PC 기반 지상파 DMB 수신기를 위한 H.264 복호 SW 모듈 최적화†

윤동환(conan@lena.uos.ac.kr), 김용한(yhkim@uos.ac.kr)
서울시립대학교 공과대학 전자전기컴퓨터공학부

Optimization of H.264 Decoder Software Module for PC-based T-DMB Receivers

Youn, Dong-hwan (conan@lena.uos.ac.kr) and Yong Han Kim (yhkim@uos.ac.kr)
Dept. of Electrical and Computer Eng., University of Seoul

본 논문에서는 PC 기반 지상파 DMB(Terrestrial Digital Multimedia Broadcasting, T-DMB) 수신기를 위한 SW 최적화에 대해 설명한다. 이 수신기는 PC 외부에 지상파 DMB 신호를 안테나로 수신하여 복조하고 채널 복호하는 프론트 엔드(front-end) 수신 모듈을 이용, USB를 통하여 RS(Reed-Solomon) 부호화된 MPEG-2 TS(Transport Stream) 데이터를 읽어 들여 RS 복호, TS 역다중화, 비디오 복호, 오디오 복호 등의 SW 처리 과정을 거쳐 디스플레이 상에 수신 내용을 표시하게 된다. 본 논문에서는 저사양 PC에서도 T-DMB를 수신할 수 있도록 H.264|MPEG-4 AVC(Advanced Video Coding) 복호 과정을 최적화한 결과에 대해 설명한다.

주제어: H.264, AVC, 복호기 최적화, 지상파 DMB, T-DMB

1. 서론

지난 2004년 8월 확정된 지상파 DMB(Terrestrial Digital Multimedia Broadcasting, T-DMB) 표준[1]은 제한된 대역폭에 좋은 품질의 다채널 서비스를 제공하기 위하여 성능 평가를 거쳐 비디오 코덱으로는 H.264|MPEG-4 AVC(Advanced Video Coding)[2](이하 H.264|AVC로 부름.)가 선정되었고 오디오 코덱으로는 MPEG-4 ER-BSAC(Error-Resilient Bit-Sliced Arithmetic Coding)가 선정되었다.

이 중 H.264|AVC는 MPEG-4 Part.2 Visual[3]과는 달리 사각 프레임의 압축률 향상을 목표로 제정되었다. 실제로 H.264|AVC는 공간 영역(spatial domain)에서의 화면 내 예측(Intra Prediction), 1/4 또는 1/8의 해상도를 갖는 움직임 벡터(motion vector)의 사용, 다양한 블록 단위 사용 움직임 보상(motion compensation), 움직임 보상 시에 다중 참조 프레임(multiple reference frame)의 사용, 정수 변환(integer transform)의 사용, CAVLC(Context Adaptive Variable Length Coding) 혹은 CABAC(Context Based Adaptive Binary Arithmetic Coding)의 사용, 루프 내(in-loop) 디블록킹 필터(In-loop deblocking filter)의 사용 등으로 MPEG-4 Part 2 Visual SP(Simple Profile)에 비해서 30퍼센트 이상의 압축 효율 개선 효과를 내는 것으로 알려져 있지만 디코더의 복잡도는 3배 정도에 이르는 것으로 알려져 있다.

본 논문에서는 T-DMB 수신기를 위한 H.264|AVC 복호기의 최적화에 대해 다룬다. 이를 위해 각 루틴별 최적화와 메모리 연산을 줄이는 등의 일반적인 사항에 대한 최적화에 대해서 설명한다.

2. T-DMB 수신기 프로파일링(profiling)

PC 기반 T-DMB 수신기의 병목 부분(bottleneck)이 어느 곳이며 어느 정도인지 알아내기 위하여 인텔에서 만든 VTUNE 성능 분석기(Performance Analyzer)[4]라는 프로그램을 이용해서 프로파일링(profiling)을 수행하였다. 좀 더 구체적으로는 VTUNE에서 제공하는 샘플링(sampling) 방법을 이용해서 성능 측정을 했다.

그림 1에 T-DMB 수신기를 프로파일링한 결과를 나타내었다. 그림에 나타난 모듈 중에서 T-DMB와 관련된 것은 4 가지이다. JM_DLL.dll은 H.264|AVC 복호기 모듈이고 DMB_Receiver.exe는 T-DMB 시스템과 관련된 모듈이며 BSACdll.dll은 MPEG-4 ER-BSAC 오디오 복호기 모듈이다. 이외에 YUV 4:2:0 포맷을 RGB 포맷으로 변환해 주는 역할을 하는 yuv2rgb.dll이 있다.

† 본 연구는 대학 IT 연구센터 육성 지원사업의 연구 결과로 수행되었음.

그림에서 보는 것처럼 H.264|AVC 복호기 모듈이 T-DMB 수신기 성능에 가장 큰 영향을 주고 있고, 중점적으로 최적화를 수행해야 하는 부분임을 알 수 있다.

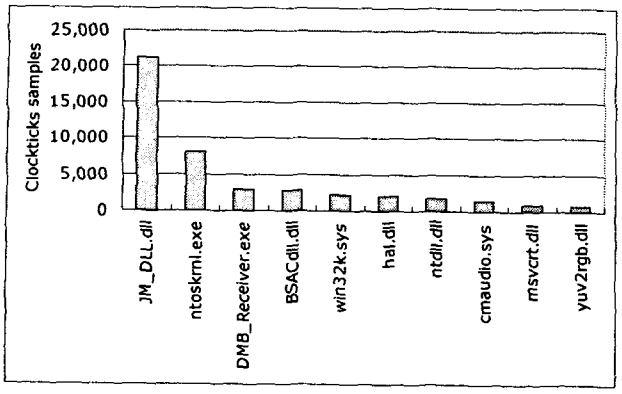

그림 1 T-DMB 수신기 프로파일링

3. H.264|MPEG-4 AVC 복호기 프로파일링

H.264|AVC 복호기 모듈을 최적화하기로 정하였으므로 어떠한 모듈이 성능 저하에 가장 큰 영향을 주고 있는지 파악할 필요가 있다.

아래 그림 2에 H.264|AVC 복호기를 프로파일링한 결과를 보였다. 여기서 사용한 복호기는 ITU-T에서 제공하는 참조 소프트웨어(reference software)[5] 버전 7.3을 기본으로 해서 DLL(Dynamic Linked library)로 제작된 것이다.

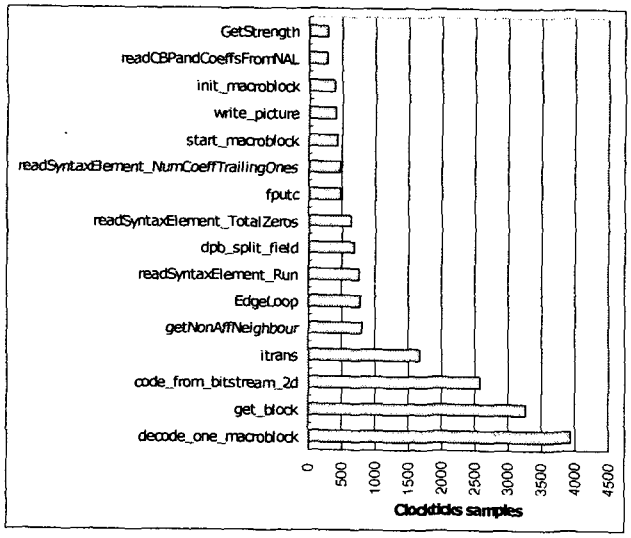

그림 2 H.264|AVC 복호기 프로파일링

그림에서 보는 것처럼 decode_one_macroblock(), get_block(), code_from_bitstream_2d(), itrans(), getNonAffNeighbour(), EdgeLoop() 등의 함수들이 수행 시간의 대부분을 차지하고 있음을 알 수 있다.

프로파일링한 결과를 그림 3에 나타낸 H.264|AVC 복호기 블록다이어그램에 따라 살펴보면 움직임 보상을 위한 화소 보간, 엔트로피 복호, 역변환, 화면 내 예측, 루프 내 디블록킹 필터 등의 순으로 수행 시간을 차지하고 있음을 알 수 있다. 이 데이터를 바탕으로 최적화를 진행하였다.

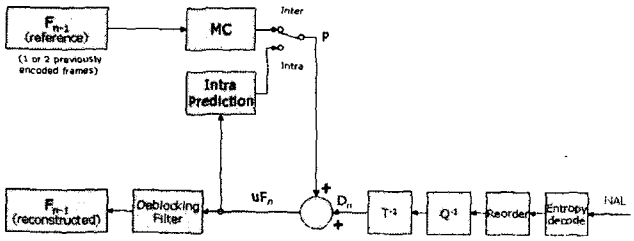

그림 3 H.264|AVC 복호기 블록다이어그램

4. H.264|MPEG-4 AVC 복호기 최적화

4.1 T-DMB 제한 사항

본 논문은 범용의 H.264|AVC 복호기의 최적화를 목적으로 하는 것이 아니라 T-DMB 수신기를 위한 최적화를 목표로 하므로 T-DMB 표준에 지정되어 있는 비디오 객체의 규격에 맞는 복호기를 만들어서 최적화를 해야 한다. T-DMB에서 규정한 비디오 객체의 규격 중 중요한 사항을 표 1에 정리하였다.

표 1 T-DMB에서의 비디오 객체 제한 사항

프로파일	베이스라인 프로파일을 사용한다.
	하나의 픽처는 하나의 슬라이스로만 구성된다.
	잉여 픽처는 사용하지 않는다.
	POC(picture order count) 타입은 2를 사용한다.
	참조 프레임은 3장까지 사용한다.
레벨	최대 지원 해상도는 QCIF(352*288)이다.
	수직 MV 범위는 [-64, +63.75]로 한다.
	MaxDPB는 445.5kbytes로 한다.

4.2 각 루틴 별 최적화

4.2.1 움직임 보상을 위한 화소 보간

H.264|AVC에서는 움직임 벡터의 해상도가 휘도 성분에 대해서는 1/4 화소까지로 색차 성분의 경우에는 1/8 화소까지로 되어 있어 움직임 보상을 위해서는 화소의 보간이 필요하다.

화소 보간에 사용되는 필터 계수는 (1/32, -5/32, 5/8, 5/8, -5/32, 1/32)인데 중간의 네 개의 계수인 -5/32, 5/8, 5/8, -5/32를 5로 묶고 전체 계수를 32로 묶어서 처리하였다. 이와 같이 처리하게 되면 곱연산은 1회로 줄어들게 되고 나머지 곱연산은 시프트 연산으로 대체되게 되어 성능 향상을 이룰 수 있다. a라는 화소 값을 보간하기 위해서 A~F 6개의 화소를 이용하는 경우의 예를 보도록 하겠다.

a=A/32-5*B/32+5*C/8+5*D/8-5*E/32+F/32와 같은 형

태의 수식은 a=(A+(-B+(C+D)>>2-E)+F)>>5와 같은 형태의 수식으로 변환될 수 있다.

이 방법 외에도 화소의 위치가 1/4, 2/4, 3/4에 따라 각기 다른 필터 계수를 사용하는 방법[6]이 있으나 이렇게 하면 화소 보간의 횟수는 줄일 수 있으나 앞에서 말한 방법에 따라 곱셈 연산을 줄일 수 없게 된다.

4.2.2 엔트로피 복호

부호 값 자체를 해당하는 테이블에서 검색하여 찾는 방식에서 부호의 비트 패턴에 따라 테이블에서 해당하는 값이 자동으로 선택되는 방향으로 수정하면 성능의 향상을 얻을 수 있다.

4.2.3 역변환(inverse transform)

H.264|AVC에서 사용되는 변환은 기존의 영상 압축부·복호기와는 달리 4x4 블록 단위로 변환이 이루어지며 4x4 DCT(Discrete Cosine Transform)에 약간의 변형을 가한 정수 변환(integer transform) 형태를 띠고 있다.

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

그림 4 H.264 복호기에서의 역변환

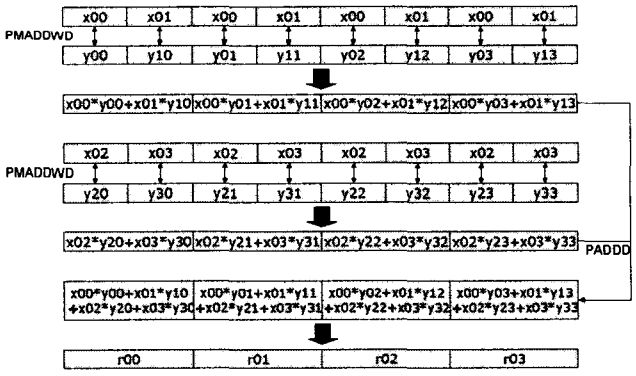

그림 5 SIMD 명령을 이용한 4x4 행렬곱

그림 4에 H.264 복호기에서의 역변환[7]에 대해 나타내었고 그림 5에는 SIMD(Single Instruction Multiple Data) 명령을 이용한 4x4 행렬곱의 구현[7]에 대해 나타내었다. 이를 이용하면 H.264에서 사용하는 역변환을 효율적으로 구현할 수 있다.

4.2.4 화면 내 예측

getNonAffNeighbour() 함수를 이용해서 주위 매크로블록의 위치를 계산하여 화면 내 예측에 이용하게 되는데, 입력된 x 좌표 값과 y 좌표 값을 가지고 분기 명령으로 위치를 얻게 되어 있는 부분을 계산식을 세워서 계산하여 매크로블록의 위치를 바로 얻을 수 있도록 바꾸게 되면 성능의 향상을 가져올 수 있다.

4.2.5 기타

비트스트림을 읽어오는 함수가 비트 단위로 처리하게 되어 있는데 이것을 바이트 단위로 처리하도록 바꾸게 되면 큰 성능의 향상을 이룰 수 있다. 또한 픽처를 출력하는 부분이 바이트 단위로 되어 있는데 픽처 크기 단위로 출력하게 되면 성능이 크게 향상된다. 이 밖에도 참조 소프트웨어에는 사소한 문제점들이 있을 수 있다고 생각된다.

4.3 일반 사항

4.3.1 메모리 연산 수 줄이기

H.264/AVC 복호기 뿐 아니라 모든 영상 압축 복호기에는 필연적으로 많은 수의 메모리 연산이 수행되게 된다. 메모리 연산 자체의 수행 속도가 느린데다가 연산 수마저 많기 때문에 복호기의 성능에 매우 중요한 영향을 주게 된다. 이 문제를 줄이기 위해서는 메모리 연산에 수행되는 데이터의 단위를 가능한 한 크게 하여 연산 수를 줄이고 불필요한 메모리 복사 및 이동을 없애야 한다.

4.3.2 캐시 미스(cache miss) 줄이기

그림 6은 L1, L2 캐시(cache), 주 메모리(main memory)의 상대적인 크기 및 속도[8]를 개념적으로 나타낸 것이다. L1 캐시는 크기는 작지만 매우 빠른 속도를 가짐을 알 수 있고, 주 메모리는 크기는 매우 크지만 속도는 아주 느린 것을 알 수 있다.

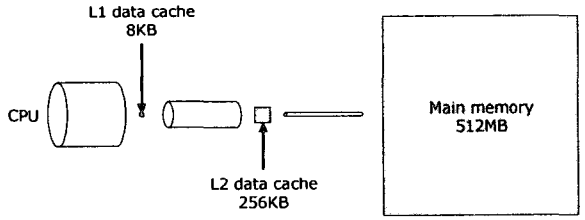

그림 6 L1, L2 캐시, 주 메모리의 상대 크기 및 속도

즉, 속도 개선을 위해서는 데이터가 캐시에 남아 있도록 하는 것이 좋다. 이를 위해 버퍼의 재사용, 자료 구조의 크기를 작게 유지하는 등의 일이 필요하게 된다.

4.3.3 분기문 예측 실패 줄이기

그림 7에 CPU의 개략적인 구조도[9]를 나타내었다. 그림에서 보는 것처럼 CPU는 분기문을 예측해서 다음 명령을 가져오게 되는데, 예측이 틀리는 빈도가 높게 되면 성능이 저하될 수 있다. 분기를 없앨 수 있는 경우 이를 없애는 방향으로 수정하면 성능 향상이 가능하다. 그 중에서도 최소값과 최대값을 구하는 매크로를 비교 루틴을 사용하지 않는 코드로 수정하게 되면 수행 시간 단축이 가능하다.

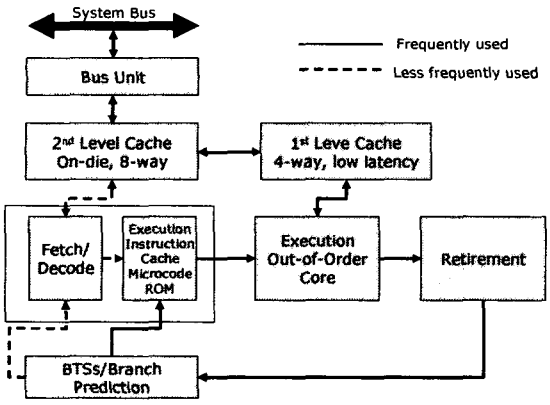

그림 7 CPU 구조도

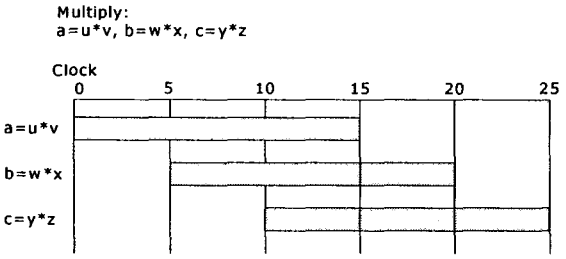

그림 8 데이터 의존도가 없는 경우의 간트 차트

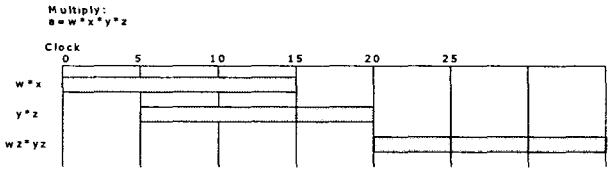

그림 9 데이터 의존도가 있는 경우의 간트 차트

표 2 실험 결과

함수명	최적화 전(clocktick s samples)	최적화 후(clocktick s samples)
getNonAffNeighbour()	804	478
get_block()	3271	1253
itrans()	1677	511
start_macroblock()	438	121
decode_one_macroblock()	3942	1766
Showbits()	1160	159
write_picture()	413	-
code_from_bitstream()	2583	292
readSyntaxElement_Run()	768	30
readSyntaxElement_TotalZeros()	638	27
readSyntaxElement_NumCoeffTrailingOnes()	476	30

4.3.4 데이터 의존도 줄이기

그림 8과 9에 각각 데이터 의존도가 없는 경우와 있는 경우의 간트 차트[10]를 나타내었다. 데이터 의존도가 있는 경우 다음 명령의 수행에 있어서 지연이 발생하게 되어 성능에 악영향을 주는 것을 알 수 있다. 소스 코드 상에서 수행 순서를 조작하여 데이터 의존도를 줄일 수 있으며 루프에서는 루프를 언롤링(Unrolling)하여 데이터 의존도로 인한 성능 저하를 줄일 수 있다.

4.4 결과

표 2에 실험 결과를 나타내었다.

5. 결론

H.264|AVC 복호기는 복잡도가 높아서 최적화를 하지 않으면 고사양의 PC에서도 실시간 복호가 힘들다. 본 논문에서는 H.264|AVC 참조 소프트웨어를 기반으로 하여 각 루틴별 최적화 방안과 일반적으로 적용될 수 있는 방법을 통한 최적화 방안을 함께 살펴보았다.

현재 최적화된 H.264|AVC 복호기 모듈을 사용해서 T-DMB 수신기를 실험해 본 결과 만족할 만한 성능을 얻을 수 있었다. 더 높은 수준의 최적화를 원한다면 H.264|AVC 복호기 모듈의 재설계를 통한 메모리 이동 및 복사의 구조적인 최적화를 이뤄야 할 것으로 보인다.

참고문헌

[1] 정보통신단체표준 TTAS.KO-07.0026, 초단파 디지털라디오방상(지상파 DMB) 비디오 송수신 정합표준

[2] ISO/IEC 14496-10, Information Technology - Coding of audio-visual objects - Part 10: Advanced Video Coding, 2003.

[3] ISO/IEC 14496-2, Information Technology - Coding of audio-visual objects - Part 2: Visual, 2001.

[4] http://www.intel.com/software/products/vtune/vpa/index.htm

[5] http://bs.hhi.de/~suehring/tml

[6] Vile Lappalainen, Antti Hallapuro, and Timo D. Hämäläinen, "Complexity of Optimized H.26L Video Decoder Implementation", 2003.

[7] Xiaosong Zhou, Eric Q. Li, and Yen-Kuang Chen, "Implementation of H.264 Decoder on General-Purpose Processors with Media Instructions", 2003.

[8] Kris Kaspersky, Code Optimization: Effective Memory Usage, 2003.

[9] Intel Corp., Intel IA-32 Intel Architecture Software Developer's Manual Volume 1: Basic Architecture.

[10] Richard Gerber, The Software Optimization Cookbook, 2002.