(19)대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) 。Int. Cl.[7]
G06F 17/14

(45) 공고일자 2005년10월19일
(11) 등록번호 10-0522262
(24) 등록일자 2005년10월11일

(21) 출원번호 10-2003-7004937
(22) 출원일자 2003년04월07일
번역문 제출일자 2003년04월07일
(86) 국제출원번호 PCT/US2001/042537
국제출원일자 2001년10월05일
(65) 공개번호 10-2003-0094213
(43) 공개일자 2003년12월11일
(87) 국제공개번호 WO 2002/29611
국제공개일자 2002년04월11일

(81) 지정국
국내특허 : 중국, 일본, 대한민국, 싱가포르,

(30) 우선권주장 09/680,665 2000년10월06일 미국(US)

(73) 특허권자
인텔 코포레이션
미국 캘리포니아주 95052-8119 산타클라라 피.오.박스 58119 미션 칼리지 불바드 2200

(72) 발명자
엘린다니
이스라엘라나나헤르젤스트리트71

라이니쉬도론
이스라엘라맛-간키쉬스트리트3

도르아브네르
이스라엘키르얏오노하망갈스트리트1818

(74) 대리인
김태홍
신정건
김두규

심사관 : 손영태

(54) 선형 변환을 효과적으로 수행하기 위한 방법 및 장치

요약

본 발명은 산술 연산수를 줄이고 회로를 단순화하며 전력 소비를 낮출 수 있는 향상된 선형 변환 수행 방법 및 장치에 관한 것이다. 상기 방법은 CDMA 분야에 나오는 복소수 U 변환을 수행한다. 상기 장치는 상이한 확산 인수의 입중 신호의 동시 분석을 가능하게 해준다.

대표도

도 1

명세서

기술분야

본 발명은 디지털 신호 처리 분야에 관한 것으로, 구체적으로는 산술 연산수를 줄이고 회로를 단순화하며 전력 소비를 낮출 수 있는 개선된 선형 변환 수행 방법 및 장치에 관한 것이다.

배경기술

선형 변환은 일반적으로 신호 또는 데이터 처리에 이용된다. 선형 변환에 의해 n 차원 벡터의 "입력 벡터"로부터 r 차원 벡터의 "출력 벡터"가 만들어진다. 많은 응용에 있어서, 입력 벡터는 처리를 요하는 소정의 신호의 디지털 샘플로 이루어진다. 그러나, 선형 변환을 응용함에 있어서 그것은 어느 특정 범위로 한정되는 것은 아니며, 모든 과학 및 기술 분야에 있어서 필수적이고 또한 매우 바람직하게도 그 수행이 효율적이다.

n 차원 벡터 집합(실수, 복소수 또는 어떤 스칼라계)에서 r 차원 벡터 집합(동일한 스칼라계)으로의 일반적인 선형 변환을 다음과 같이 주어진 차수 rxn 행렬로 표현할 수 있다.

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

일반적인 입력 n 차원 열벡터는 다음과 같은 형태로 나타낸다.

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

선형 변환은 다음과 같은 식으로 주어진 행렬과 벡터의 곱셈을 통해 입력 벡터 x로부터 r 차원의 출력 열벡터 y를 만든다.

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

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

선형 변환의 간단한 수행은 r · n 개의 곱셈과 r · (n-1) 개의 덧셈을 필요로 한다.

본 발명의 전개 및 응용에 있어서 이진 행렬이 매우 중요하다. 이진 행렬은 그 엔트리로 단지 ±1 값을 포함한다. 이하, 그러한 행렬을 U-행렬로 부르고, U-행렬로 표현되는 선형 변환을 U-변환으로 부를 것이다. 상기에서, 주어진 변환이 U-변환이면, 그 간단한 수행은 r · (n-1) 개의 덧셈/뺄셈을 필요로 한다. 다른 종류의 이진 행렬은 그 엔트리로 단지 0, 1 값을 포함한다. 그러한 행렬을 0-1-행렬로 부른다. 0-1-행렬로 표현되는 선형 변환을 0-1-변환으로 부른다. 상기에서, 주어진 변환이 0-1-변환이면, 그 간단한 수행은 평균적으로 r · (n-1)/2 개의 덧셈을 필요로 한다.

본 명세서에서 용어 "이진 변환"은 전술한 두 종류의 변환, 즉 U-변환 및 0-1-변환을 의미한다. 용어 U-벡터는 그 성분으로 ±1 값을 갖는 벡터를 의미하며, 마찬가지로 그 성분으로 0, 1 값을 갖는 벡터를 0-1-벡터로 부른다.

이진 변환 처리는 입력 벡터의 성분의 덧셈 및 뺄셈으로 이루어진다. 이것은 전자 주문형 집적 회로(ASIC)와 같은 하드웨어로 구현되며, 덧셈기, 뺄셈기와 같은 구성 요소를 필요로 한다. 이러한 구성 요소들을 사용하는 데에는 많은 비용과 에너지가 소비되며, 또한 그러한 구성은 상당한 영역을 차지한다. 수많은 기술 분야에서 이러한 자원의 소비를 낮추면서 이진 변환을 하드웨어로 구현할 수 있는 기술에 대한 필요성이 커지고 있다.

다양한 형태의 디지털 신호 처리 분야에서 U-변환이 폭넓게 사용되고 있다. 예컨대, 이미지 처리, 무선 통신과 같은 각종 통신 기술 분야에서 사용되고 있다.

최근, 직접 시퀀스 코드 분할 다중 접속(DS-CDMA) 대역 확산 통신 시스템에 대한 관심이 세계적으로 증가하고 있다. IS-95[TIA/EIA/IS-95-A, "Mobile Station Base Station Compatibility Standard for Dual-Mode Wideband Spread spectrum Cellular system", Feb 27, 1996]는 개발중인 DS-CDMA 시스템의 일례이다.

상기 CDMA 전송 기술에 있어서는, 다중 사용자 데이터가 합성 신호로 보내지며 의사 잡음(PN) 코드, 즉 랜덤 잡음 특성(예컨대, 상호 상관도가 낮음)을 갖는 U-시퀀스와 곱해진 후 전송된다. 이러한 확산 특성은 전송시 잡음(다중 경로 잡음 포함), 전파 방해(jamming) 또는 검파에 대한 저항력을 제공한다. 또한, 채널화(channeling) 코드보다 긴 암호화(scrambling) 코드도 적용된다. 이러한 통신 방식의 제2 세대(IS-95-B) 및 제3 세대(3G) 광대역(WB) CDMA 표준은 다중 코드 검출을 수행하는 수신기를 필요로 한다. 즉, 상이한 채널화 코드에 따라 이전에 확산된 각각의 결합된 채널을 역확산시켜야 한다. 이 때, U-변환이 적용되며, 그 변환 행렬을 포함하는 확산 코드는 흔히 아다마르(Hadamard) 행렬의 행(row)이다. 이것은 많은 예 중에 하나이며, 여기서 효과적인 U-변환 기술은 자원의 소비를 낮추면서 계산 태스크를 달성하는 데 적합하다.

특정한 종류의 선형 변환의 수행을 개선하기 위한 몇가지 공지된 매커니즘이 있다. 광범위한 예 중에 선택된 관련된 몇가지 예를 이하 설명한다. 컨볼루션을 수행하고 DSP에서 필터로 사용되는 토플리츠(Toplitz) 변환은 $O(n \cdot \log_2(n))$ 개의 덧셈 및 곱셈 연산(여기서, n은 영역 공간의 차원)을 필요로 하는 전통적인 고속 퓨리에 변환(FFT) 알고리즘을 사용함으로써

효율적으로 수행될 수 있다. 표준에 비해, 간단한 방법은 $O(n^2)$ 개의 덧셈 및 곱셈을 필요로 한다. 상세한 설명을 원한다면, [James W. Cooley and John W. Tukey, "An algorithm for the machine calculation of complex Fourier series", *Mathematics of computation*, 19(90):297-301, April 1965.]를 보라.

CDMA 기술을 포함하는 디지털 신호 처리에 사용되는 특정한 종류의 U-변환에는 아다마르 행렬로 표현되는 왈시-아르마르(Walsh-Hadamard) 변환이 있다. 아다마르 행렬은 다음과 같이 반복적으로 정의할 수 있다.

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

그리고, 모든 정수 n에 대해 제곱하면,

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

이러한 변환을 수행함에 있어서 복잡도가 낮고 에너지를 보존하는 알고리즘은 고속 왈시-아다마르 변환(FWT)으로서, nxn 아다마르 행렬에서 덧셈/뺄셈수를 $n \cdot \log_2(n)$ 개로 줄인다. 이것은 그 연산을 최적으로 줄인다. 이러한 방법을 강조하는 기본적인 원리는 FFT의 경우와 유사하다.

또 다른 형태의 본 발명에 의해 U-변환의 형태를 변형시켜, 토플리츠 U-변환의 수행에 효율적인 방법 및 장치를 제공한다. 이러한 변환은 소정의 U-시퀀스나 복소수 U-시퀀스와의 전체 또는 부분 컨볼루션을 나타낸다. 토플리츠 U-변환의 무선 통신 응용에는 초기 시간 동기화와, 실수 또는 복소수 의사(PN) 또는 골드(gold) U-시퀀스와의 컨볼루션을 이용하는 탐색기(searcher)가 포함된다. 골드 시퀀스는 2개의 PN 시퀀스의 Z_2 합으로 구성된다.

또 다른 형태의 본 발명에 의해 상기 이진 형태를 일반화하여, 비교적 소수의 상이한 엔트리를 갖는 rxn 행렬로 표현되는 선형 변환의 수행에 효율적인 방법 및 장치를 제공한다. 이렇게 향상된 형태의 본 발명의 응용에는 복소수 U-변환과, 엔트리가 {0,1,-1}인 행렬로 표현되는 선형 변환이 포함된다. 이러한 총괄적인 본 발명의 바람직한 실시예를 일반 소거 방법(GEM ; generalized-elimination-method)으로 부를 것이다.

광대역 CDMA 및 파생 가능한 다른 향상된 DSP와 같은 장래 기술들은 상이한 확산 인수의 입중 신호의 동시 분석으로부터 이익을 얻을 것이다. 이것은 네트워크를 과중시키지 않으면서 팩스, 정상적인 전화 통화 및 인터넷 데이터와 같은 상이한 종류의 정보를 동시에 수신할 수 있는 다중 코드 채널을 제공한다. 또 다른 형태의 본 발명은 이러한 종류의 선형 연산에 효율적인 방법 및 장치이다. 그것은 추가의 프로세서를 갖는 본 발명의 U-이진 형태의 변형된 버전을 포함한다. 이 추가의 프로세서는 전체적으로 덧셈 비율이 낮은 각종 서브섹션에 U-이진 방법을 적용하는 구성을 발견하기 위해서 상기 U-이진 방법이 필요로 하는 덧셈수에 관한 정보를 이용한다.

본 발명의 주된 응용에는 추가적으로 무선 멀티미디어 시스템, 개인 위성 이동 시스템, GPS 위성 기반 위치 시스템 등이 포함된다. 이동 통신 및 다른 이동 컴퓨터 기반 시스템 환경에서는, 선형 처리에 사용되는 현재 소비량을 줄이는 것이 필수적이다. 본 발명을 이동 전화 기술에 적용하면, 배터리 수명을 연장시킬 수 있고, 회로를 줄일 수 있으며, 응답 시간을 단축할 수 있다.

도면의 간단한 설명

본 발명의 이러한 특징 및 다른 특징과 이점은 첨부한 도면을 참조하여 다음의 상세한 설명을 읽으면 보다 명확하게 이해가 될 것이다.

도 1은 본 발명의 바람직한 실시예에 따라, 0-1-이진 변환 행렬을 이용하여 변환을 수행하는 장치에 사용되는 메모리의 내용을 갱신하는 동작을 개략적으로 보여준다.

도 2는 본 발명의 바람직한 실시예에 따라, U-변환 행렬을 이용하여 변환을 수행하는 장치에 사용되는 메모리의 내용을 갱신하는 동작을 개략적으로 보여준다.

도 3은 본 발명의 바람직한 실시예에 따라, 토플리츠 변환 행렬을 이용하여 변환을 수행하는 장치에 사용되는 메모리의 내용을 갱신하는 동작을 개략적으로 보여준다.

도 4는 본 발명의 바람직한 실시예에 따라, 덧셈수를 줄이면서 선형 변환을 수행하기 위한 예시적인 장치의 블록도이다.

도 5는 본 발명의 일 실시예에 따라 rxn U-행렬 A와 n 차원 벡터 X의 곱셈을 구현한 일례를 보여준다.

실시예

본 발명의 방법을 설명함에 있어서 기본적인 2개의 새로운 용어가 있다. 그 중에 한 용어는 등가 벡터이다. 동일한 차원의 2개의 0이 아닌 벡터는 그 중에 각각의 어느 한 벡터가 다른 벡터의 곱일 때 등가라고 한다. 이 용어는 소정의 행렬의 2개의 행 또는 2개의 열에 적용될 수 있다. 다른 용어는 소정의 행렬의 열에서의 선두(lead) 성분이다. 본 발명에 사용된 정의는 상기 행렬의 각 열에서의 최고 인덱스의 0이 아닌 성분이다. 이러한 정의는 소정의 행렬에서 일정하게 사용되는 미리 정의된 열의 인덱스 순서에 따라 달라진다. 새로운 순서를 재정의하면 새로운 선두 성분이 재정의된다. 예컨대, 소정의 행렬의 선두 성분은 어떤 순서가 목적에 따라 선택하기에 편리하게 보이는지에 따라 각 열의 최하위의 0이 아닌 성분이거나 최상위의 0이 아닌 성분이 될 수 있다.

0-1-행렬

본 발명의 바람직한 실시예는 0-1-이진 선형 변환을 수행하는 효율적인 방법을 제공한다. 이하, 그 방법을 "0-1-방법"으로 부른다. 다음의 예는 0-1-방법의 개론 및 그 주된 사상의 개요이다.

예: 0-1-이진 4x14 행렬 A가

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

이고, 14 차원 입력 벡터 x가

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

로 주어지면, 4 차원 "출력" 벡터 y는 다음과 같이 계산된다고 가정하자.

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

본 발명의 바람직한 실시예에 따라, 먼저 행렬 A에 서로 동일한 라인이 있는지를 검사한다. 서로 동일한 라인이 없으므로, 다음 단계로 절차를 진행한다. 다음과 같이, 출력 벡터 y를 각각의 입력 벡터 좌표인 계수와 함께 열의 합으로 표현한다.

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

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

이 단계에서, 0인 열은 그 계수와 함께 소거된다. 입력 벡터에 대해 본 발명의 바람직한 실시예에 따라 수행하는 제1 단계는 모든 반복되는 0이 아닌 열의 계수를 모아 합하는 것이다. 그리하여, 6개의 덧셈으로 다음과 같이 줄어든 벡터 방정식을 얻는다.

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

$$+(x_6+x_{13})\begin{bmatrix}1\\1\\1\\1\end{bmatrix}$$

이 표현식은 다음의 정의

$$w_1=x_1+x_7+x_{12},\quad w_2=x_2+x_9,\quad w_3=x_3+x_{14},\quad w_4=x_4+x_{11},\quad w_5=x_5,\quad w_6=$$

$$x_6+x_{13},$$

에 의해 다음과 같이 단순화될 수 있다.

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

그러나, 이것은 열수를 줄이려는 최초의 과제와 동일한 과제를 갖는데, 지금까지 이러한 식의 감축은 모두 다 이루어졌다. 열을 더 감축하기 위해서, 상기 벡터 방정식을 다음과 같이 동일한 2개의 벡터 방정식으로 분할한다.

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

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

이 2개의 벡터 방정식 각각마다 독립적으로 상기 바람직한 실시예의 제1 단계를 수행한다. 이와 같이, 동일한 0이 아닌 열의 계수를 모아 합한다. 그리하여, 6개의 추가의 덧셈으로 다음의 2개의 벡터 방정식을 얻는다.

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

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

이제, 출력 벡터 y의 계산을 완료하는 데 단지 4개의 추가의 덧셈만을 필요로 한다는 것을 분명히 알 수 있다. 이와 같이, 총 16개의 덧셈으로 희망하는 결과를 얻었다. 이러한 계산 과정을 수행하는 종래 기술의 방법은 28개의 덧셈을 필요로 한다는 것을 쉽게 확인할 수 있다. 본 발명의 상기 바람직한 실시예에 대한 이전의 설명으로부터, 행렬의 차원이 증가할수록 상기 방법의 효율성이 비례적으로 증가한다는 것을 알 수 있다.

일반적으로 본 발명의 0-1-이진 형태는 rxn 차원의 이진 0-1-행렬 A와 n 차원 벡터 x의 곱셈에 효율적인 방법 및 장치이다.

행렬 A와 입력 벡터 x를 다음과 같이 가정하자.

$$A=\begin{bmatrix} a_{11} & a_{12} & \cdot & \cdot & \cdot & \cdot & a_{1n} \\ a_{21} & a_{22} & \cdot & \cdot & \cdot & \cdot & a_{2n} \\ \cdot & & & & & & \\ \cdot & & & & & & \\ \cdot & & & & & & \\ a_{r1} & & & & & & a_m \end{bmatrix} \text{ where } a_{ij}=0,1 \text{ for all } 1 \le i \le r \ , \ 1$$

$\le j \le n.$

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

입력 벡터의 엔트리는 실수 또는 복소수이거나, 어떤 스칼라계(예컨대, Z_2)에 속한 것일 수 있다. 행렬 A와 벡터 x의 곱셈의 결과를 계산하는 것을 목적으로 한다. 그 결과는 다음과 같다.

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

설명할 절차는 반복적이며, 반복할 때마다 다음의 일련의 단계 또는 그 일부가 반복된다. 본 발명의 바람직한 실시예에 따라, 제1 단계는 서로 동일한 행이 있는지를 검사하는 것이다. 가능하다면 이 단계는 입력 벡터의 처리를 시작하기 전에 예비 준비 단계로서 수행해야 한다. i 라인이 j 라인과 동일하다면, y_i가 y_j와 동일하다고 추론할 수 있다. 그러므로, 동일한 2개의 라인 중 하나를 생략한다. 명확성을 위해서, 인덱스가 더 큰 라인을 항상 생략한다. 이와 같이, j>i 일 때, j 라인을 생략한다. 이 초기 동작은 본 발명의 실행의 마지막 단계, 즉 (y_i와 같으므로) 이 단계에서 알려진 y_j가 벡터 y의 적절한 위치에 다시 삽입될 때 보정된다. 동일한 라인에 대한 생략은 행렬 A에서 서로 동일한 라인이 없을 때까지 계속된다. 사실상 이 단계는 여러 경우에 스킵된다. 그러나, 서로 동일한 라인이 있을 가능성이 어느 정도 있을 경우에는 수행된다. 또한, $\log_2(r)>n$ 일 때에는 언제나 수행되는데, 그 이유는 이러한 조건 하에서는 언제나 서로 동일한 라인이 존재하기 때문이다.

표기의 복잡성을 피하기 위해서, 상기 검사 과정에서 변형된 행렬에 최초의 행렬과 동일한 부호 A를 그대로 부여하고, 차원 rxn의 부호도 그대로 유지한다. 다음 단계는 서로 동일한 열을 소거하기 위한 합산 과정이다. 이 절차는 덧셈에서 최소 비용으로 변형된 행렬의 열수를 줄일 것이다. 먼저, 다음과 같이, 행렬과 벡터의 곱 y=A・x를 행렬 A의 열과 각각의 x 성분과의 곱의 합으로 분해한다.

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

이 합의 각각의 요소는 행렬 A의 열 벡터와 대응하는 벡터 x의 스칼라 계수와의 곱으로 구성된다. 이 합은 다음과 같이 더 간결하게 나타낼 수 있다.

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

여기서, v_j(모든 j에 대해)는 A의 j 열이다.

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

본 발명의 바람직한 실시예에 따라, 이 단계에서 0인 열을 그 계수와 함께 생략한다. 다음에, 서로 동일한 0이 아닌 열을 함께 그룹화하여 재배열하는데, 여기서 각각의 상이한 열은 공통 인수로서, 대응하는 스칼라 계수의 합과 곱해진다. 그러므로, 생성된 합은 반복되는 열의 생략을 통해 최초의 0인 아닌 열의 시퀀스로부터 구한 모든 상이한 0이 아닌 열의 서브시퀀스 $w_1, \cdots\cdots, w_m$를 포함한다. 각각의 열 w_j를 이 열과 동일한 모든 열의 대응하는 최초의 계수의 합인 계수 t_j와 곱한다. 명백한 것은 m ≤n 이고, 그 간격 n-m은 최초의 시퀀스 $v_1, v_2, \cdots\cdots, v_n$에서의 반복수라는 것이다. 전술한 과정은 다음과 같은 수학식으로 공식화될 수 있다.

$$y = A \bullet x = \sum_{1 \le j \le n} x_j v_j$$

$$= \sum_{1 \le k \le m} \sum_{j \text{ such that: } v_j = w_k} x_j v_j$$

$$= \sum_{1 \le k \le m} \left(\sum_{j \text{ such that: } v_j = w_k} x_j \right) w_k$$

이제 1 ≤k ≤m 임을 고려하여 다음과 같이 정의한다.

$$t_k = \sum_{j \text{ such that: } v_j = w_k} x_j$$

계산 태스크는 지금까지 최초의 것 $x_1, \cdots\cdots, x_n$의 합과 같이 새로운 계수 $t_1, \cdots\cdots, t_m$를 계산하는 것이다. 그 대가는 n-m 개의 덧셈이다. 따라서, 곱 y=A · x는 다음과 같이 주어진다.

$$y = A \bullet x = \sum_{1 \le k \le m} t_k w_k$$

이러한 계산의 관리면에 있어서, 요구되는 합의 참여 요소를 판정하는 것은 입력 벡터가 도착하기 전에 예비 준비 단계로서 행렬마다 한 번 수행된다. 요약하자면, 행렬 B는 각각의 열이 $w_1, \cdots\cdots, w_m$인 rxm 행렬이고,

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

이라고 하면, y=A · x=B · t 이다. 이와 같이, 상기 과정은 rxn 행렬 A와 n 벡터 x와의 곱에 대한 최초의 문제를 rxm 행렬 B와 m 벡터 t와의 곱에 대한 문제로 축소시킨다. 이로써 얻는 이득은 m이 작을수록 커진다.

다음 단계에서는 이전 단계의 행렬 B를 2개 이상의 하위 행렬로 분할하며, 이 때 라인은 그대로 유지된다. 각각의 하위 행렬은 그 행수가 줄어들어, 다음 반복 절차에서 상기 열 모음 절차의 이득이 커진다. 행렬 B의 라인을 $u_1, u_2, \cdots\cdots, u_r$로 표시한다. 이러한 요건 하에 곱을 다음과 같이 표현할 수 있다.

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

여기서, 각각의 라인은 벡터 t와 스칼라곱으로 곱해진다. 이러한 표현식에 있어서, y=B · t를 계산하는 태스크는 r개의 스칼라곱 $u_1 \cdot t, u_2 \cdot t, \cdots\cdots, u_r \cdot t$을 계산하는 태스크를 결합한 것과 같다.

이것은 본 발명의 바람직한 실시예에 따라 벡터곱 $B_1 \cdot t$ 및 $B_2 \cdot t$에 의해 행렬을 계산함으로써 수행할 수 있으며, 여기서 각각의 행렬 B_1 및 B_2는 행렬 B의 라인의 부분 집합을 포함하고 이러한 2개의 부분 집합을 서로 분리하여 결합하면 행렬 B의 모든 라인의 집합을 구성한다. 보통, 제1 반복 절차를 제외하고, 2개의 하위 행렬은 그 라인수가 동일하거나 거의 동일하다. 그러나, 행렬 A의 특성에 따라 일부 특수한 경우에는 2개 이상의 행렬로 분할할 수도 있다. 그러한 경우는 행렬 A에 특별한 내부 순서가 없어, 그 엔트리가 임의적으로 선택될 수 있을 때이다. 다음에, 본 발명의 바람직한 실시예는 일반적으로 모든 분할된 하위 행렬을 "$\log_2$(열수)>행수"의 상태로 만드는 제1 라인 분할을 포함한다. 다음에, 계속된 반복 절차에서 라인을 2개의 거의 동일한 반 라인으로 분할한다. 행렬이 임의적인 것으로 간주되는 경우가 본질적으로 최악의 경우의 시나리오이다.

전술한 수평 분할 대신에 또는 그 이전에 삽입될 수 있는 또 다른 단계는 수직 분할이다. 본 발명의 바람직한 실시예에 따라 전술한 합 $y= \sum_{1 \le k \le m} t_k w_k$은 다음과 같이 2개로 분할된다.

$$y= \sum_{1 \le k \le p} t_k w_k + \sum_{p+1 \le k \le m} t_k w_k$$

여기서, p는 1과 m-1 사이에서 선택된다. 따라서, B_1이 열의 집합이 $w_1, \cdots\cdots, w_p$인 행렬이고, B_2가 열의 집합이 $w_{p+1}, \cdots\cdots, w_m$인 행렬일 때, y는 다음과 같이 유지된다.

$$y = B_1 \bullet t' + B_2 \bullet t''$$

여기서, 대응하는 벡터는 $t'=(t_1,\cdots\cdots,t_p)$ 및 $t''=(t_{p+1},\cdots\cdots,t_m)$이다. 그러므로, 본 발명의 바람직한 실시예에 따라, 2개의 낮은 차원의 곱 $B_1 \cdot t'$ 및 $B_2 \cdot t''$는 독립적으로 계산되고, 결국 2개의 r 벡터인 그 결과값들이 서로 합산된다. 이 단계의 효과는 행렬 B의 열 $w_1,\cdots\cdots,w_m$의 인덱스를 미리 재배열함으로써 더 커질 수 있다.

수직 분할은 다른 절차만큼 자주 사용되지 않는다. 그것은 주로 행수가 실질적으로 열수를 초과할 경우(이러한 상황은 DSP 응용에서 흔하지 않음)에 사용된다. 이 단계의 근본적인 결함은 상기 수평 분할에서 평행하지 않은 추가의 r 스칼라 덧셈으로 이루어지는 2개의 곱 $B_1 \cdot t'$ 및 $B_2 \cdot t''$의 결과값들을 합산해야 한다는 것에 있다. 상기 수평 분할에서와 같이, 행렬 A의 특성에 따라 2개 이상의 행렬로 수직 분할될 수 있다.

마지막으로 상기 각 단계는 반복되는 반복 절차에 적용되므로, 반복 매커니즘을 형성한다.

다음에, 상기 방법에 의해 가능한 감축 한도를 제공한다. rxn 0-1-행렬 A($r < \log_2(n)$)인 경우, s*(n, r)로 표기되는 최악의 경우에서의 덧셈수는 다음과 같은 표현식으로 주어진다.

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

다음의 표는 소수의 행을 갖는 행렬에서의 덧셈수의 감축 한도를 보여준다.

$s^*(2) = -1$ $\qquad$ $s^*(n, 2) = n - 1$

$s^*(3) = 2$ $\qquad$ $s^*(n, 3) = n + 2$

$s^*(4) = 7$ $\qquad$ $s^*(n, 4) = n + 13$

$s^*(5) = 22$ $\qquad$ $s^*(n, 5) = n + 49$

최악의 경우에서의 감축 한도는 다음과 같다.

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

표준 (종래 기술) 방법은 u(A)-r 개의 덧셈을 필요로 하며, 여기서 u(A)는 행렬 A에서의 수 1'ns이다. 평균적으로(A가 임의적일 때는 제외), $u(A)=(n-2) \cdot r/2$ 이다. s*(n, r)/n은 n이 무한대로 가고 r이 일정할 때(또는 $r \leq c \cdot \log_2(n)$, 여기서 $1 > c > 0$) 1에 가까워지므로, 이 방법은 (r이 유한하고 n이 무한할 때) 점근적으로 최적이 된다.

A가 rxn 0-1-행렬이고 행렬 A의 차원 rxn에 대한 가정이 없을 때, 본 발명에 의해 곱 $A \cdot x$를 계산하는 데 필요한 덧셈수의 한도는 $(1+\delta)n \cdot r/\log_2(n)$ 이며, 여기서 $1 > \delta > 0$ 이고, δ는 r과 n이 모두 무한대로 갈 때 0으로 간다. r > n 인 경우에, (이 경우에 대해) 더 좁은 한도가 존재하며, 이는 수직 분할을 적용한 결과이다.

$(1+\delta)n \cdot r/\log_2(r)$, 여기서 $1 > \delta > 0$ 이고, δ는 r과 n이 모두 무한대로 갈 때 0으로 간다.

이러한 과정의 효율성을 평가하기 위해서는, 본 발명이 관리 동작은 늘리고 덧셈은 줄이는지를 관찰해야 한다. 그러나, 덧셈이 복잡도의 주된 부분이어서, 본 발명을 통한 그 덧셈수의 감축은 탁월한 효과를 보여준다. 또한, 대부분의 관리 동작은 데이터 벡터의 처리를 시작하기 전에 행렬마다 한 번 수행된다. 이와 같이, 상기 제안된 방법은 실질적으로 전력 소비 및 회로를 절감시켜준다. 상기 0-1-이진 형태는 특히 주어진 행렬이 일정한 희소도 가질 때 효율적이다. 이것은 흔히 이 실시예의 응용이 본 발명의 실수 행렬 형태에서 기재한 바와 같이 일반적인 실수 행렬과 벡터의 곱의 계산에서 분포 산술과 결합될 때 마주친다.

또한 본 발명의 이러한 형태는 대수(algebraic) 코드(참조: [Shu Lin, Daniel J. Costello, Jr "Error Control Coding Fundamentals and Applications" Prentice-Hall, Inc., Englewood Cliffs, N.J., 1983])에 적용 가능하며, 여기서 코딩

및 디코딩 과정에서 Z_2-행렬은 Z_2-벡터(또는 Z_{2n}-벡터)와 곱해진다. 이것은 Z_2-행렬이 Z_{2n}-벡터)와 곱해지는 모든 상황에서 진실이다. 마지막으로, 상기 0-1-이진 실시예와 다음의 U-이진 실시예는 지표(characteristic) 2계를 제외한 모든 스칼라계에서 쉽게 서로 교환 가능하다. 특정한 경우마다 더 효율적인 실시예를 선택할 수 있다.

U-행렬

본 발명의 U-행렬의 바람직한 실시예(이하, "u-방법"이라고도 함)의 설명에 필요한 예비 개념은 벡터의 등가에 대한 개념이다. 동일한 차원의 2개의 0이 아닌 벡터는 그 중에 각각의 어느 한 벡터가 다른 벡터의 곱일 때 등가라고 한다. 이 실시예의 경우, 유의할 점은 2개의 U-벡터는 서로 동일하거나, 그 중에 하나가 다른 것의 (-1) 곱일 때 등가라고 한다. 다음의 바람직한 실시예는 상기 실시예와 대부분의 상세한 설명에서 유사하다. 주된 차이점은 다음의 본문에서 소거는 동일한 벡터보다는 등가 (행 또는 열) 벡터에 대해 행해진다는 점이다. 이것은 소거율을 높이고 따라서 효율성이 증가된다. 이 실시예를 일례를 통해 설명하고 그 사상의 본질을 증명한다.

예: 다음의 5x13 U-행렬 A와 13 차원 입력 벡터 x의 곱을 고려해보자.

행렬 A와 입력 벡터 x는 다음과 같이 주어진다.

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

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

그 곱의 결과는 다음과 같다.

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

이 과정의 제1 단계는 서로 등가인 라인이 있는지를 검사한다. 이 단계는 사실상 입력 벡터가 도착하기 전에 행해지는 1회성 동작이다. 상기 검사 결과, 주어진 행렬에서 서로 등가인 라인, 즉 라인 2와 라인 4[사실상 라인 4는 라인 2의 (-1)과 이 곱과 동일함]를 발견하였다. 따라서, $y_4=y_2$ 이다. 그러므로, y_4와 y_2를 모두 계산할 필요는 없으므로, y_4의 계산을 생략한다. 이와 같이, 라인 4를 행렬 A로부터 소거한다. 이렇게 생성된 행렬 A'는 다음과 같다.

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

대응하는 출력 벡터는 다음과 같다.

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

따라서, 다음과 같이 유지된다.

y'=A'・x

이것은 제1 감축이다. 다음 단계에서 곱 A'・x는 다음과 같이 A'의 열과 각각의 x 성분과의 곱의 합으로 분해된다.

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

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

다음에, 상기 합에서 각 열 벡터에 대한 정규화를 수행하고, 그 결과 각 변형된 벡터의 상위 성분은 1이 된다. 이와 같이, 벡터의 상위 성분이 (-1)인 경우에는 벡터와 그 계수에 모두 (-1)을 곱한다. 그러나, 벡터의 상위 성분이 1인 경우에는 그러한 변형을 가하지 않는다. 상기 합을 정규화한 형태는 다음과 같다.

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

$$(-x_7)\begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$$

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

이 정규화된 합에서 각 벡터의 상위 성분은 1이고, 상이한 벡터의 수가 줄어들었다. 다음 단계는 동일한 열의 계수를 모아 합하는 단계이다. 따라서 8개의 덧셈으로 다음과 같은 방정식을 얻는다.

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

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

이제, 새로운 계수를 다음과 같이 정의한다.

$$w_1 = (-x_1) + (-x_4) + x_6 + x_9, \quad w_2 = x_2 + (-x_7) + x_8 + (-x_{10}) + x_{11},$$

$$w_3 = -x_3 + x_{13}, \quad w_4 = x_5, \quad w_5 = -x_{12}$$

그러므로, 상기 방정식을 다음과 같은 형태로 기재할 수 있다.

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

이 단계에서 계수 w_1, w_2, w_3, w_4, w_5는 프로세서에게 통지된다. 다음 단계에서 이 벡터 방정식을 다음과 같이 2개의 방정식으로 수평 분할한다.

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

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

이제, 이들 방정식 각각에 대해 제1 반복 절차의 동일한 방식으로 독립적으로 처리한다. 상위 방정식은 정규화를 필요로 하지 않는데, 그 이유는 이전 방정식의 정규 상태를 물려받았기 때문이며, 따라서 하위 방정식만 정규화를 필요로 한다. 그러므로, 그 결과는 다음과 같다.

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

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

다음 단계에서 동일한 벡터의 계수를 모아 합한다. 따라서, 추가의 6개의 덧셈으로 다음과 같은 결과를 얻는다.

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

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

명백하게, 4개의 추가의 덧셈으로 벡터 y'를 얻었다. 최초의 출력 벡터를 구하기 위해서 이제는 단지 $y_4=-y_2$를 상기하면 된다. 이와 같이, 이 과정은 총 18개의 덧셈을 필요로 하였다. 전통적인 종래 기술의 방법은 상기 출력 벡터를 계산함에 있어서 총 60개의 덧셈을 필요로 한다는 것을 알 수 있다.

본 발명의 U-이진의 바람직한 실시예는 rxm U-행렬 A와 n 차원 입력 벡터 x의 곱에 효율적인 방법이다. 그것은 일반적으로 0-1의 바람직한 실시예보다 더 많은 이득을 제공하고 더 많이 응용될 수 있다.

행렬 A와 입력 벡터 x를 다음과 같이 가정하자.

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

$1 \le j \le n.$

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

입력 벡터의 엔트리는 실수 또는 복소수이거나, 2보다 큰 지표를 갖는 어떤 스칼라계에 속한 것일 수 있다. 곱 y=A・x의 결과를 계산하는 것을 목적으로 한다.

제1 단계는 등가 라인을 소거하여 주어진 행렬을 변형시키는 것이며, 이 단계는 예비 준비 단계의 일부일 수도 있다. i 라인이 j 라인과 등가라면, y_i가 $\pm y_j$와 동일하다고 추론할 수 있다. 그러므로, 본 발명의 제1 단계의 바람직한 실시예는 2개의 등가 라인 중 하나를 생략하기로 결정한다. 명확성을 위해서, 인덱스가 더 큰 라인을 항상 생략하는 것으로 정책을 정한다. 이와 같이, j>i 일 때, j 라인을 생략한다. 이 초기 동작은 이 과정의 마지막 단계, 즉 이 단계에서 알려진 y_j가 벡터 y의 자신의 위치에 다시 삽입될 때 번복된다. 이러한 소거 과정은 변형된 행렬에서 서로 등가인 라인이 없을 때까지 계속된다.

이 단계는 서로 등가인 라인이 있을 실질적인 가능성이 있을 경우에 수행된다. 이러한 경우는 흔히 U-방법의 다중곱(multi-product)(본 발명의 또 다른 형태) 응용의 경우이다. 또한, 이 단계는 $\log_2(r) \geq n$ 일 때에는 언제나 수행되는데, 그 이유는 이러한 조건 하에서는 언제나 서로 등가인 라인이 존재하기 때문이다. 그러나, 언제나 서로 등가인 라인이 존재하지만 고려해야만 하는 다른 조건도 있다. 예컨대, 아다마르 행렬의 하위 행렬에서의 경우이다. 표기의 복잡성을 피하기 위해서, 상기 변형된 행렬에 최초의 행렬과 동일한 부호 A를 그대로 부여하고, 차원 rxn의 부호도 그대로 유지한다.

다음 단계는 서로 등가인 열을 소거하는 단계이다. 이 단계는 덧셈에서 최소 비용으로 변형된 행렬의 수평 차원(즉, 열수)을 줄일 것이다. 이 단계와 관련된 모든 관리는 0-1-형태에서와 같이, 일반적으로 데이터 벡터가 도착하기 전에 예비 준비 단계로서 행렬마다 한 번 수행된다. 먼저, 다음과 같이, 곱 y=A・x를 행렬 A의 열과 대응하는 x 성분과의 곱의 합으로 표현한다.

다음과 같이, v_j(모든 j에 대해)를 A의 j 열로 하면,

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

이고, 따라서 $A \bullet x = x_1 v_1 + x_2 v_2 + \cdots\cdots + x_n v_n$ 이다.

다음에, 행렬 A의 각 열 벡터의 상위 성분을 검사하여, +1이 아닌 성분을 +1로 정규화한다. 그러므로, a_{1j}=+1 일 때에는 정규화가 필요없고, a_{1j}=-1 일 때에만 정규화를 수행하여 대응하는 j 열 벡터에 -1과 그 대응하는 계수 x_j를 곱한다. 그 결과 상기 합의 새로운 표현식을 얻을 수 있는데, 여기서 각 벡터의 상위 엔트리는 +1과 같다. 이러한 표현식에서 등가 열은 항상 동일하다.

다음에, 0-1-실시예에서와 같이, 동일한 열을 함께 그룹화하여 재배열하는데, 여기서 각각의 열은 공통 인수로서, 대응하는 스칼라 계수의 합과 곱해진다. 그러므로, 생성된 합은 반복되는 정규화된 열의 생략을 통해 정규화된 열 벡터의 시퀀

스로부터 구한 모든 상이한 정규화된 열 벡터의 서브시퀀스 $w_1, \cdots\cdots, w_m$(반복되지 않음)를 포함한다. 각각의 상이한 정규화된 열 w_j를 이 열과 동일한 모든 정규화된 열의 대응하는 정규화된 계수의 합인 계수 t_j와 곱한다. 명백한 것은 m ≤n 이고, 그 간격 n-m은 최초의 정규화된 열의 시퀀스 $a_{11}v_1, a_{12}v_2, \cdots\cdots, a_{1n}v_n$에서의 반복수라는 것이다.

전술한 과정은 다음과 같은 수학식으로 공식화될 수 있다.

$$A \bullet x = \sum_{1 \le j \le n} x_j v_j$$

$$= \sum_{1 \le k \le m} \sum_{j \text{ such that } a_{1j} v_j = w_k} (x_j a_{1j}) a_{1j} v_j$$

$$= \sum_{1 \le k \le m} \sum_{j \text{ such that } a_{1j} v_j = w_k} a_{1j} x_j) w_k$$

이제 1 ≤k ≤m 임을 고려하여 다음과 같이 정의한다.

$$t_k = \sum_{j \text{ such that: } v_j = w_k} a_{1j} x_j$$

본 발명의 이 부분에서 수반되는 주된 계산 태스크는 정규화된 최초의 것의 합과 같이 새로운 계수 $t_1, \cdots\cdots, t_m$를 계산하는 것이다. 그 대가는 n-m 개의 덧셈이다. 따라서, 곱 A・x는 다음과 같이 주어진다.

$$A \bullet x = \sum_{1 \le k \le m} t_k w_k$$

열 소거 과정을 통한 이득은 최초의 행렬 A의 구조에 따라 달라진다. 그것은 아다마르의 하위 행렬이나 주기적인 PN 행렬과 같이, 무선 통신에서 신호 벡터를 코딩 및 디코딩할 때 일반적으로 사용되는 일부 행렬에서 중요하다. 또한, 그것은 행렬 A에서의 라인수인 r이 행수 n보다 작을 때 중요하다. 특히, r ≤ log(n) 일 때, $m-n \ge n-2^{r-1} > 0$ 이다. 이것은 0-1-행렬에 대해서도 대부분 진실이다.

본 발명의 바람직한 실시예의 남은 부분인 수평 및 수직 분할, 반복 절차는 0-1의 대응부와 같다.

다음에, 본 발명에 의해 가능한 감축을 검토하기 위해서, 유의할 점은 먼저 덧셈이 복잡도의 상당한 부분을 차지한다는 점이다. rxn U-행렬 A($r \le \log_2(n)$)인 경우, s(n, r)로 표기되는 최악의 경우에서의 덧셈수는 다음과 같은 표현식으로 주어진다.

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

여기서,

$s(1) = -1$ $s(n\ ,\ 1) =$

$n - 1$

$s(2) = 0$ $s(n,2) =$

n

$s(3) = 3$ $s(n,3) =$

$n + 3$

$s(4) = 8$ $s(n,4) =$

$n + 8$

$s(5) = 19$ $s(n,5) =$

$n + 19$

$s(6) = 38$ $s(n,6) =$

$n + 38$

$s(7) = 75$ $s(n,7) =$

$n + 75$

$s(8) = 144.$ $s(n,8) =$

$n + 144$

$s(9) = 283$ $s(n,9) = n$

$+ 283$

다음과 같은 한도는 항상 유효하다.

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

전통적인 종래 기술의 방법은 (n-1)·r 개의 덧셈을 필요로 한다. s(n, r)/n은 n이 무한대로 가고 r이 일정할 때 1에 가까워지므로, 이 방법은 점근적으로 최적이 된다. 후술할 본 발명의 더 복잡한 다중곱 실시예는 최악의 경우에서의 덧셈수에 대한 정확한 한도를 요구한다. 이것은 다음의 회귀 공식에 의해 가능하다.

r이 짝수일 때 : $s(r) = 2^{r-1} + 2s(r/2)$

r이 홀수일 때 : $s(r) = 2^{r-1} + s((r+1)/2) + s((r-1)/2)$

A가 rxn U-행렬이고 행렬 A의 차원 rxn에 대한 제한이 없을 때, 본 발명에 의해 최악의 경우에서 곱 A·x를 계산하는데 필요한 덧셈수의 한도는 항상 $(1+\delta)n \cdot r/\log_2(n)$ 이며, 여기서 $1 > \delta > 0$ 이고, δ는 r과 n이 모두 무한대로 갈 때 0으로 간다. r > n 인 경우에, (이 경우에 대해) 더 좁은 한도가 존재하며, 이는 수직 분할을 적용한 결과이다.

$(1+\delta)n \cdot r/\log_2(r)$, 여기서 $1 > \delta > 0$ 이고, δ는 r과 n이 모두 무한대로 갈 때 0으로 간다.

그러나, 행렬 A에 특정 종류의 구조가 있는 경우에는 본 발명의 바람직한 실시예가 곱 A·x를 계산하는 데 필요로 하는 덧셈수는 급격히 떨어진다. 그것에 대한 가장 일반적인 조건은 현재의 본문의 범위를 넘는 것이나 일부 예는 언급될 것이다. 그러나, A가 nxn 아다마르 또는 주기적인 PN 행렬일 경우에는, 단지 $n \cdot \log_2(n)$ 개의 덧셈과 n개의 스칼라 메모리만을 필요로 하며, 이러한 관점에서 최적이다. 이것은 0-1-행렬에 대해서도 유효하다.

산업적 응용예

전술한 본 발명의 바람직한 실시예의 구현을 통해 이익을 얻을 수 있는 몇몇 기술 분야가 있다. 그러한 기술 분야 중 하나는 U-행렬과 벡터의 곱을 포함하는 몇몇 절차를 포함하는 이미지 처리 분야이다. 무선 통신 CDMA, IS-95 및 더 진보된 제3 세대 광대역 CDMA에는, U-행렬과 벡터의 곱을 사용하는 몇몇 과정이 있다. 이러한 과정은 본 발명에 의해 수행됨으로써, 에너지 소비, 회로 및 때로는 실행 시간이 줄어든다.

IS-95-B의 다중 코드 환경에서, 아다마르 64 중에서 8개의 라인은 신호 샘플을 엔트리로 하는 벡터가 곱해지는 U-행렬을 포함한다. 또 다른 응용은 역확산기가 수행하는 이웃 검출(neighbor detection)이다. 여기 저기에 소수의 아다마르 라인으로 구성된 행렬과 데이터 벡터의 곱이 존재한다. 또 다른 응용은 탐색기이다. 탐색기는 상호 스칼라곱을 계산하여 상호 상관도가 높은 시퀀스를 탐색한다. 상호 상관 시퀀스는 U-시퀀스인 PN 시퀀스로부터 추출되며, 그 스칼라와 데이터 벡터의 곱은 필수 계산이다. 초기 검사는 탐색기의 일례이다.

일반 소거 방법

본 발명의 상기 이진 형태는 다음의 과정들의 반복을 특징으로 한다. 0인 라인과 등가 라인(이것이 의미하는 것은 둘 중 하나가 다른 하나의 스칼라곱이라는 것이다)을 생략; 0인 열을 그와 관련된 입력 벡터의 스칼라 성분과 함께 생략; 등가 열을 함께 그룹화하여 합함; 상기 두가지 특징이 없어진 후에 그 행렬을 수평 및 수직 분할. 본 발명의 바람직한 실시예에 따라, 이러한 반복적인 일련의 단계는 모든 스칼라계에서 모든 행렬과 벡터의 곱에 적용될 수 있다. 본 발명의 이러한 폭넓은 바람직한 실시예를 일반 소거 방법(GEM)으로 부른다.

등가 열의 합산은 기본적으로 다음의 과정의 반복이다. $v_1, v_2, \cdots\cdots, v_n$을 변환 행렬 A의 열로 하고 $x=(x_1, x_2, \cdots\cdots, x_n)$을 입력 벡터로 하여, A・x를 계산하는 것을 목적으로 한다. 인덱스를 적절히 재배열한 후, 열 $v_{k+1}, \cdots\cdots, v_n$(2 ≤k < n)의 각각은 k 열의 스칼라곱이고, k < j ≤n 일 때, $v_j=z_jv_k$가 유효하다고 가정하자. 다음에 n 차원 벡터 x를 감축된 k 차원 벡터

$x'=(x_1, x_2, \cdots\cdots, x_{k-1}, x'_k)$ 여기서, $x'_k=x_k+x_{k+1}\cdot z_{k+1}+\cdots\cdots+x_n\cdot z_n$

로 대체하고, rxn 행렬을 열이 $v_1, v_2, \cdots\cdots, v_k$인 감축된 rxk 행렬 A'로 대체한다. 그러면, A'・x'=A・x가 유효하다. 본 발명의 바람직한 실시예에 따라, 필요하다면 인덱스 변경을 포함한 이러한 과정은 서로 등가인 열이 없을 때까지 반복된다. 실제로, 이 과정은 본 발명의 U-행렬 실시예에서 일어나는 정규화 과정을 일반화한 간단한 과정이다. 사실상 상기 모든 등가 열 감축 과정은 동시에 이루어질 수 있다. 이 단계가 행해질 때 그것을 전체 과정의 제1 반복 단계로 간주할 수 있다. 다음에 행렬을 수평 분할한 후, 각각의 분할된 행렬에 대해 상기 실시예에서와 같이 반복적인 방식으로 상기 열 소거를 반복한다.

U-실시예 및 다음 예에 의해 증명되는 바와 같이, 등가 열을 소거하는 효율적인 방법은 먼저 각각의 열을 최상위 성분으로 나누고 대응하는 계수를 동일한 최상위 성분과 곱하는 것이다. 그 결과, 각각의 변형된 열의 최상위 성분은 1이 된다. 그러나, GEM 응용에 있어서 때로는 변환 행렬의 열의 최상위 성분이 0이어서 상기 나눗셈이 불가능한 경우도 있다. 이러한 경우에는 각각의 0이 아닌 열을 최상위의 0이 아닌 성분으로 나누고 대응하는 계수를 동일한 최상위의 0이 아닌 성분과 곱하는 간단한 해결책이 있다. 그 결과, 각각의 변형된 열의 최상위의 0이 아닌 성분은 1이 된다. 이에 의한 바람직한 결과로서, 그것들이 동일하기만 하여도 변형된 행렬에 서로 등가인 열이 존재하게 된다.

이러한 본 발명의 바람직한 실시예가 효율성을 높이는 경우를 언급하기 위해서, 엔트리가 비교적 작은 유한 집합 S인 행렬을 고려해 보자. 사실상 이 집합 S는 유한 필드, 필드의 곱셈군의 유한 부분군이거나 곱셈으로 폐쇄된 필드의 유한 부분집합이고, 더 일반적으로는 필드의 임의의 유한 부분 집합일 수 있다. 그것은 전술한 이진 경우(여기서, S={0,1} 또는 S={1,-1})와, 다른 매우 일반적인 상황(여기서, S={0,1,-1} 또는 S={1,-1,j,-j} 또는 S={0,1,-1,j,-j})을 포함한다. 그 이득은 S의 크기를 줄이고 그 일반 규칙은 GEM이 덧셈수를 $\log_{|S|}(n)$ 배만큼 줄이는데, 여기서 n은 주어진 변환 행렬의 열 수이다. 적용 가능하다면, GEM도 후술할 복잡하고 일반적인 실시예와 효율성에 대해 비교해야만 한다.

이 방법을 설명하기 위해서, U_1-행렬로 불리는 U_1=def={1, -1, j, -j} 집합 중에서 엔트리가 선택되는 행렬에 대해 설명할 것이다. 이러한 종류의 행렬은 실제로 자주 나타난다. 그러나, 다시 강조해야 할 점은 GEM이 어떤 특정한 종류의 행렬로 한정되지 않는다는 점이다. rxn U_1-행렬과 벡터의 곱에서 덧셈수의 한도는 $C = n + 4^{r-1} + o(4^{r-1})$ 이다.

예: 다음의 3x15 U_1-행렬 A와 15 차원 복소수 입력 벡터 x의 곱을 고려해보자.

행렬 A와 입력 벡터 x는 다음과 같이 주어진다.

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

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

그 곱의 결과는 다음과 같다.

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

이 예에서 행해지는 모든 덧셈은 복소수 덧셈이다. 제1 단계는 서로 등가인 라인이 있는지를 검사한다. 서로 등가인 라인이 없다. 다음 단계에서 곱 A · x는 다음과 같이 A의 열과 각각의 x 성분과의 곱의 합으로 분해된다.

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

$$x_7 \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} +$$

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

$$\begin{bmatrix} 1 \\ -1 \\ j \end{bmatrix} + x_{15} \begin{bmatrix} j \\ -j \\ -1 \end{bmatrix}$$

다음에, 상기 합에서 각 열 벡터에 대한 정규화를 수행한다. 즉, 각각의 벡터를 그 상위 성분의 역수와 곱하고 각각의 계수를 대응하는 벡터의 동일한 상위 성분과 곱한다. 그 결과 각 변형된 벡터의 상위 성분은 1이 된다. 벡터의 상위 성분이 이미 1인 경우에는 그러한 변형을 가하지 않는다. 상기 합을 정규화한 형태는 다음과 같다.

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

$$+ (-1)x_7 \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} +$$

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

$$x_{14} \begin{bmatrix} 1 \\ -1 \\ j \end{bmatrix} + j\cdot x_{15} \begin{bmatrix} 1 \\ -1 \\ j \end{bmatrix}$$

이 정규화된 합에서 각 벡터의 상위 성분은 1이고, 열수(n)가 행수(r)에 비해 충분히 클 때(요건 : $n>4^{r-1}$) 상이한 벡터의 수가 상당히 줄어든다. 다음 단계는 동일한 열의 계수를 모아 합하는 단계이다. 따라서 7개의 덧셈으로 다음과 같은 방정식을 얻는다.

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

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

이제, 새로운 계수를 다음과 같이 정의한다.

$$w_1=x_1+(-j)\cdot x_2+j\cdot x_{10}+x_{14}+j\cdot x_{15},\quad w_2=j\cdot x_3$$

$$w_3=(-1)x_4+x_{12},\quad w_4=x_5+(-j)x_{13},\quad w_5=(-j)x_6+(-1)x_7,$$

$$w_6=j\cdot x_8,\quad w_7=(-1)x_9,\quad w_8=(-1)x_{11}$$

그러므로, 상기 방정식을 다음과 같은 형태로 기재할 수 있다.

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

$$\begin{bmatrix}1\\-j\\-1\end{bmatrix}+w_8\begin{bmatrix}1\\j\\1\end{bmatrix}$$

이 단계에서 계수 w_1,....., w_8는 프로세서에게 통지된다. 다음 단계에서 이 벡터 방정식을 다음과 같이 등가의 2개의 방정식으로 수평 분할한다.

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

$$\begin{bmatrix}1\\-j\end{bmatrix}+w_8\begin{bmatrix}1\\j\end{bmatrix}$$

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

일반적으로 양쪽 모두 벡터 방정식이다. 그러나, 현재의 축소형 예에서는 제2 방정식이 스칼라 방정식으로 변형되었다. 이제, 이들 방정식 각각에 대해 제1 반복 절차의 동일한 방식으로 독립적으로 처리한다. 상위 방정식은 정규화를 필요로 하지 않는데, 그 이유는 이전 방정식의 정규 상태를 물려받았기 때문이며, 따라서 제2 벡터 방정식이 스칼라 방정식으로 변형되지 않은 응용에서는 제2 벡터 방정식만 정규화를 필요로 한다. 상위 벡터 방정식을 축소한 결과는 다음과 같다.

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

이 단계의 경우 4개의 추가의 덧셈을 필요로 한다. 이제 간단한 방식으로 계산을 완료하면, 양쪽의 방정식에 대해 13개의 추가의 덧셈이 필요하다. 따라서 이 예에서는 본 발명의 바람직한 실시예가 총 24개의 덧셈을 필요로 하였다. 전통적인 종래 기술의 방법은 42개의 덧셈을 필요로 할 것이다. 이와 같이, 더 큰 규모로 감축할 수 있다.

토플리츠 행렬

본 발명의 바람직한 실시예를 설명하기 위해서 일례를 제시할 것이다.

예 : 길이 8의 시퀀스를 u=(1,1,-1,-1,1,-1,1,-1)이고 데이터 벡터 $x=(x_1, x_2, \cdots\cdots, x_{10})$의 3개의 연속된 가정을 검사하는 것이 바람직하다고 가정하자. 최대 상호 상관도를 탐색하는 것을 목적으로 한다. 다음의 3개의 합을 계산할 필요가 있다.

$$y_1 = 1 \cdot x_1 + 1 \cdot x_2 + (-1) \cdot x_3 + (-1) \cdot x_4 + 1 \cdot x_5 + (-1) \cdot x_6 + 1 \cdot x_7 + (-1) \cdot x_8$$

$$y_2 = 1 \cdot x_2 + 1 \cdot x_3 + (-1) \cdot x_4 + (-1) \cdot x_5 + 1 \cdot x_6 + (-1) \cdot x_7 + 1 \cdot x_8 + (-1) \cdot x_9$$

$$y_3 = 1 \cdot x_3 + 1 \cdot x_4 + (-1) \cdot x_5 + (-1) \cdot x_6 + 1 \cdot x_7 + (-1) \cdot x_8 + 1 \cdot x_9 + (-1) \cdot x_{10}$$

이것은 다음의 토플리츠 행렬 곱셈으로 표현할 수 있다.

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

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

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

다음 단계는 다음과 같이 보수 벡터를 모으는 것이다.

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

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

이 때 각각의 괄호 안의 항을 고려하여 보조 정리를 이용하면, x, y가 스칼라이고 v, u가 벡터일 때, 다음과 같이 된다.

$$xv + yu = \frac{1}{2}(x+y)(v+u) + \frac{1}{2}(x-y)(v-u)$$

그러므로,

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

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

유사하게,

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

그러므로 4개의 덧셈으로 다음과 같은 결과를 얻는다.

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

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

이제 상기 U-이진 방법을 적용할 수 있다. 따라서, 다음 단계는 각각의 열의 상위 성분이 1이 되도록 정규화하는 단계이다.

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

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

다음 단계는 공통 열의 계수를 모아 합하는 단계이다. 따라서 6개의 추가의 덧셈으로 다음의 결과를 얻을 수 있다.

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

$$\left.x_4)\right)\begin{bmatrix}1\\1\\-1\end{bmatrix}+\frac{1}{2}(x_2-x_{10})\begin{bmatrix}1\\1\\1\end{bmatrix}+\left((-x_3)+x_5\right)\begin{bmatrix}1\\-1\\-1\end{bmatrix}$$

편의성을 위해 다음과 같이 정의한다.

$$w_1=\frac{1}{2}(x_1+x_9)+(-x_6)+x_7+(-x_8)\,, \qquad w_2=\frac{1}{2}(x_1-x_9)+\frac{1}{2}(x_2+x_{10})+(-x_4)$$

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

따라서, 다음과 같이 기재할 수 있다.

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

다음 단계에서 이 벡터 방정식을 다음의 2개의 방정식으로 수평 분할한다.

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

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

제1 방정식에서 동일한 열을 모으면, 다음과 같이 2개의 추가의 덧셈을 얻는다.

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

희망하는 출력 벡터 y는 이제 5개의 추가의 덧셈으로 얻을 수 있다. 이와 같이, 본 발명을 구현한 방법은 이러한 계산을 수행함에 있어서 17개의 덧셈을 필요로 한다. 종래의 방법은 21개의 덧셈을 필요로 할 것이다. 명백한 것은 이러한 축소형 예를 통해 얻는 이득은 크지는 않지만, 대규모 응용에서의 이득은 상기 "스퀘어(square)"(비토플리츠) U-방법의 이득에 필적할 만하다.

다음은 본 발명의 이러한 형태의 일반적인 설정을 설명한다.

U-시퀀스를 $u_1,u_2,\ldots\ldots\ldots\ldots,u_n$ 이라 하고, 실수 또는 복소수 스칼라 데이터의 시퀀스를 $x_1,x_2,\ldots\ldots\ldots\ldots,x_{n+m-1}$ 으로 가정하자.

데이터 시퀀스의 요소는 실수 또는 복소수이거나 2보다 큰 지표의 스칼라계에 속한 것일 수 있다. 다음의 합을 계산하는 것이 바람직하다고 가정하자.

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

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

.

.

.

.

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

$m < \log_2(n)$ 일 경우 r=m이라 하고, $m \geq \log_2(n)$ 일 경우 본 발명의 바람직한 실시예는 m개의 합을 r개의 연속된 합으로 나누기 시작하며, 여기서 $r < \log_2(n)$ 이다. 제1 블록에서 설명된 방법과 동일한 방법으로 모든 블록을 처리한다. 첫번째 r개의 합은 토플리츠 행렬과 벡터의 곱으로 표현될 수 있다. 다음은 다음과 같은 rx(n+r-1) 토플리츠 행렬과 (n+r-1) 차원 벡터를 고려해 보자.

$$A=\begin{bmatrix} u_1 & u_2 & u_3 & u_4 & \cdots & u_n & 0 & \cdots & 0 & 0 \\ 0 & u_1 & u_2 & u_3 & \cdots & u_n & 0 & \cdots & \cdots & \cdots \\ 0 & 0 & u_1 & u_2 & \cdots & \cdots & u_n & 0 & \cdots & \cdots \\ 0 & 0 & 0 & u_1 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & & & \cdots & & & & \cdots & \cdots & \cdots \\ \cdots & & & \cdots & & & & \cdots & 0 & 0 \\ \cdots & & & \cdots & & & & & \cdots & 0 \\ 0 & 0 & \cdots & \cdots & 0 & u_1 & u_2 & \cdots & \cdots & u_n \end{bmatrix}$$

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

첫번째 r개의 합은 다음의 벡터로 주어진다.

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

여기서, y=A · x

본 발명의 바람직한 실시예의 중요한 착상은 상기 U-방법을 적용할 수 있도록 주어진 문제를 재편성하는 것이다. v_1, v_2,....., v_{n+r-1}을 그 순서에 따라 행렬 A의 열로 하자. 다음의 모든 "중간" 열 벡터는 U-벡터이다.

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

또한 주의할 점은 "사이드(side)" 열 벡터의 "매칭(matching)" 쌍의 합산 및 감산도 U-벡터이다.

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

$$\begin{bmatrix} u_1 \\ -u_n \\ -u_{n-1} \\ \cdot \\ \cdot \\ -u_{n-r+2} \end{bmatrix}$$

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

$$\begin{bmatrix} u_1 \\ u_2 \\ -u_n \\ \cdot \\ \cdot \\ -u_{n-r+3} \end{bmatrix}$$

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

$$\begin{bmatrix} u_{r-1} \\ u_{r-2} \\ \cdot \\ \cdot \\ u_1 \cdot \\ -u_n \end{bmatrix}$$

필요한 예비 준비 절차 후에 상기 방법을 도입할 수 있다. 상기에 따라 재배열하면 다음과 같다.

$$A \bullet x = \sum_{1 \le j \le n+r-1} x_j v_j$$

$$= \sum_{1 \le j \le r-1} x_j v_j + x_{n+j} v_{n+j} + \sum_{r \le j \le n} x_j v_j$$

다음에 본 발명은 x, y가 스칼라이고 v, u가 벡터일 때 다음의 규칙을 이용한다.

$$xv + yu = \frac{1}{2}(x+y)(v+u) + \frac{1}{2}(x-y)(v-u)$$

따라서,

$$A \bullet x = \sum_{1 \le j \le r-1} \frac{1}{2}(x_j + x_{n+j})(v_j + v_{n+j}) + \frac{1}{2}(x_j - x_{n+j})(v_j - v_{n+j}) + \sum_{r \le j \le n} x_j v_j$$

이 과정은 2r-2개의 덧셈을 필요로 하고, 달성된 형태는 사실상 rx(n+ r-1) U-행렬과 n+ r-1 벡터와의 곱이다. 이렇게

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

가 U-벡터

되는 이유는 상기 모든 벡터 이기 때문이다. 따라서 남은 계산은 U-방법으로 행할 수 있다. 실제 합산/감산 $x_j \pm x_{n+j}$ 을 제외한 모든 형태의 U-형태로의 변형은 입력 벡터가 도착하기 전에 "한 번의 작업"으로 처리된다.

최악의 경우에서의 덧셈수는 다음과 같다.

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

일반적인 환경, 즉 합 m이 계산되고 $m > \log_2(n)$ 일 때, 최악의 경우에서의 덧셈수의 한도는 $(1+\delta)(n+3\log_2(n)) \cdot m/\log_2(n)$ 이며, 여기서 $1 > \delta > 0$ 이고, δ는 m과 n이 모두 무한대로 갈 때 0으로 간다.

본 발명의 또 다른 바람직한 실시예에 따라 GEM의 구성 요소를 상기 방법과 통합할 수 있다. 그러한 경우에 일부 "사이드" 열은 사이드 열을 결합하는 제1 단계에 의해 처리되지 않은 채 남아 있게 된다.

(0,1,-1)-행렬

토플리츠 행렬은 더 일반적인 종류의 행렬(그 엔트리는 0, 1 또는 -1)의 일부이다. 그러한 행렬을 여기서는 (0,1,-1)-행렬로 부른다. 상기 토플리츠와 관련된 더 폭넓은 형태의 사상, 즉 그 방법을 이제 설명할 것이다. A는 rxn 차원의 (0,1,-1)-행렬이고 x는 n 차원 입력 벡터이며 곱 A · x를 계산하는 것이 바람직하다고 가정하자. 이 곱은 A의 열을 대응하는 x 성분과 곱한 것의 합으로서 표현할 수 있다. 따라서, v_j(모든 j에 대해)를 A의 j 열로 표기함으로써, 다음의 식을 얻는다.

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

A의 일부 또는 모든 열은 엔트리 0을 포함한다. 표기의 편의상, 첫번째 k 열 $v_1, v_2, \ldots\ldots, v_k$의 각각이 엔트리 0을 포함하고 (존재한다면) 남은 n-k 열 $v_{k+1}, v_{k=2}, \ldots\ldots, v_k$이 U-벡터(즉, 0의 엔트리가 없음)가 되도록 인덱스를 (방법적으로, 물론 프로세서를 사용하는 일없이) 배열한다. 분명한 점은 혼합된 벡터 $v_1, v_2, \ldots\ldots, v_k$의 각각은 2개의 U-벡터의 평균이라는 점이다. 그러므로, $1 \le j \le k$ 일 때 2개의 U-벡터 u_j, w_j가 존재하며, 따라서

$$v_j = \frac{1}{2}(u_j + w_j)$$

본 발명의 바람직한 실시예는 행렬마다 한 번 예비 준비 단계로 벡터 $u_1, w_1, \ldots\ldots, u_k, w_k$를 찾는다. 그러므로, 상기에 의해서 다음과 같이 된다.

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

그러나, 괄호의 오프닝과 함께 rx(n+k) U-행렬과 (n+k)-벡터의 곱의 표현식을 얻는다. 본 발명의 바람직한 실시예에 따라, 이 태스크는 본 발명의 상기 U-형태에 의해서 수행된다. 본 발명의 상기 토플리츠 형태는 이러한 폭넓은 형태의 특수한 경우이다.

산업적 응용예

이 탐색기는 일반적으로 토플리츠 행렬과 데이터 벡터의 곱으로 표현된다. 이것은 CDMA와 광대역 CDMA 모두에서 그러하다.

실수 행렬

본 발명의 또 다른 바람직한 실시예에 따라, 분포 산술법을 통해 어떠한 실수 선형 변환에 대해서도 계산 동작수를 줄일 수 있다. 앞으로 본 명세서에서 "실수 행렬 방법"으로 부를 것이다. 선형 변환은 엔트리가 고정된 수의 이진 디지트를 갖는 실수(정수일 필요는 없음)인 행렬로 표현된다. 행렬은 이진 행렬과 이진 계수의 합으로 분해된다. 다음 단계에서 본 발명의 이진 실시예를 적용한다. 이러한 방법을 설명하기 위해서 다음 예를 고려해 보자.

예 : (편의상) 정수 엔트리를 갖고 십진법으로 기재된 3x8 U-행렬 A를 다음과 같이 가정한다.

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

또한, 8 차원 입력 벡터를 다음과 같이 가정한다.

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

다음과 같이 3 차원 출력 벡터를 계산하는 것이 바람직하다.

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

프로세서는 정상적으로는 이진법으로 동작하므로, 행렬 A를 이진법으로 표현하면, 다음과 같다.

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

이 표현식은 다음과 같이 행렬을 3개의 이진 행렬의 합으로 표현하는 분포 산술법의 사용 가능성을 보여준다.

$$A=2^0\cdot A[0]+2^1\cdot A[1]+2^2\cdot A[2]$$

여기서,

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

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

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

따라서, 이러한 본 발명의 실시예에 따른 제1 단계는 중요도에 따라 A의 엔트리의 비트를 포함하는 이진 행렬 A[0], A[1], A[2]를 구성하는 것이다. 이러한 조건 하에서 바람직한 실시예에 의한 결과는 다음의 등식과 같다.

$$A\bullet x=2^0\bullet A[0]\bullet x+2^1\bullet A[1]\bullet x+2^2\bullet A[2]\bullet x$$

다음 단계는 다음과 같이 A[0], A[1], A[2]의 수평 체이닝에 의해 형성되는 24x3 이진 행렬 A*를 구성하는 것이다.

$$\mathbf{A^*=}$$

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

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

또한, 다음과 같이 벡터 x의 3개의 복제본의 각각과 상기 합으로부터 얻은 대응하는 이진 가중치와 함께 구성되는 24 차원 열 벡터를 형성하는 것이다. 이것은 입력 벡터가 도착되기 전에 행해진다.

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

주의할 점은 2^l과의 곱에는 단지 인덱스 이동 동작만이 요구된다는 점이다. 이 실시예의 열쇠는 다음과 같이 단정하는 것이다.

$$y=A\cdot x=A^*\cdot x^*$$

따라서 이어지는 단계는 0-1-방법을 통해 A* · x*를 계산하는 것이다. 그러므로, 다음 단계는 공통의 0이 아닌 열의 계수를 모아 합하여, 새로운 계수를 생성함으로써, 행렬의 크기를 축소한다.

그러므로 다음과 같이 유지된다.

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

여기서,

$$w_1=2^2\cdot x_7+2^2\cdot x_8+2^1\cdot x_4+2^1\cdot x_5+2^0\cdot x_5$$

$$w_2=2^2\cdot x_1+2^1\cdot x_6+2^0\cdot x_1+2^0\cdot x_7$$

$$w_3=2^2\cdot x_3$$

$$w_4=2^2\cdot x_2+2^1\cdot x_2+2^1\cdot x_3$$

$$w_5=2^1\cdot x_5+2^0\cdot x_2+2^0\cdot x_1+2^0\cdot x_3+2^0\cdot x_8$$

$$w_6=2^2\cdot x_1+2^1\cdot x_6+2^0\cdot x_1+2^0\cdot x_7$$

이것은 16개의 덧셈을 필요로 한다. 다음에, 상기 생성된 행렬을 행 분할한다. 그러므로,

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

$$y_3=w_1+w_3+w_5$$

이제, 다음과 같이 제1 방정식에서 공통 열의 계수를 합한다.

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

이 단계는 2개의 덧셈을 필요로 한다. 나머지는 2개의 추가의 덧셈으로 행해진다. 이 예에서는 변환을 계산함에 있어서 총 20개의 덧셈 연산을 필요로 하였다. 그러나, 종래의 방법은 총 29개의 등가 연산을 필요로 하며, 여기서 "등가"는 곱 연산을 포함하는 덧셈도 고려한다는 것을 의미한다.

분명한 점은 이것은 현저한 감축의 예가 아니라, 본 발명의 실시예의 개념을 쉽게 설명하기 위한 것이다. 그러나, 행렬의 파라미터(각각의 엔트리의 디지트 차원 및 디지트수)가 클 경우에는 본 발명을 통해 상당한 감축을 달성할 수 있다.

일반적으로 이러한 조건 하에서 본 발명의 바람직한 실시예는 실수의 비제한 선형 변환의 효율적인 계산에 관한 것이다. 변환을 나타내는 rxm 행렬 A는 다음과 같다.

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

여기에서 행렬의 엔트리는 실수이다. A · x를 계산하는 것이 바람직하며, x는 실수 또는 복소수 스칼라 엔트리의 n차원 벡터이며,

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

로 표시된다.

행렬 A의 엔트리는 소수점 이전과 이후에 고정된 갯수의 디지트(digit)를 갖는 이진 기반으로 표현된다. 본 발명의 두가지 선호되는 실시예는 옵션인 솔루션(solution)으로 제공될 것이고, 그리하여 서로간의 선택은 변환 행렬의 구조에 의존한다. 제1은 0-1 분해(decomposition)로 호칭되고, 제2는 U 분해로 호칭된다.

행렬 A의 엔트리는 2^{m_1}에 의하여 한계가 되며, 적절한 양의 정수 m_1 과 m_2에 대하여 소수점 위로 m_2 디지트의 정확도를 갖는 것으로 가정한다. 본 가정은 본 발명의 범위를 제한하지 않는데, 왜냐하면 프로세서가 마주치는 모든 스칼라는 소수점 전후에 유한한 갯수의 디지트를 갖기 때문이다. $m = m_1 + m_2 + 1$이라 놓는다. 분산 연산이 적용되어 곱 A · x가 위의 예에서 다루어진 유형의 m개 이진 곱으로 분해될 것이다.

먼저 본 발명은 행렬의 엔트리가 음수가 아닌 경우를 설명하며, 분해(decomposition)는 0-1 기반이다. 2^{m_1}에 의하여 한계가 되며, 소수점 위로 m2 디지트를 가지는 실수 t는 다음과 같이 표현된다.

$$t = \sum_{-m2 \le k \le m1} t_k \cdot 2^k$$

여기에서 각 t_k 는 0 또는 1이다. 이는 표준 이진 표현이다. 전형적인 분산 연산의 인수(argument) A는 아래와 같이 m개의 rxn 0-1 이진 행렬의 합으로 분해될 수 있다.

$A= \sum_{-m2 \le k \le m1} 2^k \cdot A[k]$

상기의 합 A[m1]은 최상위 비트(MSB)의 0-1 행렬이다. A[-m2]는 최하위 비트(LSB)의 0-1 행렬이다. 일반적으로, 각 매트릭스 A[k]는 엔트리가 A의 엔트리의 k번째 비트로 구성된 0-1 행렬이며, 각 비트는 이의 대응하는 엔트리에 위치한다. 분산 법칙에 의하여 다음과 같이 된다.

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

이는 본 발명의 현재 실시예의 기초가 된다.

다음으로 A*를 상기 합의 이진 행렬을 합에서 나타나는 순서대로 수평으로 정렬하여 구성된 rx(m · n) 0-1 행렬이라고 하면,

A* = [A[-m2],.....,A[0],.....A[m1]] 이다.

이는 입력 벡터의 도착이 시작되기 전에 주어진 임의의 행렬에 대하여 한번 수행될 것이다.

추가로, m · n 차원의 열 벡터 x*가 다음과 같이 주저진다.

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

고려중인 실시예를 강조하는 핵심적 주목 대상은 다음과 같다.

$A \cdot x = A^* \cdot x^*$

우변(the latter)은 rx(m · n) 0-1 행렬과 원래 벡터의 천이된 m개의 복제로 구성된 (m+ 1) · n 차원 벡터의 곱이다.

곱셈 $A^* \cdot x^*$의 계산은 본 발명의 0-1-이진-실시예에 의해서 수행된다. 각각의 정수 2승에 의한 곱셈은 비트를 이동함으로써 구현될 수 있으므로 단지 조금의 추가적인 복잡도가 더해질 뿐이다.

이러한 본 발명의 실시예는 곱셈의 결과의 복잡도를 상당히 줄일 수 있다.

$l=\log_2(mn)=\log_2(m)+\log_2(n)$이고 C(A)가 본 발명에서 $A^* \cdot x^*$를 계산하기 위해서 추가적으로 필요한 수라고 하면,

만약 $l \ge r$이면

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

특히 이 경우에서 개략적인 경계는 $C(A) < 2m \cdot n.$이라는 것을 알수 있다.

만약 $l \geq r$이라는 가정을 버리면,

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

이고 1 > δ>0 이고 δ는 m · *n*과 r이 무한대로 되는 경우 0으로 된다.

A · *x*의 곱셈을 계산하는 방법에 대한 종래의() 선행기술은 곱셈 연산을 포함하는 덧셈을 고려했을때 평균적으로 *n* · *r* · *m/2*번의 덧셈에 해당하는 것이 필요하다.

일부 경우에서는 특히 r=1인 경우 또는 r이 비교적 작은 경우 위에서 언급된 실시예의 다른 변형이 더 절약을 가능하게 하기 위해서 바람직할 수 있다. r=1인 경우에 이러한 문제는 두 벡터 사이의 스칼라 곱으로까지 감소될 수 있다는 것을 주지하여야 한다. 이것은 그 자체로 과학기술에서 보편적인 계산이고 이러한 것의 효율적인 수행이 많은 도움이 된다. 이러한 변형에 있어서 A^{**} 행렬은 다음과 같은 수직 시퀀스로 $A[-m_2], A[-m_2], \ldots\ldots\ldots, A[0], \ldots\ldots\ldots A[m_1]$ 인 연쇄(chaining) 행렬로 형성된다.

따라서 *r* · *(m+ 1)xn* 0-1-행렬은 다음과 같이 주어진다.

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

다음으로, $A^{**} \cdot x$의 계산은 0-1-행렬에 대한 본 발명의 바람직한 실시예에 의하여 이루어진다. $A^{**} \cdot x$의 곱은 $A[-m_2] \cdot x, \ldots\ldots, A[0] \cdot x, \ldots\ldots\ldots, A[m_1] \cdot x$ 등의 모든 곱들을 포함한다. 따라서, 이진 곱셈에 있어서의 쉬프트를 이용하여 원하는 결과를 다음과 같은 합에 의하여 얻을 수 있다.

$$y = \sum_{-m_2 \leq k \leq m_1} 2^k \cdot A[k] \cdot x$$

이는 음수가 아닌 원소를 가진 행렬 부분에 대한 결론이다.

행렬 *A*가 실수이고 양/음의 부호를 갖는 원소를 갖는 경우, 행렬 *A*는 음이 아닌 원소를 가진 두 행렬의 차, 즉 $A = A_1 - A_2$로 표현될 수 있다. 이러한 표시를 따를 경우, 지금 논의중인 본 발명의 특징에 있어 $y=A \cdot x$의 계산은, $y_1 = A_1 \cdot x$ 및 $y_2 = A_2 \cdot x$의 처음 각각을 별개로, 또는 위의 방법에 의해 결합된 형태로 계산하는 것에 의해서 수행된다. 그리고 최후 단계는 $y=y_1 - y_2$를 수행함으로써 이루어진다.

상기 실수 행렬과 관련된 본 발명의 바람직한 실시예에 있어서의 0-1-이진 옵션은 분해 이진 행렬의 경우 특히 효율적이다. 즉 $A[-m_2], A[-m_2+1], \ldots\ldots, A[0], \ldots\ldots, A[m1]$는 다소 드물다. 이는 다양한 사이즈를 가진 *A* 의 원소에 대한 귀결이자, 상기 포인트를 넘는 이진 숫자의 불균일성 때문이다. 이러한 경우 상기 균일 디지털 포맷에 대한 제로 패딩(zero padding)의 필요성은 더 높은 수준의 희소성을 낳게 한다.

U-이진 분산 연산에 기초한 본 발명의 바람직한 실시예의 또다른 형태는, 이하 부분에서 설명할 것이다. 본 발명의 이러한 형태는 양/음 부호를 가지며 더 빠른 *U*-방법에 기초한 원소를 갖는 행렬에 더욱 적용하기 쉽다는 장점을 가지고 있다. 실질적으로 이는 상기 행렬의 원소가 균일한 사이즈 및 정밀도를 갖는 경우 상기 0-1 버전에 대해 더욱 효율적이다.

이하의 전개는 본 발명의 설명을 계속하기 위해 필요하다. 2^{m1} 으로 바운드된 실수 t 및 상기 포인트를 넘는 m_2 디지트는 *U*-이진 합으로서 다음과 같이 표현될 수 있다.

$$t = \sum_{-m_2-1 \le k \le m_1-1} s_k \cdot 2^k + s_{m_1} \cdot (2^{m1} - 2^{-m2-1}).$$

여기서 모든 s_k 는 $-m_2-1 \le k \le m_1-1$에 대해서 ±1이고 t가 음수가 아닌 경우 $s_{m1}=1$, 음수인 경우 $s_{m1}=-1$이다.

따라서 양/음 부호를 갖는 원소로 된 rxn 실수 행렬 A는 다음과 같이 하여 *U*-행렬의 합으로 분해될 수 있다.

$$A = \sum_{-m_2-1 \le k \le m_1-1} 2^k \cdot A[k] + (2^{m1} - 2^{-m2-1}) \cdot A[m_1].$$

여기서 각 행렬 $A[k]$는 rxn *U*-행렬이다.

여기서 A^* 는 $rx((m+1)\cdot n)$ *U*-행렬이라 하자.

$$A^* = [A[-m_2-1],\ A[-m_2],\ldots\ldots\ldots\ldots,A[0],\ldots\ldots\ldots\ldots,A[m_1-1],\ A[m_1]]$$

이 행렬은 입력 데이터 벡터가 토착하기 전까지 각 행렬에 대해 한 번의 작업으로서 구성된다.

이에 더하여 $(m+1)\cdot n$차 열 벡터 x^* 은 다음과 같이 정의된다.

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

이 경우 다음이 성립한다.

$$A \cdot x = A^* \cdot x^*$$

이는 $rx((m+1)\cdot n)$ *U*-행렬과 $(m+1)\cdot n$차 벡터의 곱이다. 이 곱의 계산은 본 발명의 *U*-행렬 실시예를 적용하여 이루어진다.

0-1 이진 분해와 관련된 이상의 방법에서처럼, 이 경우도 $r=1$ 또는 작은 r인 경우의 수직 버전이 있다. 이는 앞서 설명한 것에 전적으로 유사하며, 세부사항을 반복할 필요는 없겠다.

$$l = log((m+1)\bullet n) = log(m+1) + log(n)$$

라 하자. 이는 $l \geq r$일 경우에 성립하며, 상기 방식을 실행하는데 필요한 덧셈의 수는 다음에 의해 바운드된다.

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

위에서와 같이, $C(A)$는 $A \cdot x$를 계산하기 위해 상기 U-실시예에서 요구되는 덧셈의 수로 정의된다. 더욱 일반적으로 하자면, 다음이 성립한다.

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

여기서 $1>\delta>0$ 이며 δ는 $(m+1)\cdot n$ 및 r이 무한대로 감에 따라 0으로 가까와진다.

산업상 응용예

선형 변환은 보통 기술 및 과학의 거의 모든 분야에서 사용된다. 통신 기술에 있어서 본 발명의 실수 행렬 특징에 관한 적용예는 멀티 유저 디텍터(MUD) 행렬{예컨대 디코릴레이터(decorrelator) 또는 MMSE(minumum mean square error) 행렬 등}과 디스프레더(despreader)의 출력 벡터를 곱한 값을 포함한다. 이는 또한 최소 자승의 계산에도 적용된다. 이산 푸리에 변환(DFT)가 전부 또는 일부 계산되는 FIR(Finite Impulse Response) 필터가 그것이다. 특히 부분 DFT 및 FFT가 효율적이지 못한 경우의 소형 및 중형 사이즈 FIR 필터는 본 발명에 의해 성능이 개선될 수 있는 선형 변환의 또다른 종류이다. 이는 특히 부분적으로 계산될 때 또는 고차원의 빠른 알고리즘이 그다지 유용하지 않을 정도로 너무 사이즈가 크지 않은 경우에 특히 그러하다.

FIR을 사용하는 처리 회로와 같은 일부 디지털 신호 처리 응용예에 있어서는, 상대적으로 긴 두 벡터의 코릴레이션(correlation)이 필요하다. 이들 벡터 중 하나는 FIR 필터의 탭을 나타내는바, 이는 제2벡터의 연산을 하며, 필터링되어야 할 입력을 나타낸다. 부분 컨벌루션(convolution)로 이루어지는 필터링 연산은 벡터와 토플리츠(Toplits) 행렬의 곱으로 표시된다. 이는 본 발명의 실수 행렬 특징에 의해 효율적으로 이루어진다.

복소 행렬

예 : 다음과 같은 합을 계산하는 것을 원하는 경우라 가정하자.

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

여기서 입력 스칼라 $x_1, x_2, \ldots x_6$은 복소수이다.

종래의 선행기술에서는 66번의 실수 덧셈이 필요하다. 이것은 (± 1 $\pm j$)같은 종류의 요소를 복소수로 곱하는 것에는 2번의 실수 덧셈이 필요로 하고 두개의 복소수를 더하는 것은 2번의 실수 덧셈이 필요하다는 것을 염두에 두면 알수 있다.

이러한 계산을 수행하는 데 있어서 본 발명의 바람직한 실시예의 두가지 주요한 대안이 제시된다. 제1의 바람직한 실시예는 위상 회전 및 GEM으로 일컬어질수 있는 것으로, 다음과 같은 사실을 사용한다.

$$\frac{1}{2}(1+j)\bullet(1+j) = j$$
$$\frac{1}{2}(1+j)\bullet(1-j) = 1$$
$$\frac{1}{2}(1+j)\bullet(-1+j) = -1$$
$$\frac{1}{2}(1+j)\bullet(-1-j) = -j$$

따라서 위의 합들을 본원에서 소위 위상 회전으로 부르는 $\frac{1}{2}(1+j)$ 로 곱함으로써, 세트에서부터 다음과 같은 계수를 얻는다.

$\{1,-1,j,-j\}$:

$$\frac{1}{2}(1+j)\bullet y_1 = j\bullet x_1 + 1\bullet x_2 + (-j)\bullet x_3 + (-1)\bullet x_4 + 1\bullet x_5 + (-1)\bullet x_6$$
$$\frac{1}{2}(1+j)\bullet y_2 = (-1)\bullet x_1 + j\bullet x_2 + j\bullet x_3 + (-j)\bullet x_4 + (-j)\bullet x_5 + 1\bullet x_6$$
$$\frac{1}{2}(1+j)\bullet y_3 = 1\bullet x_1 + (-j)\bullet x_2 + (-j)\bullet x_3 + j\bullet x_4 + (-1)\bullet x_5 + j\bullet x_6$$

본 발명의 바람직한 실시예에 따르면 이러한 합들은 GEM에 의해서 계산된다. 이러한 예에 있어서 크기가 작기 때문에 이 경우에 있어서 종래의 방법과 비교했을때 이득이 작지만, 더 큰 차원에 대해서는 상당하게 된다. 이 예와 같이 차원이 작을 때 상기 위상 회전 단계 이후에 종래의 다른 계산방법이 적용될 수 있다. 최종적으로 각 합의 결과는 $(1-j)$로 곱해져서 원하는 결과를 얻게 된다. j 또는 (-1)로 곱하는 것은 "유기적(organizational)" 연산이며 시간과 노력이 미미하다는 것을 주지하여야 한다.

바람직한 실시예의 제2의 대안은 복소-U-방법이라 일컬어지는 것이다. 이것은 합을 다음과 같이 각각의 계수의 괄호를 열어서 분리한다.

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

나머지는 U-방법을 적용하여 계산된다. 이것은 테이블 s(n,r)에 의해서, s(12,3)= 12+ 3 =15이기 때문에, 복소수의 덧셈이므로 최대 30번의 실수 덧셈을 필요로 한다.

위의 예를 통해서 기본적인 원리를 기술한 후 이제 일반적인 경우에 대해서 상세한 설명을 한다. 복소 행렬에 대해서 본 발명의 바람직한 실시예를 기술하기 위해서 세트가 계수: U_1= $\{1,-1,j,-j\}$, U_2= $\{1+j,1-j,-1+j,-1-j\}$를 사용한다고 간주하자. U_1-벡터 또는 U_1-행렬은 U_1 내에서 엔트리를 가진다. 마찬가지로 U_2-벡터 또는 U_2-행렬은 U_2 내에서 엔트리를 가진다. 그러한 행렬과 벡터는 무선 응용에 있어서 보편적이다. 뒤부분에서 U_2를 복소수로 곱하는 것은 2개의 실수 덧셈이 필요로 하는데 비해 U_1을 복소수로 곱하는 것은 비교적 적은 복잡도를 가진다는 것을 고려해야 한다.

풀어야 할 첫번째의 계산상 문제는 다음과 같다: rxn U_2-행렬 A가 주어지고 n-차원 복소-열 입력 벡터 x에 대해서 $y = A \cdot x$의 곱을 계산하는 것이 필요하다고 가정하자. r=1인 경우, 실제로 스칼라 벡터인 경우,가 포함된다. 이러한 계산에 대해서 2가지 접근이 제시된다. 각각에 대해서 적절한 것이 바람직하다. 동일한 방법이 약간의 변형으로 데이터 벡터가 실수일 경우에 적용이 가능하다.

첫째로 본 발명의 바람직한 실시예인 위상 회전 및 GEM이 제시된다.

$B=\frac{1}{2}(1+j)A$ 이고 $z=\frac{1}{2}(1+j)y$ 로 가정하면 B는 rxn U_1-행렬이고 z=B・x이다. 다음으로 z=B・x의 합이 GEM에 의해 계산된다. 일단 z가 계산되면, 출력 벡터 y는 $y=(1-j)\cdot z$의 곱에 의해서 얻어진다. 초기 위상 회전 단계의 결과에 기인한 종래의 방법에 비해 이득은 2r・(n-1)갯수의 덧셈을 절약할 수 있다. 이러한 이득은 r=1인 즉 스칼라 곱셈인 경우에 대해서도 존재한다. 또한 GEM의 적용을 통해서 이득이 얻어진다.

본 발명의 제2의 바람직한 실시예는 U-복소-방법이라 일컬어지는 것이다. 우선 할 일은 A를 다음과 같이 A_1과 A_2가 U-행렬인 A=A_1+jA_2로 표현하는 것이다. 그후 A・x = A_1 ・x + jA_2 ・x라는 등식을 고려한다. 본 등식은 A・x 가 rx2n U 행렬 A* = [A1, A2]와 2n 복소-차원 열벡터 $x^*=\begin{bmatrix} x \\ jx \end{bmatrix}$ 의 곱을 통해 계산될 수 있음을 의미한다. 이는 다음 등식으로 표현된다. A・x = A*・x*. 이제 A*・x*는 U 방법(U method)에 의하여 계산될 것이다. 본 발명의 바람직한 실시예 내에 또 다른 대안이 존재하며, 이는 r이 작을 경우에 타당하다. $A^{**}=\begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$라고 하면, 이는 2rxn U 행렬이다. 그리고나서 곱 A**・x를 계산하기 위해 U 방법을 적용한다. 사실 이것은 곱 y1=A1・x 및 y2=A2・x 양자 모두를 계산한다. 상기 절차는 합 y = y1 + y2로 완료된다.

상기 문제의 변형이 CDMA에서의 PN 상관기의 toplitz 행렬 표현을 포함하는 일부 응용예에 발생할 수 있다. 또한 이러한 설정에서 상기 행렬은 0 엔트리를 가질 수 있다. 따라서, U'_1= {0, 1, -1, j, -j} 및 U'_2= {0, 1, -1, j, -j} 라고 한다. U'_1 벡터 또는 U'_1 행렬은 U'_1 에 자신의 엔트리를 갖는다. 마찬가지로 U'_2 벡터 또는 U'_2 행렬은 U'_2 에 자신의 엔트리를 갖는다. rxn U'_2 행렬 A가 주어 지고, n차원 복수-열 입력 벡터 x이다. 목표는 곱 y=A・x 를 계산하고자 하는 것이다. r = 1인 경우가 포함되며, 이는 사실상 스칼라(scalar) 곱이다.

위상 회전 및 본 발명의 GEM 선호되는 실시예를 먼저 논의할 것이다.

$B=\frac{1}{2}(1+j)\cdot A$ 및 $z=\frac{1}{2}(1+j)y$ 라고 하면 B는 rxn U'_1 행렬이고 z = B・x이다. 곱 z = B・x는 GEM의 응용에 의하거나, 차원이 낮은 경우에는 어쩌면 좀더 종래적인 방법에 의하여 계산된다. 따라서 z가 일단 계산되면, 출력 벡터 y는 곱 y = (1-j)・z에 의하여 얻을 수 있다.

본 발명의 선호되는 또 다른 실시예에 따라서, A는 먼저 합 A = A1 + A2으로 표현되며, A1과 A2는 (0,1,-1) 행렬 이며, Toplitz일 수도 있다. 그리고나서 등식 A = A1 + jA2에 의해서, 곱 A・x는 rx2n (0,1,-1) 행렬 A* = [A1, A2]와 2n 복소-차원 열벡터 $x^*=\begin{bmatrix} x \\ jx \end{bmatrix}$의 곱으로써 계산될 수 있다. 마지막으로, 곱 A*・x*가 Toplitz에 의하거나 또는 본 발명의 더 일반적인 (0,1,-1) 관점에 의하여 계산될 것이다.

본 발명의 또 다른 선호되는 실시예(옵션임)는 비-Toplitz 대응물(counterpart)을 반영하여, r이 작은 경우에 효율적이다. $A^{**}=\begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$라고 하면, 이는 2rxn (0,1,-1) 행렬이다. 그리고나서 곱 A**・x를 계산하기 위해 본 발명의 (0, 1, -1) 관점(aspect)을 적용한다. 사실 이것은 곱 y1=A1・x 및 y2=A2・x 를 계산한다. 마지막으로 상기 절차는 합 y = y1 + y2로 완료된다.

복잡한 본 장(chapter)은 이제 일반 복소수 rxn 행렬 $A\in C^{rxn}$ 과 실수 또는 복소수 n 차원 벡터 x의 곱을 계산하기 위한 방법으로 결론지어질 것이다. 본 발명의 선호되는 일 실시예에 따라, 먼저 A를 합 A = A1 + A2로 표현하며, 여기에서 A1

과 A2는 실수 행렬이다. 다음으로 등식 A · x = A1 · x + A2 · x에 의하여 A · x 가 rx2n 실수 행렬 A* = [A1, A2]와 2n 차원의 열벡터 $x^*=\begin{bmatrix} x \\ jx \end{bmatrix}$의 곱으로써 계산될 수 있다. 왜냐하면 A · x = A* · x*이기 때문이다. 마지막으로 A* · x*가 실수 행렬 방법에 의하여 계산될 것이다.

본 발명의 또 다른 선호되는 실시예(옵션임)에 따르면, $A^{**}=\begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$라고 놓고, 이는 2rxn 실수 행렬이다. 그리고나서 실수 행렬 방법을 적용하여 곱 A** · x를 계산한다. 이는 y1 = A1 · x 와 y2 = A2 · x를 계산한다. 마지막으로 본 절차는 합 y = y1 + j · y1로 완료된다.

따라서 U2 계수를 갖는 toplitz 행렬과 벡터의 곱은 위의 toplitz 기법의 응용에 의하여 행해진다.

TOPLITZ 행렬, (0,1-1) 행렬, 및 복소 행렬 기법의 산업적 응용예

IS-95 검색기(searcher) : IS-95 CDMA 시스템은 이동 수신기에 데이터를 전송하기 위한 주파수 스펙트럼의 동일한 세그먼트를 동시에 사용하기 위해 작은 지리적 위치내에서 다수의 상이한 기지국(base station)을 허용한다. 상이한 기지국으로부터의 데이터가 구별될 수 있는 방법은 전송되는 데이터를 확산시키기 위해 사용되는 PN 시퀀스에 의한 것이다. 각 기지국은 PN 시퀀스의 상이한 위상을 전송한다. 이동 수신기에 있는 검색기 메커니즘의 임무는 주위 기지국으로부터 전송되는 상이한 파일롯 신호를 그들의 PN 위상을 조정하여 식별하는 것이다. 이는 또한 동일한 기지국에서 도착하는 몇몇 다중경로 신호(에코를 의미함)를 구별하는 데도 이용된다. 유사한 프로세스가 초기 동기화 절차에도 적용된다.

검색기는 각 가정에 대하여 논리적으로 생성된 PN 시퀀스를 가지고 수신된 신호를 부분 상관함으로써 다수의 가정(hypothesis)을 검사할 것이 요구된다. 그리고나서 시퀀스는 각 가정에 대하여 천이되어, 고정된 숫자의 신호 요소(칩, chip)동안에 상관이 수행된다. 일반적으로 검색기는 주어진 윈도우에서 모든 가정을 검색하는 것이 요구되며, 상기 윈도우에서 매번 시퀀스가 하나씩 천이된다. 검색기는 DS-CDMA에서 행렬 A를 구성함으로써 구현될 수 있으며, 상기 행렬의 행(row)은 전술한 천이 PN 시퀀스로 구성된다. 검색 결과는 벡터 $y=A \cdot x$의 형태로 주어지게 되는 바, 여기서 x는 단일 칩 주기 동안에 샘플링된 입력 신호를 말한다. 바람직한 실시예에 따라, 벡터 y의 자원 소비량이 감소하도록 앞서 언급했던 효율적 선형 변환을 위한 신규한 알고리즘이 구현되었다. 본 발명의 여러 응용예는 광대역 CDMA에 관한 표준안을 탐색해 가는 과정에서 또한 유용하게 사용될 수 있을 것이다.

멀티-프로덕트(Multi-Product)

본 발명의 또 다른 실시예는 벡터곱(vector product)에 의한 U-행렬의 부분합이 바람직한 경우와 관련되어 있다. 이는 CDMA 통신의 응용예에서 나타날 수 있는 바, 이 경우 서로 다른 속도(확산 인자)을 갖는 여러 코드가 동시에 테스트된다. 본 실시예를 연구하는 것은, 이미 본 발명의 전술한 특징들을 실질적으로 구사할 수 있게 된 독자들에게 더욱 유용할 것이다. 이 다소 복잡한 방법에 대한 최초의 개념이 생길 수 있도록, 상세한 예를 많이 들지 않고 이하의 예를 소개하도록 하겠다. 그러나 본 실시예의 모든 특징을 설명할 수 있는 적합한 사이즈의 예는 존재하지 않는다. 독자는 본 발명의 요약에 있어서의 항목 6을 또한 참조하는 것이 좋다.

예시: 다음 5x8 U-행렬을 살펴보자.

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

그리고 8차 입력 벡터는 다음과 같다.

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

(멀티 프로덕트 용어상에 있어서) 제1행 및 제2행의 확산 인자는 2, 제3행 및 제4행의 확산 인자(spreading factor)는 4, 그리고 제5행의 확산인자는 8이라 가정하자. 이는 첫 두 행에서 모든 두 연속한 원소가 합하여지고, 제3행 및 제4행에서는 모든 네 연속한 원소가 합하여지고, 제5행에서는 모든 행이 합하여진다. 이러한 약속에 의해 확산 인자는 감소하지 않으며, 각 행의 확산 인자는 이전 행의 확산 인자와 같거나 큰 값인 것으로 나타나게 된다. 이 용어는 나중에 정확히 정의하게 된다.

이상에 의해 이하의 합이 계산될 수 있음을 알 수 있다.

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

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

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

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

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

우선 행렬의 가로측 차수에 가장 낮은 확산 인자가 반영된, 4개의 4x2 U-행렬로 나뉘어지게 된다. 그리고 나서 이러한 U-방법이 지극히 기본적인 수준에서 적용되는 바, 이는 이러한 새 특징에 의해 덧셈이 어떻게 줄어드는가를 보여주기 위함이다. 따라서, 등가인 행들을 제거하는 것에 의해, 각 행의 모든 첫 합을 계산하기 위해서는 오직 두 번의 덧셈이 필요할 뿐이다.

$x_1 + x_2$

$-x_1 + x_2$

$x_1 - x_2$

$x_1 + x_2$

$-x_1 - x_2$

이와 유사하게 오직 두 번의 덧셈이 각 행의 두 번째 합을 위해 필요하다.

$-x_3 + x_4$

$-x_3 - x_4$

$x_3 + x_4$

$-x_3 + x_4$

$x_3 + x_4$

또한 계속 이와 같게 된다. 따라서, 이 부분에 있어 총 8번의 덧셈이 필요하게 된다. 필요한 제3,4행의 합을 위해 4번의 추가적인 덧셈이 필요하며, 필요한 제5행의 합을 위해 또 다른 4번의 추가적인 덧셈이 필요하다. 따라서 총 16번의 덧셈이 본 발명의 바람직한 실시예의 응용에 있어 필요하다. 종래 기술에 있어서는, 통상의 무지막지한 방법을 사용하는 경우 같은 작업을 위해 28번의 덧셈이 필요하였다.

본 발명의 이러한 특징상의 환경은, 앞에서의 예에서와 마찬가지로 같은 간격의 서브 합(sub-sum)이 각 행에서 필요한, 경우에 따라 고차일 수 있는 *U*-행렬을 포함하게 된다. 입력 벡터는 실수이거나 또는 복소수이다. 상기 행렬은 행들에 의하여 여러 개의 서브 행렬(sub-matrix)로 나뉘어질 수 있는 바, 이들은 각각 앞에서의 예에서 사용된 방법에 의해 개별적, 독립적으로 계산될수 있다. 이러한 실시예에서는, 덧셈을 줄일 수 있는 최적의 서브 분할(subdivision)을 찾아 복잡함을 줄이는 방법이 종합된다. 이를 위한 도구로서는 동적 프로그래밍에 기초한 추가적인 프로세서 또는 도구가 있는 바, 이들은 상기 테이블, 즉 복잡도 $s(n,r)$에 관한 한정·회귀 공식의 사용에 의해 상기 여러 서브 분할을 분석하게 된다. 매우 정확한 공식화가 본 실시예의 전개를 위해 필요하다. 몇가지 새로운 정의를 미리 할 필요가 있다.

$v=(v_1, v_2, \ldots, v_n)$을 벡터라 하고, n을 나누는 양의 정수 p(줄여서 $p \mid n$)를 정의 하자.또한 $v[p]$를 v를 길이 p인 부분으로 분할한 벡터로 이루어져 있는 벡터라 하자. 따라서 다음과 같이 된다.

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

각 부분은 다음과 같이 나타낸다.

$$v[p,k]= (v_{(k-1)p+1},v_2,\ldots\ldots\ldots\ldots\ldots\ldots,v_{kp}) \qquad \text{for } 1 \le k \le n/p.$$

이러한 맥락에서 정수 p는 확산 인자로 불려진다. 멀티 벡터는 행렬의 구조와 유사한 구조를 가지고 있음을 주목하자. 다음 항목, 즉 멀티 벡터의 멀티 스칼라 곱은 행렬 곱과 통상의 스칼라 곱의 크로스 곱이다. 두개의 n차 벡터 $v=(v_1, v_2, \ldots, v_n)$ 및 $w=(w_1, w_2, \ldots, w_n)$에 대해서 v와 w의 p차 멀티 스칼라 곱은 다음과 같이 정의된다.

$$v \bullet w[p]=\left(v[p,1] \bullet w[p,1],\ v[p,2] \bullet w[p,2],\ldots\ldots\ldots\ldots\ldots,v[p,n/p] \bullet w[p,n/p]\right)$$

여기에서 내적 : $v[p,1] \cdot w[p,1]$, $v[p,2] \cdot w[p,2]$, ……은 통상의 스칼라 곱이다. 주의할 점은 이 곱의 결과가 n/p 차원 벡터라는 것이다.

A가 라인 $A_1, \ldots\ldots, A_r$을 갖는 $r \times n$ 매트릭스라고 하고 $\boldsymbol{p}=(p_1, p_2, \ldots\ldots, p_r)$이 양의 정수 벡터라고 하자. $1 \le i \le r$인 i 모두에 대해 n이 p_i로 나누어 떨어지면 n이 $\boldsymbol{p}$로 나누어떨어진다고 말한다. 이것은 $\boldsymbol{p} \mid n$으로 표기된다. 이제 $\boldsymbol{p} \mid n$이라고 가정한다. x를 n차원 벡터라고 하고, $A \cdot x[\boldsymbol{p}]$로 표기되는 A 와 x의 $\boldsymbol{p}$-다중곱(multi-product)을 벡터들의 벡터라고 정의하자.

$$A \bullet x[p] = (A_1 \bullet x[p_1],\ A_2 \bullet x[p_2],\ldots\ldots\ldots\ldots,A_r \bullet x[p_r])$$

본 발명의 이 실시예는 이후에 기술하게 될 셋업에서의 이러한 곱의 계산을 향상시킨다.

다중곱 시스템.

다중곱 시스템(즉, 간략히 MP 시스템이라 함)은 파라미터로서 정수 n, r 및 양의 정수 벡터 $\boldsymbol{p}=(p_1, p_2, \ldots, p_r)$을 포함하는 세팅이며, 여기서 $p_1 \mid p_2$, $p_2 \mid p_3$, $p_3 \mid p_4$, ……, $p_{r-1} \mid p_r$ 및 $p_r \mid n$이다.

정수 $p_1, p_2, \ldots, p_r$ 은 시스템의 확산 인자(spreading factor)라고 한다. MP 시스템의 파라미터들은 다음과 같이 표현된다:

$$P = (r, n, \mathbf{p})$$

이제 이들 파라미터에 $r \times n$ U-행렬 A와 n차원 실수 벡터 x를 부가한다. 목표는 곱 $A \cdot x[\mathbf{p}]$을 효율적으로 계산하는 데 있다. MP 시스템 전체는 다음과 같이 표현된다:

$$S = (r, n, \mathbf{p}, A, x)$$

모든 정수 $p_1, p_2, \ldots, p_r$도 2의 멱수인 경우, 시스템은 이진 다중곱 시스템(binary multi-product system), 즉 간략히 BMP 시스템이라고 한다.

M-1 방법.

본 발명의 이 특징은 U법(U-method)을 MP 시스템에 그대로 적용한 것이다. 상기 예가 이것을 나타낸 것이다. 실제로, 이것은 이후에 기술하게 될 최적에 가까운 세분(sub-optimal subdivision) 이후에 MP 시스템의 서브시스템에 적용되는 것이 통상적이다.

MP 시스템 $S=(r,n,p,A,x)$이라고 하자. 본 방법은 최소 확산 인자 p_1에 대해 행렬을 부분 행렬(sub-matrix)로 수평 분할하는 것에 기초하고 있으며, 각 부분 행렬의 폭은 p_1이다. 본 방법은 벡터 x의 처음 p_1개의 실수와 행렬 A의 U-계수의 합을 계산하는 것으로 시작한다. 이것은 U-이진법에 의해 모든 라인에서 동시에 행해진다. 이어서, 본 방법은 그 다음 p_1개의 열로 가서 동일한 프로세스를 수행한다. 이와 같이 하여 본 방법은 n개의 변수 모두에 대해 수행될 때까지 계속된다. 그 다음에, 본 방법은 각 라인(여기서, $p_i > p_1$)으로 가서 통상의 방법으로 합산 프로세스를 완료한다.

각각의 부분 행렬에 대한 U-법의 첫번째 단계는 다른 라인들의 직선 카피, 즉 네거티브 카피(negative copy)인 라인들을 모두 스캔하는 것이다. 따라서, 예를 들어 $p_1=2$ 인 경우, 각 섹션에서는 r에 상관없이 최대 2번의 덧셈이 필요하다. 일반적으로 모든 섹션에서 U-이진법에 의해 고려되는 라인은 겨우 2^{p1-1} 개 이다. 게다가, 본 명세서에서 목적으로 하는 응용분야의 하나는 A가 아다마르 행렬(Hadamard)일 경우이다. 이 경우, 각 섹션에는 비등가 라인(non-equivalent line)이 겨우 p_1개 있을 뿐이다. 따라서, 각각의 부분 행렬에 얼마만큼의 비등가 라인이 나타날 수 있는지에 관한 경계를 포함하는 다른 소스가 삽입된다. 그 소스는 사용 중의 행렬의 유형의 결과인 z(테이블로 저장됨)로 표기된 함수에 포함되어 있다. 그의 파라미터는 p_1과 r이고, $z(p_1, r)$ 로 표현된다. 예를 들어, A가 아다마르 행렬인 경우, $z(4,6)=4$, $z(8,5)=5$ 이며, A가 일반적인 U-행렬인 경우, $z(4,40)=8$ 이다. $z(p_1, r) \le \min\{2^{p1-1}, r\}$이 항상 성립하며, A가 아다마르 행렬인 경우 $z(p_1, r)=\min(p_1, r)$ 이다.

M-1법의 복잡도의 계산은 전술한 U-이진법의 설명에 나오는 $s(r)$의 회귀식(regression formula) 및 테이블에 기초하게 된다. 양의 정수 y 모두에 대해 $s'(y)=s(y)+y$ 라고 하자. 또한 $s'(y) < 2^{y-1} + 2^{y/2+1}$ 이고 이 부등식은 꽤 엄격하다는 것을 상기한다. 이것은 다음 식에서의 복잡도의 크기에 대한 직관적인 이해를 제공한다. M-1법에서 행해지는 덧셈의 횟수의 경계는 따라서 이하의 항에 의해 정해진다.

$$C(n,r,\mathbf{p},z) = (n/p_1)(p_1 + s(z(p_1,r))) + (n/p_2)(p_2/p_1 - 1) + (n/p_3)(p_3/p_1 - 1)$$

$$+ \ldots\ldots\ldots\ldots\ldots\ldots$$

$$\ldots\ldots\ldots\ldots + (n/p_r)(p_r/p_1 - 1)$$

$$= n\bullet\left(1 + (s(z(p_1,r)) + r - 1)/p_1 - (1/p_2 + 1/p_3 + \ldots\ldots\ldots\ldots + 1/p_r)\right) =$$

$$= n\bullet\left(1 + s'(z(p_1,r))/p_1 - (1/p_1 + 1/p_2 + 1/p_3 + \ldots\ldots\ldots\ldots + 1/p_r)\right)$$

일부 사소하지만, 이 식에 있어서는 사소한 실례가 존재하지 않음에 주목할 필요가 있다. 첫번째 항은 행렬이 1 라인을 갖는 경우, 즉 r = 1인 경우라면 다음과 같이 정의된다.

$$C(n,r,p) = n - n/p_1$$

두번째 항은 균일한 확산 계수인 경우, 즉 p1 = p2 =‥‥‥= pr인 경우라면 다음과 같이 정의된다.

$$C(n,r,p,z) = n\bullet\left(1 + s(z(p_1,r))/p_1\right)$$

명확하게, M-1 방법은 r이 p_1에 비례해서 큰 경우에 비효율적임을 의미한다. 그것은 보다 스마트한 방법에 대한 발판이 주로 그 방법을 토대로 전개될 것이다. 이러한 보다 강한 방법은 행렬을 수평으로 분할하고 각각의 부분 행렬을 분리하도록 상기 M-1 방법을 적용함으로써 동작한다. 보다 작은 부가의 총 번호를 가져오는 분할 구조를 소수의 계산에 의해 찾기 위해서는 전술한 식의 다음의 간단한 버전을 갖도록 유용하게 사용될 것이다. 그래서, 다음과 같이 정의된다.

$$C^*(n,r,p,z) = 1 + s'(z(p_1,r))/p_1$$

다중 시스템의 분할 및 다중 벡터의 일반화된 개념. 다음에, 식은 행렬을 수평 라인 상에서 분할하는 명령으로 이루어진다. 이러한 명령은 벡터 r을 분할함으로써 나타낼 것이다. 그 결과는 동일한 폭인 n의 소수의 하위 문제점에 대해 최초의 문제점을 차단할 수 있는데, 이들 각각은 전술한 M-1 방법에 의해 해결된다. 그 분할은 효율성을 최대화하기 위해 본 발명의 알고리즘으로 분리될 것이다. 확산 계수의 r 차원 정수 벡터 p = (p1, p2,‥‥,pr) 및 이하의 U-행렬을 이용하고,

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

정수 k, m(여기서, 1 ≤k ≤m ≤r 임)을 이용하면, 벡터 p의 첫번째 섹션은 다음과 같이 정의된다.

$$p(k,m)=(p_k,\ldots\ldots,p_m).$$

상기 구성 요소들을 지수들 k 내지 m으로 간단하게 이용하면, 행렬 A의 섹션은 다음과 같이 정의한다.

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

이와 마찬가지로, 라인들을 지수들 k 내지 m으로 이용하는 것을 의미한다.

다음에, 정수 벡터를 고려하면 r = (r(1),····,r(t+ 1))은 다음의 식을 만족시키며,

$$k = r(1) < r(2) < \ldots\ldots\ldots < r(t) < r(t+1) = m+1,$$

이는 섹션으로 분할하는 수단을 의미한다. 먼저, r은 이하의 방법으로 벡터 p[r]의 벡터로 p의 분할을 작성하기 위한 도구이다.

$$p[r] = \left((p_{r(1)}, \ldots\ldots, p_{r(2)-1}), (p_{r(2)}, \ldots\ldots, p_{r(3)-1}), \ldots\ldots\ldots\ldots, (p_{r(t)}, p_2, \ldots\ldots, p_{r(t+1)-1}) \right)$$

부분 벡터는 다음의 사항을 나타낸다.

$$p[r,1] = (p_{r(1)}, \ldots\ldots\ldots\ldots, p_{r(2)-1})$$

$$p[r,2] = (p_{r(2)}, \ldots\ldots\ldots\ldots, p_{r(3)-1})$$

.

.

.

$$p[r,t] = (p_{r(t)}, \ldots\ldots\ldots\ldots, p_{r(t+1)-1})$$

행렬 A의 열들은 다음과 같은 방법으로 벡터 r을 분할함에 따라 유사하게 분할된다.

모든 정수들 1 ≤q ≤t에 대해서;

$$A[r,q] = \left(a_{ij} : r(q) \leq i < r(q+1),\ 1 \leq j \leq n \right)$$

주어진 분할에 대한 M-방법. 이 섹션은 낮은 복잡성의 부분 분할을 찾고 이 실시예에 대해 중심적인 메카니즘의 구성 내의 메인 단계이다. 행렬의 라인 분할을 제공하여 각각의 부분 행렬에 따라서 M-1 방법을 분리적으로 실시하는 것을 가정한다. 그 목적은 부가의 총 번호를 평가해서, 소수의 부가의 분할이 다음 스테이지에서 찾도록 함에 있다. MP 시스템 S = (r, n, p = (p1, p2,····,pr), A, x, z) 및 분분 분할 벡터 r = (r(1),····,r(t))을 고정하는데, 여기서 1 ≤k = r(1) < r(2) <·····< r(t) < r(t+ 1) = m+ 1 ≤r+ 1이다.

1 ≤q ≤t의 경우, Sq MP 서브 시스템은 다음과 같이 주어진다.

$$S_q = \left(r(q+1) - r(q),\ n,\ p[r,q],\ A[r,q],\ x, z \right)$$

모든 부분 시스템에 대한 부가의 전체 번호는 다음과 같다.

$$C(n,r,p,z) = \sum_{1 \leq q \leq t} C\left(n,\ r(q+1) - r(q),\ p[r,q], z \right) =$$

$$= n \cdot \left(\sum_{1 \leq q \leq t} s'\left(z(p_{r(q)},\ (r(q+1) - r(q)) \right) / p_{r(q)} - (1/p_k + \ldots\ldots\ldots + 1/p_m) \right) +$$

$$t)$$

다음 목적은 이 조건을 최소화하는 분할을 찾는 효율적인 방법을 발전시키는 데 있다. 각 스테이지에서 반복하는 가산 부분 조건인 1/pk + ·····+ 1/pm과 n 계수를 계산함이 없이 계산을 실행하는 것은 보다 효율적일 것이다. 따라서, 다음과 같이 정의한다.

$$C^*(n,r,r,p,z) = \sum_{1 \le q \le t} s'\left(z(p_{r(q)}, (r(q+1) - r(q))\right)/p_{r(q)} + t$$

근접한 최적의 분할. 이 스테이지에서 전술한 조건 C(n, r, r, p, z)에 의해 주어딘 부가의 양을 최소화하는 분할 특성을 연구한다. 먼저, 상기 분할이 최초의 문제점의 주어진 부분 구간과 접해 있는 최소의 최악의 경우를 갖는 부가의 번호로 추상적인 식을 제공할 것이다.

다음에, MP 시스템 S = (r, n, p = (p1, p2,····,pr), A, x, z) 및 정수 k,

m(여기서, 1 ≤k ≤m ≤n 임)을 고려하면 다음과 같이 정의된다.

$$h(k,m) = min\{C(n\,,\, m - k + 1,\, \mathbf{r}\,,\, p(k,m),z) : for\ \mathbf{r}=(r(1),.....,r(t+1))$$

$$where\ \ k=r(1)<r(2)<.....<r(t)<r(t+1)=m+1\}$$

$$h^*(k,m) = min\{C^*(n\,,\, m - k + 1,\, \mathbf{r}\,,\, p(k,m),z)\ :\ for\ \mathbf{r}=(r(1),.....,r(t+1))$$

$$\text{where}\ \ k=r(1)<r(2)<.....<r(t)<r(t+1)=m+1\}$$

이것은 다음과 같은 등식을 유지한다.

$$h(k,m) = n\bullet\left(h^*(k,m) + (1/p_k + ... + 1/p_m)\right)$$

이것에 후속하는 순환 공식은 다음 등식을 유지하고,

$$h(k,m) = \quad min\{C(n,\ m\text{-}k+1,\ p(k,m),\ z)\ ,\ \ min\{h(k,q\text{-}1) + h(q,m):\ for\ all\ k < \ q \le$$

$$m\}\}$$

이 등식에서, min(공집합)= 무한대이다.

$$h^*(k,m) = \quad min\{C^*(n,\ m\text{-}k+1,\ p(k,m)\ ,z)\ \ min\{h^*(k,q\text{-}1) + h^*(q,m):\ for\ all\ k < \ q \le$$

$$m\}\}$$

이 등식에tj, min(공집합)= 무한대이다.

이후에, 이러한 등식들은 부분 구조(substructure)들이 k와 m 사이의 간격들인 후속하는 동적 코드에 의해 이용된다. 이러한 코드(h*)의 복잡도를 줄이는 것은 그 경로 상에 h를 대체하여 최상의 부분 구조를 찾아야 할 것이다. 최적의 부분 분할로 제1 단계를 찾기 위해서, 우리는 모든 k 및 m에 대한 식 h(k,q-1)+ h(q,k)를 최소화하는 q를 계산할 것이며, 여기서, 1≤k≤m≤m 이며, h*(k,q-1)+ h*(q,k)를 최소화하는 동일한 q를 갖는다. 따라서, h*(k,m)=c*(n,r,p(k,m),z)일 때, q(k,m) = k이고, 다른 방법으로, q(k,m)= 최소의 q 이고, 여기서, k＜q≤m 및 h*(k,m)= h*(k,q-1)+ h*(q,k)이다.

이제, 최적의 부분 분할 코드가 형성될 수 있다.

최적의 부분 분할을 찾는 동적 프로그래밍 코드. 그 후속하는 코드들은 MP 시스템 P=(r,n,p,z)의 변수를 데이터로서 수신하고, M 방법을 최적으로 실행하는 부분 분할(r)을 출력으로서 발생한다. 추가적으로, 최적의 M 방법의 복잡성으로 복귀한다. 이것을 효율적으로 실행하기 위해서, 우리는 미리 s'(r)의 테이블을 계산하고 기억시킬 필요성이 있다. 이것은 회귀 공식을 이용하여 그 테이블을 계산하는 고속 코드에 의해 행하여 진다.

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

최적의 부분 분할 테이블(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.
```

이후에 남겨진 것은 이러한 절차에 의해서 최적의 부분 분할 벡터를 구축한 테이블q(k,m)로부터 얻어진다. 이것은 최적의 벡터(r) 성분을 포함하는 집합(R)을 생성하는 후속하는 코드에 의해 행해진다.

최적의 부분 분할 벡터(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.
```

이후, M 방법이 그 부분 분할을 실행할 최적의 부분 분할(r)이 발견된다.

구현예

다음의 본 발명의 양호한 실시예는 상기 방법들의 상세한 구현예들인 본 발명의 알고리즘을 제공한다. 알고리즘은 상기 방법들을 수행하기 위해 필요한 메모리 및 에너지 자원의 추가 저감을 가능하게 한다. 상기 알고리즘은 장치를 구축하기 위해 필요한 명령들을 제공하고 상기 방법들을 개선하기 위한 이중의 목표를 가진다. 이러한 구현예에 관한 기본적인 하나의 가정은 변환 행렬이 일반적이고 소정의 엔트리 세트(집합)를 갖는 행렬 세트(집합)에서 임의적으로 선택한 것으로서 고려될 수 있다. 또 다른 하나의 가정은 상기 기본적인 구속인

행의 수<로그(열의 수)

를 만족하도록 행의 수는 열의 수와 비교해서 충분히 작다. 따라서 다른 환경에서 적합한 단계들의 일부인 이러한 등가선에 대한 체크는 여기서는 여분의 단계이다.

이후 설명되는 매핑은 한 위치에서 다음 위치로 데이터를 흐름을 채널화한다. 각각의 위치에는 소정의 반복으로 행렬의 열에 대응하는 이진 어드레스가 할당된다. U-행렬의 경우 열의 (-1)-성분은 어드레스의 1에 대응하며 마찬가지로 열의 1-성분은 어드레스의 0에 대응한다. 유사한 대응 관계가 U_1-행렬에 대해서 정의된다.

기술될 구현예는 입력 x_i-신호들이 그들의 대응 열에 의해서 결정되는 프리세트 수신지에 부가되는 제1 스텝을 포함하며, 여기서 각 x_j에는 2의 멱 및/또는 부호가 곱해진다. 선행 반복으로 본 발명에 따라 열이 분할될 때마다 각각의 2 개의 스플릿의 할당된 어드레스는 소정의 열의 어드레스와 같거나 작다. 이것에 의해 처리되기전과 소실되기 전에 그의 수신지로 보내질 메모리의 일부 위치에 모든 정보가 기억 가능하다. 또한, 소정의 열의 스플릿 절반 중 하나가 0 이면, 이 열을 나타내는 어드레스는 다음 반복을 위해 사용될 것이다. 이러한 속성은 데이터의 이동을 최소화하도록 설정된 데이터 흐름(플로우)맵(data-flow map)의 새로운 구성에 의해서 달성된다. 각각의 반복에서 어드레스의 최소수는 현재의 내용이 다음 반복에 의해서 사용 가능한 모든 내용을 포함하여 손상되지 않은 상태로 남겨진다.

다음의 구현예의 메카니즘은 본 발명의 상기 GEM의 양호한 실시예를 고려하면 양호하게 이해될 것이다. 수 1를 포함하는 수 S의 유한 세트(집합)가 고려되고, 이 상황에서 r 라인의 행렬은 r-차원 열벡터 성분이 집합 S에 속하는 정규화된 r-차원 열벡터의 가능한 0이 아닌 구성으로 이루어지면 완전한 S-r-행렬이라 불리며, 각각의 구성은 정확하게 한번 나타나며 정규화 규약은 가장 바닥의 1이 아닌 원소가 1이어야 한다는 것이다.

다음의 저차원의 예를 관측한다.

행렬

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

이 완전 {0,1}-2-행렬이고,

행렬

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

이 완전 U-2-행렬이며,

행렬

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

이 완전 U-3-행렬이고,

행렬

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

이 완전 {0,1}-3-행렬이며,

행렬

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

이 완전 {1,-1,j,-j}-2-행렬이다.

다음의 구현예는 모든 가능한 구성의 대부분이 변환 행렬의 열에서 나타나도록 행의 수가 열의 수에 비해 충분히 크다는 가정에 의거하고 있다. 따라서, 모든 이전 구현예에서 행해진 제1 반복은 수정된 벡터와 완전한 S-r-행렬과의 곱에 대한 계산을 저감한다. 그에 따라 보다 개선된 모든 반복은 개개의 수정된 벡터와 저차원의 완전 S 행렬과의 곱을 계산한다.

상기 실시예의 행렬에서 열들이 나타나는 순서화 방법은 이 구현예의 어드레스 설정의 또 다른 특징을 반영한다. 그것은 열을 $|S|$-베이스 어드레스로 변환하기 위한 일관성 있는 규칙에 의거한 어드레스이며, MSB는 벡터의 바닥 성분이고 LSB는 상부 성분이다.

그 기본적인 결과는 제1 반복 후 균일한(오히려 명백한) 어드레스 넘버링을 갖는 유일의 균일한 완전 변환 행렬이 있다는 것이다. 균일한 행렬은 단지 S와 r에만 의존하고 초기 변환 행렬에는 의존하지 않는다. 이것은 나머지 프로세스에서 균일성을 가져오며 그에 따라 제1 단계 후 특정의 변환 행렬과 무관하고 일반적인 데이터 흐름 구조의 생성을 가능하게 한다. 이것은 하드웨어 구현예와 본 섹션의 주요 목표인 본 발명에 기초하고 있는 장치 구조에서 매우 유용하다.

이러한 구현예의 추가 형태 및 별개의 형태는 적절한 변환 행렬에 대한 가산 수의 축소와 소요 판독 및 기록 메모리 할당의 축소이다. 따라서, 장치의 단가와 프로세스의 성능에 필요한 에너지는 저감된다. 이어지는 코드의 각각의 효율성은 수개의 입력 x-벡터와 동일한 초기 변환 행렬을 처리하는 경우에 개선됨을 관측하였다.

최종적으로, 적절한 견해를 갖기 위해 다음의 구현예는 본 발명의 다수의 가능한 효율적인 구현예 중 몇개의 예라고 이해하여야 한다.

실시예 1:0-1-2진 행렬

다음의 단계의 순서는 본 발명의 0-1-행렬 형태의 일구현예를 기술한다. 데이터는 rxn 0-1 행렬 $A=(a_{ij}: 0\le i<r, 0\le j<n)$과 입력 실수 벡터 또는 복소 벡터 $x = (x_0, ..., x_{n-1})$ 로 구성된다. 행렬 A의 열은 $v_0, ..., v_{n-1}$로 표시된다. 이러한 단계의 순서에 의해서 곱셈 $y = (y_0, ..., y_{r-1})^T = A \cdot x$이 계산된다. 할당된 판독 및 기록 메모리는 각각의 반복에서 프로세스에 참가하는 열들을 나타내는 1 내지 2^r-1로 레이블된 2^r-1 어드레스를 포함한다. 다음의 정의는 본 발명의 양호한 실시예의 설명 부분이다.

정의

1) 모든 $k\ge0$, $j\ge1$에 대하여, $l(k,j) = \lceil (j-1)\cdot(r/2^k) \rceil$ 를 정의한다.

2) 모든 0≤m<r에 대하여, $Y_m = 2^m$를 정의한다. 이러한 어드레스는 출력 벡터 $y = (y_0, ..., y_{r-1})^T$의 성분을 프로세스의 끝에서 포함할 것이며, Y_0는 y_0의 어드레스이고, Y_1는 y_1의 어드레스이다.

3) k≥1 및 j≥1에 대해서 각각의 위치에서 다음 위치로 데이터의 이동을 제어하는 다음과 같은 함수 F_{kj}, G_{kj}를 정의한다. 먼저 l = l(k,j), m = l(k+ 1, 2j), h = l(k,j + 1)-1라고 하자.

다음에 각각의 정의 v≥1에 대해서 다음과 같이 정의한다.

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

4) 모든 벡터 $v=(v_0,.....,v_{r-1}) \in \{0,1\}^r$에 대해서 정의하면

$$\sigma(v)=\sum_{0\le j<r} 2^j v_j$$ 이다.

코드

1.초기화 : 1 내지 2^r-1의 모든 어드레스를 0으로 한다.

2.제1 단계 : j가 0 에서 n-1로 진행하는 동안

$v_j \neq 0$이면 x_j를 어드레스 $v=\sigma(v_j)$에 가산한다.

3.주요 부분 :

k가 0에서 $\lceil \log(r) \rceil - 1$로 진행하고

j가 1에서 2^k로 진행하는 동안

l(k,j+ 1)-1 > l(k,j)이면

p가 l에서 $2^{l(kj+1-l(kj)-1}-1$로 진행하는 동안

$v=2^{l(kj)} \cdot p$로 놓아

(소스)어드레스 v가

$F_{kj}(v) \neq 0$ 이고 $F_{kj}(v) \neq v$이면

(소스)어드레스 v에 상주하는 값을 (수신지)어드레스 $G_{kj}(v)$ 값에 가산하고

또한 이값(어드레스 v의)을 (수신지)어드레스 $F_{kj}(v)$ 값에도 가산한다.

4.출력 구하기:

이 단계에서, 모든 어드레스 Y_i는 모든 0≤i<r 에 대해서 출력 성분 y의 값을 포함한다.

복잡성

이 용어 정의에서 기본적인 단계는 소스라 불리는 메모리의 한 위치에서 하나의 수(number)를 판독하고 수신지라 불리는 메모리의 또 다른 위치에 놓인 수(number)에 그것을 가산하는 것이다.

상기 코드에 기초한 장치는 A · x의 계산의 주요 부분을 계산하기 위해 기본적인 단계의 다음과 같은 수를 필요로 한다.

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

이 수는 r에 만 의존한다.

따라서 A · x의 전체적인 계산은 많아야 다음의 기본적인 단계의 총계를 필요로 한다.

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

챠트 1:0-1-2진 행렬

도 1은 상기 코드의 주요 부분을 수행하는 장치가 채용하는 메모리의 내용을 갱신하기 위한 동작을 개략적으로 도시하며 그에 따라 양호한 형태에서 본 발명의 0-1-2진 형태를 구현하고, 0-1-2진 행렬은 4 개의 행을 포함한다. 각각의 반복에서 각 행렬의 열은 2진 방식으로 메모리의 어드레스에 의해서 표현된다. 바닥(0 또는 1) 성분(수평 표시에서 가장 우측 성분)은 MSB이고 상부(0 또는 1) 성분(수평 표시에서 가장 좌측 성분)은 LSB이다. 어드레스 $Y_m=2^m$, 0 · m · 3은 출력 벡터 $y = (y_0, ..., y_3)^T$의 성분을 프로세스의 끝에서 포함하며, Y_0는 y_0의 어드레스이고 Y_1는 y_1의 어드레스이다. 화살표는 하나의 어드레스의 내용을 취하여 그것을 또 다른 어드레스에 가산하는 조치를 나타낸다. 이것은 상기 코드로 표시한 바와 같이 각각의 반복에 참가하는 어드레스의 (증가)순서와 반복 순서에 따라서 행해진다.

실시예 2: U-행렬

다음 단계의 순서는 본 발명의 U-행렬 형태의 구현예를 기술한다. rxn 행렬 $A=(a_{ij}: 0\le i<r, 0\le j<n)$과 입력 실수 벡터 $x = (x_0, ..., x_{n-1})$ 로 구성된다. 행렬 A의 열은 $w_0, ..., w_{n-1}$로 표시된다. 이러한 단계의 순서에 의해서 곱셈 $y = (y_0, ..., y_{r-1})^T$=A · x이 계산된다. 모든 소정의 반복으로 각각의 위치는 벡터 x에 종속하는 하나의 실수 또는 복소수를 포함한다. 할당된 판독 및 기록 메모리는 각각의 반복에서 프로세스에 참가하는 열들을 나타내는 0 내지 $2^{r-1}+r-2$로 레이블된 $2^{r-1}+r-2$ 어드레스를 포함한다. 다음의 정의 및 이전 실시예의 정의는 본 발명의 양호한 실시예의 설명을 위해 필요한 부분이다.

정의

1) 각 r-차원의 U-벡터에 대해서, $u = (u_0,.....u_{r-1})$를 다음과 같이 정의한다.

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

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

2) 바이폴라 2진 집합 U={1,-1}과 논리 비트 2진 집합 B={0,1}간의 대응 관계를 다음과 같이 정의한다.

$$(-1)' = 1$$
$$1' = 0$$

따라서 r-차원의 u-벡터에 대해서 $u=(u_0,.....,u_{r-1})$를 다음과 같이 정의한다.

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

3)$Y_0=0$, 모든 l · m · r-l: $Y_m=2^{r-1}+m-l$를 정의한다. 이러한 어드레스는 프로세스의 끝에서 출력 벡터 $y=(y_0, ..., y_{r-1})^T$의 성분을 포함할 것이며, Y_0는 y_0의 어드레스이고 Y_1는 y_1의 어드레스이다.

3) 모든 k≥0 및 j≥1에 대해서 맵 F_{kj}, G_{kj}, $Sign_{kj}$은 다음과 같이 정의된다. 먼저 l = l(k,j), m = l(k+ 1, 2j), h = l(k,j + 1)-1라고 하자. 모든 정수 $v\neq 0$에 대해서 다음과 같이 정의한다.

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

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

$$G_{kj}(v)= \begin{cases} v \bmod 2^m & \text{if } Sign_{kj}(v) = 1 \\ 2^m - 2^l - (v \bmod 2^m) & \text{if } Sign_{kj}(v) = -1 \end{cases}$$

코드:

1.초기화: 0 내지 $2^{r-1}+r-2$의 모든 어드레스를 0으로 놓는다.

2.제1 단계: j가 0에서 n-1로 진행하는 동안

$Sign(w_j)\cdot x_j$를 어드레스 $\pi(h(w_j))$에 가산한다.

3.주요 부분:

k가 0에서 $\lceil \log(r) \rceil - 1$로 진행하고

j가 1에서 2^k로 진행하는 동안

l(k,j+ 1)-1 > l(k,j)이면

1)(소스)어드레스 $Y_{l(k,j)}$에 상주하는 값을 (수신지)어드레스 $Y_{l(k+1,2j)}$에 가산한다

2)p가 l에서 $2^{l(kj+1)-l(kj)-1}-1$로 진행하고

$u=2^{l(kj)}\cdot p$로 놓아 소스 어드레스 u에 대해 수행하고

$G_{kj}(u)=u$이면

소스-어드레스 u에 상주하는 값을 수신지 어드레스 $Y_{l(k+1,2j)}$의 값에 가산한다

$F_{kj}(u)=u$이면

(소스)어드레스에 상주하는 값을 (수신지)어드레스 $Y_{l(k,j)}$ 값에 가산한다

$G_{kj}(u)=0$이면 (수신지)어드레스 $Y_{l(k,j)}$ 값에 (-1)을 곱한 (소스)어드레스 u에 상주하는 값을 가산하고 또한 이 값(어드레스의)을 어드레스 $F_{kj}(u)$ 값에도 가산한다

그렇지 않으면 $Sign_{kj}(u)$를 곱한 소스-어드레스 u에 상주하는 값을 수신지-어드레스 $G_{kj}(u)$ 값에 가산하고 또한 이값(어드레스 u의)을 어드레스 $F_{kj}(u)$ 값에도 가산한다.

4.출력 구하기:

이제 모든 어드레스 Y_i는 모든 $0 \le i < r$에 대해서 y_i의 값을 수용한다.

복잡성

i)상기 구현예에 있어 U 설정에서 기본적인 단계는 소스라 불리는 메모리의 한 위치에서 하나의 수를 판독하고 그곳에 1 또는 -1인 부호를 곱한 다음에 그 결과를 수신지라 불리는 메모리의 또 다른 위치에 놓인 수에 가산하는 것이다.

ii)그 복잡성은 다음의 항들로 형식화된다. $0 \le k \le \lceil \log(r) \rceil - 1$과 $1 \le j \le 2^k$에 대해서

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

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

$k = \lceil \log(r) \rceil$과 $1 \le j \le 2^k$에 대해서

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

이면

$$u_k=\sum_{1\le j\le 2^k} u_{kj} = r$$

이다.

iii)상기 코드에 기초한 장치는 A · x의 계산을 충족하기 위한 기본적인 단계의 다음 번호를 필요로 한다.

(1)제1 단계의 경우 n 기본 단계를 필요로 한다.

(2)모든 $0 \le k \le \lceil \log(r) \rceil - 1$과 $1 \le j \le 2^k$에 대해서

$$2 \cdot u_{kj} - u_{k+1,2j} - u_{k+1,2j-1} + 1$$

기본 단계들은 (k,j) 단계에서 행해진다.

(3)그러므로 모든 0≤k≤ $\lceil log(r) \rceil - 1$에 대해서 주요 부분의 k 반복은 다음과 같은 기본적인 단계의 수를 필요로 한다.

$$2^k + 2 \cdot u_k - u_{k+1}$$

(4)따라서 A・x의 상기 계산의 주요 부분은 다음과 같은 기본적인 단계의 수를 필요로 한다.

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

이 수는 r에 만 의존한다.

(5)그러므로 A・x의 전체 계산은 다음 기본적인 단계의 총계이다.

$$C^u_{n,r} \equiv n + C^u_r$$

<u>챠트 2: U-행렬</u>

도 2는 상기 코드의 주요 부분을 수행하는 장치가 채용하는 메모리의 내용을 갱신하기 위한 동작을 개략적으로 도시하며 그에 따라 양호한 형태에서 본 발명의 U-2진 형태를 구현하고, U-2진 행렬은 4 개의 행을 포함한다. 각각의 반복에서 각 행렬의 열은 메모리의 2진 어드레스에 의해서 표현된다. 열의 (-1)-성분은 어드레스의 1에 대응하고 열의 1-성분은 어드레스의 0에 대응한다. 2진 해석이 적용되며, 바닥 성분(수평 표시에서 가장 우측 성분)은 MSB이고 상부 성분(수평 표시에서 가장 좌측 성분)은 LSB이다. 특정 어드레스 $\mathbf{Y}_0$=0, $\mathbf{Y}_1$=8,$\mathbf{Y}_2$=9,$\mathbf{Y}_3$=10는 출력 벡터 $y = (y_0, \ldots\ldots, y_3)^T$의 성분을 프로세스의 끝에서 포함하며, Y_0는 y_0의 어드레스이고 Y_1는 y_1의 어드레스이다. 화살표는 하나의 어드레스의 내용을 취하여 그것에 1 또는 -1인 부호를 곱하고 가산될 그 결과를 또 다른 어드레스에 내용에 가산하는 조치를 나타낸다. 하나이 창화살표는 부호가 1임을 의미하고 2중 창화살표는 부호가 -1임을 의미한다. 이것은 각각의 반복에 참가하는 어드레스의 (증가)순서와 반복 순서에 따라서 행해진다.

<u>실시예 3: U_1-행렬</u>

단계들의 다음 순서는 변환 행렬의 엔트리가 세트 $U_1 \equiv \{1,-1,j,-j\}$에 속하는 경우 GEM 방법의 일구현예를 기술한다. 이것은 종종 응용예에서 나타나는 GEM의 서브케이스 중 하나이다. 데이터는 rxn U_I-행렬 $A=(a_{ij}: 0:0 \le i < r, 0 \le j < n)$과 입력 복소 벡터 $x = (x_0, \ldots\ldots\ldots, x_{n-1})$ 로 구성된다. 행렬의 열은 $w_0, \ldots\ldots, w_{n-1}$로 표시된다. 이러한 단계의 순서에 의해서 곱셈 $y = (y_0, \ldots\ldots, y_{r-1})^T = A \cdot x$이 계산된다. 모든 소정의 반복으로 각각의 위치는 벡터 x에 종속하는 하나의 실수 또는 복소수를 포함한다. 할당된 판독 및 기록 메모리는 0 내지 $4^{r-1}+r-2$로 레이블된 $4^{r-1}+r-2$ 어드레스를 포함한다. 이것은 이전 실시예에서 설정된 정의와 다음의 정의를 이용한다.

정의

1)세트 U_1와 세트{0,1,2,3} 사이의 대응 관계(역대응 관계)를 다음과 같이 정의한다.

$$1' = 0, \qquad 0* = 1$$
$$(-1)' = 1, \qquad 1* = -1,$$
$$j' = 2, \qquad 2* = j,$$
$$(-j)' = 3, \qquad 3* = -j$$

따라서, r-차원의 U_1-벡터에 대해서, u = (u_0, ..., u_{r-1})는 다음과 같이 정의된다.

$$\alpha(u)=\sum_{0\le j<r} 4^j u'_j.$$

2)r-차원의 U_1-벡터에 대해서, u = (u_0, ..., u_{r-1})는 다음과 같이 정의된다.

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

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

3)Y_0=0을 정의하고 모든 l≤m<r에 대해서 $Y_m=4^{r-1}$ + m -1이다. 이는 출력 벡터 성분의 어드레스가 될 것이다.

4)모든 k≥0, j≥1에 대해서 맵 F, G는 다음과 같이 정의된다. l = l(k,j), m = l(k+1,2j), h = l(k,j + 1) -1이라고 하고, $v=\sum_{0\le j<r-2} 4\cdot v_j$로 4 베이스로 표시된 정수 υ≥0를 취하여 다음과 같이 정의한다.

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

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

$$G_{k,j}(v)= \sum_{1\le j<m-1} 4^j\cdot((v_j)^*\cdot((v_{m-1})^*)^{-1})'$$

코드

1. 초기화: 0 내지 4^{r-1}+ r-2의 모든 어드레스를 0으로 놓는다.

2. 제1 단계: j가 0 내지 n-1로 진행하는 동안 Sign(w_j) · x_j를 어드레스 σ(g(w_j))에 가산한다.

3. 주요 부분:

k가 0에서 $\lceil \log(r) \rceil - 1$로 진행하고

j가 1에서 2^k로 진행하는 동안

l(k,j+ 1)-1 > l(k,j)이면

1)어드레스 $Y_{l(k,j)}$에 상주하는 값을 어드레스 $Y_{l(k+1,2j)}$에 가산한다

2)p가 1에서 $4^{l(kj+1-l(kj)-1}$-1로 진행하고

u=$4^{l(kj)}$ · p로 놓아 소스 어드레스 u에 대해 수행하고

G_{kj}(u)=u이면

소스-어드레스 u에 상주하는 값을 수신지 어드레스 $Y_{l(k+1,2j)}$의 값에 가산한다

F_{kj}(u)=u이면

(소스)어드레스에 상주하는 값을 (수신지)어드레스 $Y_{l(k,j)}$ 값에 가산한다

$G_{kj}(u)=0$이면 (수신지)어드레스 $Y_{l(k,j)}$ 값에 $Sign_{kj}(u)$을 곱한 (소스)어드레스 u에 상주하는 값을 가산하고 또한 이 값(어드레스의)을 어드레스 $F_{kj}(u)$ 값에도 가산한다

그렇지 않으면 $Sign_{kj}(u)$를 곱한 (소스)어드레스 u에 상주하는 값을 (수신지)어드레스 $G_{kj}(u)$ 값에 가산하고 또한 이값(어드레스 u의)을 어드레스 $F_{kj}(u)$ 값에도 가산한다.

4.출력 구하기

이제 모든 어드레스 Y_i는 모든 $0 \le i < r$에 대해서 y_i의 값을 수용한다.

복잡성

i)상기 구현예에 있어 U_l-설정에서 기본적인 단계는 소스라 불리는 메모리의 한 위치에서 복소수를 판독하고 그것에 1 또는 -1 혹은 j 또는 -j인 부호를 곱한 다음에 그 결과를 수신지라 불리는 메모리의 또 다른 위치에 놓인 복소수에 가산하는 것이다.

ii)그 복잡성은 다음의 항들로 형식화된다.$0 \le k \le \lceil \log(r) \rceil - 1$과 $1 \le j \le 2^k$에 대해서

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

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

$k = \lceil \log(r) \rceil - 1$과 $1 \le j \le 2^k$에 대해서

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

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

iii)상기 코드에 기초한 장치는 $A \cdot x$의 계산을 충족하기 위한 기본적인 단계의 다음 수를 필요로 한다.

(1)제1 단계의 경우 n 기본 단계를 필요로 한다.

(2)모든 $0 \le k \le \lceil \log(r) \rceil - 1$과 $1 \le j \le 2^k$에 대해서

$$2 \cdot u_{kj} - u_{k+1,2j} - u_{k+1,2j-1} + 1$$

기본 단계들은 (k,j) 단계에서 행해진다.

(3)그러므로 모든 $0 \le k \le \lceil \log(r) \rceil - 1$에 대해서 주요 부분의 k 반복은 다음과 같은 기본적인 단계의 수를 필요로 한다.

$$2^k + 2 \cdot u_k - u_{k+1}$$

(4)따라서 A · x의 상기 계산의 주요 부분은 다음과 같은 기본적인 단계의 수를 필요로 한다.

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

이 수는 r에 만 의존한다.

(5)그러므로 A · x의 전체 계산은 다음 기본적인 단계의 총계이다.

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

실시예 4 : 토플리츠 U-행렬

단계들의 다음의 순서는 U-계수를 가진 본 발명의 토플리츠 행렬의 구현예를 기술한다. 데이터는 U-행렬 t_0, ..., t_{n-1}과 입력 실수 또는 복소 벡터 x=(x_0,.....,x_{n+r-2})로 구성된다. U-순서로부터 rx(n+ r-1) 토플리츠 행렬 A≡(a_{ij}≡t_{i-j}: 0≤i<r, 0≤j≤n+ r-2)가 형성된다. 여기서 모든 k<0 또는 k≥n에 대해서 t_k≡0이다. 이러한 단계들에 의해서 곱셉 y=(y_0,.....,y_{n-1})=A · x가 계산된다. 단지 제 1 단계 만이 U-행렬의 실시예의 것과 상이하며 그에 따라 이 단계 만을 도입하는 것이 필요하다. 구현예의 이전 실시예에서 열거된 모든 정의들이 여기서 적용 가능하다.

코드

1.초기화: 0 내지 2^{r-1}+ r-2의 모든 어드레스를 0으로 놓는다.

2.제1 단계:

1)j가 0에서 r-1로 진행하는 동안 (1/2) · $t_{n-r+j+1}$ · x_j를 어드레스

$$\pi(h(t_j,t_{j-1},.....,t_1,t_0,\ t_{n-1},t_{n-2},.....,t_{n-r+j+1}))$$

에 가산하고 (1/2) · $t_{n-r+j+1}$ · x_j를 어드레스

$$\pi(h(t_j,t_{j-1},.....,t_1,t_0,\ -t_{n-1},-t_{n-2},.....,-t_{n-r+j+1}))$$

에도 가산한다.

2)j가 r-1에서 n-1로 진행하는 동안 t_{j-r+1} · x_j를 어드레스

$$\pi(h(t_j,t_{j-1},.....,t_{j-r+1}))$$

에 가산한다.

3)j에 대해 n에서 n+ r-2로 진행하는 동안 (1/2) · t_{j-r+1} · x_j를 어드레스

$$\pi(h(t_{j-n},t_{j-n-1},.....,t_1,t_0,\ t_{n-1},t_{n-2},.....,t_{j-r+1}))$$

에 가산하고 (1/2) · t_{j-r+1} · x_j를 어드레스

$\pi(h(t_{j-n}, t_{j-n-1}, \ldots\ldots, t_1, -t_0, -t_{n-1}, -t_{n-2}, \ldots\ldots, -t_{j-r+1}))$

에도 가산한다.

3.주요 부분: 이제 알고리즘은 알고리즘의 주요 부분을 구현하는 U-행렬에서와 같이 속행하고 출력이 동일 어드레스에 기억된다.

복잡성

i)상기 토플리츠 구현예에서 기본적인 단계는 U-구현예와 동일하다. 즉, 소스라 불리는 메모리의 한 위치에서 수를 판독하고 그것에 1 또는 -1인 부호를 곱한 다음에 그 결과를 수신지라 불리는 메모리의 또 다른 위치에 놓인 수에 가산하는 것이다.

ii)상기 코드에 기초한 장치는 A・x의 계산을 충족하기 위한 기본적인 단계의 다음 수를 필요로 한다.

(1)제1 단계에 대해서

$$2\cdot(r-1) + n - r + 1 + 2\cdot(r-1) = n + 3r - 3$$

기본적인 단계가 필요하다.

(2)A・x의 토플리츠 계산의 주요 부분은 r라인을 가진 U-코드의 주요 부분의 구현예에 의해서 행해진다. 따라서 C_r^u 기본적인 단계를 필요로 한다.

(3)따라서 A・x의 전체적인 토플리츠 계산은

$$C_{n,r}^T \equiv n + 3r - 3 + C_r^u$$

기본적인 단계의 총계를 필요로 한다.

챠트 3: 토플리츠-행렬

도 3은 상기 코드의 초기 부분을 수행하는 장치가 채용한 적합한 메모리 위치에 입력 데이터를 송신하는 동작을 개략적으로 도시하며, 토플리츠 행렬은 4개의 행을 포함한다. 이러한 초기 부분을 제외하곤 모든 다른 형태는 U-행렬 구현예 및 장치의 것과 동일하고 여기서 동일 부분에 대한 설명이 적용 가능하다. 여기서도 역시 하나의 창화살표는 부호가 1임을 의미하며 이중 창화살표는 부호가 -1임을 의미한다.

실시예 5: 토플리츠 U_1-행렬

다음의 본 발명의 양호한 실시예는 U_1-계수를 가진 본 발명의 토플리츠 행렬 형태의 일구현예이다. 데이터는 U_1-순서 $t_0, \ldots, t_{n-1}$과 입력 복소 벡터 $x=(x_0,\ldots\ldots,x_{n+r-2})$로 구성된다. U_1-순서로부터 $rx(n+r-1)$ 토플리츠 행렬 $A\equiv(a_{ij}\equiv t_{i-j}: 0\le j<r, 0\le j\le n+r-2)$가 형성된다. 여기서 모든 $k<0$ 또는 $k\ge n$에 대해서 $t_k\equiv 0$이다. 아래에 열거된 단계들의 순서에 의해서 곱셉 $y=(y_0,\ldots\ldots,y_{n-1})=A\cdot x$가 계산된다. 단지 제 1 단계 만이 U_1-행렬의 실시예의 것과 상이하며 그에 따라 이 단계 만이 기술될 필요가 있다. 이전 실시예에서 열거된 모든 정의들이 여기서 적용 가능하다.

코드

1.초기화: 0 내지 $4^{r-1}+r-2$의 모든 어드레스를 0으로 놓는다.

2.제1 단계:

1)j가 0에서 r-2로 진행하는 동안 (1/2)・$t_{n-r+j+1}$・x_j를 어드레스

$$\alpha\ (g(t_j, t_{j-1}, \ldots\ldots\ldots\ldots t_1, t_0, t_{n-1}, t_{n-2}, \ldots\ldots\ldots\ldots, t_{n-r+j+1}))$$

에 가산하고 -(1/2)・$t_{n-r+j+1}$・x_j를 어드레스

$$\alpha\ (g(t_j, t_{j-1}, \ldots\ldots\ldots\ldots t_1, t_0, -t_{n-1}, -t_{n-2}, \ldots\ldots\ldots\ldots, -t_{n-r+j+1}))$$

에도 가산한다.

2)j가 r-1에서 n-1로 진행하는 동안 t_{j-r+1}・x_j를 어드레스

$$\alpha\ (g(t_j, t_{j-1}, \ldots\ldots\ldots, t_{j-r+1}))$$

에 가산한다.

3)j가 n에서 n+ r-2로 진행하는 동안 (1/2)・t_{j-r+1}・x_j를 어드레스

$$\alpha\ (g(t_{j-n}, t_{j-n-1}, \ldots\ldots\ldots t_1, t_0, t_{n-1}, t_{n-2}, \ldots\ldots\ldots, t_{j-r+1}))$$

에 가산하고 (1/2)・t_{j-r+1}・x_j를 어드레스

$$\alpha\ (g(t_{j-n}, t_{j-n-1}, \ldots\ldots\ldots, t_1, -t_0, -t_{n-1}, -t_{n-2}, \ldots\ldots\ldots, -t_{j-r+1}))$$

에도 가산하고 -(1/2)・t_j・x_j를 어드레스

$$\alpha\ (g(t_{j-n}, t_{j-n-1}, \ldots\ldots\ldots, t_1, -t_0, -t_n, -t_{n-1}, \ldots\ldots\ldots, -t_{j-r+2}))$$

에도 가산한다.

3.주요 부분: U_1-행렬 알고리즘의 주요 부분에서와 같이 진행하고 데이터는 동일 방식으로 수신된다.

복잡성

i)상기 토플리츠 구현예에서 기본적인 단계는 U_1-구현예와 동일하다. 즉, 소스라 불리는 메모리의 한 위치에서 수를 판독하고 그것에 1 또는 -1인 부호를 곱한 다음에 그 결과를 수신지라 불리는 메모리의 또 다른 위치에 놓인 수에 가산하는 것이다.

ii)상기 코드에 기초한 장치는 A・x의 계산을 충족하기 위한 기본적인 단계의 다음 수를 필요로 한다.

(1)제1 단계에 대해서

$$2\cdot(r-1)+n-r+1+2\cdot(r-1)=n+3r-3$$

기본적인 단계가 필요하다.

(2)A・x의 토플리츠 계산의 주요 부분은 r라인을 가진 U_1-코드의 주요 부분의 구현예에 의해서 행해진다. 따라서 $C^{U_1}_{r}$ 기본적인 단계를 필요로 한다.

(3)따라서 A・x의 전체적인 토플리츠 계산은

$$C^{T_1}_{n,r} \equiv n+3r-3 + C^{U_1}_{r}$$

기본적인 단계의 총계를 필요로 한다.

<u>실시예 6: 실수 행렬, 2진 0-1 표시</u>

본 발명의 다음의 양호한 실시예는 행렬-엔트리의 0-1-표현을 갖는 본 발명의 실수 행렬 형태의 일구현예이다. 데이터는 넌네가티브 엔트리를 갖는 rxn 실수 행렬 A=(a_{ij}: 0≤i<r, 0≤j<n)과 입력 실수 또는 복소 벡터 x=(x_0,.....x_{n-1})로 구성되어 있다. 알고리즘에 의해 y=(y_0,.....,y_{n-1})=A・x가 계산된다.

행렬의 엔트리가 모든 0≤i<r, 0≤j<n에 대해서 다음의 0-1-2진 표현에 의해서 주어진다고 가정한다.

$$a_{ij}=\sum_{-m_2\le k\le m_1} t_{ijk}\cdot 2^k$$

여기서: 모든 - m_2≤k≤m_1에 대해서 t_{ijk}∈{0,1}이다.

단지 제1 단계 만이 0-1-행렬의 구현예의 것과 상이하므로 이 단계 만이 기술될 필요가 있다. 0-1-행렬 구현예 섹션에서 열거된 모든 정의들이 여기서 적용 가능하다. 또한 모든 0≤j<n, - m_2≤k≤m_1에 대해서 다음의 r-차원 {0,1}-벡터를 정의한다.

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

1.초기화: 0 내지 2^r-1의 모든 어드레스를 0으로 놓는다.

2.제1 단계:

카운터 j가 0에서 n-l로 진행하고

k가 $-m_2$에서 m_1으로 진행하는 동안

$2^k \cdot x_j$를 어드레스 $\sigma(y_{jk})$ 값에 가산한다.

3.주요 부분: 주요 부분은 r 행에 대해 {0,1}-행렬 구현예의 알고리즘의 주요 부분에서와 같이 진행되고 출력은 계산 프로세스의 끝에서 동일한 어드레스에 기억된다.

복잡성

i)상기 구현예에서 기본적인 단계는 0-1-2진 구현예와 동일하다. 즉, 소스라 불리는 메모리의 한 위치에서 수를 판독하고 그것을 수신지라 불리는 메모리의 또 다른 위치에 놓인 수에 가산하는 것이다.

ii)상기 코드에 기초한 장치는 $A \cdot x$의 계산을 충족하기 위한 기본적인 단계의 다음 수를 필요로 한다.

(1)제1 단계에 대해서

$(m_1 + m_2 + 1) \cdot n$

기본적인 단계가 필요하다.

(2)코드의 주요 부분은 r라인을 가진 상기 구현예 0-1-2진 코드의 주요 부분에 의해서 행해진다. 따라서 $C_r^{0,1}$ 기본적인 단계를 필요로 한다.

(3)따라서 $A \cdot x$의 전체적인 계산은

$$C^{Re-0-1} \equiv (m_1 + m_2 + 1) \cdot n + C^{0,1}_r$$

기본적인 단계의 총계를 필요로 한다.

실시예 7: 실수 행렬, 2진 U-표현

본 발명의 다음의 양호한 실시예는 행렬-엔트리의 U-표현을 갖는 본 발명의 실수 행렬 형태의 일구현예이다. 데이터는 rxn 실수 행렬 $A=(a_{ij}: 0 \le i < r, 0 \le j < n)$과 입력 실수 또는 복소 벡터 $x=(x_0,.....x_{n-1})$로 구성되어 있다. 알고리즘에 의해 $y=(y_0,.....,y_{n-1})=A \cdot x$가 계산된다.

행렬의 엔트리가 모든 $0 \le i < r, 0 \le j < n$에 대해서 다음의 U- 표현에 의해서 주어진다고 가정한다.

$$a_{ij} = \sum_{-m_2-1 \le k \le m_1-1} t_{ijk} \cdot 2^k + t_{ijm_1} \cdot (2^{m_1} - 2^{-m_2-1})$$

여기서: 모든 $-m_2-1 \le k \le m_1$에 대해서 $t_{ijk} \in U$이다.

이러한 표현의 실체에 대해서는 본 발명의 실수 행렬 형태에서 이미 언급되었다. 단지 제1 단계 만이 U-행렬 구현예의 것과 상이하므로 이 단계 만이 기술될 필요가 있다. U-행렬 구현예 섹션에서 열거된 모든 정의들이 여기서 적용 가능하다. 또한 모든 $0 \le j < n, -m_2-1 \le k \le m_1$에 대해서 다음의 r-차원 U-벡터를 정의한다.

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

1.초기화: 0 내지 $2^r + r - 1$의 모든 어드레스를 0으로 놓는다.

2.제1 단계: j가 0에서 n-1로 진행하고

k가 $-m_2 - 1$에서 $m_1 - 1$로 진행하는 동안

$2^k \cdot \mathbf{Sign}(u_{jk}) \cdot x_j$를 어드레스 $\pi(h(u_{kj}))$에 가산한다.

k=m_1에 대해 행하고

$(2^{m1} - 2^{-m2-1}) \cdot \mathbf{Sign}(u_{jk}) \cdot x_j$를 어드레스 $\pi(h(u_{kj}))$에 가산한다.

3.주요 부분: 주요 부분은 r 행에 대해 u-행렬 구현예의 알고리즘의 주요 부분에서와 같이 진행되고 출력은 계산 프로세스의 끝에서 동일한 어드레스에 기억된다.

복잡성

i)상기 구현예에서 기본적인 단계는 상기 U-구현예와 동일하다. 즉, 소스라 불리는 메모리의 한 위치에서 수를 판독하고 그것에 1 또는 -1인 부호를 곱한 다음에 그 결과를 수신지라 불리는 메모리의 또 다른 위치에 놓인 수에 가산하는 것이다.

ii)상기 코드에 기초한 장치는 A · x의 계산을 충족하기 위한 기본적인 단계의 다음 수를 필요로 한다.

(1)제1 단계에 대해서

$(m_1 + m_2 + 2) \cdot n$

기본적인 단계가 필요하다.

(2)코드의 주요 부분은 r라인을 가진 U-코드의 상기 구현예의 주요 부분에 의해서 행해진다. 따라서 C_r^u 기본적인 단계를 필요로 한다.

(3)따라서 A · x의 전체적인 계산은

$$C^{Re\text{-}U} \equiv (m_1 + m_2 + 1) \cdot n + C_r^u$$

기본적인 단계의 총계를 필요로 한다.

도 4는 본 발명의 양호한 실시예에 따른 축소된 가산수를 갖는 선형 변환을 수행하기 위한 일례의 장치의 블록도이다. 장치(500)는 2 개의 입력과 하나의 출력을 가진 곱셈기(10), 그의 입력중 하나의 입력이 그의 출력에 전송되게 선택하는 멀티플렉서(MUX)(9), 2 개의 입력과 하나의 출력을 가진 가산기(11), 그 다음에 이어지는 2 개의 어드레스 버스 라인, 즉 "add_a" 및 "add_b" 및 2 개의 출력, 즉 "data_a" 및 "data_b"을 가진 이중 포트 RAM(DPRAM)(13)으로 구성되어 있다. MUX의 동작은 어드레스 발생기(501)에 의해서 제어되며, 이 어드레스 발생기에 의해 DPRAM(13)의 메모리 어드레스에 대한 엑세스가 활성화된다. 어드레스 발생기의 동작은 카운터(3)에 의해서 제어된다.

곱셈기(10)의 출력은 MUX(9)의 하나의 입력 "C"에 접속된다. MUX(9)의 출력은 가산기(11)의 하나의 입력 "A"에 접속된다. 가산기(11)의 출력은 DPRAM(13)의 입력에 접속된다. DPRAM(13)의 하나의 출력 "data_a"은 가산기(11)의 입력

"B"에 접속된다. DPRAM(13)의 다른 출력 "data_b"은 곱셈기(12)의 입력 "E"에 접속된다. 곱셈기(12)의 다른 입력 "F"은 어드레스 발생기(501)의 출력 "부호(sign)"에 접속된다. 곱셈기(12)의 다른 입력 "F"은 MUX(9)의 다른 입력 "D"에 접속된다. 카운터(3)는 어드레스 발생기(501)의 2 개의 입력에 접속된다. 어드레스 발생기(501)의 출력 "H"은 곱셈기(12)의 "부호" 입력에 접속된다. 어드레스 발생기(501)의 출력 "G"은 MUX(9)의 제어 입력 "S"에 접속된다. 어드레스 발생기(501)의 출력 "J"은 DPRAM(13)의 제1 어드레스 입력 "add_a"에 접속된다. 어드레스 발생기(501)의 다른 출력 "1"은 DPRAM(13)의 제2 어드레스 입력 "add_b"에 접속된다. 변환 행렬은 일련의 코드(행렬에서 비트들의 행)(D_0,D_1,...,D_s)를 어드레스 발생기(501)에 공급하고 최상위 비트 D_0를 곱셈기(10)에 공급하는 옵션의 RAM/ROM(1)에 기억될 수 있다. 입력 벡터는 입력 신호의 샘플을 곱셈기(10)에 공급하는 또 다른 옵션의 RAM/ROM(2)에 기억될 수 있다. 이와는 달리 기억 메모리(1,2)는 입력 벡터의 요소와 변환 행렬에서의 그의 대응 요소가 장치(500)에서 동기되어 제공된다면 제거 가능하다. 예컨대, 이러한 요소들은 A/D 변환기 또는 순서 발생기에 의해서 제공 가능하다. 장치(500)의 모든 구성 요소들은 "clock_in" 입력을 통해 공통 클록에 의해서 제어된다.

동일한 입력 세트 $[x_1\ x_2\ \cdots\ x_n]$, 상이한 코드 D_0,D_1, ...,D_s를 표현하는 (n 행들을 포함하는)동일한 U 행렬를 이용하여 이후 장치의 동작이 설명된다. 장치(500)의 동작은 2 단계로 나누어질 수 있다. "Stage 1"라고 표시된 제1 단계 동안에 입력 데이터(즉, 샘플 $[x_1\ x_2\ \cdots\ x_n]$)가 수신되고 각 구성 요소와 변환 행렬의 대응 요소의 곱이 계산되어 DPRAM(13)에 기억되며, "Stage 2"라고 표시된 제2 단계 동안에 수신된 데이터는 처리되고 추가 입력 데이터가 가산기(9)에 들어가는 것을 차단한다. 카운터(3)는 연산의 누산수를 카운트하고 그 카운트(카운터의 출력)를 이용하여 두 단계 사이를 구별한다. "Stage 1"의 연산수는 입력 벡터의 길이 n이다. 이 수를 초과하는 연산은 "Stage 2"와 관련이 있다.

본 발명의 양호한 실시예에 따르면, 어드레스 발생기(501)는 카운터(3)의 출력에 링크되는 비교기(4)를 포함한다. 비교기는 카운터(3)의 현재 출력을 판독하여 그것을 입력 벡터의 길이 n와 비교하고, 현재의 연산이 ""Stage 1" 또는 "Stage 2"에서 실행되는지를 나타내는 대응 신호를 제공한다. 이 신호는 입력 "C"와 "D" 사이에서 전환하도록 MUX(9)의 입력 "S"을 제어하기 위해 사용된다.

어드레스 발생기(501)는 또한 프로그램된 값을 기억하고 그 내용을 처리하기 위해 입력 어드레스 "add_a"와 "add_b"를 통해 DPRAM(13)의 어드레스를 결정하기 위한 룩업 테이블(LUT)로서 사용되는 비동기 메모리(5)(예, ROM)을 포함한다. LUT의 크기는 (C-n)x(1+ 2r)이고, 그의 내용은 3 개의 필드, 즉 "소스" 필드, "부호" 필드 및 "수신지" 필드를 포함한다. 각각의 연산(카운터(3)로 카운트된 각각의 클록 사이클)에 대해서 3 개의 대응 소스 부호와 수신지값이 있다. 소스 필드는 분할된 변환 행렬의 각각의 열과 관련된 정보와 특정 열의 두 부분을 정규화하는 것과 관련된 표지(indication)를 포함한다. 소스 필드는 DPRAM(13)에 대한 입력 "add_b"의 값을 결정한다. 부호 필드는 각각의 분할 열 또는 서브열의 하위 성분을 나타낸다. 수신지 필드는 대응 어드레스의 어느 내용이 처리를 위해 선택되는 지에 따라서 DPRAM(13)에 대한 입력 "add_a"의 값을 결정한다.

어드레스 발생기는 또한 일련의 비트 D_1, ...,D_s 에 각각 접속된 입력을 포함하는 s(s=r-1) 인버터 세트 6_1,6_2,....,6_s를 또한 포함한다. 각 인버터의 출력은 s 멀티플렉서 세트 7_1,7_2,....,7_s로부터의 대응하는 MUX의 하나의 입력에 접속된다. 일련의 비트 D_1,.....D_s는 또한 s 멀티플렉서 세트 7_1,7_2,....,7_s로부터의 대응하는 MUX의 다른 하나의 입력에도 공급된다. s 멀티플렉서 세트 7_1,7_2,....,7_s로부터의 각 MUX의 출력은 불변 세트 D_1, ...,D_s 또는 반전된(즉, D'_1,....D'_s) 세트의 DPRAM(13)의 입력 "address_a"로의 전송을 가능하게 하기 위해 최상위 비트 D_0의 값에 의해서 제어된다. 세트 D_1, ...,D_s (또는 D'_1,....D'_s)는 s 멀티플렉서의 대응 세트 8_1,8_2,....,8_s의 하나의 입력으로 입력되고, s 멀티플렉서는 LUT로부터 도달하는 "add_a"의 MSB(r번째 비트)에 의해서 공급되는 추가 멀티플렉서 8_r를 구비한다. 비교기(4)의 출력에 의해서 "add_a"의 선택이 세트 7_1,7_2,....,7_s로부터 또는 LUT로부터 도달 가능하다. 입력 "add_a"(즉, 수신지)은 가산기(11)의 입력"B"으로 공급되는 DPRAM(13)의 제1 출력 "data_b"을 제어한다. LUT(즉, 소스)로부터 취한 입력 "add_b"은 LUT로부터 추출된 대응 "부호"값이 "data_b"의 각각의 값에 곱해지고 곱셈기(12)의 입력 "D"으로 그 곱이 공급되는 곱셈기(12)의 입력"E"으로 공급되는 DRRAM(13)의 제2 출력 "data_b"을 제어한다. DPRAM(13)에서의 "기록" 동작은 동기적이다. 즉 각 셀의 내용은 클록 레이트에 따라(예, 클록 신호가 상승할 때) 중복 기록된다. 한편, DPRAM(13)에서의 "판독" 동작은 비동기적이다. 즉 각 출력은 어드레스 입력이 클록과 무관하게 변화되는 순간에 변화된다.

<u>단계 1에서의 연산</u>

이 단계에서 카운터(3)는 단계 1이 수행되는 동안에 제1 심볼 시간(n 클록 사이클)을 카운트하기 시작한다. 단계 1 동안에, MUX(9)는 그의 입력 "C"로부터 흐르는 데이터를 그의 출력과 가산기(11)의 입력 "A"으로 흐르게 하며, 입력 "D"는 차단한다. 입력 심볼 $[x_1\ x_2\ \cdots\ x_n]$은 곱셈기(10)의 하나의 입력에 제공된다. 코드의 MSB D_0는 곱셈기(10)의 다른 입력에 제공된다. 이 단계에서, 입력 "add_a"에 의해서 결정되는 수신지(출력 "data_a")는 DPRAM(13)으로부터 추출되고 입력 벡터의 성분 $[x_1\ x_2\ \cdots\ x_n]$에 가산되어 MSB D_0가 곱해진다. 현재의 심볼 시간을 카운트하는 카운터(3)는 현재의 심볼 시간이 종료한 ROM(5)과 어드레스 발생 비교기(4)에 표지를 제공하며 비교기(4)는 입력 "C"에서 다른 입력 "D"으로 MUX(9)의 입력 선택을 전환한다. 마찬가지로 비교기(4)는 RAM(1)이 아니라 LUT에서 데이터를 선택하도록 멀티플렉서 $8_1,8_2,\ldots,8_s$를 구동한다.

단계 2에서의 연산

이 단계에서, 카운터(3)는 단계 2가 수행되는 동안에 다음 심볼 시간을 카운트하기 시작한다. 단계 2 동안에, MUX(9)는 그의 입력 "C"로부터 흐르는 데이터를 그의 출력과 가산기(11)의 입력 "A"으로 흐르게 하며, 입력 "D"는 차단한다. 이 단계의 각 클록 사이클에서 DPRAM(13)에서 선택된 어드레스는 소스를 나타내는 "add_b"를 통해 엑세스된다. 소스 데이터는 DRRAM(13)의 제2 출력 "data_b"에서 곱셈기(12)의 입력 "E"으로 공급되며 곱셈기에서 LUT로부터 추출된 대응의 "부호"값이 곱해진다. 이 곱은 MUX(9)의 입력 "D"으로 공급됨으로써, 가산기(11)의 입력 "A"에서 나타난다. 동시에 현재의 수신지값의 내용은 가산기(11)의 입력 "B"에서 나타난다. 두값은 가산기(11)에 의해서 가산되고 그 결과는 DPRAM(130)의 동일 어드레스(이전 수신지값에 대응하는)에 기억된다. 이 합산 프로세스의 끝에서, 대응의 r 변환점(y_i's)은 DPRAM(13)의 어드레스 #0과 #2 내지 (2^{r-1}+ r-2)에 기억된다. 이러한 프로세스는 모든 변환점(y_i's)이 계산되어 DPRAM(13)의 상이한 어드레스에 기억된 다음에 단계 2가 (소정 카운트에 따라)종료된다는 표시를 제공할 때까지 지속된다. 그 때, 현재의 심볼 시간은 또한 종료되고, 비교기(4)는 입력 "D"에서 다른 입력 "C"으로 다시 MUX(9)의 입력 선택을 전환하고 LUT가 아니라 RAM(1)에서 데이터를 선택하도록 멀티플렉서 $8_1,8_2,\ldots,8_s$를 구동함으로써 단계 1에서 동작하도록 장치(500)를 다시 구동한다.

상기 방법을 구현하기 위해 사용되는 장치의 일실시예가 도 5에 도시되고 있다. 도 5는 n 차원의 벡터 X와 rxn U-행렬 A와의 곱셈을 일구현예를 도시하고 있다. 행렬의 표현은 0 또는 1을 포함하며 0은 1에 대응하고 1은 -1에 대응한다.

이 실시예에서 벡터는 실수 벡터이며 v 비트는 각 성분에 전용된다. 이어지는 구성에 있어서 행렬 A는 요소 1(RAM 또는 ROM)에 기억되고 벡터 X는 요소 2(RAM 또는 ROM)에 기억된다. 행렬 및 벡터는 A/D 변환기 또는 일부 순서 발생기 등과 같이 동기화 소스 또는 메모리 디바이스 등의 내부 또는 외부 디바이스로부터 발생될 수가 있다.

모듈 1,2,3,13을 제어하는 클록은 전체 시스템을 동기화한다. 모듈 3은 0 내지 C를 카운트하는 카운터이고, C는 다음과 같이 정의된다.

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

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

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

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

단계 1에서 요소(1 및 2)로부터의 입력 신호는 요소(3)로부터 들어오는 어드레스에 따라서 요소(13)에 삽입되고, 각각의 어드레스는 $\log_2 n$ 리스트 상위 비트를 포함한다.

또한, 카운터는 크기 (C-n)x(1+ r+ r)의 비동기 ROM(요소 5)의 어드레스 발생기로서 이용된다. 카운터의 모든 비트는 현재의 카운트가 n 보다 큰지를 체크하는 비교기인 요소(4)로 들어간다. 요소(5)는 3 개의 필드, 즉 부호 필드, 소스 필드, 수신지 필드를 포함한다. ROM에서 라인의 순서는 n 사이클 다음에 카운터에 의해서 단계 2의 제1 연산이 유도되도록 되어야 한다.

구문(syntax) 목적을 위해 s=r-1을 정의한다. 데이터 버스는 요소 1 비트-D1-Ds에서 요소 6_1-6_s(인버터)로 진행한다. 요소(71-7s)는 D0의 상태에 따라서 D1-Ds 및 61-6s의 출력 사이에서 선택한다.

(D0=0=〉 7_1-7_s=D1-Ds, Do=1=〉 71-7s=6a-6s)

요소 8_1-8_s는 요소(4)(stage1_2n)의 비교기의 출력 상태에 따라서 요소(5)(add_a=수신지)의 lon_2(n)LSB의 출력과 7_1-7_s 사이에서 선택한다. 상태 1_2n=0=〉 8_1-8_s=add_a, Stage_2n=1=〉 8_1-8_s=7_1-7_s

요소 8_r은 요소(5)의 "0" 내지 r 비트(add_a MSB) 사이에서 선택한다. 8_1-8_r로부터의 출력은 $(2^{r-1}+r-1)x(V+\log_2 n)$의 크기로 이중 포트 RAM인 요소(13)의 제1 어드레스 버스로서 이용된다. 이러한 어드레스는 요소(13)에 대한 입력인 data_in 버스의 수신지와 요소(13)로부터의 출력인 data_a 버스를 정의한다.

(요소(5)의 출력(소스 필드)인)Add_b는 요소(13)의 제2 어드레스 버스이다(이 어드레스는 요소(13)로부터의 출력인 data_b를 제어한다).

부호는 멀티플렉서에서 data_b를 곱셈하는 요소(13)로부터의 1 비트 출력이다.

data_in 버스는 요소(9)의 출력과 data_a를 함께 합산하는 가산기인 요소(11)(곱셈기)로부터 도달된 요소(13)에 대한 입력이다. data_in는 동기식으로 add_a 수신지로 진행하고 data_a 및 data_b의 판독 동작은 비동기적이다.

요소(13)의 동작 이전에 0의 삽입에 의해 초기화된다. 요소(9)는 요소(10)와 요소(12)의 출력 사이에서 선택한다. 요소(10)는 요소(1)의 LSB를 요소(2)로부터의 데이터 버스에 곱한다. C 사이클 다음에 합성 벡터는 요소(13)의 어드레스 0 및 어드레스 2^{r-1} 내지 $(2^{r-1}+r-2)$에 기억된다.

당업자에게는 본 발명의 다수의 변형례 및 수정례가 자명한 것이다. 따라서, 본 발명은 본 발명의 사상 또는 필수적 특징을 일탈하지 않는 다른 특정 형태로 실시 가능하다. 상세한 실시예는 단지 예증의 목적으로 기술되었을뿐 본 발명의 범위를 한정하는 것은 아니며 본 발명의 범위는 실시예가 아니라 첨부된 청구 범위이다. 청구 범위의 등가물의 범위 및 내용 범위 내에 속하는 모든 변경도 본 발명의 청구 범위 내에 있는 것으로 간주된다.

(57) 청구의 범위

청구항 1.

실수 또는 복소수 또는 유한 필드 상에서 적어도 하나의 n 차원 입력 벡터의 rxn 행렬로 표현된 선형 변환의 수행 효율성을 개선하기 위한 방법으로서,

감소된 차원을 갖는 변형된 행렬을 생성하기 위해 초기 rxn 변환 행렬을 변형하는 단계로서, 상기 변형 단계는,

존재하는 경우 등가 행(equivalent row)들의 그룹을 식별하고, 상기 등가 행들의 그룹으로부터 하나의 행을 선택하여 유지하고, 상기 등가 행들의 그룹에서 하나 이상의 등가 행들을 생략하고, 각각의 생략된 행과 상기 선택된 행 사이의 비율을 메모리에 기억하는 단계와,

상기 초기 변환 행렬의 0 열(column)들을 생략하는 단계와,

정규화된 행렬을 생성하기 위해 잔여 열들과 이들의 선두(lead) 요소들의 역(inverse)들을 각각 곱함으로써 상기 초기 변환 행렬의 잔여 열들을 정규화하는 단계와,

상기 정규화된 행렬의 동일한 열들의 그룹을 식별하고, 상기 동일한 열들의 그룹으로부터 하나의 열을 선택하여 유지하고, 상기 동일한 열들의 그룹에서 하나 이상의 동일한 열들을 생략하는 단계를 포함하는, 상기 변형 단계와,

채널 상에서 신호에 대응하는 n 차원 입력 벡터를 수신하는 단계와,

상기 생략된 0 열들에 대응하는 입력 벡터의 요소들을 생략하고, 상기 입력 벡터의 요소들과 상기 초기 변환 행렬의 잔여 열들을 정규화하는데 사용되는 선두 요소들을 곱함으로써 상기 입력 벡터를 정규화하고, 상기 식별된 동일한 열들의 그룹에 따라 정규화된 입력 벡터 요소들을 합산하는 단계, 및

상기 변형된 벡터와 상기 변형된 행렬을 곱함으로써 r 차원 출력 백터를 획득하는 단계로서, 상기 획득은 상기 변형된 매트릭스에 따라 상기 변형된 백터의 요소들의 합을 누산하고, 상기 메모리의 상기 저장된 하나 이상의 비율들을 사용하여 상기 하나 이상의 생략된 행들에 대응하는 상기 r 차원 출력 백터의 요소들을 발생하는 것을 포함하는, 상기 r 차원 백터 획득 단계를 포함하는 선형 변환 수행 방법.

청구항 2.

제1항에 있어서, 상기 초기 rxn 변환 행렬을 변형하는 단계는,

상기 변환 행렬을 수개의 부분 행렬(sub-matrix)로 분할하여 r 차원 출력 백터를 획득하는 단계와,

각각의 부분 행렬의 곱의 결과로 생긴 다수의 출력 벡터들을 통합하는 단계를 더 포함하는 것인 선형 변환 수행 방법.

청구항 3.

제2항에 있어서, 상기 변형된 행렬은 상기 변환 행렬의 열들의 서브 세트(부집합)을 포함하는 것인 선형 변환 수행 방법.

청구항 4.

제3항에 있어서, 각각의 부분 벡터(sub-vector)가 부분 행렬에 대응하도록 입력 벡터를 수개의 부분 벡터로 분할하는 단계를 더 포함하며, 출력 벡터는 각각의 부분 행렬의 곱의 결과로 생긴 출력 벡터를 가산함으로서 구해지는 것인 선형 변환 수행 방법.

청구항 5.

제1항에 있어서, 상기 변형된 행렬을 수개의 부분 행렬들로 분할하는 단계를 더 포함하며, 출력 벡터는 개개의 대응하는 부분 벡터들에 의해서 각각의 부분 행렬의 곱들의 결과로 생긴 출력 벡터들을 가산함으로서 구해지는 것인 선형 변환 수행 방법.

청구항 6.

삭제

청구항 7.

삭제

청구항 8.

삭제

청구항 9.

선형 변환을 수행하기 위한 장치로서,

선형 변환에 대응하는 일반적인 실수 및 복소수 행렬들의 차수를 감소시키고 이들을 정규화하는 수단으로부터의 미리 정해진 변환 데이터 및 입력 데이터를 수신하는 제1 및 제2 입력들과,

상기 미리 정해진 변환 데이터에 대응하는 선형 변환의 출력 벡터를 생성하기 위해 상기 입력 데이터 및 상기 미리 정해진 변환 데이터에 관해 동작하며, 가산기 및 메모리를 포함하는 변환 및 누산 회로와,

상기 변환 및 누산 회로에 접속되며, 상기 메모리의 셀들에 엑세스하기 위해 대응하는 어드레스들을 발생하는 제어 및 어드레스 발생 회로로서, 이 제어 및 어드레스 발생 회로는 상기 입력 데이터가 상기 제1 입력을 통해 수신되고 행렬의 열들에 대응하는 어드레스들에 따라 상기 메모리에 저장되는 데이터 수신 모드와 누산 동작들이 상기 메모리의 하나의 셀로부터 상기 메모리의 다른 셀로 데이터를 가산하거나 감산하는 데이터 처리 모드 간의 선택을 제어하는, 상기 제어 및 어드레스 발생 회로를 포함하는, 선형 변환 장치.

청구항 10.

제9항에 있어서, 상기 변환 및 누산 회로는 상기 데이터 수신 모드 동안 상기 변환 데이터의 대응하는 요소와 상기 입력 데이터의 각각의 요소를 곱하는 제1 곱셈기와, 상기 데이터 처리 모드 동안 상기 제어 및 어드레스 발생 회로의 사인(sign) 출력과 상기 메모리로부터의 데이터를 곱하는 제2 곱셈기를 더 포함하는 선형 변환 장치.

청구항 11.

제10항에 있어서, 상기 메모리는 듀얼 포트 RAM을 포함하는 것인 선형 변환 장치.

청구항 12.

삭제

청구항 13.

제9항에 있어서, 상기 제어 및 어드레스 발생 회로의 출력에 기초하여 상기 데이터 수신 모드와 상기 데이터 처리 모드간 선택하는 멀티플렉서 회로를 더 포함하는 것인 선형 변환 장치.

청구항 14.

제9항에 있어서, 상기 제어 및 어드레스 발생 회로는,

사전 프로그램된 처리 및 제어 데이터를 기억하는 제2 메모리와,

상기 데이터 수신 모드 및 상기 데이터 처리 모드간 전환하는 비교기 회로를 포함하는 것인 선형 변환 장치.

청구항 15.

제14항에 있어서, 상기 제어 및 어드레스 발생 회로는,

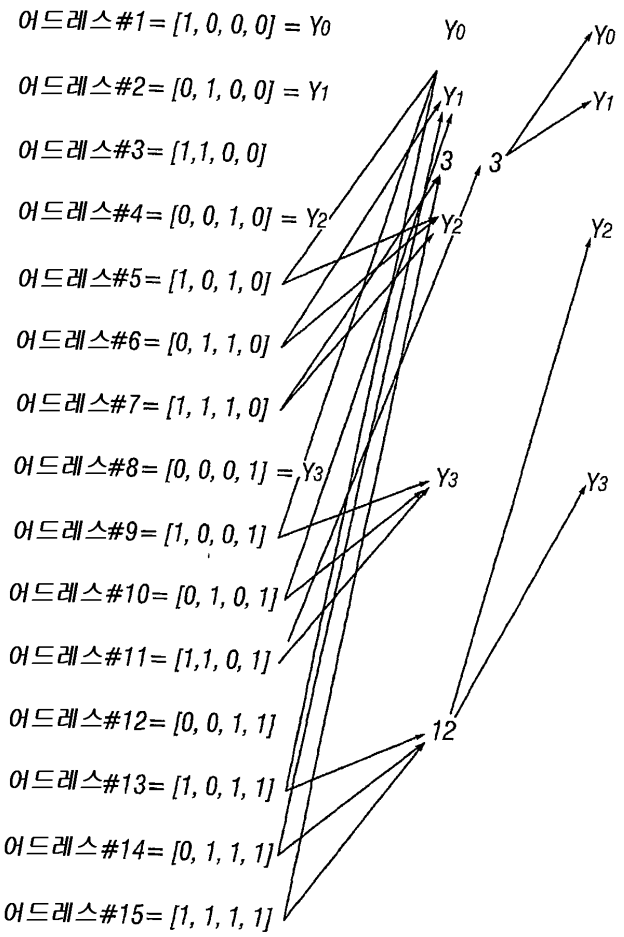
변환 데이터를 수신하는 적어도 하나의 직접 입력과 상기 변환 데이터를 대응 인버터를 통해 공급하는 또 다른 입력을 각각 갖는 멀티플렉서의 제1 세트로서, 상기 제1 세트는 상기 변환 데이터에 의해서 제공되는 소정의 값에 의해 반전된 변환 데이터 또는 변환 데이터로 변환하도록 제어되는, 상기 멀티플렉스의 제1 세트와,

상기 멀티플렉서의 제1 세트에서 선택된 대응 멀티플렉서의 출력에 접속된 적어도 하나의 입력과 상기 제2 메모리에 접속되는 또 다른 입력을 각각 가진 멀티플렉서의 제2 세트로서, 상기 제2 세트는 상기 제2 메모리에 기억된 데이터를 변환하여 상기 제2 어드레스의 적어도 일부분을 상기 제1 메모리에 제공하도록 혹은 상기 제1 세트로부터의 각 멀티플렉서의 출력을 상기 제2 세트로부터의 대응 멀티플렉서의 출력으로 변환함으로써 상기 제1 메모리에 제1 어드레스를 제공하도록 상기 비교기 회로에 의해서 제어되는, 상기 멀티플렉스의 제2 세트와,

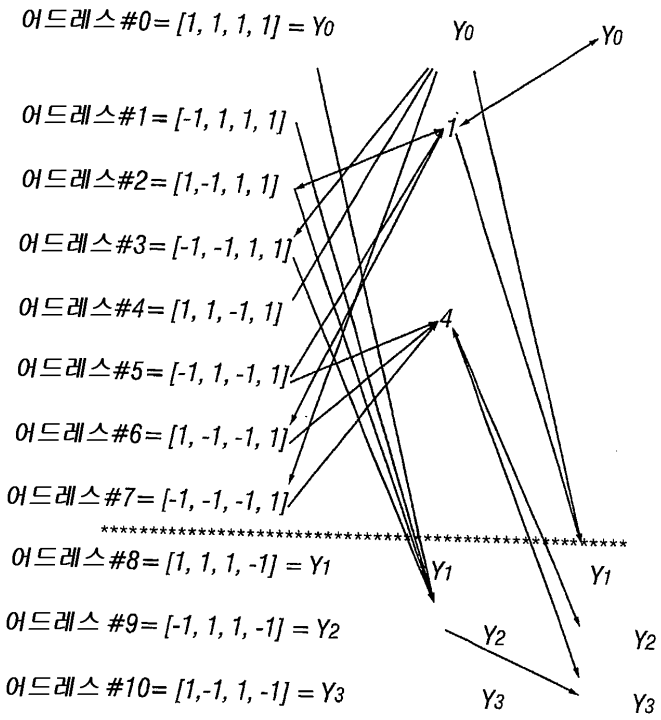
상기 데이터 처리 모드에서 상기 멀티플렉서의 제2 세트와 결합하여 동작하며 접속되지 않는 입력과 상기 제2 메모리에 접속된 입력을 가지며 상기 비교기 회로에 의해서 제어됨으로써 상기 제2 어드레스의 나머지 부분을 제공하는 멀티플렉서를 포함하는 것인 선형 변환 장치.

도면

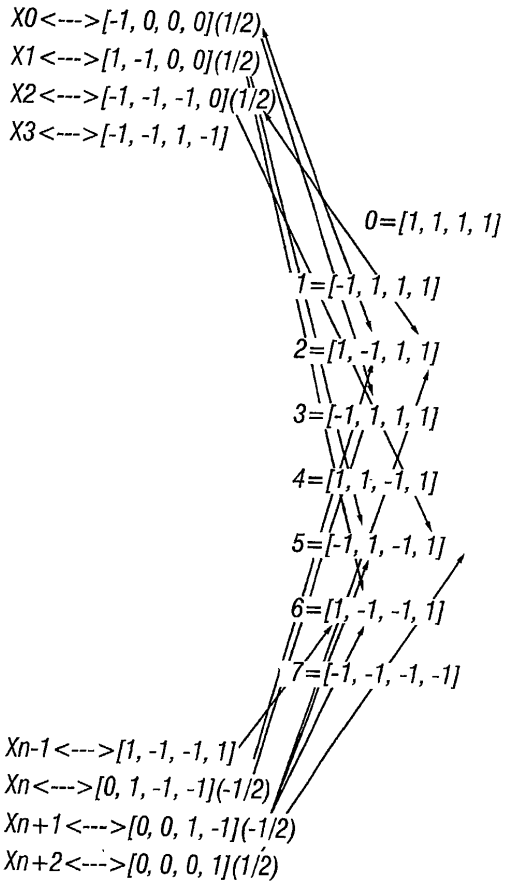
도면1



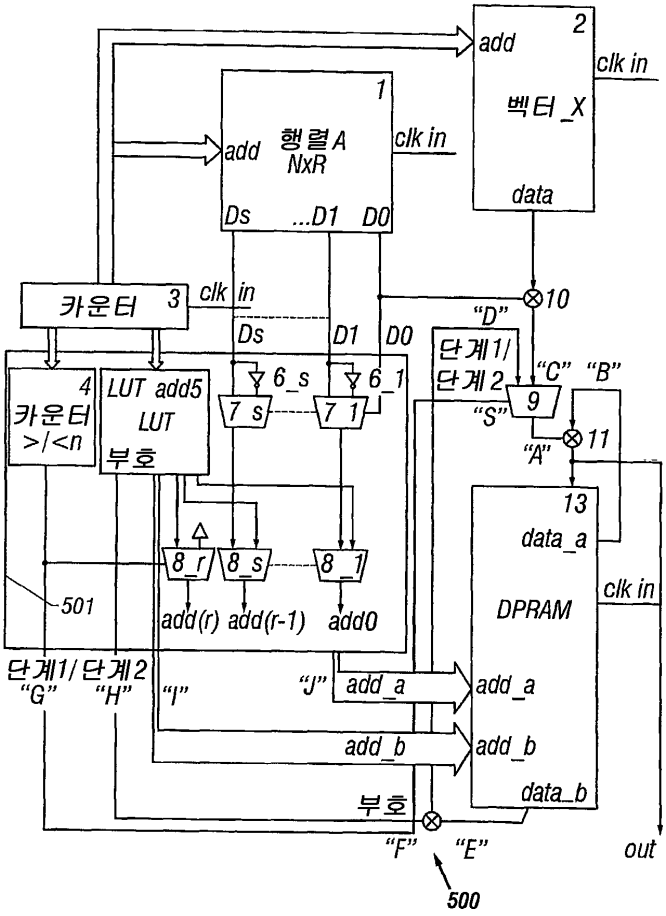
도면2


어드레스#0= [1, 1, 1, 1] = Y0
어드레스#1= [-1, 1, 1, 1]
어드레스#2= [1,-1, 1, 1]
어드레스#3= [-1, -1, 1, 1]
어드레스#4= [1, 1, -1, 1]
어드레스#5= [-1, 1, -1, 1]
어드레스#6= [1, -1, -1, 1]
어드레스#7= [-1, -1, -1, 1]
****
어드레스#8= [1, 1, 1, -1] = Y1
어드레스 #9= [-1, 1, 1, -1] = Y2
어드레스 #10= [1,-1, 1, -1] = Y3
Y0
Y0
1
4
Y1
Y1
Y2
Y2
Y3
Y3

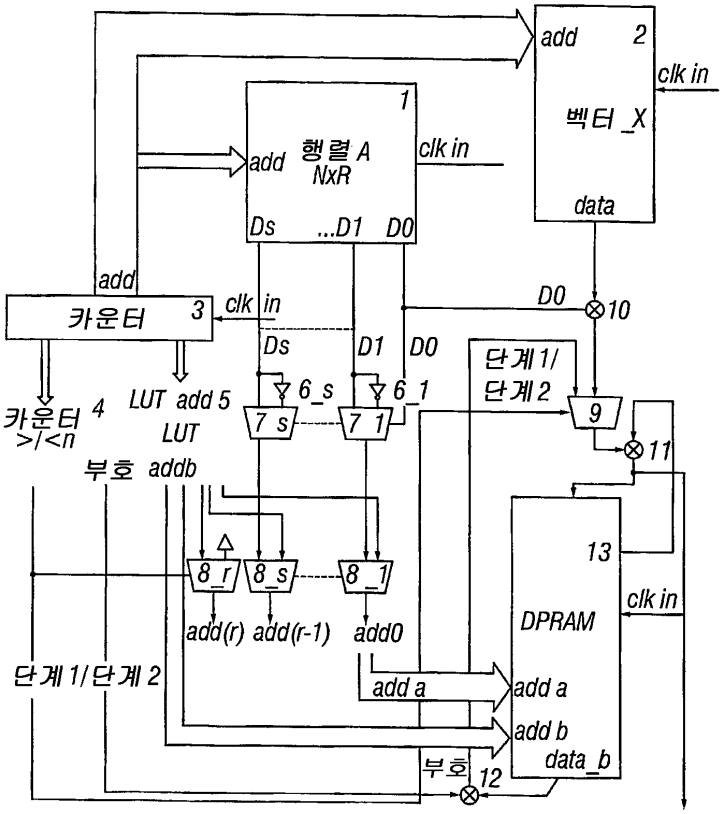
도면3


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)

도면4


2
add
clk in
벡터_X
data
1
add
행렬A
NxR
clk in
Ds
...D1
D0
카운터
3
clk in
Ds
D1
D0
10
"D"
단계1/
단계2
"C"
"B"
9
"S"
11
"A"
4
카운터
>/<n
LUT add5
LUT
부호
6_s
6_1
7_s
7_1
8_r
8_s
8_1
501
add(r)
add(r-1)
add0
13
data_a
clk in
DPRAM
단계1/단계2
"G"
"H"
"I"
"J"
add_a
add_a
add_b
add_b
data_b
부호
"F"
"E"
out
500

도면5


add
2
clk in
벡터 _X
data
1
add
행렬 A
NxR
clk in
Ds
...D1
D0
add
카운터
3
clk in
D0
10
Ds
D1
D0
단계 1/
단계 2
카운터
>/<n
4
LUT add 5
LUT
6_s
6_1
7_s
7_1
9
11
부호 addb
8_r
8_s
8_1
13
clk in
DPRAM
add(r)
add(r-1)
add0
단계 1/단계 2
add a
add a
add b
data_b
부호
12