2002년 한국정보처리학회 춘계 학술발표논문집 제 9 권 제 1 호

# 동영상 플레이어를 위한 BREW(Binary Runtime Environment for Wireless) 프로그래밍의 최적화 기법

윤민홍, 류은석, 유혁
고려대학교 컴퓨터학과
e-mail : {mhyun, esryu, hxy}@os.korea.ac.kr

# Method of optimizing on BREW(Binary Runtime Environment for Wireless) for video decoding

Min-Hong Yun, Eun-Seok Ryu, Hyuck Yoo
Dept. of Computer Science and Engineering, Korea University

**요 약**

휴대폰 사용 인구가 급증하고, 최근 출시되는 휴대폰의 사양이 좋아짐에 따라, 휴대폰을 위한 응용 프로그램(application program) 개발자 역시 늘어나고있다. 하지만, 휴대폰이 PC 에 비해 제한된 자원을 갖는 장치이기 때문에, 고성능의 PC 에서는 고려하지 않았던 최적화 문제가 발생하고 이를 해결해야 할 필요가 있다. 더욱이 최근에는 휴대폰으로 동영상을 보려는 시도가 늘고있는데, 동영상 플레이어는 기타 대부분의 응용 프로그램보다 많은 연산이 필요한 응용 프로그램이므로 최적화 문제가 더욱 절실해졌다. 본 논문에서는 휴대폰에서 특히, BREW platform 에서 프로그래밍 시에 필요한 최적화 기법을 소개한다.

## 1. 서론

휴대폰 사용자의 상당 수가 휴대폰을 단순히 전화하는 도구로만 생각하지 않고, 휴대폰을 통해 일정관리나 게임 등의 많은 일을 하고싶어 한다. 사용자의 요구가 다양해지고 요구하는 응용 프로그램(application program)의 복잡도가 증가함에 따라 응용 프로그램이 휴대폰의 CPU 와 메모리 자원을 많이 요구하게 됐다.

특히, 최근 들어 휴대폰의 LCD 성능이 좋아지고, 표현 가능한 색상의 수가 증가함에 따라 많은 휴대폰 사용자가 휴대폰을 통해 동영상을 보려는 욕구가 늘어나고, 그에 대한 연구도 활발히 진행되고 있다. 동영상 플레이어 즉, 디코더(decoder)는 많은 연산을 하는 응용 프로그램이기 때문에 CPU 를 많이 사용한다[1].

본 논문에서는 BREW(Binary Runtime Environment for Wireless) 플랫폼(platform) [1] 에서 IDCT 와 Motion Compensation 의 디코딩 과정을 거친 중간 데이터(Intermediate Data)[2]를 사용한 동영상 플레이어[3][4]를 구현하면서 얻은 최적화 프로그래밍 기법을 소개하도록 한다.

## 2. 실험 및 연구 환경

실험과 구현은 Qualcomm 사가 제공하는 BREW 에뮬레이터(Emulator)에서 작업을 한 뒤, 실제 단말기에서 이루어졌다.

단말기는 LG 전자가 제조한 CX-300K 제품이다. 이 제품은 ARM7TDMI 프로세서(processor)를 사용하고

[1] BREW(Binary Runtime Environment for Wireless)는 미국 Qualcomm사가 CDMA (코드분할다중접속) 방식의 이동통신기기용으로 개발한 차세대 애플리케이션 플랫폼이다[2].

[2] MPEG을 사용하지 않고 중간 데이터(Intermediate Data)를 사용하는 까닭은, 중간 데이터를 사용하면 연산횟수를 줄일 수 있기 때문이다.

메모리의 크기는 1832 Kilobytes 이다. ARM7TDMI 프로세서는 88MHz 의 64bit RISC 프로세서로 31 개의 범용 레지스터(general purpose register)를 갖고있으며, 메모리 버스(memory bus)는 명령어와 데이터에 대해서 32 비트 단위로 작동한다[5]. LCD 는 120x160 의 해상도이며 8 비트 색상으로 256 칼라를 지원한다.

사용한 BREW 버전은 1.0.2.5 으로 2001 년 10 월에 릴리즈(release)된 것이다. 개발에 사용된 BREW SDK 는 버전 1.0.1 을 사용했으며 개발 언어는 C 이다. 휴대폰 상에서 테스트 하기 위한 컴파일에 사용된 ARM 컴파일러는 ARM BREW Builder 1.0 을 사용했다.

## 3. 프로그래밍 최적화 기법

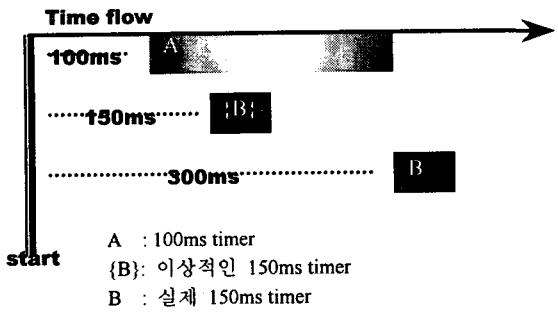
응용 프로그램(application)의 실행 성능을 극대화 시키기 위한 기법에 대해 생각하기 전에, PC 상의 프로그래밍보다 휴대폰상의 프로그래밍이 상대적으로 갖게 되는 제한 사항에 대해서 생각한다.

그 다음 비디오 디코딩 (Video Decoding)과 관련된 제약점을 극복하기 위한 프로그래밍 기술에 대해 설명하고, 메모리 접근의 최소화와 CONVERTBMP()함수 호출의 최소화를 통한 최적화 기법들에 대해서 설명한다.

### 3.1 전화기 환경의 제약

시스템적인, 즉 하드웨어적인 특징은 CPU 의 계산 능력(computation power)문제를 들 수 있다. 1GHz 가 넘는 PC 에 비해 88MHz 는 매우 열악한 환경이 아닐 수 없다. 또한 2 Megabytes 가 안 되는 메모리 역시 큰 제한 사항중의 하나이다. 부동소수점 연산을 프로세서(processor)가 지원해주지 않는 다는 것도 큰 제약 사항이다.

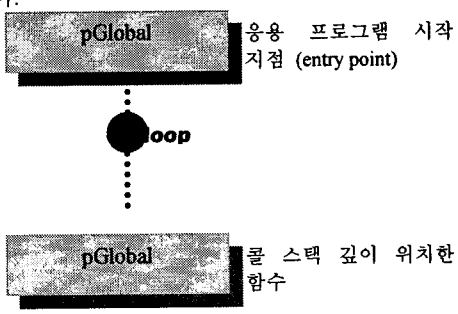
BREW 플랫폼(platform)이 가지고 있는 제약사항은 타이머(timer) 여러 개가 동작하지 않는 다는 것이다. 즉, 하나의 응용 프로그램에서는 하나의 타이머만 동작 가능하다. 예를 들어 2 개의 타이머 A, B 를 각각 100ms, 150ms 단위로 동시에 동작시키고, 타이머 A 에 의해 동작하는 루틴(routine)이 한번 처리되는데 200ms 가 걸린다고 가정하자.




**[그림 1] BREW 의 timer**

[그림 1]처럼 타이머 A 와 B 가 동시에 작동 하면 짧은 타이머 A 가 먼저 익스파이어(expire)되어 타이머 A 에 콜백 함수(callback function)로 등록된 루틴이 처리된다. 에뮬레이터와 같은 이상적인 타이머 동작이라면 타이머 {B}가 작동한 후 150ms 가 되면 타이머 {B}의 콜백 함수가 호출되어야 하지만, 실제로는 A 의 루틴이 처리된 다음에, 즉 등록된 콜백 함수가 리턴(return)한 다음에 타이머 B 의 루틴이 처리된다. 뿐만 아니라 BREW 는 반복적인 타이머 호출을 지원하지 않기 때문에, 루틴 A 를 100ms 를 주기로 호출하기 위해서는 A 에서 리턴하기 직전 타이머에 콜백 함수를 재등록 해야 한다.

BREW 에는 BREW SDK 에서 사용하는 내부 변수와의 충돌을 피하기 위해 전체적으로 하나의 전역 변수(global variable)만 존재할 수 있다. 이 전역변수는 최대 크기 64 kilobytes 까지 갖을 수 있는 구조체이다. 또한 이 전역 변수는 함수 외부에 선언된 변수가 아니고 응용프로그램의 시작점(entry point)이 되는 함수의 인자(argument)로 넘어오는 포인터로만 접근이 가능하다. 그러한 이유로 콜 스택(call stack) 깊게 위치한 함수에서 전역변수를 사용하기 위해서는 BREW 응용 프로그램을 처음 동작 시키는 함수부터 함수 인자(function parameter)로 이 전역 변수에 대한 포인터를 유지시켜야 한다. 이렇게 모든 함수에서 전역변수를 접근하기 위해 인자를 유지한다면, 함수 호출 때 사용되는 스택에 푸쉬(push)하고 팝(pop)하는 동작 때문에 함수 호출을 많이 하는 응용 프로그램의 경우 그 응용 프로그램을 수행할 때 전체적인 속도저하가 올 수 있다.



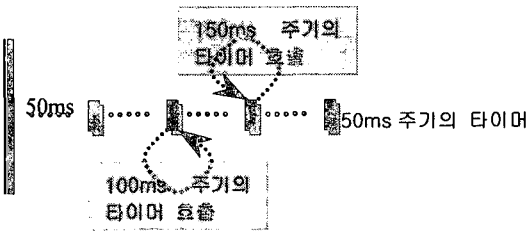

**[그림 2] 콜 스택과 전역변수**

[그림 2]에서와 같이 콜 스택 윗부분에서 많은 횟수의 반복(loop)이 있고 콜 스택의 깊이가 깊다면, 전역변수를 사용할 필요가 없는 함수에서조차 그것에 대한 포인터를 유지해야 아래부분의 함수에서 전역변수에 접근하기 때문에 수없이 많은 푸쉬와 팝의 반복이 발생하고, 이것이 응용프로그램의 실행속도가 현저히 떨어지게 할 수 있다.

### 3.2 제약점을 극복하는 프로그래밍 기법

a. 타이머 분할

3.1 에서 많은 지면을 할당해 언급했듯 BREW 는 실제적으로 하나의 반복되지 않는 타이머만 작동한다. 따라서 사용해야 하는 타이머가 여러 개인 경우 타이머 디멀티플레싱(de-multiplexing)을 해야 한다. 즉, 가장 작은 타이머로 타이머 하나를 동작시키고, 타이머를 통해 누적된 시간을 사용해 큰 타이머를 동작시켜야 한다.




[그림 3] 타이머 디멀티플레싱

[그림 3]을 보면 50ms 주기의 타이머를 사용해 100ms 주기의 타이머와 150ms 주기의 타이머를 디멀티플레싱하고 있는 것을 볼 수 있다.

b. 스택 오버플로우(stack overflow) 방지

단말기의 경우 함수 호출에 사용되는 스택의 크기가 작기 때문에 많은 지역변수를 사용하면 쉽게 스택 오버플로우가 발생하게 된다. 스택 오버플로우가 발생하면 지역 변수들의 내용이 변경되기 때문에 함수가 제대로 동작하지 않을 뿐더러 발견하기도 힘들다.

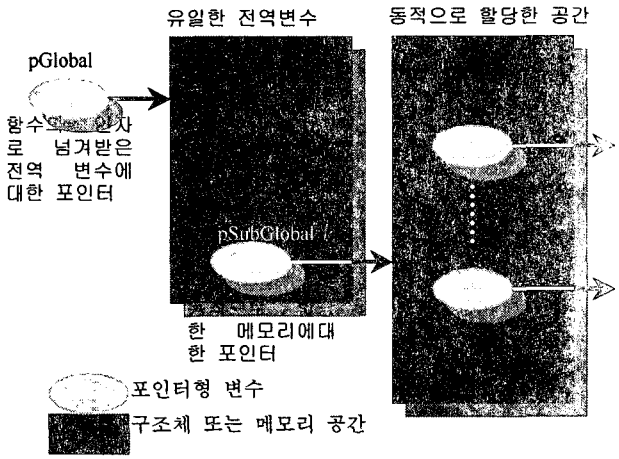
콜 스택의 깊이와 함수에 사용된 파라메터에 따라 차이가 있겠지만, 경험상 얻은 결론으로 50 bytes 정도가 적정선이다.

### 3.3 최적화 방법

a. 메모리 접근의 최소화

현재까지의 휴대폰 시스템에는 캐쉬(cache)가 존재하지 않기 때문에 레지스터에 들어있지 않는 data 를 접근하기 위해서는 매번 메모리에 접근해야 한다. 메모리에 접근한다는 것은 버스를 사용해야 한다는 것을 의미하기 때문에, 가능한 포인터의 사용을 줄여 간접적으로 데이터에 접근하는 하는 것을 줄이고 직접적으로 접근해야 한다. 3.1 에서 언급했듯 하나의 응용프로그램에는 전역 변수가 단 하나만 존재하고 그 크기가 64KB 에 한정 되어 있기 때문에 더 큰 자료를 전역변수에 저장하고 싶으면, 프로그램 실행 도중 동적으로 메모리를 할당해서 사용해야 한다.

[그림 4]의 경우를 생각해보자. 함수의 인자로 유일한 전역변수의 주소를 갖는 포인터(pGlobal)를 넘겨받았다. 또 전역변수 하나의 공간(64KB)으로는 모자라 전역변수 내에 포인터 변수(pSubGlobal)를 두어 동적으로 메모리 공간을 할당한 곳의 주소를 갖도록 했다. 이렇게 되면 동적으로 할당한 공간에 있는 변수에 접근하기 위해서 3 번의 메모리 접근을 해야 한다. 따라서 접근을 많이 하는 변수를 동적으로 할당한 공간에 위치 시키지 말고 유일한 전역변수에 위치 시켜 가능한 메모리 접근 횟수를 줄여야 한다. [그림 2]와 같이 반복해서 함수를 호출하는데 호출된 함수에서 포인터를 여러 번 따라가야 원하는 데이터를 얻을 수 있는 곳에 데이터가 있으면 더더욱 비효율적이므로 이런




[그림 4] 포인터가 사용된 메모리공간

경우에는 포인터를 가능한 적게 따라가게 최적화 시켜야 한다.

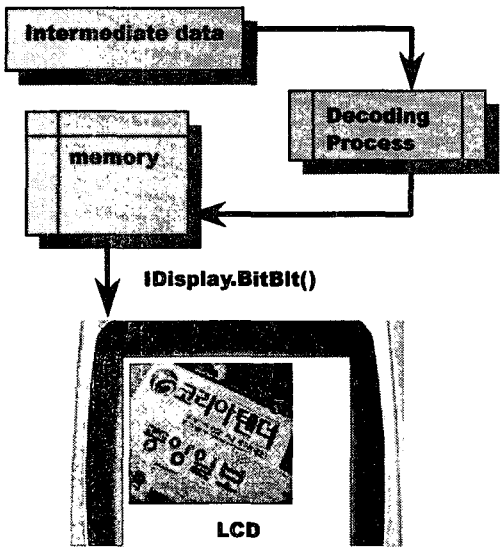
b. CONVERTBMP()호출의 최소화

CONVERTBMP()함수의 호출을 줄이는 것은 매우 중요한 일이다. 왜냐하면 이 함수가 하는 일은 메모리 복사가 많이 일어나는 함수 이기 때문이다.

BREW 개발환경은 IDisplay 라는 인터페이스(interface)가 BitBlt()라는 함수를 노출(expose)하고 있다. 일반적으로 동영상 플레이를 하기 위해서는 즉, 디코딩 과정을 거치기 위해서는 비트맵(bitmap) 파일을 리소스(resource)형태로 읽어와 화면에 뿌려주는 것이 아니고, 메모리에 화면에 뿌려줄 데이터가 위치할 영역을 확보한 다음 그곳에 연산을 통해 디코딩된 각 픽셀(pixel) 정보를 담아야 한다. 즉 픽셀 정보는 스트림으로 읽어오는 것이 아니고 동적인 계산에 의해 메모리에 채운 다음 그 정보를 화면에 뿌려줘야 한다는 것이다.

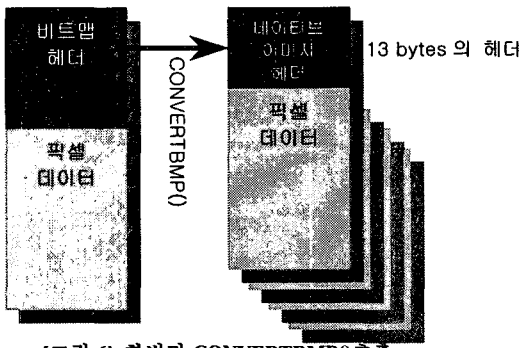
[그림 5]에서 보는 것과 같이 디코딩 과정에 의해 픽셀 정보로 채워진 메모리를 IDisplay 인터페이스의 BitBlt()라는 함수를 호출함으로써 화면에 뿌려준다. 그러나 실제 단말기에서는 에뮬레이터에서와는 다르게 CONVERTBMP()라는 함수를 호출해줘야 한다. 이 함수는 비트맵파일을 장치(device)에 맞는 네이티브 포

맷(native format)으로 바꿔주는 일을 한다[6]. 그렇지만 이 함수는 112x96 크기의 화면을 보여주기 위해서 호출하면 약 150ms 정도의 시간이 지나야만 return 한다. 즉, 매 화면을 보여줄 때마다 150ms 가 필요하다면 사용자를 만족시킬 수 있는 프레임 율(frame rate)을 얻기 힘들다. 또 이 함수는 새로운 메모리를 할당하기 때문에 함수를 호출한 다음 반드시 메모리 해제(free)를 하기 위해 SYSFREE()라는 함수를 호출하기 때문에 함수 호출에 시간을 더 낭비하게 된다.




[그림 5] 메모리에 프레임이 보이는 과정

이 함수의 호출을 한번으로 줄임으로 성능을 향상시킬 수 있다. 프로그램 초기화 과정에서 dummy 값으로 채워진 그렇지만 비트맵 헤더(header)는 맞게 작성된 비트맵 데이터를 대상으로 한번 호출한 다음 결과가 저장된 메모리를 반복해서 사용하도록 한다. 단, 앞 13byte 는 네이티브 포맷의 헤더부분 이므로 그 부분 다음부분부터 픽셀의 정보를 채운다.




[그림 6] 한번만 CONVERTBMP()호출

[그림 6]에서 보는 바와 같이 CONVERTBMP()의 호출의 결과로 네이티브 이미지가 담긴 메모리 영역을 얻을 수 있다. 이 중 이미지의 헤더부분만 유지한 채 픽셀의 칼라 정보가 들어가는 픽셀 데이터 메모리에 새로 화면에 뿌려줄 프레임의 데이터를 바꾸면 된다. 이로 얻을 수 있는 효과는 CONVERTBMP()를 한번만 호출하며 CONVERTBMP()가 동적으로 할당한 메모리를 해제 시켜 주기 위해 SYSFREE()역시 한번만 호출되면 된다는 것이다. 이 과정은 성능향상을 위해 어려운 최적화과정을 거치는 것이 아니고 간단히 150ms 가 걸리는 함수의 호출을 하지 않기 때문에 뛰어난 성능 향상의 효과를 얻을 수 있다.

## 4. 결론 및 향후 연구과제

휴대용 단말기가 갖는 특징들, 특히 PC 에 비해 갖게 되는 열악한 시스템 환경을 이해하고, 적절한 크기의 메모리 사용과 메모리 접근의 최소화를 통해 응용 프로그램의 실행시간을 줄이도록 해야 한다. 또한 프로파일링(profiling)을 통해 호출에서 return 까지의 시간이 오래 걸리는 CONVERTBMP()같은 함수들의 호출을 가능한 줄여야 전체적인 실행 시간을 줄일 수 있다. 타이머의 경우 작은 주기로 디멀티플레싱하여 큰 주기의 타이머를 대체해야 한다.

위 과정을 통해 초당 1.1~1.5 프레임 율을 보였던 디코더의 성능을 초당 1.6~2 프레임 율을 가질 수 있도록 제작할 수 있었다.

지금까지의 실험은 특정 휴대폰에 탑재된 BREW 에 대해서 이루어졌다. 더 정확한 최적화 기법을 찾기 위해서는 BREW 가 탑재된 다른 휴대폰에서도 실험이 이루어져야 할 것이다. 또 타이머를 디멀티플레싱 해도 콜백 함수로 등록된 루틴을 처리하는데 걸리는 시간이 타이머의 간격보다 길다면 시간이 지날 수록 타이머가 늦게 작동되는 현상이 발생하므로 최소 타이머 간격을 설정하는 데에도 주의를 기울여야 한다.

## 참고문헌

[1] 한상범, "계산 자원을 최소화하는 화상 회의 기법 연구", 고려대학교 석사 논문 , August 2000.

[2] Qualcomm, " BREW SDK User' s Guide" , " http://www.qualcomm.com/brew/" , July 2001

[3] Jin Hwan Jeong, Chuck Yoo, "A Server-centric Streaming Model", NOSSDAV, pp.25-34

[4] 한종민,한상범,정진환, 유 혁, "BREW 기반의 동영상 플레이어 구현", 제 29 회 정보과학회 추계학술발표회 논문집, Vol. III, 691, Oct. 2001.

[5] ARM, "ARM7TDMI (Rev 4) Technical Reference Manual", "http://www.arm.com/arm/documentation?OpenDocument", 2001

[6] Qualcomm, "BREW SDK Reference", " http://www.qualcomm.com/brew/" , July 2001