ISSN 1975-8359(Print) / ISSN 2287-4364(Online)
The Transactions of the Korean Institute of Electrical Engineers Vol. 65, No. 4, pp. 666~672, 2016
http://dx.doi.org/10.5370/KIEE.2016.65.4.666

OpenCL을 이용한 GPGPU 기반 지문개선 알고리즘 가속화

Accelerating Fingerprint Enhancement Algorithm on GPGPU using OpenCL

김 대 희* · 박 능 수†
(Daehee Kim · Neungsoo Park)

Abstract - Recently the fingerprint is widely used as one of biometrics to improve the security of financial mobile applications, because of its user convenience and high recognition rate. However, in order to apply fingerprint algorithms to finance and security applications, the recognition rate and processing speed of the fingerprint algorithms have to be improved further. In this paper, we propose the parallel fingerprint enhancement algorithm on general-purpose computing on graphics processing unit (GPGPU) using OpenCL. We discuss the analysis of the parallelism in the fingerprint algorithm as well as the exploration of optimization parameters of the parallel fingerprint algorithm to improve the performance. The experimental results showed that the execution of parallel fingerprint enhancement algorithm on GPGPUs was accelerated from 29.4 upto 69.2 times compared with the execution of the original one on the host CPUs.

Key Words : Fingerprint, GPGPU, OpenCL

1. 서 론

최근 스마트 폰 등 모바일 디바이스를 활용한 금융 거래가 활성화되면서 보안 향상을 위해 생체 인식 기술에 대한 관심이 증대하고 있다. 다양한 생체인식 기술 중에서도 지문 인식은 사용하기에 쉽고 간편하여 많은 응용에서 활용되고 있다. 하지만 여전히 금융 거래와 보안 등의 중요한 응용을 위하여 인식률과 속도에 대한 지속적인 개선을 시도하고 있다. 따라서 지문 인식률을 높이기 위하여 지문 영상 개선하는 것은 지문 영상 처리에 중요한 과정이다[1]. 특히 지문 영상 처리 속도를 향상시키기 위하여 FPGA[2]와 GPGPU(General-Purpose computing on Graphic Processor Units)[3]을 활용한 다양한 연구가 진행되고 있다[4, 5]. 본 논문은 미국 NIST에서 배포한 PCASYS 지문 영상 개선 알고리즘을 GPGPU를 이용하여 성능을 개선하고자 한다.

본 논문의 GPGPU 병렬 프로그램 모델은 OpenCL(Open Computing Language)을 기반으로 한다. OpenCL은 개방형 범용 컴퓨팅 프레임워크로 GPU, DSP 등 다양한 하드웨어에서 프로그램을 실행 가능하게 해준다. 본 연구에서는 지문 영상 개선을 위한 OpenCL 기반 GPGPU 병렬 프로그램의 성능을 최적화하기 위하여 병렬 지문 개선 알고리즘을 분석하고 GPGPU의 구조와 OpenCL의 환경을 고려하여 이를 최적화하고자 한다. 또한 실험을 통하여 제안한 OpenCL 기반 GPGPU 병렬 프로그램이 AMD 시스템에서 약 69.2배, Tesla 시스템에서 약 36.8배, 그리고 GeForce 시스템에서 약 29.4배로 성능 가속화되었음을 확인하였다.

본 논문의 구성은 2장에서는 PCASYS 지문 개선 알고리즘에 대해 소개하고, 3장에서는 GPGPU를 이용해 지문 개선 알고리즘의 병렬화 및 성능 최적화에 대하여 설명한다. 4장에서는 실험결과를 분석하고 마지막 5장에서 결론과 함께 향후 연구를 소개로 마친다.

† Corresponding Author : Dept. of Computer Science and Engineering, Konkuk University, Korea
E-mail : neungsoo@konkuk.ac.kr
* Dept. of Computer Science and Engineering, Konkuk University, Korea
Received : February 19, 2016; Accepted : March 5, 2016

2. PCASYS 지문 개선 알고리즘

PCASYS는 미국 NIST(National Institute of Standards and Technology)에서 배포한 지문 분류 알고리즘[6, 7]으로 본 연구에서는 PCASYS 내의 지문 개선 알고리즘을 병렬화하여 성능을 향상시키고자 한다. PCASYS의 지문 개선 알고리즘의 순서는 다음 그림 1과 같다.

PCASYS는 512×480 크기의 8비트 그레이스케일 지문 영상을 입력 받아 처리하도록 되어 있다. 현재 지문 입력 장비들은 다양한 크기의 영상을 생성할 수 있다. 따라서 PCASYS는 이러한 다양한 크기의 영상을 입력 받아 세그멘테이션(segmentation) 과정을 거쳐 512×480 크기의 지문 영상만을 추출하여 사용한다. 본 연구에서는 세그멘테이션 과정을 생략하고 세그멘테이션 된 지문 영상을 이용한다.

화질 개선 알고리즘의 경우 작은 크기의 타일로 나누어 좌상의 위치에서부터 순차적으로 지역 화질 개선을 하고 개선된 타일 영상을 이어 붙여서 생성한다. PCASYS의 경우 지문 융선 방향

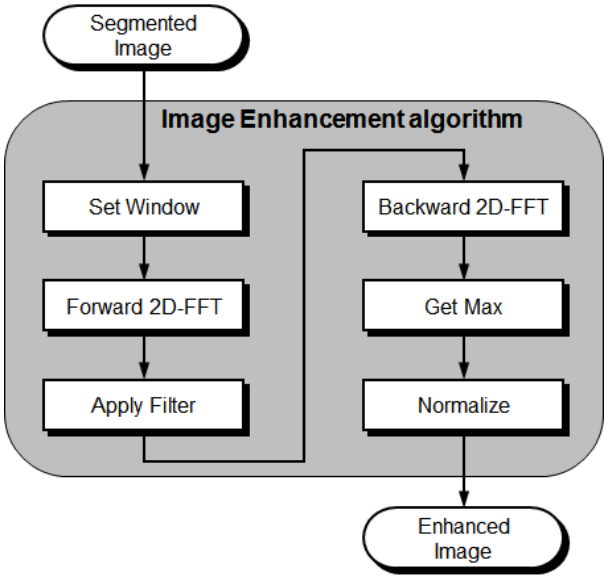


그림 1 PCASYS 지문 개선 알고리즘
Fig. 1 PCASYS fingerprint image enhancement algorithm

(fingerprint ridge direction) 알고리즘이 16×16 픽셀을 이용하므로 이와 맞추어 지문 개선 알고리즘도 16×16 픽셀을 이용하도록 구성하였다. 이는 지문 융선 방향 알고리즘이 융선 흐름 평균(ridge flow average) 계산을 할 때 16×16 타일 영상의 경계에서 발생하는 인공적인 개선의 영향을 최소화하기 위해서다. 타일 경계에서 생기는 인공적인 개선 영향은 최종 방향 어레이의 잘못된 융선 흐름점들을 만들 수 있기 때문이다.

PCASYS 지문 개선 알고리즘은 세그멘테이션 된 지문 영상의 좌우상하 테두리 8 픽셀은 건너 띄고 내부 지문 영상의 화질 향상에 집중하도록 구성이 되어 있다.

입력된 영상은 32×32 크기의 윈도우로 분할되어 지문 개선처리가 되며 16×16 개선된 영상 타일을 생성한다. 따라서 입력된 지문 영상에 대하여 그림 2와 같이 각 테두리의 8 픽셀을 제외하고 32×32 크기의 윈도우가 16 픽셀씩 슬라이딩 하며 중첩되어 입력이 되고 16×16 개선된 영상 타일을 순차적으로 생성한다. 개선된 영상 타일들을 순차적으로 이어 붙여서 개선된 지문 영상을 만들게 된다. 지문 개선 알고리즘에 필요한 중첩된 윈도우의 연산 횟수는 $[(512-16)/16-1]\times[(480-16)/16-1]=30\times28$가 된다.

분할된 윈도우들에 대하여 각각 Forward 2D-FFT 연산을 적용한다. 고속 푸리에 변환(Fast Fourier Transform) 알고리즘을 이용하면 공간 데이터로 이루어진 원본 영상을 주파수 데이터 영상으로 변환 된다. 이 때 다음 식 (1)을 이용하여 변환한다. 먼저 윈도우의 각 픽셀 값을 복소행렬 $A+iB$에 연결한다. 실제 픽셀 값들이 A에 대입되고 B는 0으로 대입된다. 그렇게 하여 복소행렬 $X+iY$를 도출한다.

$$X_{jk}+iY_{jk}=\sum_{m=0}^{31}\sum_{n=0}^{31}(A_{mn}+iB_{mn})\exp(\frac{-2\pi i}{32}(mj+nk)) \quad (1)$$

2D-FFT 결과에 대하여 특정 주파수대역을 필터링 하는 밴드패스 필터를 적용하는데 이때 고주파(449Hz 이상), 저주파(150Hz 이하)를 0으로 치환하고 나머지 통과 주파수 대역의 FFT 결과값은 식 (2)를 적용한다. 푸리에 변환 요소인 $X+Y$에 파워스펙트럼을 취한 것과 0.3을 제곱 한 것을 서로 곱함으로써 노이즈 감쇠 효과를 얻을 수 있다.

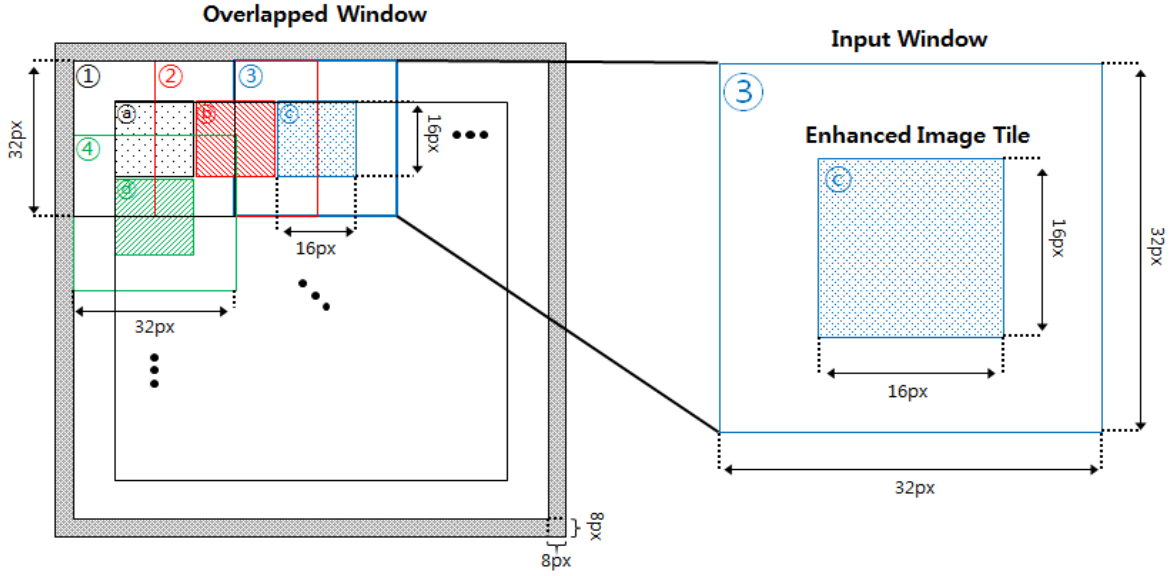


그림 2 중첩 윈도우 연산
Fig. 2 Overlapped window operations

$$U_{jk}+iV_{jk}=(X_{jk}^2+Y_{jk}^2)^{0.3}(X_{jk}+iY_{jk}) \quad (2)$$

다음 각각의 윈도우에 Backward 2D-FFT를 적용한다. 푸리에 변환 특성상 변환된 값을 역으로 다시 변환하면 원본 값을 도출해 낼 수 있다. 필터를 적용한 주파수 데이터를 원래의 이미지 데이터로 되돌리기 위해 다음 식 (3)을 참고하여 실행한다. 여기서 $U+iV$가 필터가 적용된 주파수 데이터의 복소행렬이고 $C+iD$가 역변환된 값이다.

$$C_{mn}+iD_{mn}=\sum_{j=0}^{31}\sum_{k=0}^{31}(U_{jk}+iV_{jk})\exp(\frac{2\pi i}{32}(jm+kn)) \quad (3)$$

Backward 2D-FFT가 처리된 후 각 윈도우내의 최대 절대 값을 찾아 다음 식 (4)를 이용하여 정규화 한다. P는 각 윈도우내의 픽셀 값을 의미하고 MAX는 각 윈도우내의 픽셀 값 중 가장 큰 값을 의미한다.

$$(P\times(127/MAX))+128.5 \quad (4)$$

마지막으로 각 윈도우의 가운데 16×16 픽셀만을 결과 이미지에 취합하며 중복되지 않게 연결하여 최종 이미지로 만든다. 이렇게 하여 최종 이미지의 명암(contrast)이 전체적으로 증가하여 이미지가 개선됨을 볼 수 있다.

3. GPGPU를 이용한 지문 개선 알고리즘 병렬화

본 연구는 PCASYS의 지문 개선 알고리즘을 GPGPU를 이용하여 성능을 개선하고자 한다. 특히 GPGPU를 활용 할 수 있는 프레임 워크 인 OpenCL[8]을 기반으로 병렬화하고자 한다. 지문 개선 알고리즘의 경우 입력과 출력 영상을 처리하는 부분을 제외하면 그림 2와 같이 윈도우 단위로 작업을 분할하여 진행을 하게 된다. 따라서 총 윈도우 처리 작업을 최대한 병렬화하여 성능 향상 시키고자 한다.

3.1 워크그룹 매핑 최적화

PCASYS 지문 개선 알고리즘은 세그멘테이션 된 지문 영상이 윈도우 단위로 분할되어 진행된다. 지문 영상이 중첩되어 각 작업 윈도우에 입력으로 전달 된다. 각각 윈도우 별로 계산이 되어 작은 개선된 타일이 생성 되고 이 타일 영상들을 이어 붙여서 개선된 지문 영상을 만들게 된다. 총 작업 윈도우의 개수는 512×480 입력 영상에서 각각 테두리의 8 픽셀을 제외한 영역에 윈도우를 중첩이 되도록 슬라이딩하므로 $[(512-16)/16-1]\times[(480-16)/16-1]=30\times28$가 된다. 각 작업 윈도우 간에는 비록 입력 영상이 서로 중복이 되어 사용되더라도 개선된 이미지 타일은 독립적으로 생성하기 때문에 개선된 타일 영상 간에는 상호 의존성이 없다. 따라서 그림 3처럼 하나의 윈도우를 하나의 워크그룹에 대응하여 병렬처리를 할 수가 있다.

OpenCL의 경우 GPGPU 내의 계산 유닛(compute unit)에 각각의 워크그룹을 할당하여 처리를 하게 된다. 기본적으로 그림 2에서와 같이 하나의 워크그룹에 하나의 작업 윈도우를 할당하고 이를 하나의 계산 유닛에서 처리할 수 있다. 경우에 따라서 하나의 워크그룹에 다수의 작업 윈도우를 할당 할 수가 있는데 이때 할당할 수 있는 최대 윈도우의 개수($AW_{\max}$)는 다음과 같이 구할 수 있다.

$$AW_{\max}=\frac{LM_{size}}{W_{size}} \quad (5)$$

여기서 LM_{size}는 GPGPU 계산 유닛의 로컬 메모리 사이즈를 의미하고 W_{size}는 윈도우의 사이즈 즉, 본 논문에서는 32×32 를 의미한다. 따라서 각 워크그룹에 할당되는 윈도우의 개수는 $AW_{\max}$를 초과할 수 없고 만일 윈도우 사이즈가 로컬메모리 보다 클 경우 윈도우를 분할하여 할당하여야 한다. 지문 개선 알고리즘의 경우 30×28 개의 윈도우 어레이를 계산 유닛에 할당을 하여야 하므로 생성되는 워크그룹의 총 개수(NWG)는 다음과 같이 구할 수 있다.

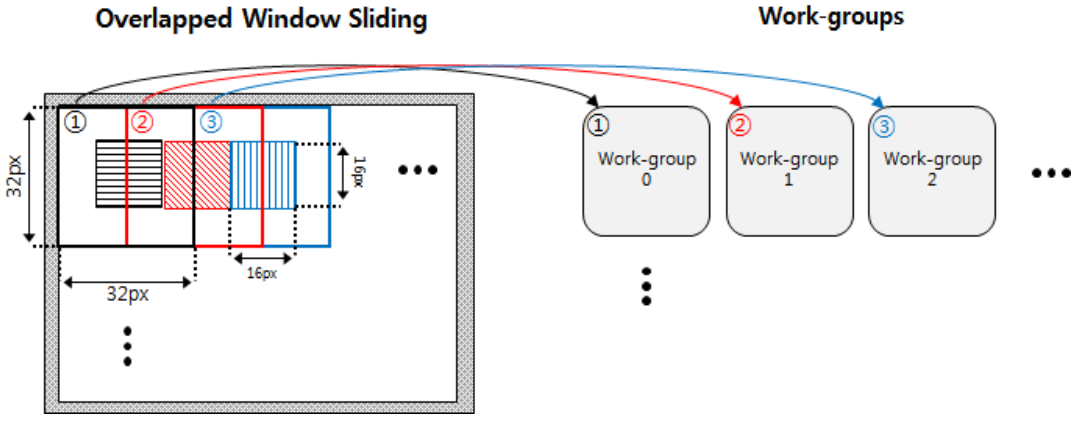


그림 3 워크그룹과 윈도우 매핑 예

Fig. 3 An example of mapping a window to a work-group

$$NWG = \left\lceil \frac{NCW}{AW} \right\rceil \times NRW \quad (6)$$

여기서 NCW는 윈도우 어레이의 열의 수(30), NRW는 윈도우 어레이의 행의 수(28), 그리고 AW는 하나의 워크그룹에 할당된 윈도우 수를 의미한다. 윈도우 어레이 열의 수가 할당된 윈도우의 배수가 아닐 경우 더미 데이터를 할당하여 처리하게 된다. 일반적으로 워크그룹의 총 개수(NWG)가 GPGPU의 총 계산 유닛 개수의 배수로 이루어질 때 성능이 최적화 될 수 있다.

3.2 워크아이템 할당 최적화

GPGPU가 OpenCL기반으로 최적화를 할 때 수용할 수 있는 최대 계산유닛을 이용하여 모든 자원이 유휴 되지 않게 워크 그룹을 매핑하고 각 계산 유닛에서 워크 아이템이 최대한 동작 할 수 있도록 워크아이템을 할당하여야 한다. GPGPU의 계산에서 하나의 워크아이템은 하나의 스레드에 할당이 되어 처리가 된다. 각 워크그룹에 할당되는 워크 아이템의 개수(Local Work Size: LWS)는 다음과 같다.

$$LWS = N_{thread} \times W_h \times AW \quad (7)$$

여기서 N_{thread}는 윈도우의 한 행 연산에 할당하는 스레드의 개수, W_h는 윈도우의 높이를 의미한다. 하나의 워크그룹에 할당할 수 있는 최대 스레드 개수($TWG_{\max}$)는 GPGPU의 특성에 따라 다르다. 워크그룹에 할당되는 워크아이템의 개수(LWS)는 $TWG_{\max}$ 보다 작아야 한다. 하나의 워크그룹은 하나의 계산 유닛에서 처리가 되므로 하나의 워크그룹에 있는 모든 워크아이템들을 효율적으로 처리하기 위하여 계산 유닛의 모든 스레드가 동시에 최대한 활용이 될 수 있도록 하여야 한다. 이를 위하여 LWS가 계산 유닛에 있는 프로세싱 엘리먼트의 개수($PECU_{\max}$)의 배수가 되도록 할당을 하여야 한다. 성능을 최적화하기 위하여 LWS는 다음 조건식 (8)에 따라 할당하여야 한다.

$$LWS \leq TWG_{\max} \quad \text{and} \quad LWS \propto PECU_{\max} \quad (8)$$

할당된 워크그룹의 개수(NWG)와 각 워크그룹에 할당되는 워크 아이템의 개수(LWS)를 이용하여 전체 워크 아이템의 개수(Global Work Size: GWS)를 구할 수 있다.

$$GWS = LWS \times NWG = LWS \times \left\lceil \frac{NCW}{AW} \right\rceil \times NRW \quad (9)$$

조건식 (8)을 고려한 식 (9)를 활용하여 OpenCL 디바이스에 맞게 워크 아이템을 할당함으로써 병렬처리의 성능을 최적화 시킬 수 있다.

3.3 메모리 접근 최적화

OpenCL에서는 호스트 프로세서인 CPU는 GPU 칩 외부에 있는 글로벌 메모리를 통해서 데이터를 주고받는다. 하지만 로컬 메모리 또는 프라이빗 메모리는 GPU 칩 내부에 있기 때문에 로컬 메모리에 접근 할 때 글로벌 메모리에 접근 할 때 보다 약 10배 이상 빠르다. 성능을 최적화하기 위해서는 워크 아이템이 칩 외부에 있는 글로벌 메모리의 접근을 최소화해야 한다. 하나의 윈도우를 하나의 워크 그룹에 할당하는 경우를 가정할 경우, 연산 할 때 마다 총 1024 개의 픽셀을 접근하여야 한다. 이때 $1024 \times N$번의 글로벌 메모리 접근이 필요하다. 여기서 N은 지문 개선 알고리즘에서 윈도우 단위에 연산에서 반복적으로 픽셀에 접근하는 횟수를 의미한다. 2D-FFT의 예처럼 총 $32 \times (32 \times \log_2 32)$의 연산이 필요하고 매 연산마다 2개 픽셀 읽기와 하나 픽셀 결과 쓰기를 수행하여야 한다. 따라서 글로벌 메모리의 접근 횟수를 줄이기 위하여 로컬 메모리에 작업 윈도우 데이터를 복사하고, 로컬 메모리에서 연산을 처리한 후에 최종 결과를 글로벌 메모리에 저장하여 글로벌 메모리 접근을 최소화하여 성능을 향상 시킬 수 있다.

4. 실험 및 결과

PCASYS 지문 개선 알고리즘의 GPGPU 기반 병렬 프로그램은 OpenCL을 이용하여 구현하였다. 표 1과 같은 다양한 환경에서 구현된 GPGPU 기반 병렬 프로그램의 성능을 평가하였다.

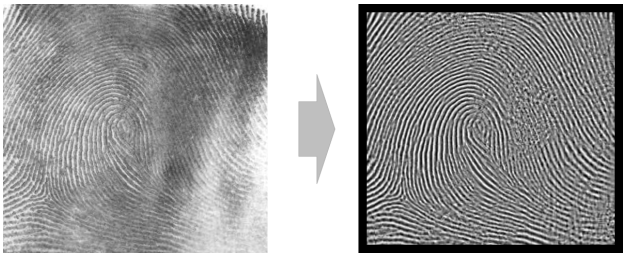


그림 4 개선된 지문 영상 예시
Fig. 4 A sample of enhanced fingerprint images

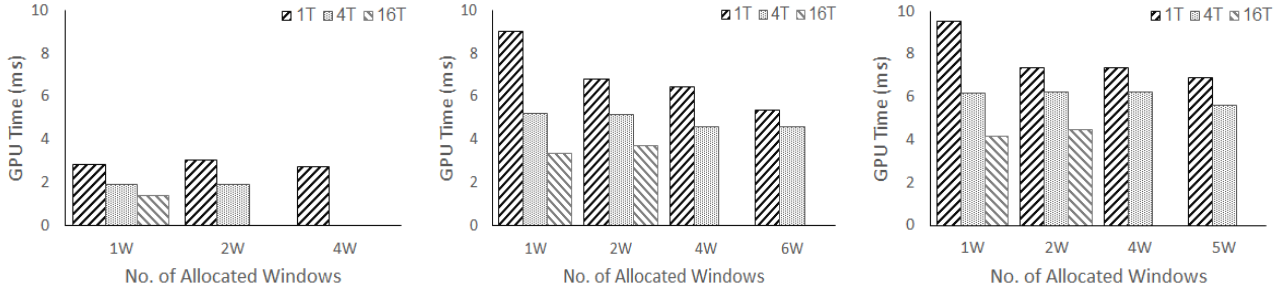
먼저 구현된 GPGPU기반 병렬 프로그램을 검증하기 위해 개선된 지문 영상을 기존의 CPU 기반 직렬 알고리즘의 개선 영상과 비교하여 확인하였다. 그림 4 병렬 프로그램으로 개선된 지문 영상의 한 사례를 보이고 있다.

GPGPU 기반 병렬 프로그램의 성능을 평가하기 위하여 지문 영상 2700개를 이용하여 각각 실행시간을 측정하고 이를 평균을 내어 평균 실행시간을 측정하였다. 그림 5는 GPGPU 기반 병렬 프로그램의 GPU 실행 시간을 비교한 결과이다. GPGPU 기반 병렬 프로그램의 성능 최적화를 알기 위하여 윈도위의 한 행에 할당할 수 있는 스레드의 개수(N_{thread})와 하나의 워크 그룹에 할당

표 1 실험 환경

Table 1 Experimental environments

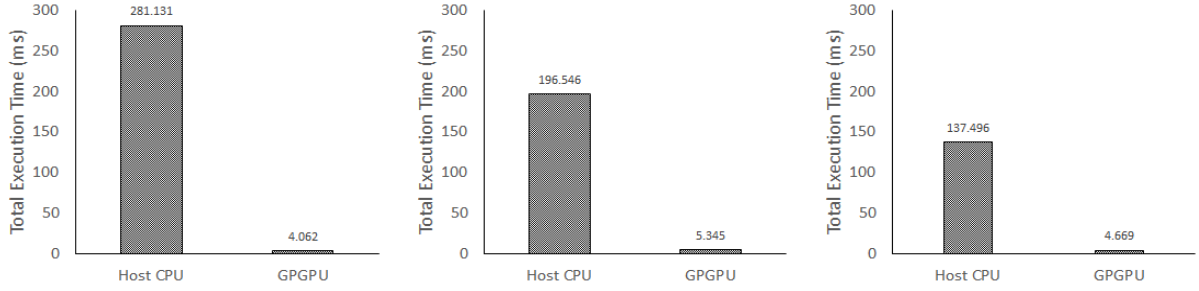
		AMD Hawaii	Tesla C2075	GeForce GTX 750
Host CPU		AMD FX-8370E 8-Cores 3.3GHz	Intel Core i7-975 Extreme 3.33GHz	Intel Core i5-4690 3.50GHz
GPU Clock		1.04GHz	1.15GHz	1.02GHz
OpenCL Version		OpenCL 2.0	OpenCL 1.1	OpenCL 1.1
GPU	Global Memory Size	4 GB	2 GB	1 GB
	Local Memory Size	32768 Bytes	49152 Bytes	49151 Bytes
	Number of PEs	2816	448	512
	Number of CUs	44	14	4



(a) AMD Hawaii (b) Tesla C2075 (c) GeForce GTX 750

그림 5 각 플랫폼 별 GPU 실행 시간

Fig. 5 The GPU Time of various GPU platforms



(a) AMD Hawaii (b) Tesla C2075 (c) GeForce GTX 750

그림 6 호스트와 GPU 간 총 실행시간 비교

Fig. 6 The comparison of the total execution time between a host and a GPU

할 수 있는 윈도우의 개수(AW)를 조절하여 그 성능을 평가하였다. 워크그룹에 할당되는 워크 아이템의 개수(Local Work Size: LWS)는 계산 성능 효율을 최대한 높이기 위하여 계산 유닛에 있는 프로세싱 엘리먼트의 개수($PECU_{\max}$)의 배수가 되도록 할당하였다. 각 GPU에서 AW를 1개, 2개, 4개, 6개와 같이 각 GPU의 로컬 메모리에 복사할 수 있는 최대 크기까지 할당하였으며, 또한 N_{thread}는 1, 4, 그리고 16개로 변화시켜서 할당하였다. 이때 한 윈도우의 높이가 32이므로 한 워크 그룹에 할당되는 스레드의 수는 각 32, 128, 512개가 할당을 할 수가 있다. 단 AMD Hawaii의 워크그룹에 할당할 수 있는 총 워크 아이템의

개수는 256으로 1W-16T의 경우 512개의 스레드 할당이 필요하므로 윈도우 1/2을 할당하여 수행하도록 하였다. 실험 결과를 분석하면, 로컬 메모리에 할당할 수 있는 윈도우를 증가할수록 실행시간이 감소하였으며, 또한 할당된 윈도우에서 최대한 많은 스레드를 할당하여 처리할수록 또한 성능이 좋아지는 것을 확인 할 수 있었다. 그러나 할당 윈도우 수에서 최대한 많은 스레드를 할당하여 처리한 성능만을 비교할 경우 본 사례에서는 할당하는 윈도우 개수가 1일 때, 즉 1W-16T의 경우에서 성능이 가장 뛰어난 것으로 확인할 수 있었다. 이는 할당하는 윈도우의 수를 늘릴 때 마다 각 윈도우 한 행에 할당할 수 있는 스레드가 줄게 되면서 각 행을 담당하는 스레드의 메모리 접근 시 스레드 간에 메모리 뱅크 충돌의 확률이 증가하기 때문이다.

호스트 CPU 실행시간과 GPGPU 실행시간을 비교하기 위해서는 GPU 계산 전 호스트 CPU에서 GPU 메모리로 데이터를 복사하는 시간과 GPU 계산 종료 후 결과를 GPU에서 호스트 CPU로 전단하는 시간을 포함하는 총 실행시간과 비교를 하여야 한다. 다음 그림 6은 호스트 CPU와 GPU의 총 실행시간을 비교한 결과로 호스트 CPU 직렬코드 실행 시간과 GPU에 N_{thread}가 16개 이면서 AW가 1일 때 호스트-GPU 간의 메모리 전달 시간을 포함하는 총 실행시간을 비교한다. OpenCL을 이용한 GPGPU 병렬 프로그램 성능 평가 결과, AMD Hawaii의 경우 약 69.2배, Tesla C2075 시스템의 경우 약 36.8배, 그리고 GeForce GTX 750 시스템에서는 약 29.4배가 향상되었다.

5. 결론 및 향후 연구

본 논문에서는 미국 NIST에서 배포한 PCASYS의 지문 개선 알고리즘을 OpenCL을 이용한 GPGPU 병렬 프로그램을 구현하여 성능을 가속화하였다. PCASYS의 지문 개선 알고리즘은 지문 영상을 32×32 크기의 윈도우로 분할 하여 영상을 개선한다. 총 30×28 개의 윈도우를 처리한다. 각각의 윈도우는 서로 데이터 종속성이 없기 때문에 독립적으로 처리가 가능하기 때문에 윈도우 단위로 병렬처리 할 수 있다. OpenCL 구현 시 성능 최적화를 위하여, 하나의 워크그룹에 할당할 수 있는 윈도우의 수와 하나의 워크그룹에 할당하는 워크 아이템의 개수 등 각 GPU의 플랫폼 특성을 고려하여 다양한 조합으로 할당을 할 수가 있다. 본 연구에서는 GPU 환경 변수와 그 조건에 따른 성능 분석을 수행하여 GPGPU 성능을 최적화 하였다.

실험을 통하여 PCASYS의 지문 개선 알고리즘의 OpenCL을 이용한 GPGPU 병렬 프로그램의 성능 평가 결과, AMD Hawaii의 경우 약 69.2배, Tesla C2075 시스템의 경우 약 36.8배, 그리고 GeForce GTX 750 시스템에서는 약 29.4배가 향상되었음을 확인하였다. 향후 본 연구는 GPGPU를 이용하여 지문 영상을 개선한 것으로 이를 기반으로 향후 GPGPU를 이용하여 지문 인식에 대한 연구를 진행하고자 한다.

감사의 글

이 논문은 2013학년도 건국대학교의 연구년교원 지원에 의하여 연구되었습니다.

References

[1] Lin Hong, Anil Jain, Sharathcha Pankanti, and Ruud Bolle, Automatic Fingerprint Recognition Systems, Springer, pp. 127-143, 2004.

[2] Francisco Fons, Mariano Fons, and Enrique Cantó, “Approaching fingerprint image enhancement through reconfigurable hardware accelerators,” In proceedings of 1st Workshop on Intelligent Signal Processing(WISP 2007), pp. 1-6, 2007.

[3] Raja Lehtihet, Wael El Oraiby, and Mohammed Benmohammed, “Fingerprint grid enhancement on GPU,” In proceedings of 20th the International Conference on Image Processing, Computer Vision, and Pattern Recognition(IPCV 2013), pp. 1, 2013.

[4] Chirag Agarwal, Akhtar Rasool, and Nilay Khare, “PFAC Implementation Issues and their Solutions on GPGPU's using OpenCL,” The International Journal of Computer Applications, Vol. 72, No. 7, pp. 52-58, 2013.

[5] Jeonghyeon Ma, Sejin Park, and Chanik Park, “Parallel Rabin Fingerprinting on GPGPU for Efficiemt Data Deduplication,” The Journal of Korean Institute of Information Scientists and Engineers, Vol. 41, No. 9, pp. 611-616, 2014.

[6] G. T. Candela, P. J. Grother, C. I. Watson, R. A. Wilkinson, and C. L. Wilson, “PCASYS-A pattern-level classification automation system for fingerprints,” Technical Report NISTIR-5647, NIST, August 1995.

[7] Craig I. Watson, Michael D. Garris, Elham Tabassi, Charles L. Wilson, R. Michael McCabe, Stanley Janet, and Kenneth Ko, “User's guide to NIST biometric image software (NBIS)”, Technical Report NISTIR-7392, NIST, January 2007.

[8] Lee Howes and Aaftab Munshi, “The OpenCL Specification version 2.0”, Khronos OpenCL Working Group, October 2015.
https://www.khronos.org/registry/cl/specs/opencl-2.0.pdf

[9] NIST Special Database 27a,
http://www.nist.gov/itl/iad/ig/sd27a.cfm

저 자 소 개

김 대 희 (Daehee Kim)

2015년 건국대학교 컴퓨터공학과(학사)
2015년~현재 건국대학교 컴퓨터공학과(석사)
관심분야 : GPGPU, OpenCL, 병렬컴퓨팅 등

박 능 수 (Neungsoo Park)

1991년 연세대학교 전기공학과(학사)
1993년 연세대학교 대학원 전기공학과 (석사)
2002년 미국 University of Southern California 전기공학과(공학박사)
2002년~2003년 삼성전자 책임연구원
2003년~현재 건국대학교 정보통신대학 컴퓨터공학부 교수
관심분야 : 컴퓨터구조, 임베디드 시스템, 병렬시스템, 멀티미디어 컴퓨팅 등