ次元のＵベクトルをすべての$0 \leqq j < n$、$-m_2 \leqq k \leqq m_1$に対して定義する。
【０４１３】
$u_{jk} = (t_{1jk}, t_{2jk}, \ldots\ldots\ldots, t_{rjk})$
１．初期化：０から$2^r + r - 1$までのあらゆるアドレスに０を入れる。
２．第１段階
for j going from 0 to n-1 do
for k going from $-m_2$-1 to m_1-1 do
add $2^k \cdot \mathrm{Sign}(u_{jk}) \cdot x_j$ to the address $\pi(h(u_{jk}))$
for k=m_1 do
add $(2^{m-1} - 2^{-m2-1}) \cdot \mathrm{Sign}(u_{jk}) \cdot x_j$ to the address $\pi(h(u_{jk}))$
３．主要部分：
ｕ行列のインプリメンテーションのアルゴリズムのｒ行に対する主要部分として進行し、そのデータはその演算処理の最後に同じアドレスに格納される。
【０４１４】
複雑性
ｉ）上記インプリメンテーションでの基礎的ステップは、Ｕインプリメンテーションと同じである、すなわち、ソースと呼ばれるメモリの１つの場所から数を読み出し、１または－１である符号をそれに掛けて、次に、その結果を宛先と呼ばれるメモリの別の場所に置かれた数に加える。
ii）上記コードに基づく装置は、Ａ・ｘの演算を完了するために次の数の基礎的ステップを要求するであろう。
（１）第１段階に対して、
$(m_1 + m_2 + 2) \cdot n$
個の基礎的ステップが必要である。
（２）コードの主要部分は、ｒラインを備えたＵコードの上記インプリメンテーションの主要部分によって行われる。すなわち、それは$C^u{}_r$個の基礎的ステップを必要とする。
（３）Ａ・ｘの演算全体は、このように次の回数の基礎的ステップを要求する。
【０４１５】
$$C^{Re-U}{}_r \equiv (m_1 + m_2 + 1) \cdot n + C^u{}_r$$

図４は、本発明の好適な実施例に従って、加算の回数を削減した線形変換を行なう典型的な装置のブロック図である。装置５００は、２つの入力および１つの出力を備えた乗算器１０、入力のうちの１つを選択しその出力に転送するためのマルチプレクサ（ＭＵＸ）９、２つの入力および１つの出力を備える加算器１１、その後ろに続く２つのアドレス・バス・ライン「ａｄｄ＿ａ」および「ａｄｄ＿ｂ」および２つの出力、「ｄａｔａ＿ａ」および「ｄａｔａ＿ｂ」を有するデュアル・ポート・ランダム・アクセス・メモリ（ＤＰＲＡＭ）１３を含む。ＭＵＸの動作は、アドレス・ジェネレータ５０１によって制御され、それは、さらにＤＰＲＡＭ１３中のメモリ・アドレスへのアクセスを可能にする。アドレス・ジェネレータの動作は、カウンタ３によって制御される。

【０４１６】

乗算器１０の出力は、ＭＵＸ９の１つの入力「Ｃ」に接続される。ＭＵＸ９の出力は、加算器１１の１つの入力「Ａ」に接続される。加算器１１の出力は、ＤＰＲＡＭ１３の入力に接続される。ＤＰＲＡＭ１３の１つの出力「ｄａｔａ＿ａ」は、加算器１１の入力「Ｂ」に接続される。ＤＰＲＡＭ１３の別の出力「ｄａｔａ＿ｂ」は、乗算器１２の入力「Ｅ」に接続される。乗算器１２の別の入力「Ｆ」は、アドレス・ジェネレータ５０１の出力「ｓｉｇｎ」に接続される。乗算器１２の出力は、ＭＵＸ９の別の入力「Ｄ」に接続される。カウンタ３は、アドレス・ジェネレータ５０１の２つの入力に接続される。ジェネレータ５０１の出力「Ｈ」は、乗算器１２の「ｓｉｇｎ」入力に接続される。ジェネレータ５０１の出力Ｇは、ＭＵＸ９の制御入力「Ｓ」に接続される。ジェネレータ５０１の出力「Ｊ」は、ＤＰＲＡＭ１３の第１のアドレス入力「ａｄｄ＿ａ」に接続される。ジェネレータ５０１の他の出力「Ｉ」は、ＤＰＲＡＭ１３の第２のアドレス入力「ａｄｄ＿ｂ」に接続される。変換行列は、オプションのＲＡＭ／ＲＯＭ１に格納され、それはアドレス・ジェネレータ５０１にコードのシリーズ（行列中の行のビット）D_0, D_1, ..., D_sを供給するとともに、最上桁のビットD_0を乗算器１０に送る。入力ベクトルは別のオプションのＲＡＭ／ＲＯＭ　２に格納されることがあるとともに、それは入力信号のサンプルで乗算器に１０を与える。代わって、入力ベクトルの要素および変換行列中のそれらに対応する要素が装置５００へ同期して提供される場合、格納メモリ１，２を除去することができる。例えば、これらの要素は、ＡＤＣまたはシーケンス・ジェネレータによって提供されてもよい。装置５００の全てのコンポーネントは、「ｃｌｏｃｋ＿ｉｎ」入力を介して共通クロックによって制御される。

【０４１７】

同じ入力セット［x_1　x_2　.....　x_n］、同じ異なるコードD_0, D_1, ..., D_sを表わす同じＵ行列（それはｎ行を含む）を使用する場合、装置の動作は、以下に説明される。装置５００の動作は２つのステージ、すなわち、入力データ（つまりサンプル［x_1　x_2　.....　x_n］）が受け取られ、各コンポーネントおよび変換行列のその対応する要素の積が計算され、ＤＰＲＡＭ１３に格納される「Ｓｔａｇｅ　１」と表示された第１ステージ、および、入力データが加算器９に入ることをブロックしている間、受信データが処理される「Ｓｔａｇｅ　２」と表示された第２ステージに分割される。カウンタ３は動作の蓄積数を係数するとともに、そのカウント（カウンタの出力）は２つのステージを識別するために用いられる。「Ｓｔａｇｅ　１」の動作回数は入力ベクトルの長さｎである。この数を越える動作は「Ｓｔａｇｅ　２」と関連する。

【０４１８】

本発明の好適な実施例によれば、アドレス・ジェネレータ５０１は、比較器４を含み、それはカウンタ３の出力にリンクされる。その比較器は、カウンタ３の現在の出力を読み、それを入力ベクトルの長さｎと比較し、現在の動作が「Ｓｔａｇｅ　１」または「Ｓｔａｇｅ　２」の中で実行されるかどうか示す対応信号を提供する。この信号は、入力「Ｃ」と「Ｄ」の間で切り替わるためにＭＵＸ９の入力「Ｓ」を制御するために使用される。

【０４１９】

アドレス・ジェネレータ５０１はさらに予めプログラムされた値を格納する非同期メモリ５（例えばＲＯＭ）を含み、入力アドレスが「ａｄｄ＿ａ」および「ａｄｄ＿ｂ」を介し

てＤＰＲＡＭ１３中のアドレスを決めるために参照テーブル（ＬＵＴ）としてそれらの内容の処理のために使用される。ＬＵＴのサイズは（Ｃ－ｎ）×（１＋２ｒ）であり、その内容は３つのフィールド、「ソース」フィールド、「符号」フィールド、および「宛先」フィールドからなる。各動作（つまり、カウンタ３によって計数される各クロック・サイクル）については、３つの対応するソース符号および宛先値がある。ソース・フィールドは、特定の列における２つの部分の正規化と関係する指示と同様に、スプリットされた（分割された）変換行列の各列と関係する情報からなる。ソース・フィールドは、ＤＰＲＡＭ１３上の入力「ａｄｄ＿ｂ」の値を決定する。符号フィールドは個々のスプリット列またはサブ列中のより低いコンポーネントを表わす。宛先フィールドは、対応するアドレスのどの内容が処理のために選択されたかに従って、ＤＰＲＡＭ１３上の入力「ａｄｄ＿ａ」の値を決定する。

【０４２０】

アドレス・ジェネレータはさらに１セットのｓ（ｓ＝ｒ－１）インバータ6_1，6_2，.....，6_sを含み、その各々はビットD_0，D_1，.....，D_sのシリーズにそれぞれ接続された入力を含む。各インバータの出力は、ｓマルチプレクサのセット、7_1，7_2，.....，7_sからの対応ＭＵＸの１つの入力に接続される。ビットD_0，D_1，.....，D_sのシリーズは、さらにｓマルチプレクサのセット、7_1，7_2，.....，7_sからの対応ＭＵＸの別の入力に供給される。セット7_1，7_2，.....，7_sからの各ＭＵＸの出力は、最上位ビットD_0の値によって制御され、セットD_1，.....，D_sをＤＰＲＡＭ１３の入力「ａｄｄｒｅｓｓ＿ａ」へ変化させないであるいは反転（つまりD'_1，.....，D'_s）して転送を可能にする。セットD_1，.....，D_s（またはD'_1，.....，D'_s）は、ＬＵＴから到着する「ａｄｄ＿ａ」にＭＳＢ（ｒ番目のビット）によって供給される追加のマルチプレクサ8_rと共に、ｓマルチプレクサの対応するセット8_1，8_2，.....，8_sの１つの入力に入力される。比較器４の出力によって、「ａｄｄ＿ａ」がセット7_1，7_2，.....，7_sからまたはＬＵＴから到着する選択を可能にする。入力「ａｄｄ＿ａ」（つまり宛先）は、加算器１１の入力「Ｂ」に供給するＤＰＲＡＭ１３の第１の出力「ｄａｔａ＿ａ」を制御する。ＬＵＴ（つまりソース）から得られた入力「ａｄｄ＿ｂ」はＤＰＲＡＭ　１３の第２の出力「ｄａｔａ＿ｂ」を制御し、それは乗算器１２の入力「Ｅ」に供給され、「ｄａｔａ＿ｂ」の各値をＬＵＴから抽出された対応する「符号」値と乗算し、その積を乗算器１２の入力「Ｄ」に供給する。ＤＰＲＡＭ１３中の「書込み」動作は同期しており、つまり、各セルの内容はクロックレート（例えば、クロック信号が立ち上がる時）に従って上書きされる。他方では、ＤＰＲＡＭ１３中の「読込み」動作は非同期であり、つまり、クロックにかかわらず、アドレス入力が変更されるときに、各出力は変更される。

【０４２１】

ステージ１の動作

このステージでは、ステージ１が動作を実行している間、カウンタ３は１シンボル時間（ｎクロック・サイクル）を計数し始める。ステージ１の間、ＭＵＸ９によって、入力「Ｄ」をブロックしている間、入力「Ｃ」からのデータ・フローをその出力および加算器１１の入力「Ａ」へ流れることを可能にする。入力シンボル［x_1　x_2　.....　x_n］は、乗算器１０の１つの入力へ提供される。そのコードのＭＳＢ　D_0は、乗算器１０の別の入力へ提供される。このステージでは、入力「ａｄｄ＿ａ」によって決定される宛先（出力「ｄａｔａ＿ａ」）は、ＤＰＲＡＭ１３から抽出され、入力ベクトルのコンポーネント［x_1　x_2　.....　x_n］に加えられ、ＭＳＢ　D_0を乗じられる。カウンタ３は、現在のシンボル時間を計数してきたが、アドレス・ジェネレータ比較器４へおよびＲＯＭ５へ現在のシンボル時間が終了し、比較器４がＭＵＸ９の入力選択を入力「Ｃ」からの別の入力「Ｄ」に切り替えるという指示を提供する。同様に、比較器４は、ＬＵＴからデータをＲＡＭ１からのデータへ選ぶために、マルチプレクサ8_1，8_2，.....，8_sを駆動する。

【０４２２】

ステージ２の動作

このステージでは、ステージ２が動作を実行している間、カウンタ３は次のシンボル時間を計数し始める。ステージ２の間、ＭＵＸ９によって、入力「Ｃ」をブロックしている間、入力「Ｄ」からのデータ・フローをその出力および加算器１１の入力「Ａ」へ流れることを可能にする。このステージにおいて、各クロック・サイクルで、ＤＰＲＡＭ１３中の選択されたアドレスは、ソースを表わす「ａｄｄ＿ｂ」によってアクセスされる。ソース・データは、ＤＰＲＡＭ１３の第２の出力「ｄａｔａ＿ｂ」から、乗算器１２の入力「Ｅ」に供給され、そこでＬＵＴから抽出された「符合」値が乗じられる。この積はＭＵＸ９に入力「Ｄ」に供給され、それによって、加算器１１の入力「Ａ」に現われる。同時に、現在の宛先値の内容が加算器１１の入力「Ｂ」に現われる。２つの値は加算器１１によって加えられ、その結果は、ＤＰＲＡＭ１３（それは前の宛先値に相当する）中の同じアドレスに格納される。この合計プロセスの終わりに、対応するｒ変換ポイント（y_i’ｓ）が、アドレス＃０および＃2^{r-1}から（2^{r-1}＋ｒ－２）のＤＰＲＡＭ１３に格納される。すべての変換ポイント（y_i’ｓ）が計算され、かつＤＰＲＡＭ１３の異なるアドレスに格納された後、（前もって定義したカウントに従って）ステージ２が終了するという指示をクロックが提供するまで、結果としてこのプロセスは継続される。そのときに、現在のシンボル時間も終了し、また、比較器４はＭＵＸ９の入力選択を入力「Ｄ」から別の入力「Ｃ」へ切り替え、そしてＲＡＭ１からのデータをむしろＬＵＴからデータへ選ぶために、マルチプレクサ8_1，8_2，.....，8_sを駆動し、それによって、ステージ１で再び動作するために装置５００を切り替える。

【０４２３】

上記方法を実行するために使用される装置の１つの実施例は図５で見られる。図５は、ｒ×ｎ　Ｕ行列Ａをｎ次元のベクトルＸで乗じた積のインプリメンテーションを図示する。行列は０または１の表現を含み、ここで０が１に相当し、１が－１に相当する。

【０４２４】

この実例において、ベクトルは実数であり、ｖビットは各コンポーネント特定する。次の構成において、行列Ａは要素１（ＲＡＭまたはＲＯＭ）に格納され、ベクトルＸは要素２（ＲＡＭまたはＲＯＭ）に格納される。行列とベクトルは、ＡＤＣまたはいくつかのシーケンス・ジェネレータなどに類するメモリ・デバイスまたは同期させられたソースのような内部または外部装置から生成される。

【０４２５】

モジュール１，２，３，１３を制御するクロックは、全体のシステムを同期させる。Ｃが以下のように定義される場合、モジュール３は０からＣまで計数するカウンタである。

【０４２６】

【数１０７】

$$C = n + 2u_0 + u_1 + u_2 + \ldots\ldots + u_{\lceil \log(r) \rceil - 1} - r.$$

$$u_k = \sum_{1 \le j \le 2^k} 2^{h(k,j) - l(k,j)}.$$

$$l(k,j) = \lceil (j-1) \cdot (r/2^k) \rceil$$

$$h(k,j) = \lceil j \cdot (r/2^k) \rceil - 1$$

【０４２７】

ステージ１では、要素３から来るアドレスに従って、要素１および２から入って来る信号は要素１３に挿入される、ここで、各アドレスは$\log_2 n$リスト重要ビット（list significant bit)を含む。

【０４２８】

加えて、カウンタは、（Ｃ－ｎ）×（１＋ｒ＋ｒ）のサイズの非同期ＲＯＭ（要素５）の

アドレス・ジェネレータとして使用される。カウンタのすべてのビットは比較器である要素４に入り、現在のカウントがｎより大きいかどうかをチェックする。要素５は３つのフィールドを包含する、すなわち、符号、ソース、宛先である。ＲＯＭにおけるラインの順序は、ステージ２の第１の動作がｎサイクルの後にカウンタによって引き起こされるようになされる。

【０４２９】

シンタックスの目的のために、ｓ＝ｒ－１を定義する。データ・バスは、要素１のビットＤ１－Ｄsから要素６＿１－６＿s（それらはインバータである）にそれぞれ入る。要素７１－７ｓは、Ｄ０の状態に従って、Ｄ１－Ｄｓと６１－６ｓの出力との間を選択する。

【０４３０】

(D0=0 => 7_1-7_s=D1-Ds, D0=1 => 71-7s=61-6s).

要素８＿１－８＿ｓは、要素４（ｓｔａｇｅ１＿２ｎ）中の比較器の出力の状態に従って、７＿１－７＿ｓと要素５（ａｄｄ＿ａ＝宛先）のlog_2（ｎ）ＬＳＢ出力との間を選択する。

【０４３１】

Stage 1_2n=0 => 8_1-8_s=add_a, Stage1_2n=1 => 8_1-8_s=7_1-7_s.

要素８＿ｒは、「０」と要素５のｒビット（ａｄｄ＿ａ　ＭＳＢ）との間を選択する。Stage 1_2n=0 => 8_r=add_a MSB, Stage 1_2n=1 => 8_r=0.　８＿１－８＿ｒからの出力は、要素１３の第１のアドレスバスとして使用され、それは、（2^{r-1}＋ｒ－１）×（Ｖ＋log_2ｎ）のサイズのデュアル・ポートＲＡＭである。このアドレスは、要素１３からの出力であるｄａｔａ＿ａバス、およびさらに要素１３への入力であるｄａｔａ＿ｉｎバスの宛先を定義する。

【０４３２】

Ａｄｄ＿ｂ（それは要素５（ソース・フィールド）の出力である）は、要素１３の第２のアドレス・バスである（このアドレスは、要素１３からの出力であるｄａｔａ＿ｂバスを制御する）。

【０４３３】

符号は要素１３からの単一ビットの出力であり、（乗算器である）要素１２でｄａｔａ＿ｂと乗算する。

【０４３４】

要素１３に関して入力であるｄａｔａ＿ｉｎバスは、要素１１から到着し、それは、ｄａｔａ＿ａおよび要素９の出力をともに合計する加算器である。ｄａｔａ＿ａとｄａｔａ＿ｂの読込み動作が非同期である間、ｄａｔａ＿ｉｎは同期した方法でａｄｄ＿ａ宛先へ行く。

【０４３５】

要素１３の動作に先立って、ゼロの挿入によって開始する。要素９は、要素１０と要素の１２の出力との間を選択する。要素１０は、要素２からのデータ・バスを要素１のＬＳＢで掛ける。Ｃサイクルの後、結果としてのベクトルは、アドレス０およびアドレス2^{r-1}から（2^{r-1}＋ｒ－２）の要素１３に格納される。

【０４３６】

本発明の多数の変形および修正は、当業者に容易に明らかになるであろう。従って、本発明は、その精神またはその本質的特質から逸脱せずに、他の特定の形式で具体化されてもよい。詳細な実施例は、単に図示されたものとして尊重され、限定的なものではなく、したがって、本発明の範囲は前述の記述によるのではなく、添付された請求項によって示される。請求項の均等の解釈および範囲内にある変更はすべて、本請求項の範囲内に包含される。

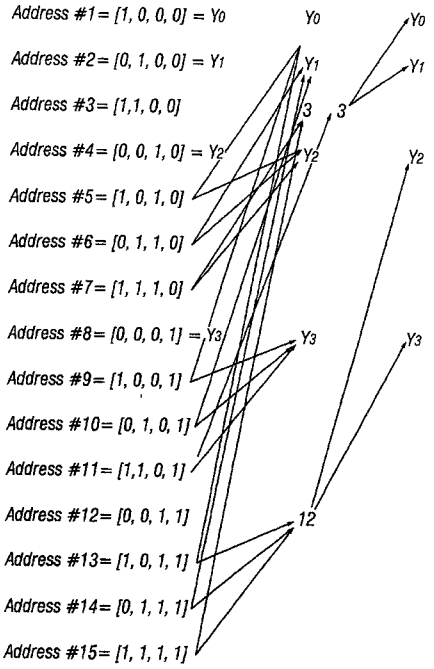
【図面の簡単な説明】

【図１】　本発明の好適な実施例に従って、０－１－２進変換行列を使用して変換を行なう装置によって使用されたメモリの内容を更新する動作を概略的に図示する。

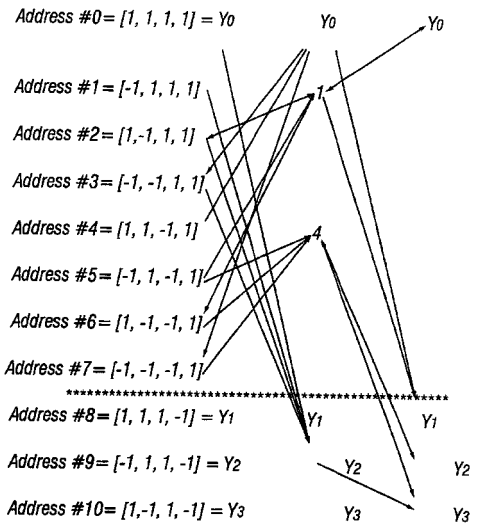
【図２】　本発明の好適な実施例に従って、Ｕ変換行列を使用して変換を行なう装置によ

って使用されたメモリの内容を更新する動作を概略的に図示する。
【図３】　本発明の好適な実施例に従って、トップリッツ変換行列を使用して変換を行なう装置によって使用されたメモリの内容を更新する動作を概略的に図示する。
【図４】　本発明の好適な実施例に従って、加算の削減数を備えた線形変換を行なうための典型的な装置のブロック図である。
【図５】　本発明の好適な実施例に従って、ｒ×ｎのＵ行列Ａをｎ次元のベクトルＸで乗じた積のインプリメンテーションの例を図示する。

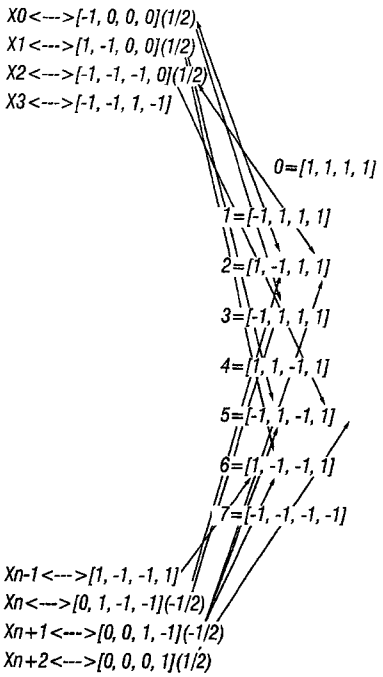
【図１】



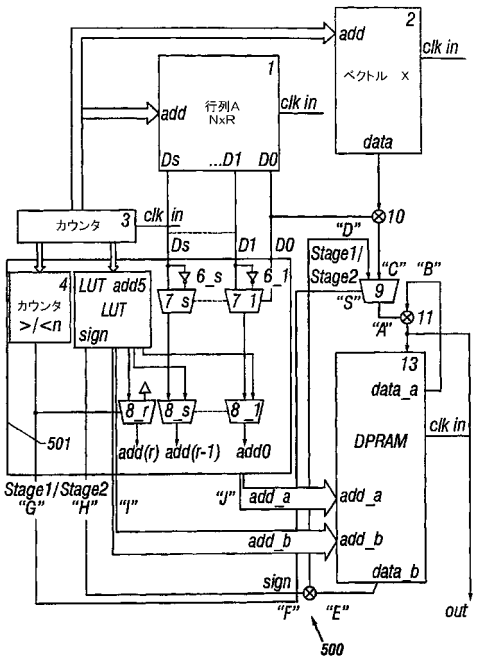
【図２】



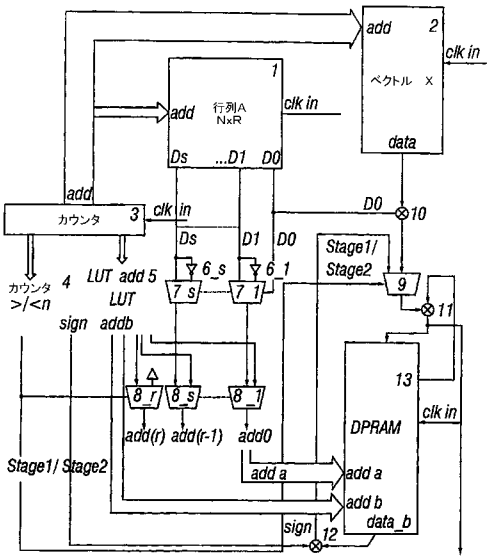
【図３】


X0<--->[-1, 0, 0, 0](1/2)
X1<--->[1, -1, 0, 0](1/2)
X2<--->[-1, -1, -1, 0](1/2)
X3<--->[-1, -1, 1, -1]
0=[1, 1, 1, 1]
1=[-1, 1, 1, 1]
2=[1, -1, 1, 1]
3=[-1, 1, 1, 1]
4=[1, 1, -1, 1]
5=[-1, 1, -1, 1]
6=[1, -1, -1, 1]
7=[-1, -1, -1, -1]
Xn-1<--->[1, -1, -1, 1]
Xn<--->[0, 1, -1, -1](-1/2)
Xn+1<--->[0, 0, 1, -1](-1/2)
Xn+2<--->[0, 0, 0, 1](1/2)

【図４】


add
clk in
ベクトル X
data
行列A NxR
Ds ...D1 D0
カウンタ
LUT add5
LUT
sign
6_s
7_s
7_1
8_r
8_s
8_1
add(r) add(r-1) add0
501
Stage1/Stage2
"A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "S"
add_a
add_b
data_a
data_b
DPRAM
sign
out
500

【図５】


add
clk in
ベクトル X
data
行列A NxR
Ds ...D1 D0
カウンタ
LUT add 5
LUT
sign addb
6_s
7_s
7_1
8_r
8_s
8_1
add(r) add(r-1) add0
Stage1/Stage2
add a
add b
data_b
DPRAM
sign

フロントページの続き

(72)発明者　ドル，アブナー
イスラエル国　キリヤット・オノ　ハマンガル・ストリート１８１８

審査官　鳥居　稔

(56)参考文献　特開平０２－２７３８６７（ＪＰ，Ａ）
特開平１０－３０８６７２（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 17/14
G06F 17/16