CUDA를 이용한 PCA 기반 얼굴인식의 가속
(Acceleration of PCA based Face Recognition using CUDA)

이 영 민 †
(Youngmin Yi)

요 약 얼굴인식은 보안 등 다수의 응용분야에서 중요하게 이용되는데, 얼굴인식을 위한 학습은 많은 계산시간이 소요된다. 따라서, 인식율을 높이기 위해 많은 이미지들을 학습하거나 높은 해상도의 이미지를 대상으로 학습할 때 가속화가 필요하다. 한편, 최근 폭넓은 분야에서 널리 이용되고 있는 그래픽스 프로세싱 유닛(GPU)은 대용량 정보처리를 빠르게 수행할 수 있어 고해상도 대용량 이미지들에 대해서도 빠른 인식이 가능하다. 본 논문에서는 주성분 분석(PCA) 기반의 얼굴인식 알고리즘의 병렬성을 분석하고 이를 GPU에서 효율적으로 병렬 수행하기 위한 방법을 제안하였다. C/OpenCV로 구현된 순차적인 버전과 비교했을 때, CUDA로 구현한 얼굴인식기는 전체 학습 시스템에서 최대 약 40배의 성능이득을 얻었다.

키워드 : 얼굴인식, CUDA, GPU, 주성분분석

Abstract Face recognition is important in many applications including surveillance, biometrics, and other domains and fast face recognition is required if she wants to train and test more images or to increase the resolution of an input image for better accuracy in recognition. Meanwhile, Graphics Processing Units (GPUs) have become widely available, offering the opportunity for real-time face recognition even for larger set of images with high resolution. In this paper, we explore the design space of parallelizing a PCA (Principal Components Analysis) based face recognition algorithm and propose a fast face recognizer on GPUs by exploiting the fine-grained data-parallelism found in the face recognition algorithm. Our best results with the CUDA face recognizer show over 40-fold speedups compared to a sequential C implementation.

Key words : Face recognition, CUDA, GPU, PCA (Principal Component Analysis)

1. 서 론

얼굴은 가장 중요한 정보 중 하나로서, 컴퓨터가 얼굴을 인식하고 새로운 얼굴의 패턴을 학습하는 연구는 많은 관심을 끌고 있다. 가장 기본적이며 성공적인 접근방법으로 PCA(Principal Components Analysis, 주성분분석)를 통한 얼굴 인식을 들 수 있다[1]. 이 기법은 얼굴이미지들에 대한 고유값(eigenvalue)을 구하여 고유성분 얼굴들을 추출(학습)하고, 이 값을 주어진 입력 얼굴 이미지의 값과 비교(식별)하는 방법이다. 이미지 수가 많고 해상도가 높을수록 얼굴 이미지들을 학습하는데 많은 시간이 소요되기 때문에 본 논문에서 GPU(Graphics Processing Unit)를 통해 학습과정을 가속하는 기법을 제안한다. 주어진 얼굴이미지를 식별하는데 필요한 연산도 학습에 사용되는 연산 중 일부이므로 제안하는 병렬화 기법을 통해 얼굴인식의 식별 과정도 가속이 가능하다.

한편, 최근 GPU를 범용으로 활용할 수 있는 병렬프로그래밍 프레임워크인 CUDA[2]가 소개되어 다양한 분야의 많은 응용에서 성공적으로 가속이 이루어졌다. 이 논문에서 우리는 얼굴인식 응용의 각 모듈들에 대한 병렬화 방법을 제안하고, 이를 CUDA로 구현하여 GPU에서 가속하였다.

2. PCA 기반 얼굴인식

PCA는 입력데이터에 변수가 많을 때, 가능한 연관된 변수들을 서로 독립적인(선형적으로 상관관계가 없는)

· 이 논문은 2012년도 정부의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2011-0013479)
· 이 논문은 2012 한국컴퓨터종합학술대회에서 'GPU를 통한 얼굴인식 가속화'의 제목으로 발표된 논문을 확장한 것임

† 정 회 원 : 서울시립대학교 전자전기컴퓨터공학부 교수
ymyi@uos.ac.kr

논문접수 : 2012년 7월 5일
심사완료 : 2012년 10월 23일

정보과학회논문지 : 시스템 및 이론 제40권 제1호(2013.2)

변수로 변환함으로써, 불필요한 변수의 수를 줄이는 수학적 기법이다[3]. 즉, 차원의 수를 줄이는 것인데, 입력 얼굴이미지를 픽셀 단위로 파악하면 차원의 수가 너무 많고 노이즈도 많기 때문에, 주성분들을 구함으로써 차원을 줄인다. 주성분들이 구해지면 이 주성분들을 축으로 하는 새로운 도메인이 성립되는데, 이를 얼굴공간(face space)라고 한다. PCA 기반 얼굴인식에서 학습과정은, 입력 얼굴이미지들의 주성분을 구하고, 각 얼굴이미지를 얼굴공간으로 투사(projection)하여 하나의 벡터로 얻는 과정이다. 한편 식별과정은, 학습에 사용되지 않은 주어진 얼굴이미지를 (학습과정에서 구한) 얼굴공간으로 투사하고, 이렇게 하여 얻어진 벡터 값을 학습 때 구한 벡터 값들과 비교하여, 가장 거리가 가까운 벡터에 해당하는 얼굴이미지로 인식하는 과정이다. 이 때 벡터의 차원은 원래 얼굴이미지의 픽셀 수보다 훨씬 적기 때문에 효율적인 계산이 가능하다. 다음 소절에서 각각 학습과 식별에 대해 더 자세하게 살펴보겠다.

2.1 학습

그림 1은 얼굴인식을 위한 학습과정의 각 모듈들과 입출력을 보여주고 있다. 첫 번째 과정은 공분산 행렬을 구하는 모듈이다. 이 모듈은 학습하는 얼굴이미지들간에 얼마나 상관관계가 있는지를 알아내기 위해 각 얼굴이미지 쌍간의 공분산을 구한다. N 장의 얼굴이미지를 입력으로 받아 NxN 공분산 행렬을 출력한다. Γ_k는 학습할 k 번째 입력 얼굴이미지, Ψ는 평균 얼굴이미지, P는 한 얼굴이미지의 픽셀 수로 정의될 때, 공분산 행렬의 i행 j열의 원소(공분산)을 구하는 수식은 식 (1)과 같다. 공분산 행렬의 크기는 NxN 이므로, 이 모듈의 계산복잡도는 O(N^2P)가 된다.

$$C_{ij} = \frac{1}{P}(\Gamma_i - \Psi)\cdot(\Gamma_j - \Psi)^T \quad (1)$$

두 번째 모듈은 공분산 행렬을 입력으로 받아, 주성분을 구한다. 주성분은 고유벡터와 고유값으로 표현되며, 우리는 Jacobi 법을 통해서 구한다.

세 번째 모듈은 두 번째 모듈에서 계산한 고유벡터를 이용하여 고유성분 얼굴을 구한다. 식 (2)에서 보듯이 m번째 고유성분 얼굴 U_m은 m번째 고유벡터 V_m를 이용하여 구해지는데, 고유벡터와 모든 얼굴이미지를 곱한 후 이를 누적하여 구한다. 따라서 고유벡터의 수가 M 개일 때 계산복잡도는 O(NMP)가 된다.

$$u_m = \sum_{k=1}^{N} v_{mk}(\Gamma_k - \Psi) \quad (2)$$

마지막으로 네 번째 모듈은 각 얼굴이미지를 고유성분 얼굴로 구성된 얼굴공간으로 투사한다. 식 (3)에서 보듯이 이것은 각 고유성분 얼굴과 각 얼굴이미지에 대해 내적을 수행하는 것에 해당한다. 얼굴공간에 투사되었다는 것은, 원래 얼굴이미지를 고유성분 얼굴들에 대한 가중치 합으로 재구성할 수 있다는 의미이며, 따라서 식 (3)으로 구해지는 M 차원 벡터를 가중치 벡터라 부른다. 학습하는 얼굴이미지 N 장에 대해서 수행하므로, 계산복잡도는 O(NMP)가 된다.

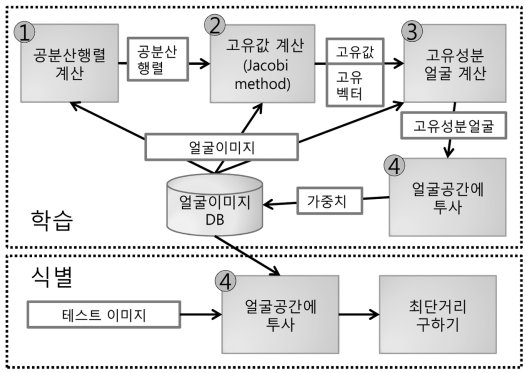

그림 1 PCA에 기반한 얼굴인식의 학습 및 식별과정

$$w_{nm} = u_m^T \cdot (\Gamma_n - \Psi) \quad (3)$$

2.2 식별

그림 1에 보듯이, 식별과정은 학습과정의 네 번째 모듈을 그대로 사용한다. 주어진 테스트 이미지와 학습된 얼굴이미지를 비교하기 위해서는, 테스트 이미지도 먼저 얼굴공간에 투사해야 하기 때문이다. 이미 학습과정에서 고유성분 얼굴들을 구했기 때문에 바로 투사를 할 수 있다.

테스트 얼굴이미지가, 학습된 얼굴이미지(Φ_f)들과 같은 얼굴공간으로 투사되어 얼굴공간 안에서의 위치(Φ)를 갖게 되면, 테스트 얼굴이미지와 학습된 얼굴이미지들 사이의 거리를 계산하여 유사도를 판단하고 가장 유사도가 높은 얼굴이미지를 식별된 얼굴로 태깅된다. 거리(ε)에 대한 계산은 유클리드 거리계산 법으로 수행한다(식 (4)).

$$\varepsilon^2 = \left\| \Phi - \Phi_f \right\|^2 \quad (4)$$

3. GPU와 CUDA

GPU는 본래 그래픽 응용의 처리를 위해 고안되었기 때문에 데이터를 병렬적으로 수행하는데 최적화되어 있다. 예를 들어 NVIDIA GTX480의 경우 480개의 스트리밍 프로세서(SP: Streaming Processor)를 가지고 있다. GPU는 계층적 구조로 이루어져 있어서 하나의 GPU는 여러 개의 스트리밍 멀티프로세서(SM: Streaming Multiprocessor)를 가지고, 하나의 SM에는 여러 개의 SP들이 집적되어 있다. GTX480의 경우, 총 15개의 SM

이 있고, 하나의 SM안에 32개의 SP들이 있다. 또한 SM 안에는 SP들이 공유할 수 있는 빠른 접근속도를 제공하는 *공유메모리*가 존재한다.

한편, 비교적 최근에 등장한 CUDA는 NVIDIA GPU를 위한 C기반의 병렬 프로그래밍 프레임워크로서 GPU를 범용으로 프로그래밍할 수 있다. C프로그래밍 언어를 확장하여 쓰레드와 GPU의 다양한 메모리를 구분하여 접근할 수 있도록 하였고, 원자연산(atomic operation)을 위한 API[4] 등과 같은 라이브러리가 제공된다. CUDA에서 규정하는 논리적인 병렬수행 주체인 *쓰레드*와 *쓰레드 블록*은 물리적으로 각각 SP와 SM에 매핑된다. 즉, SP는 쓰레드 연산을 수행하고 SM은 쓰레드 블록의 연산을 수행한다. 이러한 쓰레드 블록과 쓰레드로 구성된, GPU에서 수행되는 함수를 커널이라고 한다.

다음절에서 자세히 살펴보겠지만, 주어진 알고리즘을 병렬수행의 주체인 쓰레드에 매핑했을 때 한 쓰레드가 다른 쓰레드의 수행과 무관하게 독립적으로만 계산하는 경우는 없기 때문에, 쓰레드 사이에 동기화가 이루어져야 한다. 이 때 공유메모리를 이용하는 것도 한 가지 방법이다.

한편, 최신의 CPU들은 멀티코어 CPU로서 4개 이상의 코어가 탑재된 경우가 대부분이다. GPU만큼은 아니지만, 어느 정도 병렬 수행이 가능할 수 있다. 하지만, GPU가 작은 단위의 데이터 병렬성을 이용한 가속에 적합한 반면, CPU는 규모가 큰 단위의 태스크 병렬성을 이용한 가속에 상대적으로 더 유리한 구조를 지니고 있다. 따라서, 코어 개수의 차이뿐만 아니라 코어의 구조에 적합한 병렬성을 고려했을 때, 픽셀단위 처리가 빈번한 얼굴인식의 경우 GPU를 이용한 병렬화가 더 적합하다. 5절에서 OpenMP를 이용한 CPU에서의 병렬화 결과와 제안하는 GPU를 이용한 병렬화 결과와 비교를 하였다.

4. 병렬화 기법

이 절에서는 2절에서 설명한 알고리즘의 병렬성과 3절에서 설명한 GPU의 특성을 고려한 병렬화 방법에 대해 설명하도록 하겠다. 추가로, 4.4절에서 CPU에서 OpenMP로 병렬화하기 위한 프래그마 사용법을 설명한다.

표 1은 C/OpenCV로 구현된 얼굴인식기[5,6]에서 학습과정의 각 수행시간을 측정한 것으로 공분산 행렬 계산은 약 20%, Jacobi 법은 약 1%, 고유성분 얼굴 계산은 약 40%, 그리고 얼굴공간으로의 투사는 약 40%의 시간 비율을 가진다. 따라서, 비중이 적은 Jacobi법을 제외한 나머지 세 모듈에 대해서 가속하기로 정했다.

식별과정에서 수행시간을 측정한 결과는 표 2와 같이 얼굴공간 투사(모듈4)가 전체 수행시간의 73%로 가장 큰 비중을 차지하고 있다. 거리계산 모듈의 경우에는 수행시간이 미미하여 가속화를 고려하지 않았다.

표 1 C/OpenCV 구현에 의한 학습 실행시간(FEI in Sao Bernardo do Campo DB, 얼굴이미지의 해상도 320*240, 200장)

과정	실행시간	
	시간(ms)	비중(%)
공분산 행렬 계산(모듈1)	6,101	22
Jacobi 법(모듈2)	289	1
고유성분 얼굴 계산(모듈3)	10,543	38
얼굴공간 투사 (모듈4)	10,952	39
계산 시간	27,596	99
전체 응용시간	27,885	100

표 2 C/OpenCV 구현에 의한 식별 실행시간(Caltech DB, 얼굴이미지의 해상도 320*400, 테스트 이미지 수 170, 학습된 얼굴 190장)

과정	실행시간	
	시간(ms)	비중(%)
얼굴공간 투사(모듈4)	14,818	73
거리계산	24	0.1
계산시간	14,842	73
학습된 얼굴정보 메모리적재	5,475	27
전체응용시간	20,317	100

4.1 공분산 행렬 계산(모듈1)

공분산 행렬을 계산하는 모듈은 식 (1)에서 살펴본 것처럼 복잡도가 $O(N^2P)$로서 이미지 수의 제곱에 비례하고, 또한 픽셀 수에 비례한다. 이와 같은 병렬성을 GPU에 매핑하는 방법은 극단적으로 두 가지가 있을 것이다: 하나의 쓰레드가 하나의 픽셀을 담당하도록 매핑하는 것과 하나의 쓰레드가 하나의 얼굴이미지를 담당하도록 매핑하는 것이다. 만약 P가 작고 N이 클 경우 후자의 매핑이 유리할 수 있고, 만약 P가 크고 N이 작다면 전자의 매핑이 유리할 수 있을 것이다. 이를 일반화하여, 하나의 쓰레드가, 1이상 P이하의 값인 K개의 픽셀을 담당하도록 매핑하고, 실험적으로 최적이 되는 K의 값을 구한다. 이와 같은 쓰레드 매핑은 이후의 다른 과정에서도 마찬가지로 적용하도록 한다.

식 (1)에서 기술한 것처럼 모듈1은 두 개의 얼굴이미지의 공분산을 계산하기 위해 두 얼굴이미지 쌍의 내적을 계산해야 하는데, 이 때 합 리덕션 연산이 필요하게 된다. 순차적인 구현에서는 누적을 통해 자연스럽게 합 리덕션이 구현되지만, 병렬수행에서는 동기화가 이루어져야 합 리덕션이 올바르게 수행이 된다. 리덕션 방법은 *atomicAdd*()와 같은 원자연산 API를 사용하는 방법과 3절에서 설명한 공유메모리를 이용하여 병렬 리덕션을 구현하는 방법이 있다. 병렬 리덕션은, 아래 그림 2와 같이 T개의 쓰레드들이 $\log_2 T$만큼의 스텝을 반복하면서 병렬로 리덕션을 수행한다.

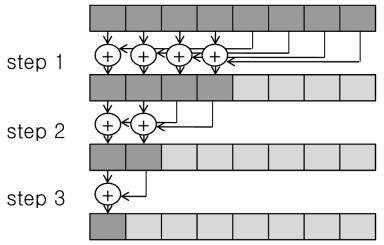

그림 2 CUDA 커널에서 구현된 병렬 리덕션

4.2 고유성분 얼굴 계산(모듈3)

고유성분 얼굴을 계산하는 모듈은 식 (3)에서 살펴본 바와 같이 O(NMP)의 복잡도를 가지기 때문에, 최대 NMP만큼 모두 병렬화할 수 있지만, GPU가 가지는 메모리 크기가 제약이 된다면 MP나 NP로만 병렬화하고, N이나 M만큼 CPU에서 순차적으로 커널호출을 반복해야 한다. 그러나 GPU의 메모리 가 충분히 크더라도 식 (3)에서 알 수 있듯이, NMP만큼 병렬화할 경우 동기화 비용이 클 수 있다. 모듈1과 달리 모듈3에서는 같은 얼굴이미지의 픽셀들에 대해서는 합 리덕션과 같은 동기화를 수행하지 않아도 되지만, 대신 서로 다른 얼굴이미지들의 동일한 픽셀 위치들에 대해서 합 리덕션을 수행하여 최종적으로 고유성분 얼굴을 계산해야 한다. 그림 3의 빨간 사각형은 한 쓰레드가 담당하는 픽셀 영역을 도시하고 있는데, 설명을 간단히 하기 위해, 한 장의 얼굴이미지는 모두 한 쓰레드 블록에서 처리된다고 가정한다. 그림 3의 작은 파란 사각형으로 도시한 영역은 각 얼굴이미지에서 같은 픽셀 영역으로서, 서로 다른 쓰레드 블록에 위치하고 있다. 따라서, 이 경우 쓰레드 블록 내의 동기화가 아닌 쓰레드 블록 간의 동기화가 되기 때문에 동기화 비용이 크다. 따라서, 우리는 MP만큼만 GPU에서 병렬화하고, CPU에서 순차적으로 N번 커널을 호출하는 구조로 설계하였다.

4.3 얼굴공간 투사(모듈4)

얼굴공간 투사 모듈의 복잡도는 O(NMP)이다. 고유성분 얼굴을 계산하는 모듈과 마찬가지로 병렬성은 복잡도만큼 확보할 수 있지만, GPU의 메모리 크기 제약을 받을 수 있다. 하지만, 모듈3과는 달리 (그리고 모듈1과는 유사하게) 얼굴이미지와 고유얼굴 쌍간의 내적을 수행하므로, GPU의 메모리가 충분히 크다면, NMP 모두 병렬 수행할 수 있다. 즉, 설명하기 쉽게 하나의 쓰레드 블록에서 하나의 얼굴이미지를 담당한다면 총 MN개의 쓰레드 블록과 쓰레드 블록 당 P개의 쓰레드가 병렬적으로 수행되는 커널을 설계한다.

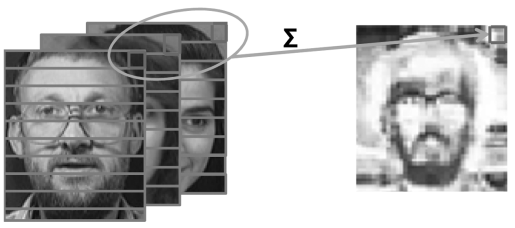

그림 3 모듈 2에서 필요한 합 리덕션

4.4 OpenMP를 이용한 CPU에서의 병렬화 방법

한편, OpenMP는 프래그마를 이용한 병렬프로그래밍 기법으로서 OpenMP 프래그마를 지원하는 컴파일러와 라이브러리가 존재해야 한다. 최근에는 멀티코어 병렬 프로그래밍이 보편화되어 대부분의 컴파일러에서 이를 지원한다. OpenMP 프래그마를 이용하여 반복문을 병렬화할 때 #pragma omp parallel for 와 같이 지정을 하면 병렬 수행이 이루어진다. 이 때, 반복문 안에 리덕션 되어야 할 변수가 있다면, #pragma omp parallel for reduction (+:변수명) 과 같이 지정하면 자동으로 합 리덕션이 이루어진다.

5. 실험 및 결과

앞서 설명한 방식대로 CUDA로 GPU커널을 작성하고GTX480[7]이 장착된 Core i7 3.7 GHz 컴퓨터에서 실험하였다. 92×112 픽셀크기를 가지는 AT&T DB[8]와 320×400픽셀크기를 가지는 Caltech DB[9]를 사용했고, 가장 높은 인식율을 얻기 위해, M 값은 N-1로 설정하였다. AT&T DB의 경우에는 병렬 리덕션 합을 적용했을 때 *atomicAdd*()에 비해 최고 150%정도 빨랐으나, Caltech DB 에서는 차이가 7% 미만이었다. 따라서, DB마다 최적의 수행시간을 보였던 리덕션 방법을 가지고 실험하였다. 표 3, 표 4는 Caltech DB(320x400 픽셀)에 대해 190장, 360장을 학습한 실험 결과이며 표 5, 표 6은 AT&T DB(92×112 픽셀)에 대해 200장, 400장을 학습한 결과이다.

각 표의 두 번째 열은 C구현으로 순차적으로 수행될 때 걸리는 시간이고, 세 번째 열은 전체 수행시간 중 해당 모듈의 수행시간이 차지하는 비중을 %로 보여준다. 이 때 마지막 행(**전체시간)은 모듈2인 Jacobi 법 수행시간을 포함한 시간을 보여주고, 마지막에서 두 번째 행(*계산시간)은 이를 제외한 시간을 보여주기 때문에 이 차이가 Jacobi 법 수행시간의 비중(%)이 된다. 네 번째 열은 CUDA로 구현된 커널만의 수행시간이며, 다섯 번째 열은 CPU와 GPU사이의 메모리 복사 시간까지 포함된 수행시간이다. 여섯 번째 열은 GPU로 병렬수행 될 때 각 모듈의 수행시간 비중을 보여준다. 일곱번째 열은 C 구현에서의 수행시간 대비 CUDA 커널만의 수행시간으로 얻어진 성능 이득을, 여덟 번째 열은 C 구현에서의 수행시간 대비 CUDA 커널 및 메모리 전송시간을 모두

포함한 수행시간으로 얻어진 성능 이득을 의미한다.

표 3에서 보듯이 픽셀 수가 128K(=320×400)인 Caltech DB의 경우, 모듈1의 33초에서 0.3초로 줄어들어 성능이득이 약 100배로 가장 컸고, 메모리 전송 오버헤드를 고려해도 58배의 성능이득이 발생한다. 모든 모듈들이 잘 가속되어 약 151초였던 수행시간이 약4초로 줄어들어 약 40배의 성능이득을 얻었다. 한편, Jacobi 법 수행하는 모듈인 모듈2의 수행시간 비중이 순차구현에서는 1%밖에 되지 않지만, 병렬구현에서는 워낙 다른 모듈의 수행시간이 줄어들면서, 전체의 40% 비중이 된다. 표 4에서는 표 3과 동일한 설정이나 N=190에서 N=360으로 학습되는 얼굴이미지 수가 증가되었다. 여전히 40배 정도의 성능이득을 얻을 수 있음을 알 수 있다.

한편, 픽셀 수가 10K정도밖에 되지 않는 AT&T DB의 경우는 계산시간 대비 메모리 복사 시간의 비율이 커지고, Jacobi법 수행시간의 상대적인 비중이 커서, 전체적인 성능이득이 줄어든다. 하지만 여전히 10배 가까운 성능이득을 얻을 수 있다.

식별단계는 표 2에서 보는 바와 같이 하나의 모듈(모듈4)에 대한 가속화만을 살펴보면 된다. 표 7의 모듈4의 성능이득을 보면, CUDA커널 수행시간만 비교했을 때 97배, 메모리 복사 오버헤드까지 고려했을 때 56배로서, 표 3에서의 모듈4의 성능이득과 유사함을 알 수 있다.

표 3 학습시간 및 CUDA구현 성능이득(Caltech DB 190장, K=500)

모듈	실행시간(ms)					성능이득	
	C	%	CUDA (커널)	CUDA (커널 + 메모리)	%		
모듈 1 (A)	9,201	22	92	159	16	100	58
모듈 3 (B)	15,858	38	268	332	34	59	48
모듈 4 (C)	16,477	39	203	254	26	81	65
*계산시간	41,536	99	562	745	76	74	56
**전체시간	41,776			987		52	42

*계산시간 = A+B+C, **전체시간 = 계산시간 + Jacobi법

표 4 학습시간 및 CUDA구현 성능이득(Caltech DB 360장, K=500)

모듈	실행시간(ms)					성능이득	
	C	%	CUDA (커널)	CUDA (커널 + 메모리)	%		
모듈 1(A)	33,124	22	331	403	11	100	82
모듈 3(B)	57,249	38	954	1,077	28	60	53
모듈 4(C)	59,607	39	735	851	22	81	70
*계산시간	149,980	99	2,019	2,331	61	74	64
**전체시간	151,466			3,821		43	39

*계산시간 = A+B+C, **전체시간 = 계산시간 + Jacobi법

유의할 사항은 학습된 정보의 양이, N과 M의 값에 따라 다르겠지만, 대개 수 GB의 용량을 가지고 있어서 이를 메모리에 적재할 때 많은 시간이 걸린다는 점이다. 이 시간을 포함하면 전체시간의 성능이득은 4배로 낮게 나온다. 그러나, 식별할 이미지 수가 아무리 많아도 학습정보는 한번만 적재하면 되고, 또한 이미 메모리에 적재되어 있다고 가정할 수 있기 때문에 전체적인 성능이득은 모듈4의 성능이득과 유사한 52배라고 할 수 있겠다.

한편, 표 8은 OpenMP로 CPU에서 병렬수행을 했을 때의 결과를 보여준다. Core i7은 총4개의 코어를 가지고 있어서 각 모듈은 이상적으로는 최대4배까지 향상될 수 있다. 표에서 볼 수 있듯이 약 3배정도의 향상을 얻을 수 있다.

표 5 학습시간 및 CUDA 구현 성능이득(AT&T DB 200장, K=40)

모듈	실행시간(ms)					성능이득	
	C	%	CUDA (커널)	CUDA (커널 + 메모리)	%		
모듈 1(A)	819	21	11	60	14	76	14
모듈 3(B)	1,403	35	25	35	8	55	40
모듈 4(C)	1,455	37	17	43	10	87	34
*계산시간	3,677	93	53	138	32	70	27
**전체시간	3,969			427		12	9

*계산시간 = A+B+C, **전체시간 = 계산시간 + Jacobi 법

표 6 학습시간 및 CUDA 구현 성능이득(AT&T DB 400장, K=40)

모듈	실행시간					성능이득	
	C	%	CUDA (커널)	CUDA (커널 + 메모리)	%		
모듈 1(A)	3,261	19	43	95	4	76	34
모듈 3(B)	5,664	33	95	113	5	59	50
모듈 4(C)	5,895	35	59	123	5	100	48
*계산시간	14,820	87	196	331	13	75	45
**전체시간	16,999		2,492			7	7

*계산시간 = A+B+C, **전체시간 = 계산시간 + Jacobi 법

표 7 식별시간 및 CUDA 구현 성능이득(Caltech DB 170장, M=189)

모듈	실행시간(ms)					성능이득	
	C	%	CUDA (커널)	CUDA (커널 + 메모리)	%		
모듈 4 (A)	14,818	73	152	263	5	97	56
거리계산(B)	24	0.1			0.3		
학습정보 적재	5,475	27			95		
*계산시간	14,842	73			5	86	52
**전체시간	20,317			5788		4	4

*계산시간 = A+B, **전체시간 = 계산시간 + 학습정보 로딩

표 8 학습시간 및 OpenMP구현 성능이득(Caltech DB 190장)

모듈	실행시간 (ms)				성능이득
	C	%	OpenMP	%	
모듈 1 (A)	9,201	22	3,228	16	2.8
모듈 3 (B)	15,858	38	5,256	34	3.0
모듈 4 (C)	16,477	39	4,696	26	3.5
*계산시간	41,536	99	13,180	76	3.2

*계산시간 = A+B+C

6. 관련연구

얼굴인식을 GPU로 가속하기 위한 노력은 최근에 제안되고 있으나 관련 연구가 많지 않다. [10]은 주성분 기반의 얼굴인식을 가속화한 논문이다. 주성분 기반 학습 알고리즘 중에서 얼굴공간에 대한 투사 모듈만을 가속화했고, 식별 알고리즘에서는 유클리드 거리계산 모듈을 가속했다. GTX480으로 약 200배의 성능이득을 보고하고 있지만, 사용된 이미지의 크기가 16×16으로 매우 작고 하나의 이미지 전체를 하나의 쓰레드가 처리하는 단순 매핑으로 구현하여, 일반적인 이미지 크기에 대해서 적용하기 어렵다.

[11]은 Local Binary Pattern(LBP) 기반의 얼굴인식기에 대해 CUDA로 구현하였는데, 입력 얼굴이미지로부터 LBP값을 구하는 모듈, LBP값으로부터 지역 히스토그램을 구성하는 모듈, kNN(k-Nearest Neighboring) 계산모듈을 각각 CUDA로 가속하여 29배의 성능이득을 얻었다. [12]는 SVM (Support Vector Machine)을 이용하는 얼굴인식 시스템에서 GPU를 이용하여 25배 성능이득을 얻었다.

뉴런을 기반으로 얼굴인식을 수행하는 네오코그니트론[13] 접근법을 GPU를 사용하여 가속한 연구가 최근에 제안되었다[14]. 네오코그니트론은 뉴런의 결과값들을 리덕션하고 학습된 정보들과 비교하여 얼굴인식을 수행하는 알고리즘인데, [14]에서 이를 뉴런단위로 병렬화하였다. 이미지 600장에 대하여, CPU에서 순차적으로 48초 걸리는 수행시간을 CUDA로 약 250배 가속하였으나, 논문에서 언급했듯이 이 때 사용된 픽셀크기는 57x57로 작기 때문에 인식률이 제한적이고, 무엇보다 네오코그니트론은 주성분 기반의 얼굴인식 알고리즘에 비해 널리 사용되는 알고리즘이 아니다.

7. 결 론

얼굴인식은 갈수록 다양한 응용에서 사용되고 있는 추세이지만, 인식률을 높이기 위해서는 많은 계산이 요구된다. 본 논문에서는 주성분 기반의 얼굴인식 알고리즘의 주요 모듈들을 GPU로 효율적으로 가속화하기 위한 병렬화 방법을 제안하였으며, 학습에서는 DB에 따라 10배에서 40배, 식별에서는 (메모리 적재 시간을 제외하면) 50배 정도의 성능이득을 얻을 수 있었다.

참 고 문 헌

[1] M. Turk, A. Pentland(1991), "Eigenfaces for Recognition," *Journal of Cognitive Neuroscience*, pp.71-86.

[2] J. Nickolls et al., "Scalable Parallel Programming with CUDA," *ACM Queue*, vol.6, no.2, pp.40-53, Mar./Apr. 2008.

[3] L. I. Smith, "A Tutorial on Principal Components Analysis," http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf, 2009.

[4] A. Functions, "CUDA C Programing Guide version 4.1," pp.98-100.

[5] OpenCV Library, http://opencv.org

[6] R. Hewitt, "Face Recognition with Eigenface," *SERVO Magzine*, Apr. 2007.

[7] C. M. Wittenbrink, E. Kilgariff, A. Prabhu, "Fermi GF100 GPU Architecture," *IEEE Micro*, vol.31, no.2, pp. 50-59, Mar. 2011.

[8] F. Samaria, A. Harter, "Parameterisation of a Stochastic Model for Human Face Identification," *IEEE Workshop on Applications of Computer Vision*, 1994.

[9] Caltech Face Database, http://www.vision.caltech.edu/html-files/archive.html

[10] N. Ashraf, Sibi. A, "CUDA-Accelerated Face Recognition," *poster presentation*, GPU Technology Conference, 2010.

[11] S. C. Tek, M. Gokmen, "CUDA Accelerated Face Recognition Using Local Binary Patterns," N.p.

[12] Eunji Lee, Sungju Lee, Bong-Su Kang, Yongwha Chung, Daihee Park, Byoung-Ki Min, "GPU based Parallel Hierarchical SVM for Video Surveillance System," *Joint Conference on Signal Processing*, pp.1-4, 2010.

[13] K. Fukushima, S. Miyake, "Neocognitron: A New Algorithm for Pattern Recognition Tolerant of Deformations and Shifts in Position," *Pattern Recognition*, vol.15, no.6, pp.455-469, 1982.

[14] G. Poli, J. H. Saito, J. F. Mari, M. R. Zorzan, "Processing Neocognitron of Face Recognition on High Performance Environment Based on GPU with CUDA Architecture," *International Symposium on Computer Architecture and High Performance Computing*, pp.81-88, 2008.

이 영 민

2000년 서울대학교 컴퓨터공학과(학사) 2007년 서울대학교 전기·컴퓨터공학부(석박사 통합). 2007년~2009년 UC Berkeley Parallel Computing Lab. 박사후연구원. 2009년~2010년 삼성전자 종합기술원 멀티코어SW그룹. 2010년~현재 서울시립대학교 전자전기컴퓨터공학부 조교수. 관심분야는 병렬 소프트웨어 설계, 이기종 컴퓨팅, 임베디드 시스템