임베디드 DSP 기반 시스템을 위한 H.264 소프트웨어 부호기의 실시간 최적화

정회원 노 시 봉*, 안 희 준**°, 이 명 진***, 오 혁 준****

Real-time Optimization of H.264 Software Encoder on Embedded DSP System

Sibong Roh*, Heejune Ahn**°, Myeong-jin Lee***, Hyukjun Oh**** *Regular Members*

요 약

H.264 영상압축표준은 우수한 부호화 성능 때문에 현재 DMB와 IPTV 등에 다양한 응용에 활용되고 있으나, 높은 계산량으로 인하여 임베디드 환경에서의 실시간 부호화기로의 활용은 매우 제한적이다. 본 논문은 DSP 시스템이 제공하는 컴파일러 옵션 최적화, 인트린식과 어셈블코드 적용, 데이터 메모리 배치 최적화 과정을 H.264 부호화기 최적화의 입장에서, 비판적, 종합적으로 분석하고 반영한 결과를 소개한다. 특히, 대표적인 DSP인 TMS320DM64x를 사용하여 적용된 최적화 방식에 따른 연산이득을 구체적으로 제시하였으며, 그 결과 CIF급의 영상은 현재시장에 유통되는 DSP기반으로 실시간 구현이 가능함을 확인하였다.

Key Words : H.264 Encoder, Embedded DSP, Software Optimization

ABSTRACT

While H.264/AVC is in wide use for multimedia applications such as DMB and IPTV service, we have limited usage cases for embedded real-time applications due to its high computational demand. The paper provides judicious guide line for optimization method selection, by presenting the detailed experiments data through the development process of a real time H.264 software encoder on embedded DSP. The experimental analysis includes an intensive profiling analysis, fast algorithm application, optimal memory assignment, and intrinsic-based instruction selection. We have realized a real-time software that encodes CIF resolution videos 15 fps on TMS320DM64x processors.

I. 서 론

ITU VCEG그룹과 ISO MPEG 그룹이 2003년에 최종 표준화한 H.264/AVC[1]는 부호화 성능 측면에서 H.263(동일 PSNR 기준)과 MPEG-4(MOS 기준) 대비 2배의 비트율 감소를 얻었지만, 이들 표준에 비하여 많게는 2~8배 정도의 계산량을 요구한다[20]-[21]. 이러한 이유로 현재 H.264 표준은 주로 실시간 인코딩을 요구하지 않거나 고가의 부호화 시스템을 활용하는 IPTV, DMB, 저장 매체 응용들에만 주로 사용되고 있다. 그러나 점차 감시용 IP 네트워크 카메라나 슬링박스와 같은 트랜스코딩 등의

※ 본 연구는 지식경제부 전략과제 (다종 미디어 융합기반 Any Device Platform 기술 개발, No-10031824)과 광운대학교 2008년도 교내연구비의 지원을 받아 수행되었음.

* (주) 디지털 아리아 기술원구소 (sibong@digitalaria.com)
** 서울산업대학교 제어계측공학과 임베디드 통신연구실 (heejune@snut.ac.kr) (°:교신저자)
*** 한국항공대학교 항공전자및정보통신공학부 (artistic@kau.ac.kr), **** 광운대학교 전자통신공학과(hj_oh@kw.ac.kr)
논문번호 : KICS2009-05-209, 접수일자 : 2009년 5월 20일, 최종논문접수일자 : 2009년 9월 7일

실시간 임베디드 환경에서도 H.264 방식 사용요구가 증가하고 있다.

상용적인 측면에서 살펴보면, 부호화만을 적용하는 경우라면 가격과 전력소모 등을 고려하였을 때 SoC 기반의 방식이 적합한 것으로 보이며 그러한 형태의 어플리케이션 프로세서가 시장에 소개, 사용되고 있다. 그러나, 트랜스코딩이나 IP 네트워크 카메라와 같이 영상 압축 이외에 다양한 신호처리 응용을 적용할 경우에 가변성이 적은 SoC 하드웨어를 사용하기보다 DSP 기반의 소프트웨어 코덱을 사용하는 것이 적절하다[4].

따라서 주로 소프트웨어 입장에서 H.264의 계산속도 향상을 위한 연구들이 지속적으로 발표되고 있다. 이러한 가속화 연구들은 하드웨어에 무관한 알고리듬 레벨 연구와 적용 하드웨어의 특성을 고려한 구현 레벨 연구들로 분류할 수 있다. 알고리즘 레벨 연구는 아키텍처와 상관없이 적용할 수 있다는 장점 등으로 학계에서 많은 연구결과가 발표되고 있으나, 아키텍처를 고려한 연구결과는 실질적인 중요성에 비하여 연구결과로의 발표가 상대적으로 적다. 아키텍처 레벨 최적화에 대한 기존의 연구들은 다음과 같은 내용들이 있다. Peng 등은[13] x264의 변형인 T264를 사용하여 TMS320DM642상에서 최적화를 하였다. 움직임 추정방법으로 다이아몬드 검색 알고리즘을 사용하고, 이를 탐색 초기에 종료하는 조건을 추가하여 움직임 추정의 정확성을 떨어뜨려 PSNR값을 감소시키는 대신에 ME과정의 수행속도를 개선하였다. 그리고 SAD, 1/2-pel 보간, 1/4-pel보간, DCT/IDCT, MV예측 과정에서 인트린식(Intrinsic)과 #pragma를 사용하여 코드를 최적화하였다. 최종적으로 Forman QCIF 영상의 경우 약 26fps 을 부호화 할 수 있었다.

Wang등[14]은 x264를 사용하여 TMS320DM642 상에서 데이터 전송에서 EDMA를 사용하고, 1/2-pel 보간에서 6-tap 필터를 사용하지 않고 2-tap과 4-tap 필터만을 사용하여 메모리 접근 횟수와 연산량을 감소시키기 위해 다소 부정확한 보간 영상을 생성하였다. 그리고 SAD, DCT, 양자화, 화면내 예측 과정을 리니어 어셈블리로 작성하였다. QP값을 24로 하여 Forman QCIF를 약 45fps로 부호화 할 수 있었다. 이전 연구 결과를 바탕으로 TMS320DM642 상에서 실시간으로 동작하는 소프트웨어 H.264인코더를 구현하였다. 그 결과로 이전 연구들 보다 높은 성능향상을 얻었으며 구체적인 성능 향상의 원인과 결과를 수치적으로 분석하였다.

국내에는 아키텍처 관련 최적화에 대한 공식적으로 논문 결과는 많지 않으며, Intel Pentium 환경에서 SIMD를 적용한 연구 결과가 보고된 바 있다[3]. 본 논문에서는 이런 점을 아쉽게 생각하여, 아키텍처를 고려하여 최적화한 결과를 분석 정리하였다.

본 연구에는 PC에서 최적화된 공개소스인 x264 부호기[17]를 사용하였으며, 따라서 알고리즘 레벨의 최적화는 별도로 언급하지 않았다. JM을 기반으로 실시간 압축을 수행하는 부호기를 구현하는 것은 많은 시간을 필요로 하기 때문에 대부분의 논문은 실용적인 면을 고려하여 x264에 기초하여 최적화를 수행하였다[13]-[15].

주요한 연구 결과로 임베디드 시스템에 H.264 부호화 기법을 적용하기 위하여 필요한 아키텍처를 고려한 소프트웨어 최적화 기법을 적용하고, 그 결과를 구체적으로 정리하였다. 각 최적화 방법의 효율성을 정량적으로 평가하기 위하여 대표적인 DSP칩인 TI사의 DM642 (TMS320DM642 Video/Imaging Fixed-Point Digital Signal Processor) 상에서 구현 결과들을 제시하였다. 그러나 본 연구에 제시된 기법들은 대부분의 신호처리용 DSP에 공통으로 적용될 수 있는 방법들로 한정하였으며, 이 방법들을 다른 업체의 DSP에서도 유사하게 적용할 수 있을 것으로 판단된다.

본 논문은 다음과 같이 구성하였다. 제2장에서는 JVT 참고구현과 본 연구의 기본코드인 x264의 PC와 DSP상에서 소프트웨어의 프로파일링(profiling) 결과를 통하여 기능별 수행 시간을 우선 비교분석하였다. 부호화 과정, 함수 호출 그래프, 중요 알고리즘의 수행시간 및 호출 횟수, 동적 메모리 사용량 등을 분석하여 CPU와 메모리에 사용량을 분석하고 제시하였다.

제3장에는 각각의 신호처리 소프트웨어 최적화 기법을 적용하기 위한 구체적인 방법과 그에 따른 실험결과를 분석 제시하였다. 단계적으로 TI 컴파일러의 옵션조정, 내부 SRAM메모리 할당 방법에 따른 변화, DSP 병렬구조에 따른 알고리즘, 특히 H.264에 추가된 기능들에 대한 알고리즘 (SAD, SATD, DCT, Quantization, 1/2-pel interpolation, 1/4-pel interpolation)을 수행하는 함수에 Intrinsic 및 #pragma를 사용하는 방법과 이를 적용한 결과 등을 제시하였다.

제4장에는 각 단계별 최적화 방식의 이득을 분석하고, 최종 성능에 대하여 분석하였다. 본 연구 결과 QCIF 영상은 51.73~86.25fps, CIF 영상은 12~15fps

의 부호화 속도를 확인하였다. 이는 최적화 전과 비교하여 약 20배 정도 속도가 향상되어 실시간 DMB 전송이나 개인용 슬링박스 (slingbox), 일반적인 성능의 감시 카메라 등에 바로 적용할 수 있다.

Ⅱ. H.264 소프트웨어 인코더 계산량 분석

JM(Joint Model) 참조 소프트웨어[16]는 H.264 표준을 정확하게 따르도록 구현되었으나, 가능한 모든 경우를 일종의 brute-force 방식으로 수행하므로 수행속도가 매우 낮다. 본 논문에서는 PC상에서 최적화가 진행되고 있는 공개 소스 프로젝트인 x264를 사용한다. x264는 ffmpeg, ffdshow, VLC media player등에 다양한 응용에 활용되고 있으며, 화질과 연산속도에서 MainConcept과 같은 최고 수준의 상용 코덱과 동일하거나 우수한 성능을 보인다[19].

그림 1은 H.264 기능을 구현한 X264 프로그램의 함수레벨 구성도이다. 최적화의 기본 규칙은 암달의 법칙(Amdal's Law)에 따라 가장 시간을 많이 차지하는 요소를 찾아내는 것이 중요하다.

그러나 x264 코덱은 x86상에서 MMX(Matrix Math Extensions), SSE(Streaming SIMD Extensions) 등을 적용한 어셈블리 코드로 PC환경만을 고려하여 최적화 되어 있다. 그림 2에는 PC와 DSP상에서 x264의 프로파일링을 수행한 결과이다. PC의 경우 어셈블러 코드는 제외하고 C 코드만을 사용하여 비교하였다. 특별한 언급이 없는 경우 실험영상은 Foreman 을 사용하였고, X264 구동 옵션은 "--no -cabac --no-ssim -bframes 0 --framerate 15 -qp 28"을 사용하였다. 두 환경에서 모두 화면 내와 화

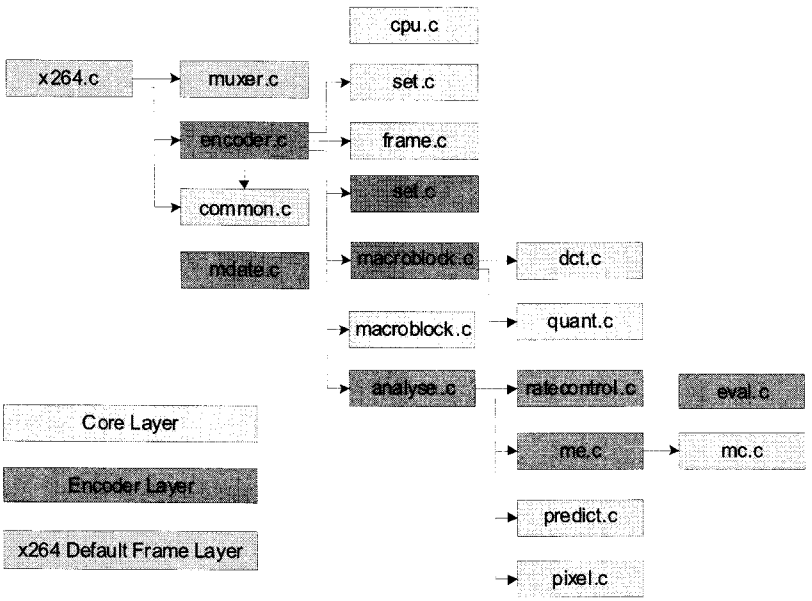


그림 1. X264 프로그램의 함수레벨 구성도

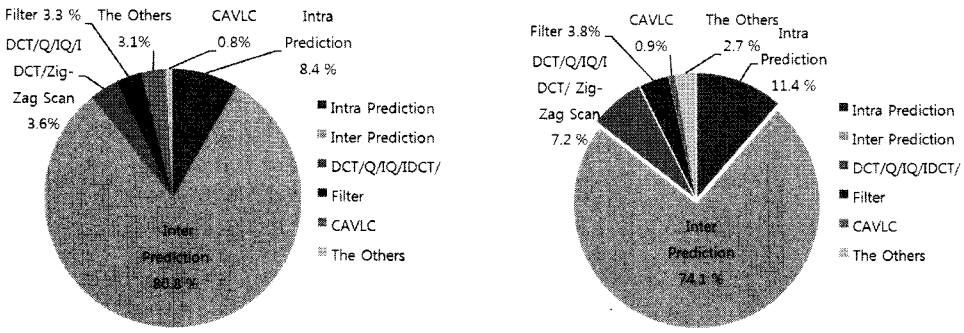


그림 2. PC와 DSP상에서 x264의 프로파일링을 수행한 결과

면 간 예측이 대부분에 수행시간을 차지하고 있다. 다른 점은 PC와 비교하여 DSP에서는 화면 내 예측은 2.3% 증가하였고, 화면 간 예측은 12.5%가 감소 되었으며, DCT/Q/IQ/IDCT/Zigzag Scan등은 2.3배가 증가하고 나머지 항목은 10배 가까이 증가되었다. 즉, 아키텍처 상에 차이가 분명히 있지만 우선적으로 예측부호화 부분이 중요하다는 것을 확인하였다. 각 x264 함수의 상세 결과는 [2]에서 확인할 수 있다.

Ⅲ. DSP 상의 최적화 기법

우선, 본 논문에서 사용한 TMS320DM642의 주요 특징은 다음과 같다. 연산모듈은 병렬처리 구조인 VelociTI.2[11]로 VLIW(Very Long Instruction Word)[12]방식에 기반을 두고 있으며 최대 8개의 명령어를 병렬로 처리할 수 있고, SIMD 형태의 PACKED명령어 지원이 가능하다. 내부 메모리는 2계층 캐시 메모리로 구성되어 있다. L1캐시는 각 16KB의 L1D(데이터)와 L1P(프로그램)으로 하드웨어적으로 제어된다. L2캐시는 사용자가 256KB안에서 선택적으로 분할하여 사용할 수 있다. L2캐시와 주변 장치의 데이터 교환은 EDMA(Enhanced DMA) 컨트롤러를 통해서 이루어진다. 외부메모리와의 데이터 교환은 EMIF(External Memory Interface)를 통해서 실행된다. 보다 자세한 내용은 [2] 또는 TI 매뉴얼[7]-[10]에서 찾을 수 있다.

본 논문에서 사용한 최적화 방법은 크게 세 가지로 나눌 수 있다. 첫 번째 방법은 컴파일러 옵션을 조정하여 효율적인 코드가 생성 되도록 설정한다. 두 번째 방법은 내부와 외부메모리를 최적화하여 수행속도를 개선한다. 세 번째 방법은 컴파일러에서 제공하는 인트린식(instrinsic) API와 #pragma 옵션을 사용하여 코드를 수정하여 최적화를 수행한다. (참고로, 인트린식과 #pragma는 모두 컴파일러를 세부 조정하여 프로그래머가 원하는 기계어를 생성하도록 하는 소프트웨어 방식이다.) 프로파일링 결과와 같이 부호화기의 가장 큰 병목은 예측기능이다. 예측을 위해서는 메모리 접근을 최적화하는 것과 SIMD 기법을 사용하여 병렬처리를 지원해야 한다.

3.1 컴파일러 옵션조정

옵션을 조정하지 않고 Intrinsic같은 API를 사용하여 성능을 향상시킬 수 있지만, Intrinsic 적용 후에 최적화 옵션들을 모두 적용한 경우에 컴파일러가 원래 C 코드를 최적화한 결과보다 더 낮은 성능이 발생할 수도 있다. 그래서 최적화의 첫 번째 단계로는 컴파일러의 옵션을 조정해야 한다. Code Composer Studio(CCS)의 C/C++컴파일러에는 다양한 옵션이 존재한다. 이를 코드특성에 맞게 적용하면 코드크기 및 수행속도를 최적화할 수 있다. x264에 적용한 옵션 및 중요한 특징은 표 1에 정리하였다.

표 1. 최적화 적용 옵션

	설 명
-o3	- o2 옵션부터는 Loop unrolling, Software pipelining등에 최적화가 수행 - o3 옵션을 사용하는 경우만 -pm옵션 적용가능
-mt	코드상에 잘못된 앨리어스(Bad Aliases)가 없다는 정보를 컴파일러에 전달
-mh	루프를 수행하기 전에 미리 데이터를 읽어 커널의 수행 속도를 향상
-pm	전체 소스 파일을 하나의 중간 파일에 모아서 최적화를 수행한다.

특히, 주의하여야 할 것은 기존의 코드에 앨리어싱 기법을 사용한 경우가 존재할 수 있다는 점이다. x264의 경우에 일부 코드가 앨리어싱을 사용하고 있으므로 다음과 같이 수정이 필요하였다. 표 2에는 각각의 옵션마다 성능 개선 정도를 정리하였다. 괄호 안의 값은 옵션을 적용하지 않은 첫 번째 줄과 비교하여 향상된 정도를 백분율로 표시하였다. -mt와 -mh 옵션을 사용한 경우 코드 크기가 변동될 수 있다. 이로 인해 L1P 스톨 사이클이 증가되는 경우도 있다. 반면에 이 옵션들은 L1D 스톨 사이클을 감소시켜준다. 마지막에 사용한 -pm옵션은 프로그램에서 사용하지 않는 코드 및 데이터를 제거하기 때문에 L1D와 L1P 스톨 사이클이 감소하였다. 이 옵션들을 개별적으로 사용하는 것보다 모두 같이 적용해야 가장 좋은 성능이 나온다. 컴파일러 옵션 조정을 통해서 68%의 성능 향상을 확인하였다.

표 2. 컴파일 옵션에 따른 성능결과 (cycles (%), QCIF, Forman, 20 frames)

	Total	L1P Stall	L1D Stall	speedup
default	3.9G	1.1G	372M	
-o3	1.3G	637M	321M	2.92x
-o3 -mt	1.4G	689M	343M	2.8x
-o3 -mt -mh	1.3G	672M	313M	2.9x
-o3 -mt -mh -pm	1.2G	565M	315M	3.15x

```
// for zigzag_scan_8x8_frame
#define ZIG(i,y,x)
level[i] = dct[x*8+y];
// for zigzag_scan_4x4_frame
#define ZIG(i,y,x,)
level[i] =dct[0][(x<<2)+4];
```

변경전

```
// for zigzag_scan_8x8_frame
#define ZIG(i,y,x)
level[i] = dct[x][y];
// for zigzag_scan_4x4_frame
#define ZIG(i,y,x)
level[i] = dct[x][y];
```

변경후

그림 3. -mt 옵션적용을 위한 수정 (dct.c)

3.2 메모리 최적화

그림 4에서 보이는 DM642의 내부메모리 구성은 레지스터, L1 캐시, L2 캐시, 외장메모리 순으로 각각의 접근 지연은 0, 1, 6~8, 수십 사이클이 소요된다[5]-[6]. 표 3에 정리된 결과로 L2 캐시의 크기를 두 배로 늘리는 경우에 전체 속도향상은 평균적으로 1.31배 증가되었다. 따라서 내부메모리 최적화 목표는 L2캐시의 크기를 128KB로 설정하고, 나머지 128KB를 효율적으로 사용하여 L2 캐시의 크기를 256KB로 사용할 때 보다 더 좋은 성능이 나오도록 정하였다.

또한, 스택을 내부메모리로 사용하면 지역변수들이 내부메모리에 할당되므로 외부메모리에서 값을 매번 불러오지 않는 변수들이 생기므로 성능이 더 좋아진다. 스택을 내부메모리에 할당하는 경우에는 스택의 크기를 적절히 설정해야 한다. 스택의 크기

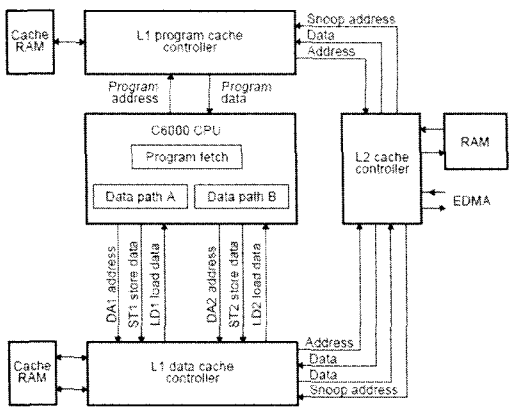


그림 4. DM642의 내부메모리 구성[6]

표 3. L2 캐시 따른 성능결과 (cycles (%), QCIF, Forman 20 frames)

	Total	L1P Stall	L1D Stall	speedup
4-way/0KB	1,373M	672M	313M	
4-way/32KB	1,013M	332M	290M	1.35x
4-way/64KB	823M	260M	172M	1.23x
4-way/128KB	623M	147M	84M	1.32x
4-way/256KB	474M	44M	38M	1.31x

표 4. L2 캐시 따른 성능결과 (cycles (%), QCIF, Forman 20 frames)

	Total	L1P Stall	L1D Stall
external mem	623M	147M	84M
internal mem	611M	141M	77M

를 결정하기 위해서 함수 호출이 가장 깊은 경우와 지역 변수의 크기를 조사하였다. 이를 고려할 경우에 최소 6KB정도로 스택의 크기를 선언해야 한다. 그러나 안정적인 동작을 위해서 스택을 8KB로 설정하였다.

L2 캐시 크기를 128KB로 설정하고 스택의 위치를 내부메모리에 할당하고 실험한 결과 스택을 내부메모리로 옮기는 경우에 L1P스톨과 L1D스톨 사이클이 각각 4%와 8% 감소하였다.

CCS C/C++컴파일러에서는 자주 사용되는 코드와 데이터를 내부메모리에 할당하는 코드로 CODE_SECTION/DATA_SECTION pragma를 사용하여 섹션을 구분하여 주고, 링크 과정에서 이를 물리적인 메모리에 할당해준다(그림 5).

```
// #pragma CODE_SECTION (symbol, "Section Name")
#pragma
CODE_SECTION(pixel_satd_wxh,".text:hotspot")
// #pragma DATA_SECTION (symbol, "section name")
#pragma
DATA_SECTION (intra4x4_cbp_to_golomb, ".far:hotspot_data");
// video.cmd
MEMORY {
CODE_MEM0: origin = 00000000h len = 00012000h
DATA_MEM0: origin = 00012000h len = 00001000h
}
SECTIONS {
.text:hotspot:{ }> CODE_MEM0
.far:hotspot_data:{ }>DATA_MEM0
}
```

그림 5. 메모리 섹션 설정

내부메모리에 할당할 함수들은 3장의 프로파일링 결과를 기반으로 자주 호출되면서 코드 크기가 작은 함수와 호출 빈도는 낮으나 L1P 스톨이 큰 함수를 기준으로 할당하였다. 또한, 내부메모리에 할당할 데이터는 상수 테이블들과 x264_t 구조체에 존재하는 캐시데이터들이다. 구조체를 내부메모리에 할당하기 위해서 MEM_alloc(segid, size, align) 함수를 이용하였다. 그 결과 상당수의 코드 영역과 예측기준 블록의 데이터가 캐시가 할당되었다. 표 5~6에서 전체 사이클은 메모리 최적화 목표보다 1.61% 높은 성능이 나왔다. L1P스톨 사이클과 L1D스톨 사이클이 각각 77.7%와 49.6%감소한 만큼 전체 수행 사이클이 향상되었다.

표 5. 내부메모리 최적화후의 성능결과 (cycles (%), QCIF, Forman 20 frames)

	Total	L1P Stall	L1D Stall
세그먼트미지정	611M	141M	77M
세그먼트지정	479M	31M	39M
최종목표치	474M	44M	38M

표 6. 인트린식 적용 지도

	Alg.	instrinsic
inraPred, 1/4 ME	SATD	_amem4(),_unpklu4(), _pack2(),_packh2(), _sub2(), _add2(), _sadd(), _ssub(), _abs()
화면간 (정수)	SAD	_amem8(), _amem4(), _dotpu4(), _subabs4()
DCT	DCT	_amem4(), _unpklu4(), _pack2(),_packlh2(), _sub2(),_sadd2(), _dotp2(), _dotpn2()
Quant	multi	_mpyu(),_mpysu(), _mpy()
1/4 interp. MC	linear interpol	_amem4(),_avgu4(), _mem4()

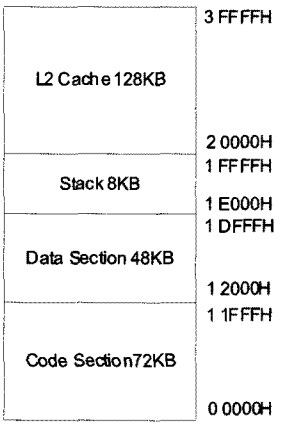


그림 6. 내부메모리 맵

메모리최적화를 수행한 후에 내부메모리 맵은 그림 6과 같다. 내부메모리128KB는 L2 캐시로 나머지 128KB영역은 스택 영역으로 8 KB를 할당하고, 코드 영역으로 72KB와 데이터 영역으로는 48KB를 할당하였다.

3.3 코드 최적화

메모리 최적화 후에도 가장 큰 수행시간을 차지하는 기능은 화면 내 예측과 화면 간 예측 함수들이다. H.264의 화면 내 예측과 화면 간 예측에는 SAD, SATD, SSD, AVG, 4x4 DCT, 양자화, 역양자화, 1/2과 1/4 픽셀 보간 함수가 필요하다. 코드 최적화에서는 intrinsic, _nassert, #pragma MUST_ITERATE, #pragma UNROLL을 사용하였다. 표 6은 부호화 과정에서 수행되는 알고리즘에 따라서 사용한 intrinsic 목록을 정리하였다. 영상처리용 DSP는 영상 신호가 일반적으로 8bit로 표현되는 특징을 이용하여 SIMD(Single Instruction Multiple Data)를 지원하는 packed 명령과 pack 형태를 변환하는 명령어 그리고 절대값 계산 및 합산을 제공하는 외적(dot product) 명령 등을 사용하였다.

각 기능 함수들에 관한 블록도와 코드 예제들은 [2]에 제시되어 있으며 본 논문에는 공간의 제약상 중요한 함수중심으로 정리하였다.

● SAD (Sum of Absolute Differences)

SAD연산을 H.264 VBS(4x4, 4x8, 8x4, 8x8, 8x16, 16x8, 16x16) 별로 필요하다. 대표적으로 SAD 4x4연산에 Intrinsic을 적용한 결과는 _amem4()를 사용하여 4바이트 단위로 데이터를 읽어 들인 후에 _subabs4()를 사용하여 4개 픽셀의 차이 값을 구하고 결과 값에 절대값을 동시에 계산한다. 마지막으로 _dotpu4()를 사용하여 바이트 단위로 각각의 결과에 1을 곱한 후 바이트 별로 덧셈을 수행하여 합을 구하여 i_sum변수에 값을 더한다. 실험결과 SAD 함수들은 평균적으로 약 1.36배 향상된 결과를 확인하였다.

● SATD (Sum of Absolute Transformed Differences)

_amem4()를 사용하여 4바이트 단위로 값을 읽고 나서 unpack Intrinsic을 사용하여 1바이트를 2바이

트로 확장하여 다른 Intrinsic을 적용하기 쉬운 형태로 데이터를 모아서 _sub2() Intrinsic을 사용하여 상위 16비트와 하위 16비트에 대해서 각각 감산을 실행한다. 점선 아래 부분부터는 실제 계산을 하는 과정으로 pack Intrinsic을 사용하여 다음 연산을 적용하기 쉬운 형태로 데이터를 모으고 가산과 감산을 수행하여 행렬식의 계산된 결과를 배열에 저장한다. 코드 최적화를 통해 SATD함수를 최적한 결과 평균적으로 약 2.36배 향상된 결과를 확인하였다.

● 정수변화 4x4 DCT 함수 최적화

정수 DCT 연산은 H.264에서 새롭게 추가된 기능으로 행렬 곱셈을 계수화하였으며 다른 연산과 마찬가지로 packed 연산을 수행하였다. sub4x4_dct 함수는 약 1.74배 향상된 결과를 확인하였다.

● SSD (Sum of Squared Differences)

SSD는 SAD와 유사한 절차를 거치나 곱셈과정에서 범위가 두 배로 늘어나므로 unpack 과정을 거치는 점이 차이가 있다. 그림 9와 같이 코드 최적화를 수행한 후에 평균적으로 약 1.69배 향상된 결과를 확인하였다.

● 양자화 및 역양자화 함수

양자화와 역양자화를 계산하는 경우에 _mpyu(), _mpysu(), _mpy() intrinsic을 적용하여 계산속도를 높일 수 있다. 그리고 연산과정에서 필요한 테이블을 내부메모리에 할당하였기 때문에 L1D 스톨과 미스가 줄어든다. quant4x4_()와 dequant_4x4() 최적화 결과 각각의 경우에 1.52배와 1.23배 의 성능 향상이 있었다.

3.4 그 외에 고려된 최적화 알고리듬

본 논문은 주로 아키텍처 중심의 최적화를 연구하였으나 몇 가지 알고리즘 레벨의 최적화도 추가로 적용하였다. x264에서는 기본적으로 움직임 추정 방법으로 UMH(uneven multi-hexagon search)를 사용하게 되어 있다. [18]에는 PC상에서 UMH가 Diamond 탐색보다 약 3배 느리고 PSNR값은 높다는 것을 알 수 있다. 본 논문에서는 Diamond 탐색을 사용하여 다소 PSNR값을 감소시키고, pixel_avg() 함수에 #pragma UNROLL과 _amem4(), _avgu4(), mem4()를 사용하여 함수의 수행 속도를 약 4.24배 향상된 결과를 얻었다.

또한, mc_croma()는 색차 성분에 대한 움직임 보상을 수행한다. 연산을 수행하기 위해 선언된 지역변수들은 int형으로 선언되어 있는데 이 변수들을 short형으로 변경하고, #pragma UNROLL을 사용하여 약 2.44배의 성능향상을 확인 하였다.

표 7. 코드 영역의 최적화 전후 성능 비교 (영상 복잡도순)

video (frames)	frame rate (fps)	PSNR (db)	bitrate (kbps)
Akio (300)	86.2	40.5	81.9
news (300)	69.7	38.8	128.4
froeman(400)	52.1	37.5	240.8
mobile (300)	51.7	34.9	670.3

표 7은 코드 최적화 결과를 종합적으로 정리하였다. L1P 스톨과 L1D 스톨은 각각 26.99%와 1.94% 감소하였고 전체 수행 시간은 54.69%가 향상되었다.

Ⅳ. 결과 분석

최적화 수행 후에 다시 프로파일링을 수행한 결과는 그림 7, 8과 같다. 화면 간 예측은 30.84% 화면 내 예측은 3.08%가 감소되었고, 나머지 연산들이 차지하는 비중은 증가되었다. 그 중에서 파일 입출력 등에 함수들이 속해 있는 The Others가 차지하는 비중이 18.47%증가 하였다.

입력 영상에 따른 차이를 확인하기 위해 4개의 표준 실험 영상을 각각을 300프레임씩 부호화하여 실험한 결과 QCIF 입력 영상에서 51.73fps~86.25fps의 성능이 나오는 것을 확인하였다. 이는 기존의 연구 결과에 비해서 30%이상 향상된 결과이다.

최종적으로 응용의 목표인 CIF 영상을 실제 보드상에서만 카메라 입력을 통해 CIF 영상을 획득하여 12~15fps의 성능으로 부호화되는 것을 확인하였다. 이 테스트 환경은 EDMA를 사용하여 입력 영

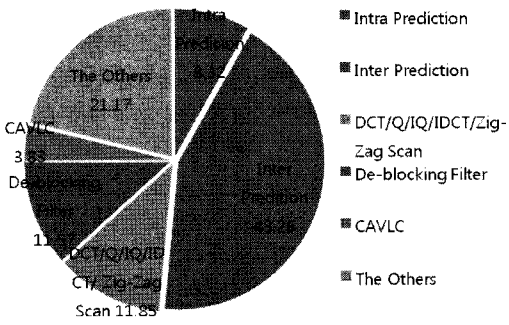


그림 7. 최적화 수행후 DSP 상에서 x264 프로파일링 결과

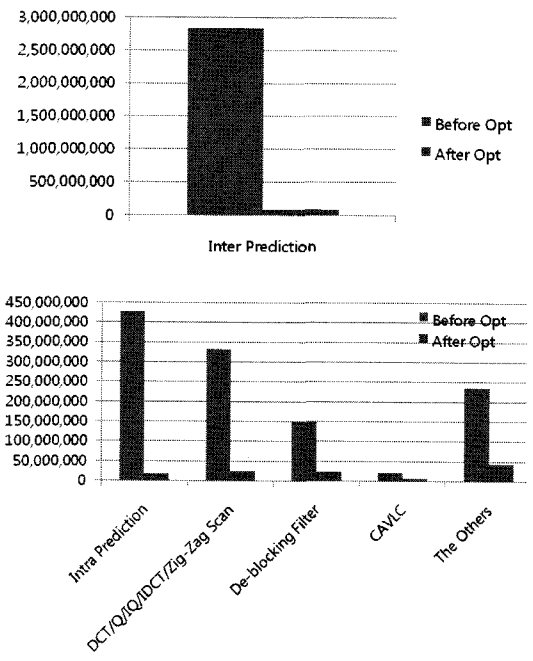


그림 8. 기능별 최적화 전후의 수행 사이클 비교

상을 카메라에서 SDRAM으로 복사한다. 압축된 H.264/AVC 비트스트림은 JTAG을 통하여 PC상에 저장하여 부호화가 제대로 수행되었는지 확인하였다.

V. 결 론

본 논문에서는 대표적인 영상처리용 DSP인 TI사의 TMS320DM642에 최적화된 실시간 H.264 소프트웨어 부호기를 구현하였다. PC상에서 최적화된 H.264 공개 소프트 인코더인 x264를 기능별로 성능을 상세히 분석하였으며, 최적화 과정을 수행하여 CIF 급 실시간 부호화에 적합한 성능을 얻을 수 있었다. 시뮬레이터상에서 QCIF 표준 입력 영상을 인코딩한 결과 51.73~86.25fps 속도로 인코딩 되는 것을 확인하였다. 이는 기존 연구 결과들에 비하여 30% 정도 향상된 결과이다. 또한, 실제 임베디드 형태의 DM642를 사용한 보드에서는 CIF를 기준으로 12~15fps 속도로 인코딩 되는 것을 확인하였다. 이는 현재 국내에서 사용하는 T/S-DMB와 슬링박스의 일반 화질 정도의 기능을 실시간으로 제공할 수 있는 성능이다.

본 연구에서 구체적으로 적용하지 않은 최적화 기법으로는 EDMA 적용과 리니어 어셈블리이다. 리니어 어셈블리를 적용하는 것은 Intrinsic과 #pragma를 적용한 결과 보다 못한 결과가 나올 수 있으므로 논문에서는 리니어어셈블리를 적용하지 않았다. 논문에서 개발된 부호화기는 네트워크 기능을 결합시키고 DM642보다 고성능의 DSP상에 적용하여 모바일 단말용 트랜스코더 및 H.264/AVC IP 네트워크 카메라 등에 적용을 할 수 있을 것으로 보인다.

참 고 문 헌

[1] Joint Video Team of ITU-T VCEG and ISO/IEC MPEG, Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC Version 1, 05. 2003.

[2] 노시봉, "TI DM64x상의 실시간 H.264 소프트웨어 부호기의 최적화 연구" 서울산업대 대학원 석사학위 논문, 2009. 2.

[3] 김용환, 김제우, 김태완, 최병호, "SIMD 명령어를 이용한 H.264 인코더의 최적화", 한국멀티미디어학회 추계학술대회, 10, 2003.

[4] 석진욱, 김범호, 이정우, 조창식, "HD급 H.264 기술의 발전 동향", 전자통신동향분석 제21권 제1호 2006년 2월.

[5] Robert Oshana, "DSP Software Development Techniques for Embedded and Real-Time Systems," Newnes, 2006.

[6] Texas Instruments, "TMS320C64x DSP Two-Level Internal Memory Reference Guide" (spru610), Aug., 2002.

[7] Texas Instruments, "TMS320C6000 CPU and Instruction Set Reference Guide"(SPRU189F), Oct 2000.

[8] Texas Instruments, "TMS320DM642 Video/Imaging Fixed-Point Digital Signal Processor" (SPRS2000L), Jan., 2007.

[9] Texas Instruments, "TMS320C6000 Instruction Set Simulator Technical Reference"(SPRU600F), Apr., 2005.

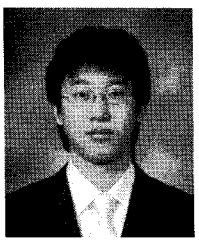
[10] Texas Instruments, "TMS320C6000 Optimizing Compiler User's Guide"(spru187l), May, 2004.

[11] J. Labrousse and G.A Slavenburg, "A 50 MHz Microprocessor with a VLIW Architecture," ISSCC, 1990.

[12] Nat Sehan, "High VelociTI Processing Teaxas Instruments VLIW DSP Architecture," IEEE

Signal Procesing Magazine, Mar, 1998.
〔13〕 Chaonan Peng, Hui Wang, Chuanzhen Li and Qin Zhang, "The Optimization of H.264 Encoder Based On TI TMS320DM642," FGCN2007, Dec., 2007.
〔14〕 Hong-Jun Wang, Yong-Jian Huang and Hua Li, "H.264/AVC Video Encoder Implementation Based on TI TMS320DM642," IIH-MSP'06, Dec., 2006.
〔15〕 Li Zhuo, Qiang Wang, David Dagan Feng, Lansun Shen, "Optimization and Implementation of H.264 Encoder on DSP Platform," ICMP 2007, Aug., 2007.
〔16〕 JM reference software Available: http://iphome.hhi.de/suehring/tml/index.htm
〔17〕 VideoLAN http://www.videolan.org/developers/x264.html
〔18〕 Loren Merritt, Rahul Vanam, "Improved Rate Control and Motion Estimation for H.264 Encoder," ICIP 2007, Oct., 2007.
〔19〕 Dmitriy Vatolin, Dmitriy Kulikov, Alexander Parshin, "MPEG-4 AVC/H.264 Video Codecs Comparison," MSU Fourth Annual MPEG-4 AVC/H.264 Video Codecs Comparison (December 2007), http://www.compression.ru
〔20〕 Nejat Kamaci, Yucel Altunbasak, "Performance comparison of the emerging H.264 video coding standard with the existing standards," ICME 2003, July, 2003.
〔21〕 정제창, "H.264/AVC 비디오 압축 표준", 홍릉과학 출판사, 2005.

노 시 봉 (Sibong Roh) 정회원



2007년 2월 서울산업대학교 제어계측공학과 학사
2007년 3월~2009년 2월 서울산업대학교 제어계측공학과 석사
2000년 3월~현재 (주) 디지털아리아 (현 주임연구원)
<관심분야> 영상압축, 임베디드 시스템

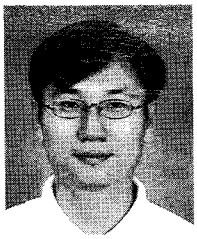
안 희 준 (Heejune Ahn) 정회원

2000년 2월 KAIST 전기및전자공학과 박사
2000년 2월~2002년 8월 독일 얼랑엔-뉴른버그대학 통신연구소 초빙연구원
2000년 2월~2002년 8월 LG전자 차세대단말연구소 선임연구원
2002년 9월~2004년 1월 (주) TmaxSoft 기술연구소 책임연구원/팀장
2004년 2월~현재 서울산업대 제어계측공학과(현 조교수)
<관심분야> 영상통신, 임베디드 SW, 인터넷 통신

이 명 진 (Myeong-jin Lee) 정회원

2001년 8월 KAIST 전기및전자공학과 박사
2001년 2월~2004년 2월 삼성전자 System LSI 책임
2004년 3월~2007년 2월 경성대학교 조교수
2007년 3월~현재 한국항공대학교 항공전자및정보통신공학부 (현 조교수)
<관심분야> 멀티미디어통신, 임베디드시스템

오 혁 준 (Hyukjun Oh) 정회원



1999년 8월 KAIST 전기 및 전자공학과 박사
1999년~2001년 미국 Stanford 대학 박사 후 과정
2001년~2004년 미국 Qualcomm 사, 3GPP CSM/MSM 개발
2004년~현재 광운대학교 전자통신공학과 (현 부교수)
<관심분야> 차세대 이동통신, 통신 모뎀 SoC 설계, 통신 신호 처리 이론 및 설계