

SIMD 명령어를 이용한 H.264 인코더 최적화

김용환, 김체우, 김태완, 최병호
전자부품연구원 디지털미디어연구센터

Optimization of H.264 Encoder using SIMD Instructions

Yong-Hwan Kim, Je-Woo Kim, Tae-Wan Kim, Byeongho Choi
Digital Media Research Center, Korea Electronics Technology Institute
yonghwan@keti.re.kr

요약

최근에 표준화가 완료된 차세대 비디오 코딩 표준인 H.264는 적은 비트율에서 높은 품질의 비디오 압축을 목표로 하기 때문에, H.263+ 및 MPEG-2/4 와 같은 이전의 표준들보다 훨씬 더 많은 연산을 필요로 한다. 본 논문은 SIMD (Single Instruction Multiple Data) 명령어를 가지는 범용 프로세서(예를 들면, 펜티엄 4)에서 H.264 S/W 인코더의 속도 최적화를 위한 알고리듬 및 구현 기술을 제안한다. 화질 저하 없이 RDO (Rate Distortion Optimization) 의 속도를 높일 수 있는 효율적인 모드 검색 건너뛰기 알고리듬을 제안하고, SIMD 명령어를 이용하여 1/4 화소 보간, SAD(Sum of Absolute Difference), SATD(Sum of Absolute Transformed Difference), SSD (Sum of Squared Difference) 등의 개별 루틴의 속도를 최적화한다. 일련의 최적화 후에 인코더는 화질 저하 없이 H.264 레퍼런스 인코더보다 평균 3배 정도의 속도 향상이 이루어진다.

1. 서론

요즘에 쓰이는 대부분의 마이크로프로세서는 멀티미디어 응용 프로그램의 빠른 실행을 돋기 위한 멀티미디어 명령어를 포함하고 있다. 예를 들면 인텔의 제온과 펜티엄 4 및 AMD의 에슬론64는 SIMD 모델인 MMX, SSE 및 SSE2 명령어들을 모두 포함하고 있다. 이러한 명령어들은 하나의 명령으로 동시에 여러 계산들을 수행할 수 있게 해준다 [1, 2].

최근에 표준화가 완료된 차세대 비디오 코딩 표준인 H.264는 기존의 MPEG-2, MPEG-4 Part 2, H.263+ 등의 코덱보다 적은 비트율에서 고화질의 비디오 압축을 목표로 설계되었다. H.264 코딩 표준에는 기존의 비디오 코덱들과 마찬가지로 블록 기반의 하이브리드 MC/Transform 모델이 채택되었지만, 코딩 효율을 높이기 위한 많은 새로운 기법들이 추가되었다. 결과적으로 새로 추가된 기법들에 의해 코딩 효율은 높아졌지만 기존의 코덱들보다 훨씬 더 많은 연산을 요구하게 되었다[3, 4].

많은 개별 모듈 및 알고리듬의 복합 작용으로 인해서 많은 연산량을 가지는 비디오 코덱의 최적화는 크

게 세 부류로 나눌 수 있다. 첫 번째는 개별 모듈의 알고리듬 최적화이다. 예를 들면 움직임 예측 모듈을 빠른 검색 알고리듬으로 대체하는 것이다. 두 번째는 코덱 전체의 데이터 흐름 구조를 최적화하는 것이다. 예를 들면, 1/4 화소 보간을 모든 블록에 대해서 미리 한꺼번에 하는 것이 움직임 예측 및 보상시에 블록마다 개별적으로 중복 수행하는 것보다 훨씬 효율적이다. 세 번째는 개별 모듈의 코드 최적화이다. 예를 들면 SAD 루틴을 SIMD 명령어로 대체하면 화질 저하 없이 몇 배의 속도 향상을 이룰 수 있다. 일반적으로 첫 번째 방법은 약간의 화질 저하를 가져올 수 있지만, 나머지 방법들은 화질 저하 없는 속도 향상을 가져다 줄 수 있다.

본 논문의 2장에서는 레퍼런스 H.264 인코더 S/W의 모듈 실행시간을 분석하고 1/4 화소 보간, RDO, SAD/SSD /SATD 루틴을 알고리듬 및 SIMD 명령어를 이용해서 화질 저하 없는 속도 최적화 기법을 제안한다[5]. 3장에서는 실험결과를 제시하고, 마지막으로 4장에서 결론 및 향후 연구과제로 끝을 맺는다.

2. H.264 인코더 분석 및 최적화 방법

H.264 레퍼런스 인코더 JM72[6]의 실행 시간을 모듈별로 측정하여 그림 1에 보인다[7]. 측정에 사용된 영상은 QCIF 크기의 foreman 100 프레임이고, 사용된 인코더의 파라미터를 표 1에 보인다.

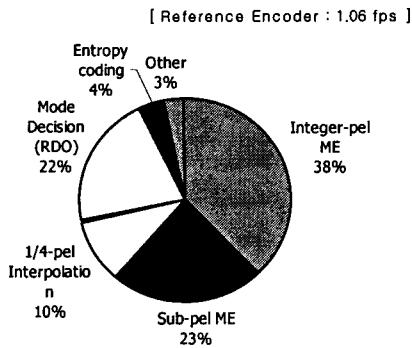


그림 1. H.264 레퍼런스 인코더의 모듈별 실행 시간

표 1. 인코더의 파라미터

파라미터	값
프로파일	베이스라인
SATD (Hadamard)	On
RDO	On
검색 범위	16
참조 프레임 개수	3
인트라 주기	32
양자화 계수	26

그림 1에서 모드 선택(Mode Decision) 루틴 내에 정수 변환, 양자화, 역양자화, 역변환 및 인트라 예측이 포함되어 있다. 움직임 예측 루틴(Integer-pel ME, Sub-pel ME)이 가장 많은 연산을 필요로 하고, RDO에 의해 모드 선택 루틴이 두 번째로 많은 연산을 필요로 한다. Integer-pel ME 루틴 내에 SATD의 연산은 Sub-pel ME 루틴 내에 포함되어 있다. 세 번째로 많은 연산량은 1/4 화소 보간 루틴이 차지한다.

2.1) 1/4 화소 보간 루틴

H.264는 1/4 화소 단위까지 움직임 예측을 수행한다. 먼저 필터 계수가 $(1, -5, 20, 20, -5, 1)$ 인 6-tap FIR 필터로 1/2 화소 보간을 수행한 후에 정수 화소와 1/2 화소를 평균해서 1/4 화소 값을 구한다. H.264에서는 움직임 벡터가 꽉쳐 경계 밖을 가리킬 수 있기 때문에 보간 루틴에서 경계 화소 패딩을 수행하는

것이 인코더의 구조적인 최적화의 한 방법이 될 수 있다. QCIF 영상의 경우 보간 루틴의 순서도는 그림 2와 같다. 16 화소 패딩된 208×176 영상은 Integer-pel ME/MC에서 유용하게 쓰이고, 48 화소 패딩된 832×704 영상은 Sub-pel ME/MC에서 유용하게 쓰인다. 왜냐하면 움직임 예측/보상 과정에서 움직임 벡터가 꽉쳐 경계 밖을 가리키는지를 계속적으로 체크할 필요를 없애기 때문이다. 6-tap 필터는 SIMD 명령어인 pmaddwd를 이용하여 화소 데이터와 필터 계수와의 벡터곱셈으로 구현할 수 있다[5]. 필터링하기 전에 패딩된 대부분의 영역은 필터링에서 제외하고 필터링이 모두 끝난 후에 패딩으로 처리한다. 그림 3은 SSE2 명령어를 이용한 6-tap 필터 연산을 행(row) 방향과 열(column) 방향으로 나누어서 보여준다. 수평 방향의 필터링의 경우 $R1 \times C3$ 에 마지막 곱셈인 $1 \times X(18)$ 을 별도로 계산해서 더해줘야 한다.

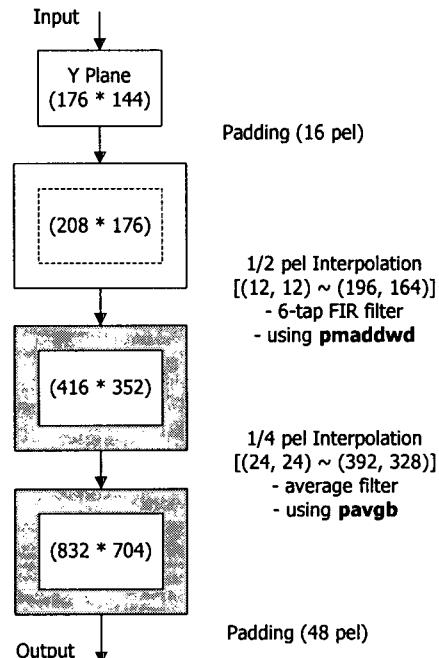


그림 2. 1/4 화소 보간 루틴의 순서도

2.2) 정수 변환/양자화 이후 계수들이 모두 0의 값을 갖는 블록 (AZCB : All Zero Coefficient Block)

AZCB 블록이 존재할 경우 적용할 수 있는 세 가지 속도 향상 기법을 제안한다.

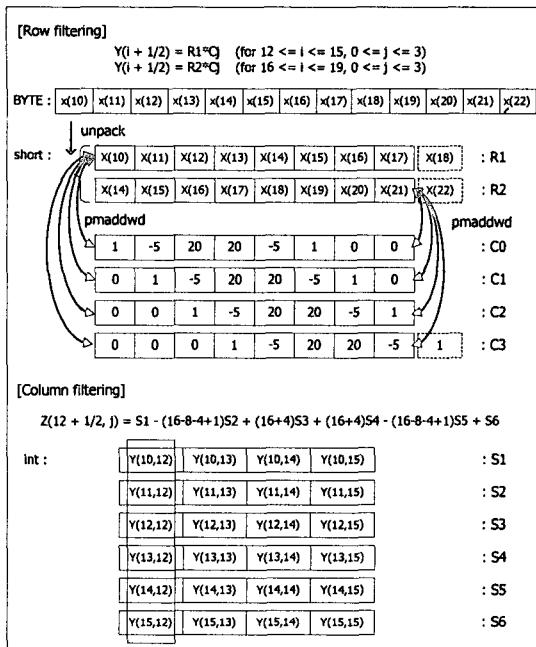


그림 3. SSE2 명령어를 이용한 6-tap 필터링 연산

① 역변환 및 역양자화 건너뛰기

H.264 인코더뿐만 아니라 현대의 하이브리드 MC/Transform 모델을 갖는 비디오 인코더들은 DCT 등의 변환을 수행하고 양자화를 수행한 후에, 디코딩된 프레임을 만들기 위해서 역양자화 및 역변환을 수행한다. 이 과정에서 AZCB는 역양자화 및 역변환을 건너뛰고 움직임 보상된 블록을 바로 디코딩되는 프레임에 복사해주면 된다. H.264 이전의 코덱들은 변환 및 양자화에 기본적으로 8x8 블록을 이용했지만 H.264는 4x4 블록을 이용하기 때문에 AZCB 발생 확률이 더 높다.

② RDO에서 P4x4 모드 검색 건너뛰기

8x8 블록의 RDO에서 P8x8 블록이 AZCB이고 RD 비용이 SRDO 보다 작으면 P4x4 모드 검색을 건너뛸 수 있다. SRDO의 값은 현재 1000으로 설정했다.

③ RDO에서 I4x4 모드 검색 건너뛰기

16x16 블록의 RDO에서 P16x16 블록이 AZCB이면 I4x4 모드 검색을 건너뛸 수 있다.

위의 세 가지 알고리듬을 모두 적용하면 화질 저하는 전혀 없이 평균 15% 정도의 전체 속도 향상을 이룰 수 있다.

2.3) SIMD 명령어를 이용한 기타 루틴의 최적화

SAD는 psadbw 명령어를, SSD는 pmaddwd 명

령어를 이용하여 구현한다. SATD는 4x4 블록의 차분을 구하고 Hadamard 변환을 수행한 후에 계수의 절대값의 합을 구하는 연산이므로 Hadamard 변환의 연산 속도를 높이는 게 관건이다. Hadamard 변환을 SIMD 명령어로 구현하는 방법은 이차원 행열 연산에 의한 방법과[5], 그림 4와 같은 Butterfly 표현식을 이용하는 방법이 있다.

<Forward & Inverse Transform>

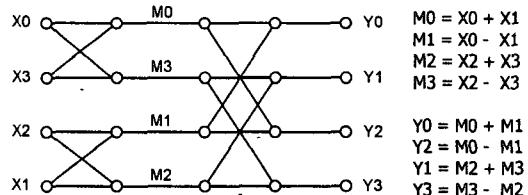


그림 4. Hadamard 변환의 Butterfly 표현

3. 실험 결과

실험은 인텔 제온 2.0 GHz CPU를 가진 PC에서 수행되었다. 제안된 알고리듬 및 구현 방법으로 최적화 후에 H.264의 모듈별 실행 시간을 그림 5에 보인다. 인코더의 파라미터는 표 1과 동일하다.

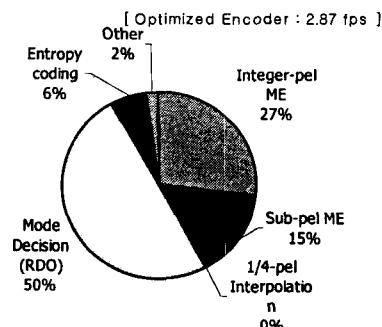


그림 5. 최적화된 H.264 인코더의 모듈별 실행시간

본 논문에서 제안한 1/4 화소 보간 루틴을 이용하고 SIMD 명령어로 구현한 결과 보간 루틴의 연산량이 획기적으로 줄어들었음을 알 수 있다. 또한 SAD/SSD/SATD 등의 루틴을 SIMD 명령어로 구현하였기 때문에 움직임 예측 루틴의 연산량도 상당히 줄어든 것을 확인할 수 있다. 표 3에 각각의 모듈의 속도 향상을 보이고, 표 4에서 여러 방법으로 구현한 Hadamard 변환 루틴의 실행 시간을 비교한다. Hadamard 루틴의 시간은 사천만번을 실행시킨 후 측정한 시간이다.

표 2. H.264 인코더의 최적화 결과 비교

영상	QP	12		18		26		32		36	
		JM72	New	JM72	New	JM72	New	JM72	New	JM72	New
Mother & Daughter	속도	0.95	2.36	1.03	2.94	1.14	3.57	1.24	3.74	1.28	3.77
	PSNR	48.96	48.97	45.18	45.18	40.01	40.01	36.58	36.58	34.47	34.47
	크기	505474	505670	210520	210632	68289	68289	30355	30355	18058	18058
Foreman	속도	0.82	2.15	0.92	2.48	1.06	2.87	1.13	3.27	1.19	3.19
	PSNR	48.12	48.12	43.60	43.61	38.21	38.22	34.98	34.98	32.91	32.91
	크기	1156725	1156824	523599	523175	157039	157184	67804	67804	42216	42216
News	속도	0.23	0.66	0.26	0.75	0.28	0.93	0.30	1.02	0.31	1.05
	PSNR	49.09	49.09	45.21	45.21	40.44	40.44	36.94	36.94	34.57	34.57
	크기	1861636	1860904	738022	738179	275885	275983	137122	136989	87021	87032
Mobile & Calendar	속도	0.19	0.45	0.20	0.50	0.23	0.59	0.25	0.68	0.27	0.74
	PSNR	47.67	47.67	42.59	42.59	36.04	36.04	31.64	31.64	29.03	29.03
	크기	8231781	8232080	4945744	4945744	2012417	2011975	765558	765558	377500	377605

표 3. SIMD 명령어를 구현한 모듈별 속도 향상 비율

모듈	속도 향상
1/4 화소 보간	3.5x
SAD	2.0x
SSD	1.9x
SATD	1.8x

모드 검색 건너뛰기 알고리듬 및 SIMD 명령어를 이용한 최적화와 데이터의 흐름과 관계된 구조적인 최적화 과정을 거친 이후에 인코더는 JM72 인코더와 비교하여 화질 저하는 전혀 없이 평균 3배의 속도 향상을 보인다. 표 2에 여러 가지 영상(200 프레임) 및 양자화 계수에 대해서 테스트한 전체 실험 결과를 보인다. Mother...와 Foreman 영상은 QCIF 크기이고, News 와 Mobile... 영상은 CIF 크기이다. 속도의 측정 단위는 fps, 크기의 측정 단위는 byte 이다.

표 4. Hadamard 변환 루틴들의 실행시간 비교

구현 방법	시간 (초)	속도 향상
C_Matrix	3.78	1.0x
C_Butterfly	0.59	6.4x
SSE_Matrix	0.56	6.8x
SSE_Butterfly	0.33	11.5x

4. 결론 및 향후 연구 과제

본 논문에서는 H.264 인코더의 성능을 측정하고 분석하고 최적화하였다. 처음부터 본 논문이 목표로 삼은 것은 화질 저하 없는 속도 최적화였기 때문에, 화질 저하를 가져올 수 있는 알고리듬, 예를 들면 빠른 움직임 예측 루틴, 은 구현되지 않았다. 그 대신에 본 논문에서는 1/4 화소 보간 루틴에서 적용한 경계 화

소 패딩을 통해서 보간 루틴 뿐만 아니라 움직임 예측 루틴의 구조적인 최적화를 구현하였고, RDO 루틴에서의 화질 저하 없는 모드 검색 건너뛰기 알고리듬을 제안하였고, SIMD 명령어를 이용하여 연산량이 많은 루틴들을 최적화하였다. 만약 빠른 움직임 예측 루틴과 좀 더 지능적인 RDO 루틴이 설계되고, SIMD 명령어를 이용한 부분 모듈의 최적화가 좀 더 이루어진다면 고화질의 PC용 실시간 베이스라인 H.264 S/W 인코더도 충분히 구현 가능하리라고 보여진다.

[참고문헌]

- [1] Richard Gerber, "The Software Optimization Cookbook", Intel Press, 2002
- [2] Intel Corp., "Intel Pentium 4 and Intel Xeon Processor Optimization - Reference Manual", Order Number: 248966-05, 2002
- [3] Draft ITU-T Rec. and FDIS of Joint Video Spec. (H.264 | ISO/IEC 14496-10 AVC), JVT-G050r1, Geneva, 23-27 May, 2003
- [4] T. Wiegant, et al., "Overview of the H.264/AVC Video Coding Standard", IEEE Trans. on Circuits and Systems for Video Technology, Vol. 13, No. 7, pp.560-576, July 2003.
- [5] X. Zhou, et al., "Implementation of H.264 Decoder on General-Purpose Processors with Media Instructions", Proceeding of SPIE Conference on Image and Video Communication and Processing, vol. 5022, Jan. 2003
- [6] <http://bs.hhi.de/~suehring/tm/download>
- [7] Intel Corp., Intel® VTune™ Performance Analyzer, version 7.0, 2003.