모바일 GPU를 활용한 플로우 기반 영상 추상화 기법의 가속

전세원, 김진우, 한탁돈
연세대학교 컴퓨터과학과
e-mail : hantack @msl.yonsei.ac.kr

Accelerating Flow based Image Abstraction by using mobile GPU

Se-Weon Jeon, Jin-Woo Kim, Tack-Don Han
Dept. of Computer Science, Yonsei University

요 약

본 논문에서는 스마트폰에 탑재되는 모바일 GPU 를 활용하여 만화 형식의 영상을 생성하는 과정을 가속하는 방법을 제시하였다. 또한 모바일 GPU 에 적합한 벡터 데이터 타입과 벡터 명령어의 사용 및 워크 그룹 크기에 의한 영향을 고려한 최적화를 적용하였다. 제안하는 모바일 GPU 가속 기법의 검증을 위해 OpenCL API 를 이용하여 구현하였다 실험 결과를 통해 제안하는 기법이 모바일 CPU 기반의 처리 방법 보다 800% 이상의 성능 향상을 있음을 확인하였다.

1. 서론

모바일 기술이 발전함에 따라 상용화된 스마트폰에 탑재되는 카메라, CPU, 메모리 등 하드웨어의 성능 또한 향상되었고 이렇게 생산된 제품들의 보급으로 인하여 사용자들의 모바일 기술에 대한 관심도 증가하였다. 스마트폰에 고성능의 카메라가 장착되고 사용자들은 그 카메라를 이용하여 단순히 촬영할 뿐만 아니라 촬영된 영상을 불에 탄 모습처럼 보이게 하고, 흑백처리를 하는 등 영상에 변화를 주는 효과들을 표현하길 원하였다. 또한 사물의 경계를 추출해내거나 영상에 포함되어 있는 사물 중에서 특정 물체를 인식하는 등의 특수 목적을 충족시키는 기능을 바라고 있다. 이런 사용자의 요구를 반영하여 영상에 각종 효과를 주기 위한 다양한 어플리케이션들이 출시되었다.

하지만 고성능의 카메라로 촬영된 고해상도의 영상 처리를 수행할 때는 픽셀수가 늘어남에 따라 요구되는 연산량이 급격하게 증가하여 모바일 CPU 만으로 처리하는 것은 무리가 있다. 지난 수년간 데스크톱이나 워크스테이션 환경에서는 이와 같이 방대한 연산량을 요구하는 작업을 효율적으로 처리하기 위해서 영상 처리뿐만 아니라 다른 분야에 CPU 대신 GPU 를 활용하면서 하드웨어 특성에 맞게 최적화시키는 연구가 이루어졌다[1-4]. SIMD 방식으로 처리하거나 워크그룹의 크기를 조절하여 GPU 에 탑재된 코어들의 사용률을 향상시키고, CPU 와 GPU 간의 데이터 이동을 최소화시키면서 GPU 에서 사용이 가능한 각기 다른 메모리들의 속도와 크기를 고려하여 그 특성에 맞게 활용하는 방법 등이 제시되었다.

최근 출시되고 있는 스마트폰에 탑재되는 모바일 GPU 의 성능이 향상되고 있고 이를 지원하는 CUDA, OpenCL 등의 API 도 발전이 이루어지고 있다. 이렇게 발전하고 있는 모바일 GPU 를 활용하여 기존 모바일 CPU 만으로 처리하던 작업들을 병렬화, 가속화하고 하드웨어 특성에 맞게 최적화하는 연구가 이루어지고 있다[5, 6].

본 논문에서는 이러한 연구의 일환으로 모바일 GPU 를 활용하여 스마트폰에 장착된 카메라로부터 실시간으로 입력되는 영상을 추상화시키는 기법을 병렬화 및 가속화하는 방법을 제안한다. 여기에는 OpenCL 을 사용한 플로우 기반 영상 추상화 기법의 구현 및 OpenCL 이 지원하는 내장 벡터 데이터 타입과 벡터 명령어들을 이용한 SIMD(Single Instruction Multiple Data: 단일 명령어 복수 데이터) 연산의 최적화와 캐시 메모리의 영향을 고려한 워크 그룹 크기 최적화가 포함된다.

2. 관련 연구

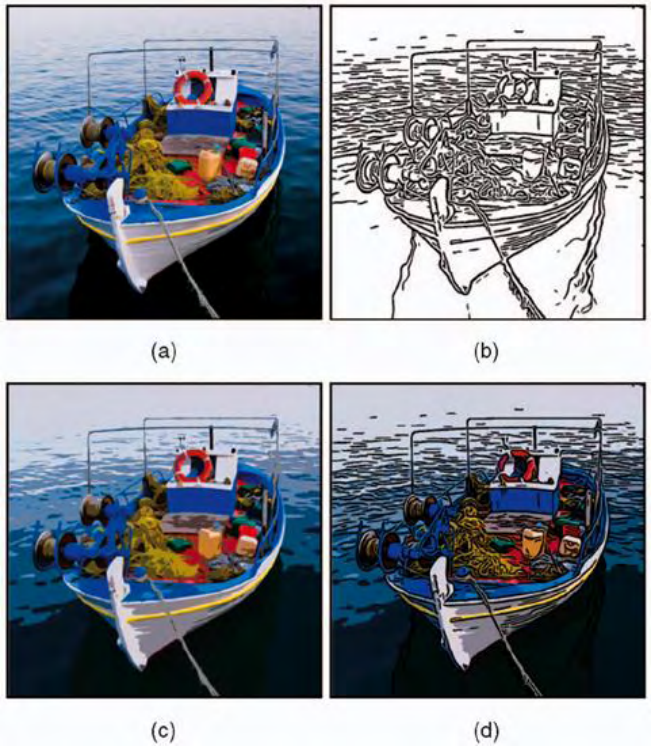
2.1 영상 기반의 비사실적 렌더링 기법

영상 처리 중에서 영상 기반의 NPR(Non – Photorealistic Rendering: 비사실적 렌더링)[7]은 입력된 영상의 픽셀 정보만을 가지고 필터링하여 비사실적인 영상을 나타내는 렌더링 기법이다.

Kang[8]은 영상 기반의 NPR 중 하나인 영상 추상화[9] 기법의 시각적 품질을 향상 시키기 위해 ETF(Edge Tangent Flow: 엣지 탄젠트 플로우)를 필터에 적용한 FBIA(Flow Based Image Abstraction: 플로우 기반 영상 추상화 기법) 제안하였다. Kang 은 영상에서 인접한 픽셀간의 두드러진 밝기 차이에 의해 만들어진 ETF 를 이용함으로써 기존의 DoG Filter(Difference

of Gaussian Filter: 가우시안 차 필터) [10]와 양방향 필터[11]를 확장하여 FDoG Filter(Flow-based Difference of Gaussian Filter: 플로우 기반 가우시안 차 필터) 와 FBL(Flow-based Bilateral Filter: 플로우 기반 양방향 필터)를 제안하였다.

FBIA 는 NPR 영상을 생성하기 위해 FBL 과 FDoG 필터링 과정을 거치게 된다. 각각의 필터의 역할을 살펴보면 FBL 필터는 이웃 픽셀간의 색상 차이를 줄여서 단순화시키고 FDoG 필터는 색상 차이가 두드러지는 픽셀을 강조하여 점과 선으로 나타낸다. 이렇게 각각 처리된 영상을 병합하여 추상화된 결과 영상을 생성하는데 그 모습은 그림 1 에 나타나 있다.



(그림 1) FBIA 의 필터에 의한 결과[8]
(a)입력 영상 (b) FDoG 필터링 결과
(c) FBL 필터링 결과 (d) 병합된 영상

Kang 이 제시한 방법은 고품질의 결과를 생성하지만 많은 연산이 요구되는 문제점이 있다. 이를 해결하기 위해 Kang 은 또한 ETF 생성 과정에서 생성한 flow 의 방향을 기준으로 삼아 이에 수직하거나 나란히 위치한 픽셀들만 계산하는 분리형 커널[12]을 FBL 과 FDoG 에 사용하여 시간 복잡도를 $O(n^2)$에서 $O(2n)$으로 감소시켰다.

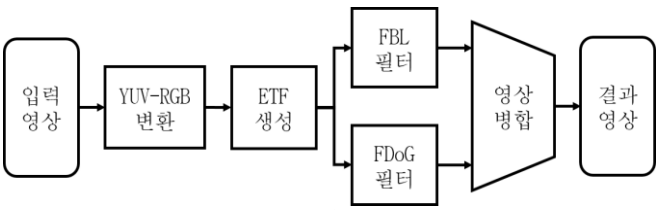
2.2 모바일 환경에서의 OpenCL

모바일 GPU 지원을 위한 API 에는 CUDA, OpenCL 등이 있다. CUDA 는 Nvidia 의 제품에만 사용이 가능하지만 OpenCL 은 칩셋 제조사에 관계 없이 대부분의 모바일 GPU 에서 사용이 가능하다. OpenCL 에는 풀 프로파일과 임베디드 프로파일이 있으며 각각의 모바일 GPU 칩셋마다 따라 지원하는 확장 기능이 다르다[2, 13]. 임베디드 프로파일은 데이터 타입이나 메모리 크기 제한 등 몇 가지 제약사항이 존재하지만 확장 기능을 통해 그 제약 사항을 상쇄할 수 있다.

모바일 환경에서 OpenCL 을 사용하여 GPU 를 GPGPU(General-Purpose computation on Graphics Processing Units) 로서 활용이 가능해짐에 따라 이를 이용한 연구 사례가 나타나고 있다. Wang[5]은 모바일 GPU 에서 영상 내에 존재하는 특정 사물을 제거하는 알고리즘을 가속화하고 메모리 구조에 맞게 데이터의 크기를 최적화하였다. Gleeson [6]은 모바일 GPU 를 활용하여 AES 암호화 과정을 가속화하였다. 또한 calcHist 를 비롯한 영상 처리 알고리즘들을 병렬로 처리하는 라이브러리를 개발한 연구 사례가 있다[14].

3. 제안하는 구조

최근의 모바일 GPU 는 일반적으로 복수의 계산 유닛들로 구성되어 있고 각각의 계산 유닛은 다수의 코어들을 포함하고 있다. 복수의 코어들은 동시에 동일한 코드를 실행하기 때문에 병렬처리에 적합하다[5, 15]. FBIA 의 각 과정에서 모든 픽셀에 대하여 동일한 연산을 수행하므로 모바일 GPU 를 활용하면 성능 향상을 기대할 수 있다. 본 논문에서는 OpenCL 을 사용하여 그림 2 에 나타난 것처럼 FBIA 를 모바일 GPU 에서 처리하도록 하였으며 각각의 코어들은 영상을 이루는 각각의 픽셀에 대한 연산을 수행한다. 카메라로부터 얻어진 YUV 컬러모델의 데이터를 RGB 컬러모델의 데이터로 변환하고 FBIA 의 각 필터를 거쳐서 결과 영상을 생성한다. 뿐만 아니라 OpenCL 에서 지원하는 벡터 데이터 타입과 벡터 명령어들을 사용하여 SIMD 방식으로 처리하고 워크 그룹 크기를 최적화하여 추가적인 성능 향상을 이끌어 내도록 하였다.



(그림 2) 모바일 GPU 에서의 FBIA 처리 과정

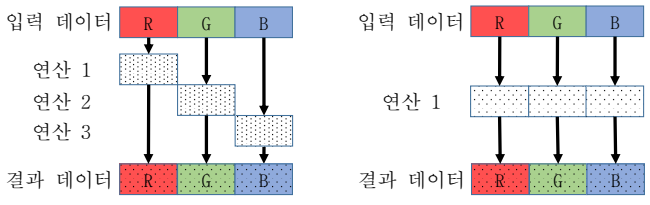
3.1 벡터명령어를 사용한 SIMD 연산 최적화

SIMD 는 서로 다른 데이터에 동일한 연산을 수행할 때 동일한 하나의 명령어를 여러 번 반복 호출하는 스칼라 방식과 다르게 서로 다른 데이터에 대해 하나의 명령어를 한번만 호출하여 처리하는 방식이다. SIMD 방식 지원을 위해 벡터 데이터 타입과 명령어들이 별도로 존재하며 CPU 에서 사용되는 Intel 의 MMX 와 SSE[16], ARM 의 NEON[17]등을 예로 들 수 있다. GPU 에서도 CUDA 와 OpenCL[4, 13] 모두 벡터 데이터 타입과 명령어들을 지원한다. SIMD 방식으로 처리할 경우 명령어 호출 및 연산 횟수가 줄어들기 때문에 처리 속도가 향상된다.

FBIA 의 FBL 필터는 입력된 영상의 각 픽셀을 이루는 RGB 채널 데이터에 대해서 이웃 픽셀과의 유사

도, 거리 등에 따라 가중치를 곱하거나 더하는 등의 동일한 연산을 수행한다. 그리고 ETF 생성과 FBL 및 FDoG 필터에서 중심 픽셀과 그 주변에 위치해 있는 비교 대상이 될 픽셀의 위치를 계산할 때에 해당 픽셀들이 X, Y 축 상에서 어느 지점에 있는지 나타내는 X, Y 의 값에 대해서도 서로 동일한 연산을 수행한다.

그림 3 에 나타난 것처럼 FBL 의 RGB 각각의 채널에 연산을 수행할 때 스칼라 방식의 연산량이 3n 이라고 하면 OpenCL 이 지원하는 SIMD 방식으로 처리할 경우에는 n 으로 감소시킬 수 있다. 마찬가지로 X, Y 위치를 계산하는 과정에서 스칼라 방식의 연산량이 2n 이라고 하면 SIMD 방식으로 처리할 경우에는 그 수를 n 으로 감소시킬 수 있다. 이렇게 감소한 연산량 만큼 FBIA 의 전체 연산량도 감소하여 처리 속도를 증가시킬 수 있다.



(a) 스칼라 방식 (b) SIMD 방식

(그림 3) 스칼라 방식과 SIMD 방식 차이

3.2 워크 그룹 최적화

GPU 는 다수의 코어들로 구성되어 있으며 다수의 코어는 동일한 코드를 동시에 실행한다. 동시에 동일한 코드를 실행하는 단위를 워크 그룹이라 부르고 워크 그룹의 크기에 따라서 GPU 사용률이 달라진다[1]. GPU 에서의 병렬성을 향상시키기 위해 워크그룹의 크기를 최대로 해야 하지만 워크그룹의 크기를 무한히 증가시킨다고 하여 성능도 그에 따라 지속적으로 향상되지는 않고 코어 수, 레지스터 그리고 로컬 메모리 사용 등에 영향을 받으므로 한계가 존재한다[3]. 뿐만 아니라 워크그룹의 크기가 같더라도 워크그룹을 이루는 X, Y, Z 축의 각 크기와 알고리즘에 따라 메모리 접근 양상이 달라져 성능 차이가 발생한다.

FDoG 과 FBL 을 비롯한 FBIA 의 각 필터들은 2 차원 영상의 픽셀 값에 연산을 하여 결과를 얻어내므로 처리 대상픽셀과 비교 대상이 되는 인접한 픽셀들의 값이 저장된 메모리에 접근해야 한다. 모바일 GPU 에도 캐시 메모리가 존재하여 연속된 메모리에 접근할수록 캐시 효율이 향상되는데 워크그룹을 이루는 X, Y 크기에 따라 메모리 접근 방식이 달라지므로 성능 차이가 발생하게 된다.

4. 실험 결과

본 논문에서 언급한 방법들의 성능 비교를 위한 실험에 사용된 기기는 삼성전자의 갤럭시 S5(G900)[18] 이다. 갤럭시 S5 에 탑재된 CPU 는 Krait 400 MP4 이고 GPU 는 Adreno 330 이며 이들을 지원하는 API 는 OpenCL 1.1 임베디드 프로파일[13]이다. 속도 비교를 위하여 각각의 테스트 프로그램을 구현하여 실험하였다. 정확한 성능 비교를 위하여 카메라로부터 실시간으로 얻어진 프레임 데이터 대신 NV21 포맷의 영상 파일을 처리하도록 하였다. NV21 포맷은 안드로이드 OS 가 탑재된 스마트폰의 기본 영상 포맷과 동일하다. 모든 실험에는 1024×1024(X×Y) 크기의 영상을 사용하였고 각 ETF 생성 및 FBL, FDoG 필터의 반복 횟수는 1 회로 하였다.

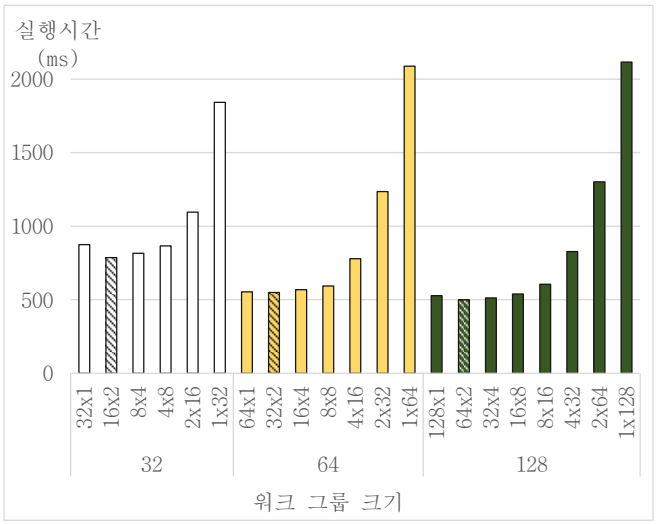
CPU 와 GPU 에서의 FBIA 처리 속도 비교를 위해 CPU 에서 동작하는 프로그램은 4 개의 스레드를 사용한 멀티 스레딩 방식으로 C 코드로 구현하였으며 GPU 에서 동작하는 프로그램은 OpenCL 커널로 구현하되 SIMD 연산 최적화 여부에 따라 데이터 타입과 명령어를 달리 사용하였다. 이때 워크 그룹의 크기는 64×2(X×Y)로 동일하게 지정하였다.

표 1 에 모바일 CPU 와 GPU 에서의 FBIA 실행 시간 및 CPU 대비 속도향상이 나타나 있고 SIMD 연산 최적화의 영향을 보이고 있다. SIMD 연산 최적화가 적용되지 않은 경우는 GPU 를 이용하여도 CPU 를 이용한 경우에 비해 속도 향상이 미미하지만 SIMD 연산 최적화가 적용된 경우는 모바일 CPU 이용 대비 800% 이상의 속도 향상을 보였다. 따라서 모바일 GPU 에서의 성능 향상을 위해 SIMD 연산 최적화는 필수적이라고 볼 수 있다.

<표 1 >CPU 와 GPU 에서의 FBIA 실행 시간

프로세서	SIMD 연산 최적화	실행시간(ms)	속도향상(%)
CPU		4385	100%
GPU	X	3542	124%
	○	499	879%

워크 그룹 크기에 의한 영향을 확인하기 위해서 SIMD 연산 최적화를 적용하고 워크 그룹의 크기(X×Y)와 워크그룹을 이루는 X, Y 각각의 크기를 조절하며 실험한 결과는 그림 4 에 나타나 있다.



(그림 4) 워크 그룹 크기에 따른 실행 시간 변화

워크 그룹 크기가 32 일 때 실행시간이 가장 길었

고 64, 128 로 증가시킴에 따라 실행시간이 감소하였으나 워크 그룹 크기가 64 에서 128 로 증가할 때 실행 시간 감소는 워크 그룹 크기가 32 에서 64 로 증가할 때에 비하여 크지 않음을 알 수 있다. 워크그룹의 크기를 결정하는 X 와 Y 의 영향을 보면 Y 가 커지면 실행 시간이 급격하게 증가함을 알 수 있다. 그 이유는 영상이 메모리에 저장될 때 X 축 방향으로 인접한 픽셀들이 메모리에서도 인접하여 저장되는데 워크 그룹의 크기를 결정할 때 Y 의 크기를 크게 하면 연속된 메모리 공간에 접근하지 않고 건너 뛰어가며 접근하므로 캐시 메모리 효율이 하락하기 때문이다.

하지만 FBIA 의 특성상 X 축 방향으로 인접한 픽셀뿐만 아니라 Y 축 방향으로 인접한 픽셀들까지 비교하여 연산하는데 X 의 크기를 최대로 하여 1 차원으로 워크 그룹을 구성하면 Y 축 방향으로 인접한 픽셀의 정보가 필요할 때에는 캐시 미스가 발생하기 때문에 그림 4 에서 빗금으로 표시된 부분에 나타난 것처럼 워크 그룹의 크기가 동일한 경우 X 의 크기가 최대일 때가 아닌 Y 의 크기가 2 일 때 가장 짧은 실행시간을 보이고 있다. 따라서 모바일 GPU 활용을 위해 워크 그룹을 이루는 X 와 Y 의 크기를 정할 때에는 사용되는 알고리즘의 특성을 바탕으로 캐시 메모리 효율을 고려해야 함을 알 수 있다. 워크 그룹 크기가 256 이상일 때는 모바일 GPU 와 API 에서 지원하는 최대 크기를 초과하여 실험 결과를 확인 할 수 없었다.

5. 결론

본 논문에서는 모바일 GPU 를 활용하여 기존의 영상 처리 기법 중 하나인 FBIA 를 병렬화 및 가속화하였고 SIMD 연산과 워크 그룹의 최적화를 통해 모바일 CPU 대비 800%이상의 성능 향상을 이루었다. 본 논문에서는 모바일 GPU 에 탑재된 로컬 메모리 활용을 통한 최적화는 다루지 않았다. 향후 연구로는 기존 데스크탑이나 워크스테이션에서 이루어진 연구 사례들을 바탕으로 로컬 메모리를 활용하여 추가적인 성능향상을 이끌어 낼 계획이다.

참고문헌

[1] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *Computing in science & engineering*, vol. 12, p. 66, 2010.

[2] B. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa, *Heterogeneous Computing with OpenCL: Revised OpenCL 1*: Newnes, 2012.

[3] K. Komatsu, K. Sato, Y. Arai, K. Koyama, H. Takizawa, and H. Kobayashi, "Evaluating performance and portability of OpenCL programs," in *The fifth international workshop on automatic performance tuning*, 2010, p. 7.

[4] L. Nyland, M. Harris, and J. Prins, "Fast n-body simulation with cuda," *GPU gems*, vol. 3, pp. 677-696, 2007.

[5] G. Wang, Y. Xiong, J. Yun, and J. R. Cavallaro, "Accelerating computer vision algorithms using OpenCL framework on the mobile GPU-a case study," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, 2013, pp. 2629-2633.

[6] J. Gleeson, S. Rajan, and V. Saini, "GPU Encrypt: AES Encryption on Mobile Devices," 2014.

[7] A. Santella and D. DeCarlo, "Visual interest and NPR: an evaluation and manifesto," in *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, 2004, pp. 71-150.

[8] H. Kang, S. Lee, and C. K. Chui, "Flow-based image abstraction," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 15, pp. 62-76, 2009.

[9] H. Winnemöller, S. C. Olsen, and B. Gooch, "Real-time video abstraction," *ACM Transactions On Graphics (TOG)*, vol. 25, pp. 1221-1226, 2006.

[10] D. Marr and E. Hildreth, "Theory of edge detection," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 207, pp. 187-217, 1980.

[11] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Computer Vision, 1998. Sixth International Conference on*, 1998, pp. 839-846.

[12] T. Q. Pham and L. J. Van Vliet, "Separable bilateral filtering for fast video preprocessing," in *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on*, 2005, p. 4 pp.

[13] T. K. Group. (2011). *The OpenCL Specification*. Available: http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf

[14] 이종환, 강승헌, 이만희, 이성철, 김학일, and 박인규, "모바일 GPU 를 이용한 실시간 병렬 영상처리 라이브러리," *정보과학회 컴퓨팅의 실제 논문지*, vol. 20, pp. 96-100, 2 2014.

[15] ARM. (2013). *ARM Mali-T600 Series GPU OpenCL Developer Guide*. Available: http://infocenter.arm.com/help/topic/com.arm.doc.dui0538f/DUI0538F_mali_t600_opencl_dg.pdf

[16] S. K. Raman, V. Pentkovski, and J. Keshava, "Implementing streaming SIMD extensions on the Pentium III processor," *IEEE micro*, vol. 20, pp. 47-57, 2000.

[17] M. Jang, K. Kim, and K. Kim, "The performance analysis of arm neon technology for mobile platforms," in *Proceedings of the 2011 ACM Symposium on Research in Applied Computation*, 2011, pp. 104-106.

[18] Qualcomm. (2014). *Samsung GALAXY S5 | Snapdragon Powered Smartphones | Snapdragon*. Available: http://www.qualcomm.com/snapdragon/smartphones/samsung-galaxy-s5