NDK를 이용한 안드로이드 애플리케이션 성능향상에 관한 연구

이재규*, 최진모*, 이상엽*, 최효섭*, 이철동*
*전자부품연구원 전북임베디드시스템연구센터
e-mail:jae4850@keti.re.kr

A Performance Improvement Study on Android Application using NDK

Jae-Kyu Lee*, Jin-Mo Choi*, Sang-Yub Lee*, Hyo-Sub Choi*, Chul-Dong Lee*
*Jeonbuk Embedded System Research Center, Korea Electronics Technology Institute

요 약

스마트폰의 급속한 확산과 함께 스마트폰 애플리케이션 시장이 빠르게 성장하고 있다. 이러한 성장세에 따라 많은 애플리케이션 개발자들이 생겨났으며, 다양한 콘텐츠와 수많은 애플리케이션이 개발되어지고 있다. 여기서 우리는 모바일 기기들의 제한적인 요소를 간과해서는 안 된다. 제한적인 모바일기기에서 유저가 만족할 만할 애플리케이션을 개발하기 위해서는 효율적인 자원 활용과 함께 효율적인 프로그래밍을 해야 할 필요가 있다. 본 논문은 안드로이드 NDK 및 SDK를 기반으로 Native C와 Java를 이용해 애플리케이션을 설계하고, 각 애플리케이션간의 알고리즘 수행속도, 프로세서 점유율측면에서 성능측정 실험을 수행했다. 실험 결과를 통해 보다 우수한 성능의 안드로이드 애플리케이션 개발 방법에 관해 연구했다. 성능측정 항목으로는 JNI delay, Integer, Floating point, Memory access algorithm, String이며, 실험은 삼성 갤럭시 S1에서 수행하였다.

1. 서론

구글은 Open Source를 기반으로 다양한 API와 Tool 그리고 SDK(Software Development Kit), NDK(Native Development Kit) 등 풍부한 개발환경을 지원한다. 구글의 체계적인 개발환경은 많은 개발자들로부터 수많은 애플리케이션이 개발되어지도록 한다. 개발자들은 제한적인 요소를 갖는 모바일 기기에서 애플리케이션을 개발할 때 반드시 프로그램의 효율성을 고려해야한다. 본 논문에서는 SDK와 NDK를 기반으로, 동일한 알고리즘을 Native C와 Java를 이용해 구현하여 각 알고리즘간의 성능측정 작업과 원인분석을 했다. 성능측정은 삼성 갤럭시 S1에서의 알고리즘 수행속도, 알고리즘이 수행되는 동안의 CPU점유율에 관해 실험을 수행했다. 결과를 바탕으로 보다 효율적인 안드로이드 애플리케이션 개발 방법에 관해 연구했다.

2. 배경이론

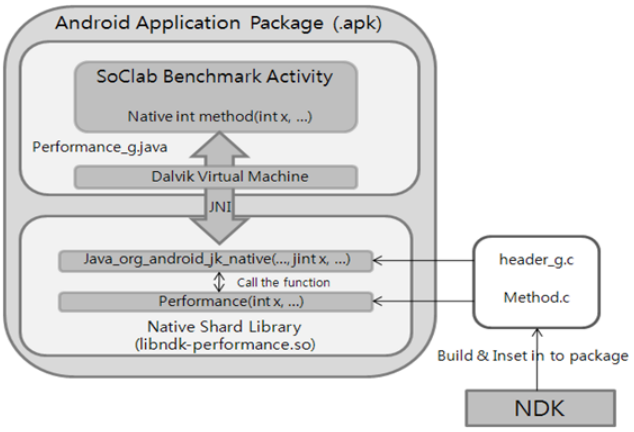
안드로이드(Android) : 안드로이드는 운영체제(OS), 미들웨어, 핵심 애플리케이션들을 포함한 모바일 기기용 소프트웨어 스택이다. 안드로이드 소프트웨어 개발도구는 자바 프로그래밍 언어를 사용하여 안드로이드 플랫폼 상에 응용프로그램을 개발하는데 필요한 도구들과 API를 제공한다.[1]

NDK(Native Development Kit) : 안드로이드 NDK란 자바 프로그래밍 언어로 작성된 안드로이드 애플리케이션에서 사용할 수 있는 C 또는 C++로 작성된 공유라이브러리(Shared libraries)를 Build할 수 있도록 하는 개발 툴체인(development tool-chain)을 포함한 Toolset이다. 현재 안드로이 NDK 최신 버전은 Android-NDK-Revision 8b 이다.

JNI(Java Native Interface) : JNI는 자바에서 다른 언어로 만들어진 소스를 접근하여 실행하게 하거나 반대로 다른 언어에서 자바 코드를 접근해서 사용할 수 있도록 하기 위한 Interface역할을 하는 것을 말한다.[2] NDK와 JNI를 이용하면 기존의 C나 C++로 설계되어 있는 Code를 재사용 할 수 있다는 장점이 있고, 속도가 느린 Java대신 Native언어 C와 C++로 설계함으로써 속도향상을 도모할 수 있다.

3. 실험구성

실험을 위해 NDK기반 Native C를 이용한 애플리케이션과 SDK기반 Java를 이용한 애플리케이션을 설계했다. Java알고리즘은 일반적인 안드로이드 애플리케이션 설계 방법과 동일하게 설계했다. Native C를 이용한 알고리즘은 Java측에서 JNI를 통해 C Method를 호출하는 형태로 구현했으며, 설계한 애플리케이션의 구조는 그림1과 같다.



(그림 1) Native C로 설계한 애플리케이션 구조

설계한 알고리즘은 모두 위와 같은 구조에 의해 설계되었으며, Java로 설계한 애플리케이션과 성능측정 실험을 했다. 성능측정 항목으로는 JNI delay와 정수형 및 부동소수점 성능측정, 메모리접근 속도를 측정하기 위한 Memory access항목과 문자열 처리 속도를 측정하는 String항목으로 이루어져 있다. 수행속도 실험 알고리즘은 각각 Fibonacci수열, 원의 넓이, Bubble Sort Worst Case, 계속적으로 입력되는 문자열 처리 알고리즘으로 구현했으며, CPU점유율 실험 알고리즘은 각각 Factorial, 연속적인 실수의 곱셈, 재귀적으로 구현한 Hanoi Tower, 연속적인 문자열 처리 알고리즘을 구현하여 알고리즘이 수행되는 동안의 평균 CPU점유율을 계산했다. 또한 정확한 수행시간 측정을 위해 'System.nanoTime()'함수를 이용해 연산 소요시간을 측정했고, 보다 신뢰할 수 있는 값을 얻기 위해 n값이 100인 조화평균을 이용해 평균값을 얻었다.

4. 실험결과

실험에 앞서 보다 정확한 실험을 위해 JNI Delay를 측정했다. 본 실험의 Java알고리즘은 Native Method를 호출하는 형태로 구현되어있다. 따라서 Native 함수로 값을 전달될 때 JNI Delay가 발생한다. 이러한 Delay가 다른 연산 수행시간에 포함되면 정확한 성능을 측정할 수 없다. JNI Delay는 1.14 msec로 측정 되었으며, 이는 연산을 수행하지 않은 No-Operation 값과 큰 차이가 없다.

<표 1> 알고리즘 수행시간 실험 결과 (단위 :sec)

Integer	Loop	250000	500000	···	2000000	3000000
	Java	0.0360	0.0417	···	0.09002	0.12640
	Native C	0.0013	0.0023	···	0.00931	0.01392
Floating point	Loop	250000	500000	···	2000000	3000000
	Java	0.0899	0.1037	···	0.39018	0.47865
	Native C	0.0534	0.0757	···	0.33535	0.42571
Memory access	Loop	2500	5000	···	20000	30000
	Java	0.512	1.8030	···	29.2370	69.0434
	Native C	0.037	0.1075	···	1.50429	3.48432
String	Loop	25	50	···	200	300
	Java	0.0073	0.0186	···	0.05421	0.06811
	Native C	0.5494	1.2043	···	4.42658	6.53887

실험은 삼성스마트폰 갤럭시 S1에서 수행했으며, 사양은 S5PC111 1GHz, RAM 512M이고, GPU는 PowerVR, Android Version 2.3이다. 표1은 알고리즘 수행시간 실험 결과를 나타내며, 표2는 CPU점유율 실험 결과이다.

<표 2> CPU점유율 실험 결과

Integer	Loop	1000000	2000000	···	4000000	5000000
	Java	45.5%	61.4%	···	82.2%	98.3%
	Native C	20.4%	24.3%	···	39.2%	45.3%
Floating point	Loop	100000	500000	···	5000000	10000000
	Java	14.4%	20.2%	···	69.3%	96.2%
	Native C	10.9%	13.9%	···	38.4%	68.5%
Memory access	Loop	20	21	···	23	24
	Java	19.4%	27.5%	···	57.9%	68.9%
	Native C	15.8%	16.2%	···	31.7%	40.2%
String	Loop	80	90	···	110	120
	Java	12.4%	12.9%	···	14.1%	16.2%
	Native C	63.7%	76.5%	···	77.2%	79.4%

위의 두 실험결과에서 볼 수 있듯이 정수형 타입이 부동소수점 타입에 비해 Java 및 Native 언어로 설계한 프로그램 모두에서 더 우수한 성능이 나타난 것을 볼 수 있다. 또한 Java 보다는 Native 언어로 설계한 프로그램의 성능이 우수한 것을 볼 수 있다. 이는 Native 및 NDK를 이용해 설계한 애플리케이션이 Package(.apk) 파일로 압축될 때 Compile 된 상태로 압축되고 실행되기 때문이다. 특히, Native언어로 설계한 알고리즘은 메모리 접근에 있어서 가장 우수하다. 가장 큰 이유로는 Native 언어로 설계한 알고리즘은 Java로 설계한 알고리즘에 비해 Dalvik 가상머신에서 제공하는 메모리보다 더 많은 메모리를 사용할 수 있고, Native언어로 설계된 알고리즘은 Dalvik 가상머신에서 수행되는 Garbage Collection의 영향도 받지 않기 때문이다. 마지막으로, String 연산에 있어서는 정반대의 결과가 나왔다. 이는 Java, C/C++, JNI 각기 다른 문자열 처리 방식(Unicode, KSC5601, UTF-8)에 있다. 따라서 Native Method로 인자 값을 전달할 때마다 형 변환이 필요하며, 이는 많은 Overhead를 발생시킨다.[3]

5. 결론

본 논문에서는 NDK기반 효율적인 안드로이드 애플리케이션 설계방법에 관한 연구를 수행했다. 결과적으로 정수형과 부동소수점, 메모리접근의 속도 및 CPU 점유율 성능이 SDK기반 Java로 구현한 것보다 NDK기반 Native C 언어로 설계한 프로그램이 더 우수했다. 특히 메모리접근 연산에 있어서는 Native C가 월등히 우수한 성능을 보였다. 반면, 문자열처리 연산에서는 Java언어로 설계한 프로그램이 훨씬 우수한 성능을 보였다. 애플리케이션을 설계할 때 각 상황에 맞게 NDK 및 Native C를 이용하면 보다 효율적인 설계에 도움이 될 것이라 생각된다.

ACKNOWLEDGMENT

본 연구는 지식경제부 및 정보통신산업진흥원의 정보통신기반구축사업 [B1120-0901-0002, IT특화연구소설립] 사업의 일환으로 수행하였음

참고문헌

[1] Android(Operating System) http://en.wikipedia.org/wiki/Android_(operating_system)

[2] Sheng Liang. The Java Native Interface: Programmer's Guide and Specification, Addison-Wesley, 1999.

[3] Dawid Kurzyniec and Vaidy Sunderam, Efficient Cooperation between Java and Native Codes – JNI Performance Benchmark, Emory University Dept. of Math and Computer science.