SIMD 기반의 효율적인 4x4 정수변환 방법

유상준°, 오승준, 안창범
광운대학교 VIA 멀티미디어 센터
{ yusj1004°, sjoh }@media.kw.ac.kr, cbahn@daisy.kw.ac.kr

An Efficient 4x4 Integer Transform Algorithm on SIMD

Sang-Jun Yu°, Seoung-Jun Oh, and Changbeom Ahn
VIA-Multimedia Center, Kwangwoon University

요 약

DCT(Discrete Cosine Transform)는 현존하는 블록기반 영상 압축 코딩기법의 핵심이 되는 부분이다. 많은 고속 방법이 제안되었으며, 최근 들어 SIMD 병렬구조를 이용한 고속방법들이 제안되고 있다. 본 논문에서는 SIMD명령어를 가지는 프로세서에서 4x4 정수변환의 속도를 최적화하기 위한 알고리즘을 제안한다.

본 논문에서 제안하는 알고리즘은 128비트 SIMD명령어로 확장이 가능하며 비슷한 구조를 가지는 Hadamard 변환에서 적용할 수 있다. 제안하는 방법을 펜티엄4 2.4G에서 구현할 경우 H.264 참조 부호화기의 4x4 정수변환 방법보다 64비트 SIMD 명령어를 사용할 경우 4.34배 128-bit SIMD 명령어를 사용할 경우 6.77배의 성능을 얻을 수 있다.

1. 서 론

영상 신호와 관련하여 JPEG, MPEG, H.264과 같은 표준들은 블록 기반의 영상 압축 방식을 채택하고 있다. 상기한 표준에서는 영상신호내의 공간적 중복을 줄이기 위해 각 블록에 2D DCT을 적용한 후 고주파 성분을 제거하는 방식으로 압축 효과를 얻는다. 따라서 보편적으로 사용되는 영상및 비디오 데이터 압축방법의 핵심 모듈인 2D DCT를 효율적으로 수행하기 위한 많은 알고리즘이 제안되었다. 2D DCT 방법에는 1D DCT를 두 번 수행하는 방법과 2D에 대해 직접 적용하는 방법이 있다 [1][2].

요즘 쓰이는 대부분의 마이크로프로세서는 멀티미디어 응용 프로그램의 빠른 실행을 돕기 위한 멀티미디어 명령어를 포함하고 있다. 예를 들면 인텔의 제온, 펜티엄IV, AMD의 애슬론 64등은 SIMD(Single Instruction Multiple Data) 모델인 MMX, SSE, SSE2 명령어들을 모두 포함하고 있다. 이러한 명령어들은 하나의 명령으로 동시에 여러 데이터를 병렬로 처리한다. 그러나 SIMD 명령어를 사용할 경우 연산에 사용되는 시간보다 레지스터에 데이터를 적절히 포장(packing)하거나 해체(unpacking)하는 과정에서 소요되는 시간이 더 큰 경우가 많다 [3][4].

본 논문에서는 SIMD 명령어를 지원하는 프로세서에서 4x4 크기의 블록을 정수변환 방법으로 2D DCT를 수행하는 최적화된 SIMD 알고리즘을 제안한다. 제안하는 구조는 곱셈기가 필요 없으며, 입력 데이터와 출력 데이터에 대한 포장과 해제하는 과정과 산술 연산에 비해 큰 비중을 차지하는 올려놓기(load), 저장하기(store), 옮기기(mov)와 같은 명령을 줄임으로써 속도 향상을 얻는다.

2. 4x4 정수변환 방법

H.264는 주파수 영역에서 시공간영역이나 시공간 영역에서 주파수 영역으로 변환할 때 4x4 DCT/IDCT(Inverse DCT)를 사용하는 대신 4x4 정수변환 방법을 사용한다.

4x4입력 X에대한 4x4 DCT식은 다음과 같다.

$$Y = AXA^T = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix}$$

여기에서 $a = 1/2,\ b = \sqrt{1/2}\cos(\pi/8),\ c = \sqrt{1/2}\cos(3\pi/8)$ 값을 가진다. 위 식을 인수분해 하면 다음과 같은 식을 얻을 수 있다.

$$Y = (CXC^T) \otimes E$$

$$= \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{bmatrix} \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & d \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix}$$

여기서 E는 스케일링 팩터(Scaling Factor) 행렬이며, $\otimes$는 (CXC^T)와 E행렬의 같은 위치의 값을 서로 곱하는 기호이다. 그리고 $d = c/b \approx 0.414$를 갖는 상수 이다.

위의 행렬식의 간략화를 위하여 $d = 0.5$로 가정하여 대입하면 다음과 같은 행렬식으로 정리된다.

$$Y = (C_f X C_f^T) \otimes E_f$$

$$= \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{bmatrix}$$

여기에서 a, b, d의 값은 각각 $1/2, \sqrt{2/5}, 1/2$의 값을 갖는다. 위 식에서 $(C_f X C_f^T)$는 H.264에서 사용되는 정 방향 정수변환의 수식을 나타내며, 행렬의 곱으로 계산할 수 있다. 수식에서 처음과 마지막 행렬은 ±1, ±2의 정수 값만을 가지고 있으며, 이 값들은 덧셈, 뺄셈, 쉬프트(shift)연산으로 간단히 계산 할 수 있다. 이것을 '곱셈이 없는(Multiplication-free)' 방법이라고 하며 참조 부호화기에서 매우 효율적으로 사용된다 [5].

3. SIMD명령어를 이용한 구현 방법

SIMD는 프로세서 성능을 증가시키는 컴퓨터 기술 중의 하나이다. 그림 1은 일반적인 SIMD 동작을 보여준다. 두 쌍의 4개의 데이터(X1, X2, X3, X4 그리고 Y1, Y2, Y3, Y4)는 같은 명령어(op)를 각각의 데이터 쌍(X1과 Y1, X2과 Y2, X3과 Y3, X4과 Y4)에 병렬로 동작 한다. 연산의 결과는 4개의 데이터 위치에 순서대로 저장된다.

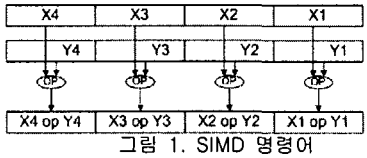

그림 1. SIMD 명령어

정수변환 방법을 SIMD로 구현하는 방법은 이차원 행렬 연산에 의한 방법과, 버터플라이(Butterfly) 표현식을 이용하는 방법, 4개의 행렬을 병렬 처리 하는 방법이 있다[6-9].

이차원 행렬 연산을 사용할 경우 곱셈 연산을 사용하기 때문에 곱셈기가 없는 프로세서인 경우에는 다른 명령어로 대치되어 성능이 저하 될 수 있다. 대신 정수변환 뿐만 아니라 일반적인 4x4 DCT변환을 수행 할 수 있다.

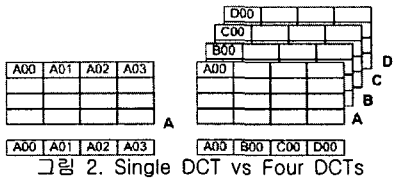

그림 2. Single DCT vs Four DCTs

두 번째 방법은 그림 2에서와 같이 서로 독립적인 행렬을 병렬 처리 하는 방법이다. 인접한 4개의 블록을 하나의 블록으로 포장하여 처리함으로써 동시에 4개의 블록이 수행되는 결과를 가져올 수 있다. 대신 연산전후에 추가적으로 데이터의 포장, 해제과정이 필요하며 이 연산들은 메모리에 접근해야 하기 때문에 많은 클럭이 소요된다. 또 다른 방법으로는 그림 3과 같은 버터플라이 표현식을 이용하는 방법이다.

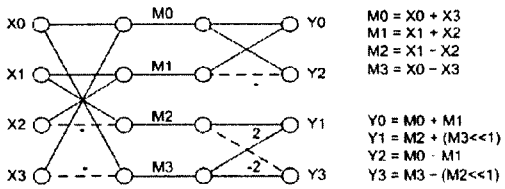

그림 3. 정 방향 정수변환의 butterfly 표현

이 방법은 구현은 용이 하지만 동시에 여러 데이터를 처리하지 못하여 SIMD 레지스터의 슬롯 사용에 대한 효율이 떨어진다. 또한 전체 소요되는 사이클 중에 메모리 접근하는 명령이 75%정도로 큰 비중을 차지해 성능이 떨어진다.

4. 제안 방법

2장에서 언급했듯이 H.264에 사용되는 기저 행렬은 ±1, ±2 값만 가지고 있다. 따라서 곱셈기를 사용하지 않고 간단히 덧셈, 뺄셈, 쉬프트 명령만을 이용해 계산하도록 구현이 간단한 정수변환 버터플라이방식을 적용하였다. 또한 최소한의 메모리 접근을 통해서 속도향상을 얻는다.

2D DCT $(C_f X C_f^T)$는 1D (XC_f) DCT를 수행하고, 그 결과를 전치(Transpose)한 후 다시 1D $(X^T C_f)$ DCT 과정을 통하여 계산된다.

$$(XC_f) = \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix}$$

(XC_f)는 행렬 X의 x_{ij}에 대한 행렬 C의 c_{ij}를 곱하는 연산이다. 정수변환의 버터플라이 표현을 보면 입력 X의 한 행에 대한 c_{ij}의 연산은 피연산자 x_{ij}에 대한 연산이 각각 달라 SIMD 구조에서 병렬로 처리하기에 효율적이지 않다. 이 점을 해결하기 위해 입력에 대해 첫 번째 1D 4x4 정수변환을 수행하기 전에 전치행렬을 적용하였다. 이렇게 함으로써 입력 X를 1x4 행렬처럼 처리할 수 있다. 그림 4와 5는 4x4입력에 대한 전치 과정과 제안하는 알고리즘의 전체 구조를 나타낸다.

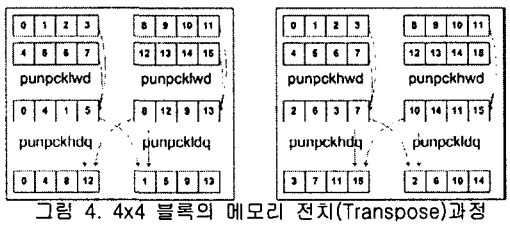

그림 4. 4x4 블록의 메모리 전치(Transpose)과정

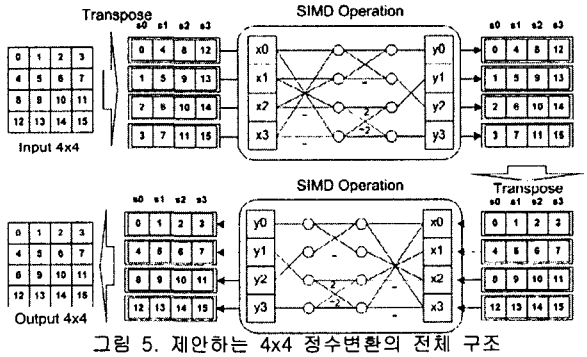

그림 5. 제안하는 4x4 정수변환의 전체 구조

그림 6은 4x4에 대한 정수변환의 핵심부분을 SIMD로 구현한 예이다. 입력 행렬 X는 그림 4의 방법으로 전치되기 있기 때문에 행과 열이 바뀐 상태가 된다. 이때 네 개의 입력 데이터 x00, x10, x20, x30를 하나의 64비트 레지스터 x0p에 올려놓는다. x0p는 행렬 X의 0번째 열의 데이터가 순차적으로 저장된 것을 의미 한다. 이렇게 4x4 입력 데이터가 x0p, x1p, x2p, x3p의 네 개의 64비트 레지스터에 올려질 수 있다. 64비트 레지스터에 그림 3의 정수변환 버터플라이 방법을 적용하면 행렬 X대한 4x4 1D DCT 정수변환 과정이 이루어진다. 이때 그림 3의 덧셈, 뺄셈, 쉬프트 연산은 SIMD 명령어인 PADDW, PSUBW, PSLLW로 대치하여 사용한다[4].

1D 4x4 정수변환의 결과 y0p, y1p, y2p, y3p를 다시 그림 4의 방법으로 전치 한 후 그림 6의 과정을 반복하면 원하는 2D 4x4 정수변환의 결과를 얻을 수 있다.

지금까지는 64비트 명령어와 64비트 레지스터를 이용한 4x4 입력에 대한 정수변환 과정을 설명 하였다. 동일한 방법을 128비트로 확장이 가능하며, 이때는 4x4 블록 두 개를 동시에 처리 할 수 있다. 이때 전치과정 후에 그림7의 과정이 추가적으로 필요 하다.

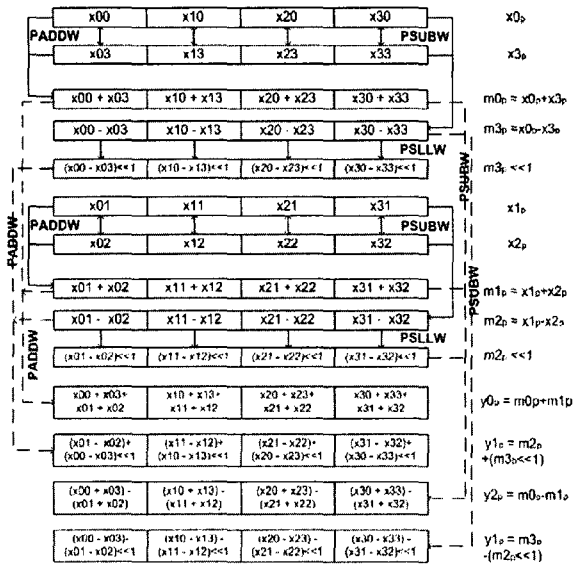

그림 6. 4x4 정수변환 핵심부분에 대한 SIMD 구현 사례

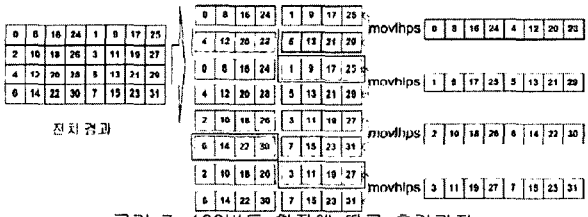

그림 7. 128비트 확장에 따른 추가과정

5. 실험 결과

실험에 사용된 프로세서는 인텔 펜티엄IV 2.4G로 64비트 MMX와 128비트 SSE2 명령어를 지원한다. 성능 분석을 위한 소프트웨어로는 VTune 7.0 버전을 사용하였다[10].

표1은 참조 부호화기의 4x4 정수변환 모듈과, MMX(64비트)를 이용한 행렬의 곱셈 모듈, 제안하는 MMX, SSE2를 이용한 모듈의 수행 시간과 마이크로 명령어 단위의 클럭수(clocktick)를 비교하여 정리한 것이다. 무작위로 생성된 입력 데이터를 MMX 경우 2000000회 SSE2는 1000000회 반복하였다. 결과적으로 4x4 정수변환을 2000000회 수행한 것이다.

표 1. 수행 시간과 성능 비교

성능비 / 방법	수행시간 (단위:초)	multiplication-free	행렬곱셈 (MMX)	제안방법1 (MMX)
multiplication-free	0.817s	1.000	0.286	0.230
행렬곱셈 (MMX)	0.234s	3.490	1.000	0.804
제안방법1 (MMX)	0.188s	4.341	1.244	1.000
제안방법2 (SSE2)	0.121s	6.776	1.941	1.561

표2는 알고리즘별 명령어에 소요되는 클럭수을 측정한 결과를 정리한 것이다. 행렬곱셈을 제외한 방법에서는 곱하기 연산이 쓰이지 않았으며, 메모리 접근과 관련된 이동에 소요된 클럭수를 전체 클럭수로 나눈 비율이 행렬곱셈 방법을 제외한 세 가지방법 중에서 제안한 방법2가 가장 낮았다.

표 2. 각 모듈별 clocktick 비교

명령어 / 방법	multiplication-free)	행렬곱셈 (MMX)	제안방법1 (MMX)	제안방법2 (SSE2)
mov [A]	522	100	114	71
add,sub,shift	102	31	26	14
mul	0	21	0	0
else	76	52	25	22
Total Clocktick[B]	700	204	165	107
[A]/[B]	0.746	0.490	0.691	0.664
성능 향상비	1.000	3.431	4.242	6.542

행렬곱셈은 [A]/[B]값이 작은 대신에 곱셈 연산이 추가되어 전체적로 낮은 성능을 보여 주는데 이러한 이유는 일반적으로 덧셈, 뺄셈, 쉬프트 연산에 비해 이동 명령어가 2배에서 6배정도의 많은 클럭이 소요되기 때문이다[10].

6. 결론

본 논문에서는 H.264에서 사용하는 정수변환 방법을 SIMD 명령어로 최적화 시키는 방법을 제안 하였다. 이 알고리즘을 적용하여 64비트 명령어를 사용하였을 경우 참조 부호화기의 정수변환 방법에 비해 4.34배, 128비트 명령어를 사용할 경우는 6.77배의 성능을 얻을 수 있었다. 또한 이 구조는 곱셈기가 없는 프로세서에서 효율적으로 쓰일 수 있다.

감사의 글

본 연구는 한국과학재단 목적기초연구 (R01-2002-000-00179-0) 과제로 수행되었음.

참고문헌

[1] Iain E.G Richardson, *H.264 and MPEG-4 VIDEO COMPRESSION*, WILEY, 2003

[2] Feig, E. and Winograd, S, "Fast Algorithms for Discrete Cosine Transform", *IEEE Transactions on Signal Processing*, vol.40, pp.2174-2193, Sept. 1992

[3] Richard G., "The Software Optimization Cookbook", Intel Press, 2002

[4] Intel Corp., "Intel Pentium 4 and Intel Xeon Processor Optimization - Reference Manual", Order Number: 248966-05, 2002

[5] Iain E., G Richardson, "H.264 White paper - White Papers describing aspects of the new H.264 / MPEG-4 Part 10 standard", May 12, 2004

[6] X. Zhou, Eric Q. Li and Yen-Kuang Chen, "Implementation of H.264 Decoder on Ceneral-Purpose Processors with Media Instructions", *Proceeding of SPIE conference on Image and Video Communication and Processing*, volume 5022, January, 2003

[7] Intel Corp., "Streaming SIMD Extensions - Matrix Multiplication" ,AP-930, June, 1999

[8] Intel Corp., "Using Streamin SIMD Extensions in a Fast DCT Algorithm for MPEG Encoding" Ver1.2 Jan. 1999

[9] Intel Corp., "Using Streaming SIMD Extensions 2(SSE2) to Implement an Inverse Discrete Cosine Transform", Version 2.0, July, 2000

[10] Intel Corp., Intel® VTune™ Performance Analyzer, Version 7.0, 2003