ARM9 프로세서용 실시간 JPEG2000 코덱의 구현
A Real-Time JPEG2000 Codec Implementation on ARM9 Processor

김영태*, 조시원*, 이동욱*
Young-Tae Kim*, Shi-won Cho*, Dong-Wook Lee*

요 약

본 논문에서는 ARM9 프로세서를 위한 실시간 JPEG 2000 코덱을 구현하였다. 구현된 코덱은 프로세서, 메모리와 같은 시스템의 리소스를 효율적으로 사용할 수 있도록 제어 코드와 데이터 관리 코드를 분리하여 설계하였다. 특히 이동전화와 같은 임베디드 환경에서는 제한된 프로세서와 내부메모리를 이용하여 양질의 서비스를 제공하는 것이 매우 중요하다. ARM9계열의 프로세서는 부동소수점을 제공하지 않기 때문에 DWT와 같이 아주 반복적으로 부동소수점 연산을 필요로 하는 동작을 실행하기위해서는 많은 연산시간이 필요하다. 제안된 코덱은 이러한 단점을 극복하기 위해 고정소수점을 이용하여 프로그램을 하였다. 또한 캐시 메모리를 고려한 코드 최적화 방법을 적용하여 연산속도를 더욱 향상시켰다.

Abstract

In this paper, we propose an real-time implementation of JPEG2000 codec on the ARM9 processor. The implemented codec is designed to separate control codes from data management codes in order to use effectively the system resources such as processor and memory. Especially, in embedded situations like cellular phones it is very important to provide good services using limited processor and internal memory. Since ARM9 series processors do not provide floating-point, large amount of computational time is required to perform the operation which needs highly repetitive floating-point computations like DWT(discrete wavelet transform). The proposed codec was programed using fixed-point to overcome this weakness. Also code optimization considering cache memory was applied to further improve the computational speed.

Keywords : JPEG2000, Embedded System, ARM9, Memory Management

I. 서 론

JPEG는 Joint Photographic Experts Group(정지영상전문가그룹)의 약자로 현재 가장 많이 쓰이는 정지영상압축 규격 중 하나이다. 1992년 JPEG이 국제 표준으로 채택된 이후, 다양한 멀티미디어 기기에 이용되고 있다. JPEG 2000은 웨이브렛 변환(wavelet transform)과 EBCOT(embedded block coding with optimized truncation)를 이용하여 JPEG의 약점인 DCT(discrete cosine transform)변환을 적용하여 생기는 블록화 현상을 방지하고, 이미지의 품질을 유지하면서 높은 압축률을 얻을 수 있다.

JPEG 2000의 적용 분야로 관심을 받고 있는 분야는 디지털 카메라와 디지털 카메라 폰 등의 임베디드 기기 응용 분야이다. 현재 디지털 카메라와 카메라 폰의 경우는 새로운 문화 조류를 형성할 정도로 보급이 확대되고 있으며, 작은 저장 메모리를 가지고 있는 여러 종류의 임베디드 기기들이 JPEG 2000 코덱을 많이 요구할 것으로 예상된다.
하지만, JPEG2000 알고리즘은 기존 JPEG 알고리즘보다 더 복잡하고, 많은 계산량을 요구한다. 임베디드 환경에는 개발비용, 프로세스의 성능, 메모리, 전원 등과 같은 시스템 리소스가 제한되기 때문에 시스템의 리소스를 효율적으로 최적화하는 코드를 적용해야 한다.

ARM계열의 프로세서는 부동소수점(floating-point)을 제공하지 않기 때문에, DWT(discrete wavelet transform)와 같이 부동 소수점 연산이 많은 부분에서는 고정소수점(fixed-point) 코딩이 필수적이며, 캐시 메모리(cache memory)를 고려한 코드의 최적화가 필요하다.

본 논문에 소개된 JPEG2000 코덱은 임베디드 환경에서 메모리 자원이 제한적인 임베디드 프로세스에서 메모리 리소스의 효율을 더 높이고, 부동소수점 연산이 많은 부분은 고정소수점을 이용하여 코딩을 하였다.

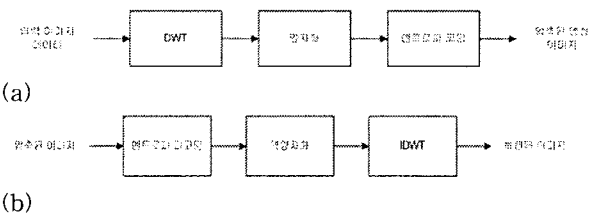
II. JPEG2000 코덱의 구조

JPEG 2000은 JPEG 표준을 만든 ISO 산하의 JPEG 그룹이 JPEG의 뒤를 이을 새로운 정지영상압축표준으로 만들었다. JPEG 기술의 기본은 DCT(discrete cosine transform)라는 일종의 푸리에 변환을 이용하여 이미지를 인코딩하는 것이다. 푸리에 변환은 임의의 신호를 여러 주파수의 삼각함수

*동국대학교 전기공학과
논문 번호 : 2007-3-1 접수 일자 : 2007. 5. 23
심사 완료 : 2007. 7. 19

의 합으로 표현할 수 있다. JPEG의 기본 기술은 사람의 눈으로 구분하기 어려운 주파수 성분을 제거하여 정보량을 낮추어 압축하는 방식이다. 압축을 푸는 디코딩 방법은 인코딩과 반대로 IDCT(inverse discrete cosine transform)를 거쳐 원하는 영상을 얻을 수 있다. 그러나, JPEG은 구현하기 쉬운 장점이 있지만, 손실과 무손실 압축이 단일화되어 있지 않고, 잡음이 많은 경우 저비트율(low bit rate) 환경에서는 성능 저하를 보인다.

JPEG 2000 표준은 여러 가지 새로운 기술의 장점만을 채택하도록 노력하였으며, 다양한 응용분야를 고려하여 표준화되었다. JPEG 2000의 특징으로는 무손실/손실 압축, 무손실 코딩에 내포된 손실, 화소의 정확도, 비트 에러와 관심영역(region of interest: ROI)부호화에 대한 견고함 등을 들 수 있다. JPEG 2000 표준 그룹은 웨이브렛 변환(wavelet transform)을 채택하였다. JPEG에서 이용하는 DCT의 단점은 삼각 함수가 주기를 가지고 무한히 반복하는 함수이기 때문에 주파수 영역에서 시간 정보를 표현하지 못한다는 점이다. 시간에 따라 주파수가 변동하는 신호를 푸리에 변환으로 나타내면 무한개에 가까운 삼각 함수를 더해야만 제대로 된 신호를 재생할 수가 있다. 실제로는 무한개의 삼각 함수 즉, 무한개의 주파수로 원래 신호를 표현할 수는 없기 때문에 주로 사용되는 범위의 주파수만을 이용하여 표현을 하게 된다. 이 경우 시간에 따라 변화가 심한 신호는 원래 모양과 다른 왜곡이 많이 발생하게 된다. JPEG에서는 이미지를 블록 단위로 압축을 하였기 때문에 사진을 확대하면 작은 블록 단위로 영상이 손상되는 것을 볼 수 있다. 그러나, JPEG 2000에서는 이미지를 웨이브렛 변환을 이용하여 압축하였기 때문에, 확대/축소를 하여도 자연스럽게 표현될 수 있다. 웨이브렛 변환의 경우는 축소할 때 고주파 성분을 따로 저장하여, 확대할 때 다시 그 정보를 이용해서 원래 이미지를 복원한다. 원본 이미지를 작은 이미지로 축소하면서, 다시 확대할 때 사용할 고주파 성분을 관심영역(region of interest), EBCOT(embedded block coding with optimized truncation) 등을 이용하여 압축한 것이 JPEG 2000이다. 고주파 성분은 같은 값들이 반복해서 나타나는 경우가 많기 때문에, 기존 JPEG보다 더 효율적으로 압축을 할 수 있다.



(a)

(b)

그림 1. JPEG 2000 코덱의 구조
(a)인코더 (b)디코더
Fig. 1. Structure of JPEG 2000 codec
(a)Encoder (b)Decoder

JPEG 2000 코덱은 그림 1과 같이 구성된다. 입력 이미지 데이터는 DWT(discrete wavelet transform)를 통하여 계수로 변환된다. 변환된 계수는 다시 양자화를 거치고, 엔트로피 부호화 과정을 거쳐서 압축된다[1][2]. 압축된 이미지 데이터는 위의 과정을 역으로 수행함으로써 원래의 이미지로 복원된다.

III. 소프트웨어 구성

3.1 JPEG2000 코덱의 구성

JPEG 2000 코덱은 사용 환경에 따라서 하드웨어 혹은 소프트웨어로 구현하게 된다. 하드웨어로 구현된 전용 코덱을 사용할 경우, 빠른 처리 속도를 보장받을 수 있지만, 설계 단계에서 기구 설계, 동작 클럭, 소비 전력, 제조 단가 등을 고려해야 하고, 많은 개발 비용과 시간의 투자가 필요하다. 반면 소프트웨어로 구현된 코덱의 처리 속도는 운영체제와 프로세스의 성능에 크게 의존하며, 하드웨어 코덱보다 처리 속도는 느리지만, 개발에 필요한 비용과 기간을 크게 줄일 수 있는 장점이 있다. 프로세스의 성능이 계속 발전하면서 별도의 하드웨어 코덱을 사용하지 않고, 소프트웨어 코덱만으로 필요한 기능을 제공할 수 있게 되었다. 물론 임베디드 환경에서 소프트웨어 코덱이 필요한 성능을 만족시키기 위해서는 적용되는 환경에 적합하도록 최적화된 코드로 코덱이 구현이 되어야 한다.

JPEG 2000 코덱 라이브러리는 이미지 데이터를 처리하는 소프트웨어 라이브러리(software library)형식으로 구성되어 있다. 소프트웨어의 구성은 크게 JPEG 2000 코덱 엔진(codec engine), 코덱 드라이버(codec driver), 데이터 관리 서비스(data management servce)의 세 부분으로 구성되어 있으며, 상용 제품에 적용이 가능한 수준으로 프로그램을 개발하는 것을 목표로 하였다. 그림 2는 구현된 코덱 라이브러리의 소프트웨어의 구성을 보여준다.

JPEG 2000 코덱 엔진은 입력 이미지를 JPEG 2000 형식으로 변환하는 기능을 수행하며, 코덱 드라이버를 통해 JPEG[3], PNM[4], BMP[5], NV21[6] 형식의 이미지를 JPEG 2000 형식으로 변환할 수 있도록 구현되어 있다. NV21형식은 YUV 형식[6]의 컬러 스페이스 모델 중 하나이며, 디지털 카메라의 기본 이미지의 기본 포맷으로 많이 사용한다. JPEG 2000 코덱 엔진에서 입력되는 데이터는 NV21[6]와 같은 YUV 형식으로 표현된 데이터 포맷을 사용한다. 인코딩을 할 경우, YUV형식의 데이터를 입력하면, JPEG 2000 형식으로 압축된 데이터를 얻을 수 있다. 반대로 디코딩을 할 경우, JPEG 2000 형식의 데이터를 입력하면, YUV 형식으로 표현된 데이터를 얻는다. YUV 형식을 사용하면 디지털 카메라 센서 장치간은 하드웨어와 직접 연결하여 데이터를 빠르게 처리할 수 있으며, RGB형식에 비해 이미지 데이터 크기가 작고, 다른 형식의 이미지로 변환하기 편리한 장점이 있기 때문에 YUV형식을 JPEG 2000 코덱 엔진의 기본 포맷으로 적용하였다.

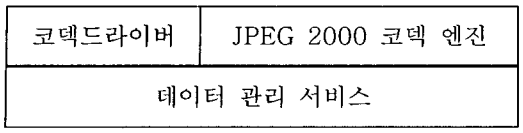


그림 2. 소프트웨어 구성
Fig. 2. Software organization

코덱 드라이버는 JPEG 2000 코덱 엔진과 입출력 데이터 변환을 위한 인터페이스를 담당한다. JPEG, BMP, PNM, RGB565 와 같은 형식의 이미지를 YUV형식으로 변환하거나, 반대로, YUV형식으로 표현된 이미지를 JPEG, BMP, PNM형식으로 변환하기 위한 함수(API)들을 제공한다.

데이터 관리 서비스는 할당한 힙 메모리(heap memory)를 관리하고, JPEG 2000 코덱 엔진과 코덱 드라이버에서 사용하는 메모리와 데이터를 관리한다. JPEG2000 코덱 엔진과 코덱 드라이버는 응용 프로그램에서 필요한 제어 코드와 데이터를 서로 분리하여, 리소스를 효율적으로 관리할 수 있는 응용 프로그램의 개발이 가능하도록 고려하였다.

미리 확보된 일정량의 힙 메모리를 가지고 있으면, 데이터 저장에 관련된 처리를 좀 더 쉽게 할 수 있으며, 운영체제에 매번 메모리를 요청하는 것 보다 빠르게 데이터를 처리할 수 있다. 힙 메모리가 필요한 메모리 요구량 보다 적게 할당되면, 중간에 메모리가 부족하여, 작업이 중단되거나, 부족한 힙 메모리를 다시 할당하게 되면, 전체적으로 메모리 효율이 떨어지게 된다.

```
static unsigned char Heap[1024*1024*3] // 힙 메모리 지정

void *pOutput = NULL;          // Target 데이터 포인터
jp2k_stream_t      in;         // 입력 스트림
jp2k_stream_t      out;        // 출력 스트림

Heap_Setup(Heap, sizeof(Heap)); // 힙 메모리 설정

in.pData  = SRC_IMAGE_DATA;   // 원본 이미지 데이터
in.Surf   = SURFTYPE_YUY2;    // 원본 이미지 형식
in.iWidth = SRC_IMAGE_WIDTH;  // 원본 이미지의 Width
in.iHeight = SRC_IMAGE_HEIGHT; // 원본 이미지의 Height

out.Surf  = SURFTYPE_JP2K;    // Target 이미지 형식
out.pData = pOutput;
stream_open(&in, Buffer_Size)
stream_open(&out, Buffer_Size);
jp2k_process(&in, &out);      // 인코딩 실행
stream_close(&in);
stream_close(&out);
```

그림 3. JPEG 2000 코덱 라이브러리 알고리즘
Fig. 3. Algorithm of JPEG 2000 codec library

그림 3은 JPEG2000 코덱 라이브러리의 실행 코드이다. 먼저 사용할 수 있는 힙메모리 크기를 설정하면, 구현된 소프트웨어는 힙 메모리 범위 내에서 이미지 인코딩/디코딩을 수행한다. 이미지 변환 작업이 종료되면, 사용하지 않는 힙 메모리는 시스템에 반환한다. 초기에 힙메모리의 크기를 정할 때, 처리하려는 이미지의 해상도를 고려하여 힙 메모리를 할당하여야 한다.

그림 4는 JPEG 2000 코덱 엔진의 힙메모리의 구조를 보여준다. JPEG 2000 코덱 엔진의 인코더는 입력 영상을 스캔라인 버퍼를 이용해 일정한 블록 단위로 이미지를 처리한다.

JPEG 2000 코덱 엔진은 메모리 사용량을 줄이기 위해, 스캔라인 버퍼(scanline buffer)를 이용하여 입력 영상을 조금씩 나누어서 인코딩과 디코딩을 수행한다. 그림 4에서와 같이 이미지를 몇 개의 블록으로 나누어서 처리하기 때문에, 최소한의 기본 메모리 버퍼 용량만큼 메모리를 사용한다. 이와 같이 블록 단위로 이미지를 처리하는 방법은 적은 메모리를 가지고 큰 이미지를 처리할 수 있다는 장점이 있다. 하지만, 적당한 기본 메모리 버퍼 용량이 확보되지 못하면, 이미지를 처리하는 시간이 오래 걸리거나, 처리하는 도중에 메모리 부족으로 종료될 수 있다. 또, 블록 단위로 이미지를 처리하기 때문에, 메모리를 필요한 만큼 이용할 수 있는 환경과 비교하여 상대적으로 압축률과 품질에 한계가 있다.

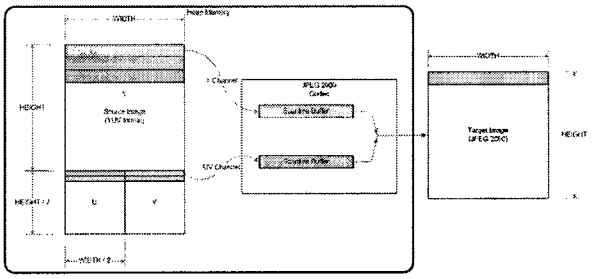


그림 4. 힙메모리 블록 다이어그램
Fig. 4. Block diagram of heap memory

스캔라인 버퍼에 저장된 소스 이미지는 32x32 바이트(byte)의 타일(tile)이라는 서로 겹치지 않는 직사각형의 조각으로 분리된다. 분리된 타일을 타일 컴포넌트(tile-components)라고 부르며, 인코더는 타일 컴포넌트 단위로 DWT(discrete wavelet transform)를 수행한다. DWT에서 처리한 웨이브렛 계수(wavelet coefficients)들은 타일 버퍼(tile buffer) 에 저장된다[7]. 타일 버퍼에 저장된 데이터는 타일 스필리터(tile splitter)를 통해 다시 32x32의 코드 블록(code block)으로 분할되고, 분할된 코드 블록은 비트 평면 단위로 웨이블릿 변환과 블록 기반 비트 플랜(bit-plane) 부호화 방법을 이용한 EBCOT(Embedded Block Coding with Optimized Truncation)[7][8][9]알고리즘에 의해 각 비트에 대한 정보를 산술 부호화기로 보낸다.

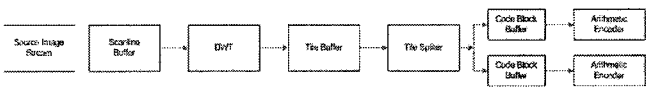


그림 5. JPEG2000 인코더
Fig. 5. JPEG 2000 encoder

그림 6은 JPEG 2000 디코더이다. JPEG 2000 코덱 엔진의 디코더는 인코더와 반대의 순서로 진행된다. 타일 스필리터 대신 타일 콤포저(tile composer)에서 서로 분리된 코드 블록을 합쳐서, IDWT(inverse discrete wavelet transform)를 위한 타일 버퍼에 저장하고, 타일 단위로 스캔라인 버퍼를 구성한다. 완성된 스캔라인 버퍼를 차례대로 디코딩한다. 복구된 이미지의 형식은 YUV 형식이다.

그림 6 JPEG 2000 디코더
Fig. 6. JPEG 2000 decoder

JPEG 2000 코덱 엔진의 디코더도 메모리 사용량을 줄이기 위해, 스캔 라인 버퍼를 이용하여 입력 영상을 조금씩 나누어서 디코딩을 수행한다. 그림 6에서와 같이 이미지를 몇 개의 블록으로 나누어서 처리하기 때문에, 최소한의 기본메모리 버퍼 용량만큼 메모리를 사용한다.

3.2 코드의 최적화

컴파일러는 소스 코드를 타겟 CPU가 사용하는 프로그램으로 변환한다. 컴파일러 최적화는 하나의 프로그램을 성능이 좋은 프로그램으로 변환하는 것이다. 컴파일러 최적화의 대상은 실행 시간, 코드 사이즈, 및 전력소모이다.

실행 시간을 향상시키는 방법에는 부동소수점 연산, 레지스터 할당(register allocation), 루프 최적화(loop optimization) 방법 등이 있다. 코드를 최적화하기 위해 먼저 프로그램의 복잡한 흐름에 대한 분석이 필요하다. 루프 문과 각 명령어의 실행 시간, 실행 횟수, 정의된 변수가 어떻게 정의되고, 어디까지 적용되는지, 그리고 사용되는 변수는 어떤 것들이 있는지 분석해야 한다.

JPEG2000은 기존 JPEG보다 압축효율이 매우 높지만, 이미지를 인코딩할 때, 부호화 과정이 복잡하고, 반복연산과 실수 연산 과정이 많다. ARM 계열의 프로세서는 부동 소수점 연산 장치를 지원하지 않는다.

ARM10 이상에서는 VFP(vector floating-point) 기능을 지원하여 부동소수점 연산이 가능하지만, 아직 만족할 만한 연산 속도를 제공하지 못한다. 정수형 데이터로만 이루어진 코드와 부동소수점으로 이루어진 데이터로 이루어진 코드를 실행을 해보았을 때, 정수형 데이터로만 이루어진 코드가 거의 2배 이상의 속도를 보여준다. 고정 소수점 코딩(fixed-point coding)의 가장 큰 장점 중 하나는 연산 속도가 매우 빠르다는 것에 있다.

부동 소수점 프로그램 코드 예제	고정 소수점 프로그램 코드 예제

int x1 = 3;
float x2 = 2.472
float y = x1 * x2;
//y = 7.416
	#define n 10
	#define _Qf(x)
	((x)*pow(2, n))
	//pow(2, n) == 2^n
	#define _Qi(x) ((x) << n)
	//x * 2^n
	
	int x1 = _Qi(3);
	int x2 = _Qf(2.472);
	int y = (x1 >> n) * x2;

그림 7. 부동 소수점 코드와 고정 소수점 코드의 비교
Fig. 7. Comparison of codes with floating-point and fixed-point

그림 7에서와 같이 부동 소수점 프로그램을 고정 소수점 프로그램으로 변환하는 방법은 간단하다. 2^n을 곱해주거나, 쉬프트 연산을 이용하여 나눗셈 연산을 수정한다. 매크로 연산(#define)은 컴파일 단계에서 적용되기 때문에 함수 호출과 같은 오버헤드가 없으며, 쉬프트 연산은 매우 빠르게 처리할 수 있다. 고정 소수점 코드는 약간의 매크로와 쉬프트 연산에 대한 오버헤드보다, 실수 연산이 없으므로 더 빠르게 동작할 수 있다. 하지만 부동 소수점 코드(floating-point code)를 고정 소수점 코드(fixed-point code)로 변환하여 계산하는 과정에서 결과 값의 차이가 발생할 수 있다. 이러한 에러 값들이 계산 하는 과정에서 누적이 되면서 원하지 않는 오차 값이 발생을 할 수 있기 때문에, 허용 가능한 최대 오차(max difference)를 줄이는 일은 대단히 중요하다. 예를 들어서 그림 7의 고정 소수점 코드에서 n을 어떻게 설정하느냐에 따라 최대 오차를 줄일 수 있다. 매우 큰 값으로 n을 설정하면 계산과정에서 오버플로우(overflow)가 발생할 수 있으며, 작은 값으로 설정을 한다면 소수점이하의 값의 버림으로 인해 오차가 커질 수 있다. 그렇기 때문에, 오버플로우가 발생하지 않는 범위에서 가장 큰 n값을 설정하는 것이 최대 오차를 줄이는 방법이다. 이때 n은 해당 시스템의 정수 연산 허용범위에서 결정을 해야 한다. 본 논문에서는 실수형 데이터를 16 비트 정수형 데이터로 변환할 때, 16비트 중 상위 5 비트를 정수 부분으로, 하위11비트를 소수점 이하 부분으로 지정하여 실험을 하였다.

ARM프로세서는 나누기 연산 명령어가 없기 때문에, 나눗셈 연산은 실행 시간이 가장 오래 걸리는 명령어 중 하나이다. 나눗셈 연산은 나누는 수의 역수를 계산하여, 역수를 곱하는 방식으로 바꾼다.

```
    z = 10.0;                    z = 1.0 / 10.0;
x = y / z;                       x = y * z;
                                =>
x = x / 10                       x = x * 0.1;
(x / y) > z                      x /(y * z)
```

log(), sin(), exp()과 같은 함수들은 연속적인 곱셈과 덧셈 연산들로 구현되는데, 일반적인 실수형 데이터를 이용한 곱셈보다 매우 느리다. 이와 같은 함수들은 결과 값을 미리 계산한 참조 테이블(lookup-table)을 만들어서 연산시간을 줄일 수 있다.

값만 정의되고, 사용되지 않는 데드 코드(dead-code)는 가장 먼저 제거되는 코드이다. 루프와 부동 소수형 연산은 임베디드 시스템의 성능을 떨어뜨리는 주요 원인중 하나이다. 고정 소수점 연산을 적용하여도 실행 속도의 향상을 얻을 수 없다면, 부동 소수형 변수 값들이 갖는 범위를 고려하여, 정수형으로 바꿀 수 있는지 검토해야 한다. 정수형 연산은 부동 소수형 보다 빠르며, 유효 숫자의 제한이 없다.

루프 최적화는 루프 문에 포함되어서 코드가 실행되는 횟수를 줄이는 방법이다. 반복 연산이 많은 코드는 아래의 예에서와 같이 루프 오버헤드(loop overhead)를 줄이고, 캐시 메모리를 효율적으로 사용하여 메모리 성능을 향상시킬 수 있도록 해야 한다.

```
for (i = 0; i < 100; i++)
{
    x = y + z[i];
}
```

=>

```
for (i = 0; i < 99; i+=2)
{
    x = y + z[i];
    x = y + z[i + 1];
}
```

또, 루프 내부에 존재하면서 연산 결과가 항상 상수 값을 유지하는 코드는 루프 문 밖으로 빼내어 루프 내부에서 실행되는 코드의 개수를 줄여 성능 향상을 얻을 수 있다.

ARM 프로세서는 RISC 방식의 로드-스토어 방식의 아키텍쳐로 데이터를 처리하려면, 먼저 메모리에서 데이터를 레지스터로 읽어야 한다. ARM은 32비트 단위로 데이터를 처리하고, 레지스터에 저장한다. 만약 레지스터에 미리 데이터가 저장되어 있다면, 메모리에서 데이터를 읽어 오는 과정이 생략된다. 메모리에 데이터를 읽고, 저장하는 명령어는 레지스터에서 처리하는 명령보다 느리기 때문에, 메모리 접근 연산을 레지스터 접근 연산으로 변환하여 사용하는 데이터들을 최대한 레지스터에 할당되도록 해야 한다. ARM 프로세서는 12개의 레지스터를 변수의 할당에 사용한다. 코드에서 12개 이상의 변수가 사용된다면, 나머지 변수들은 레지스터가 아닌 메모리에 저장되기 때문에, 함수 내에서 필요한 변수는 12개미만을 사용하는 것이 좋은 방법이다. 변수의 종류에 따라 레지스터에 할당이 가능한 변수와 불가능 변수들이 있다. 지역변수(local variable)와 함수 파라미터(function parameter)가 레지스터에 할당이 가능한 변수에 해당된다.

전역 변수(global variable)는 외부의 함수에서 값을 변경될 수 있기 때문에, 레지스터에 할당되지 않는다. 전역 변수의 경우, 함수에서 처음과 끝에서만 변수를 참조하고, 실제 처리되는 부분은 지역변수를 이용한다.

수정된 코드는 프로파일링(profiling)을 통하여 각 코드의 실행 결과를 분석한 다음, 최적화 대상이 되는 코드를 선정하여 해당 코드를 최적화함으로써 성능 향상을 얻을 수 있다.

Ⅳ. 실험 및 결과

구현된 JPEG 2000 코덱의 성능을 평가하기 위하여 이미지 해상도와 메모리 사용량에 대한 실험을 하였다. 실험에 사용된 장비는 ARM9 프로세스를 장착한 개발 보드와 ADS(ARM Developer Suite), 그리고 PC로 구성되어 있다. 실험에 사용된 이미지는 320x240, 640x480, 1600x1200, 2048x1036 해상도의 24bit BMP 이미지 파일을 이용하였다. BMP 파일을 JPEG 2000 형식으로 인코딩하는 실험과 인코딩된 파일을 다시 BMP형식으로 변환하는 디코딩 실험을 하였으며, 각 해상도별로 메모리 사용량과 압축율을 측정하였다(그림 8).

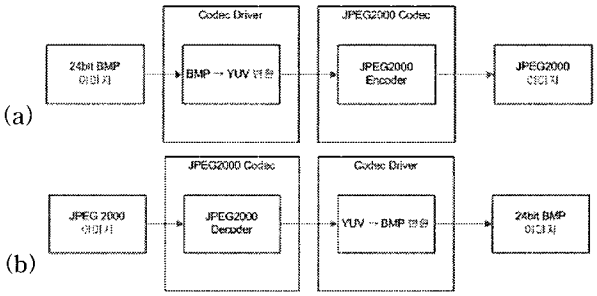


그림 8. JPEG 2000 코덱의 인코딩/디코딩 테스트
(a)인코딩 테스트 (b)디코딩 테스트

Fig. 8. Encoding/Decoding test of JPEG 2000 codec
(a)Encoding test (b) Decoding test

표 1에서는 각 해상도별로 인코딩/디코딩 처리할 수 있는 최소 힙메모리 요구량과 메모리 사용량을 비교하였다. 힙 메모리는 전체 이미지를 처리하는데 필요한 최소 메모리 크기이며, 메모리 사용량은 JPEG2000 코덱이 인코딩/디코딩 중에 사용한 최대 메모리 사용량을 측정한 값이다.
표 2는 BMP 이미지를 JPEG 2000으로 변환하였을 때, 인코딩된 이미지의 크기와 압축률, 실행시간을 측정한 결과이다.

표 1. 각 해상도별 메모리 사용량
Table 1. Amounts of memory used for each resolution

해상도	힙메모리 크기	최대 메모리 사용량
320x240	512KB	74KB
640x480	1,024KB	135KB
1600x1200	1,280KB	320KB
2048x1536	1,536KB	484KB

표 2. 각 해상도별 인코딩 결과

Table 2. Encoding results for each resolution

해상도	BMP 파일크기	JPEG2000 파일크기	이미지 압축율	처리시간
320x240	226KB	94KB	41.5%	24.34ms
620x480	931KB	314KB	33.7%	64.12ms
1600x1200	5,734KB	1,785KB	31.1%	384.78ms
2048x1536	9,345KB	2,869KB	30.7%	856.12ms

$$PSNR = 10*\log_{10}[(255*255)/MSE] \quad (1)$$

$$MSE = 1/N^2 * \sum_{x=1}^{N}[\sum_{y=1}^{N}[(f(x,y)-g(x,y))^2]]$$

$x : Height, y : Width$

$N(=x*y)$: 모든픽셀의갯수

$f(x,y)$: 좌표(x,y)의비교영상픽셀값

$g(x,y)$: 좌표(x,y)의원래영상픽셀값

표 3. 이미지 품질 비교

Table 3. Comparison of image quality

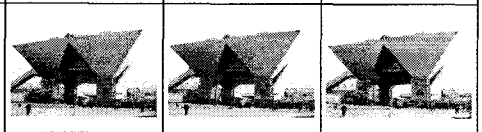
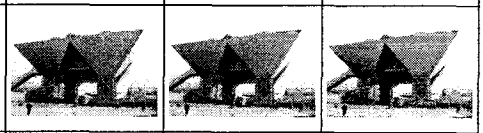
해상도	원본이미지	ARM9	Window PC
640x480			
이미지 크기	931KB	314KB	216KB
PSNR	-	31.83	29.42
1600x1200			
이미지 크기	5,734KB	1,785KB	952KB
PSNR	-	31.24	29.63

표 3은 임베디드 환경에서 인코딩된 이미지와 PC환경에서 인코딩한 이미지를 각각 원본 이미지와 PSNR(peek signal to noise ratio) 값을 비교한 것이다. PSNR은 원본 이미지와 비교 대상 이미지를 YUV로 변환하여 Y채널에 대해 식 (1)을 적용하여 측정한 값이다.

PC환경에서는 Pentium M 1.8GHz CPU와 Visual C++을 이용하였다. Visual C++에서 고정 소수점 연산을 적용할 경우, 연산 성능이 떨어질 수 있기 때문에, PC환경에서는 부동소수점을 이용한 코드를 적용하였다. 구현된 JPEG 2000코덱은 임베디드 환경에서도 PSNR값의 차이가 매우 적고, 원본 이미지와 거의 동일한 품질의 이미지를 제공하는 것을 알 수 있다.

V. 결론

본 논문에서 구현된 JPEG 2000 코덱은 별도의 백-엔드 칩셋(back-end chipset)없이 임베디드 시스템에 적용될 수 있으므로, 비용적인 측면에서 매우 유리하다. 또한 메모리를 자체적으로 관리하여 메모리 버퍼를 독립된 프로세서로 처리할 수 있으며 다른 애플리케이션과 서로 충돌 없이 임베디드 시스템에 적용할 수 있다.

임베디드용 소프트웨어는 개발 비용, 프로세스 성능, 메모리, 저장장치, 전력 소모와 같은 시스템 리소스가 제한되기 때문에, 시스템에서 지원하는 리소스를 효율적으로 사용하기 위한 최적화가 반드시 필요하다는 특수성을 가지고 있다. 이와 더불어 여러 종류의 디바이스를 지원해야 하기 때문에, 소프트웨어 모듈과 응용 프로그램도 환경에 따라 바꿔야하는 어려움이 있다. 이로 인해 운영체제와 하드웨어에 종속되고, 소프트웨어의 이식성이 떨어지는 문제가 발생하기도 한다. 이런 문제를 극복하기 위해, 코덱에서 제어 코드와 데이터 관리 코드를 분리하였다. 함수가 중심이 되고 데이터 구조가 함수를 지원하는 역할을 하는 전통적 방법보다, 데이터가 중심이 되고 코드가 이를 지원하는 역할을 할 수 있도록 코덱을 설계하여, 소프트웨어 개발의 복잡성과 비용을 줄일 수 있도록 고려하였다. 제어코드와 데이터 관리 코드를 분리하면, 임베디드 환경에서 뛰어난 안정성, 유연성, 이식성을 갖춘 소프트웨어를 개발할 수 있을 것이다. 향후 연구과제는 고정 소수점 코딩 부분의 지속적인 개선을 통해 처리 속도를 보다 향상시키는 것이다. 이미지 압축에 대한 꾸준한 연구가 향후 이미지 데이터를 이용한 임베디드 시스템에 적용될 경우 많은 성능 향상을 기대할 수 있을 것이다.

참고 문헌

[1] C. Christopoulos, A. Skodras, and T. Ebrahimi, "The JPEG 2000 Still Image Coding System: an Overview," IEEE Transactions on Consumer Electronics, Vol. 46, No. 4, pp. 1103-1127, November 2000.

[2] M. Boliek, C. Christopoulos, and E. Majani (Eds), "JPEG 2000 Part 1 Final Draft International Standard," (FDIS15444-1), ISO/IEC JTC1/SC29/WG1, N1855, August 2000.

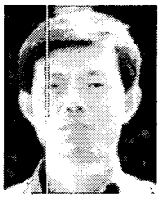
[3] International Organization for Standardization and International Electrotechnical Commission, ISO/IEC 10918-1:1994, Information technology Digital compression and coding of continuous-tone still images: Requirements and guidelines.

[4] "Netpbm Home Page," http://netpbm.sourceforge.net,h 2005

[5] M. Luse, "The BMP file format," Dr. Dobb's Journal, vol. 9, no. 10, pp. 1822, Sept. 1994.

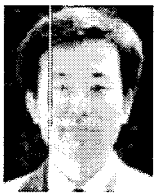
[6] "FOURCC.org Home Page", http://www.fourcc.org/, 2005

[7] H.C. Fang, T.C Wang, C.J. Lian, T.H. Chang, and L.G. Chen,"High Speed Memory Efficient EBCOT Architecture for JPEG2000," in Proc. IEEE International Symposium on Circuits and Systems, Vol. 2, Thailand, pp. 736-739, May 2003.
[8] D. Taubman, "High performance scalable image compression with EBCOT," IEEE Trans. Image Processing, vol. 9, no. 7, pp.1158-1170, July 2000.
[9] D. Taubman, E. Ordentlich, M.J. Weinberger, and G.Seroussi, ""Embedded block coding in JPEG2000,"" Signal Processing: Image Communication 17 (1) (2002) 49.72.
[10] K. Andra, C. Chakrabati, and T. Acharya, "A High-Performance JPEG2000 Architecture," IEEE Trans. on Circuits and Systems for Video Technol., vol.13, no.3, pp. 209-218, March 2003.
[11] "The JasPer Project Home Page", http://www.ece.uvic.ca/~mdadams/jasper/, 2006



김 영 태 (Young-Tae Kim)
1973년 서강대학교 전자공학과 (공학사)
1986년 University of New Mexico 전기공학과 (공학박사)
1989년~현재 동국대학교 전기공학과 교수

조 시 원 (Shi-won Cho)
1994년 동국대학교 전기공학과 (공학사).
1998년 동국대학교 대학원 전기공학과 (공학석사).
2003년~현재 동국대학교 대학원 전기공학과 박사과정



이 동 욱 (Dong-Wook Lee)
1983년 서울대학교 전기공학과 (공학사).
1985년 서울대학교 대학원 전기공학과 (공학석사).
1992년 Georgia 공대 전기공학과 (공학박사).
1993년~현재 동국대학교 전기공학과 교수