GPGPU를 이용한 가우시안 혼합 모델의 관측확률 계산 성능 향상

김형주*, 김승희**, 김상훈**, 장길진***
*UNIST 의생명과학, 컴퓨터공학 융합전공
**한국전자통신연구원(ETRI)
***UNIST 전기전자컴퓨터공학부
e-mail:khj458@unist.ac.kr

Performance Improvement in Observation Probability Computation of Gaussian Mixture Models Using GPGPU

Hyeong-Ju Kim*, **, Seung-Hi Kim***, Sanghun Kim***, Gil-Jin Jang**
*Dept of Nano Biology & Chemistry Eng., UNIST,
**Dept of Electrical & Computer Eng., UNIST,
***Electronics and Telecommunications Research Institute

요 약

범용 GPU (general-purpose computing on graphics processing units, GPGPU)는 GPU를 일반적인 목적으로 사용하고자 하는 병렬 컴퓨터 구조로써, 과학 연산 등 여러 분야에서 응용 프로그램의 성능을 향상시키기 위하여 사용되고 있다. 본 연구에서는 음성인식기에서 주로 사용되는 가우시안 혼합 모델 (Gaussian mixture model, GMM)에서 많은 연산시간을 차지하는 관측확률 계산의 성능을 향상시키고자 GPGPU를 이용하는 알고리즘을 구현하였으며, 기존 CPU 기반 알고리즘 대비 약 13배 연산시간을 단축하였다.

1. 서론

가우시안 혼합모델(Gaussian Mixture Model, GMM)은 임의의 확률 분포를 추정하기 위하여 주로 사용되며, 음성인식을 비롯한 다양한 신호처리 응용분야에서 사용되고 있다. 가우시안 혼합모델의 처리과정에서 가장 많은 시간이 소요되는 부분은 입력의 관측확률 계산이다. 특히, 실시간으로 발화된 음성신호에서 의도한 문장을 인식하여 주는 음성인식 응용프로그램의 경우, 가우시안 혼합 모델의 관측확률 계산 과정은 사용자의 체감 소요시간과 직결되는 요소이다. 따라서 이에 소요되는 시간을 줄이는 것은 사용자 편의 측면에서 매우 중요하다.

현재 판매되고 있는 GPU (graphic processing unit)의 컴퓨팅 능력은 이미 CPU (central processing unit)의 컴퓨팅 능력보다 월등히 높다. 최근 NVIDIA에서 제시되는 바에 따르면[1], 하드웨어의 컴퓨팅 능력을 나타내는 단위시간당 부동소수점 연산수(Floating-point operations per second, FLOP/s)는 GeForce GTX 680의 경우 약 3 TFLOP/s로써, Intel Sandy Bridge CPU의 약 500 GFLOP/s를 크게 웃돈다. 뿐만 아니라 메모리 대역폭 또한 GPU 190 GB/s 대비 CPU 50 GB/s로 거의 네 배에 가까운 차이를 보여준다.

범용 GPU (general-purpose graphics processing units, GPGPU)는 이러한 고성능 GPU를 일반적인 목적으로 사용 가능하도록 제작된 것이며, CUDA는 NVIDIA의 GPU를 GPGPU로 사용하기 위해 제시된 병렬 컴퓨팅 구조이다. CUDA는 또한, C나 Fortran 같은 잘 알려진 언어에서 확장된 언어로 구현되어 있기 때문에 기존의 프로그래머들이 쉽게 습득할 수 있는 장점도 있다.

이번 연구에서는 CUDA를 이용한 GPGPU의 활용으로 음성인식에서 사용되는 가우시안 혼합 모델의 확률 계산의 성능 개선을 목표로 하였다. CUDA에 대한 자세한 사항은 참조문헌을 참고하도록 한다[1][2].

2. 본론

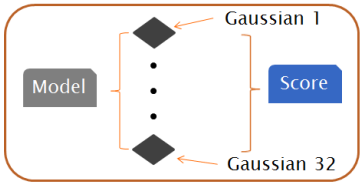
2.1. 개발 환경 및 실험 데이터

개발을 진행한 GPU의 종류는 NVIDIA GeForce GTX 550Ti와 GTX 680이었으며, 운영체제는 Windows 7 Service Pack 1, CPU는 Intel i7 SandyBridge 2600을 사용하였다. CUDA의 버전은 4.2였으며, 사용한 언어는 CUDA C였고, 프로그래밍은 Windows Visual Studio 2008에서 진행하였다. 성능 평가에 사용된 자료는 50여개의 발화로부터 추출되었으며, 음성의 특징은 음성인식에서 널리 채택되고 있는 MFCC (mel-frequency cepstral coefficients)를 사용하였다. MFCC는 12차 DCT (discrete cosine transform)에 log 에너지를 첨가하여 13차로 구성

하였으며, 여기에 시간축 차분(difference)과 차차분(acceleration) 벡터를 더하여 총 39차로 음성특징 벡터를 구성하였다.

2.2. 알고리즘 및 데이터 구조

GMM 확률 계산 알고리즘에서는 총 6647개의 모델이 사용되었다. 각 모델은 32개의 가우시안 함수로 이루어져 있으며, 각 함수는 음성특징 벡터의 차원수와 동일한 39차로 이루어져 있다.



(그림 1) GMM 구조 및 알고리즘 개요

39차 음성특징 벡터가 입력될 때마다 6,647개의 모델에서 각각 관측확률을 계산하였으며, 아래 식과 같이 계산되어진다.

$$f_Y(\boldsymbol{x}) = \sum_{i=1}^{32} \alpha_i \frac{1}{\sqrt{(2\pi)^{39}|\Sigma_i|}} exp(-\frac{1}{2}(\boldsymbol{x}-\mu_i)^T\Sigma_i^{-1}(\boldsymbol{x}-\mu_i)) \quad \cdots\cdots\cdots\cdots\cdots ①$$

위의 식에서 $\boldsymbol{x}$는 39차 특징 벡터이며, μ는 각 가우시안 함수의 평균, Σ는 공분산 행렬이다. α_i는 i번째 가우시안의 선험확률(*a priori* probability)이며, 1부터 32까지의 모든 가우시안의 선험확률의 합은 1이다.

하지만 실제 음성인식기에서는 연산량을 줄이기 위하여 식 ①을 변형하여 GMM 확률 계산을 한다. 이는 아래와 같다.

$$score_i = Gconst_i + \sum_{j=0}^{35}(x_{i_j} - \mu_{i_j})^2 * IVar_{i_j} \quad \cdots\cdots\cdots\cdots\cdots ②$$

식 ②는 식 ①의 내부 수식에 log를 취한 것이다. 이 근사법을 사용하여 관측확률을 근사함으로써 연산량을 획기적으로 줄일 수 있다. 또한, 39차를 모두 사용하는 것이 아니라 36차까지만 사용한다. 이는 경험적으로 인식기의 인식 성능에 큰 영향을 미치지 않기 때문에, 차수를 줄여 연산량을 줄인 것이다. *Gconst*는 선험확률 α_i와 상수 $1/\sqrt{(2\pi)^{39}|\Sigma_i|}$, 그리고 -1/2를 반영한 하나의 상수로서, 평균 μ와 IVar와 함께 확률 모델에서 이미 계산되어 저장된 값이다. IVar는 분산의 역수를 의미한다. 나누기 연산은 곱하기 연산보다 많은 연산량이 요구되기 때문에 미리 분산의 역수를 저장하여 사용한다. 이렇게 계산한 32개의 score를 가지고 다시 한 번 근사를 한다.

$$BestScore = \min(score_0, score_1, \ldots, score_{31}) \quad \cdots\cdots\cdots\cdots\cdots ③$$

*Gconst*에서 -1/2값을 반영하여 처리해 주었기 때문에, score들 중 최솟값이 가장 높은 확률 값을 얻을 수 있다. 따라서 32개 가우시안에서 계산된 관측확률 값들 중, 최솟값 하나만 선택하여 사용한다. 이를 위하여 각각의 가우시안 혼합 모델에 대하여 32개의 *Gconst*와 평균벡터 μ, 그리고 32×36개의 IVar, 즉 1152개의 값들을 미리 계산하여 저장한다. 이에 대한 알고리즘의 의사코드는 아래와 같다.

```
Calculation_GMM (Model, feature){
    bestScore = WORSTSCORE;

    for m in 0:31 {
        score = gconst[m];

        for d in 0:35 {
            diff = mean[m][d] - feature[d];
            score += diff^2 * ivar[m][d];
        }

        if score < bestScore
            bestScore = score;
    }
    return score;
}
```

(코드 1) GMM 확률 계산 의사코드

2.3. 적용방법

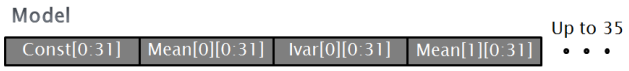
2.3.1. 기존 알고리즘 및 적용 방향

기존의 알고리즘은 하나의 CPU 코어를 사용하는 것으로 개발되었다. 하지만 단일 코어 CPU 기반 알고리즘은 참조 가능한 데이터를 읽어 오면서 매 프레임마다 제한된 개수의 모델을 선택적으로 사용하여 GMM 확률을 계산하기 때문에 속도가 빠른 편이다. Intel i7 Sandy Bridge 2600을 기준으로, 테스트 환경에서 22.1초가 소요되었다.

기존 CPU 기반 알고리즘과는 다르게, GPGPU를 이용한 알고리즘은 참조 데이터를 매 프레임마다 읽으면서 적은 양만 계산하는 것이 쉽지 않다. 또한, GPGPU에서의 최대 효율을 얻기 위해서는 병렬화를 통해 모든 리소스를 활용하는 것이 중요하다. 따라서 본 연구에서는 매 프레임마다 6647개 모델 전체를 계산하는 방법으로 진행하였다.

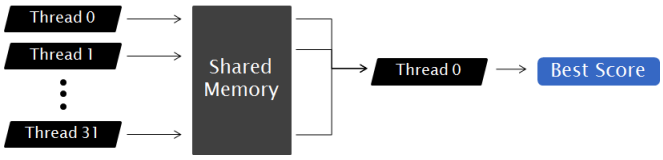
2.3.2. 결합전송 구현

결합전송은 GPGPU를 효율적으로 활용하기 위한 중요한 요소들 중 하나이다. 이는 소프트웨어적 개념인 블록과 대응되는 하드웨어 상의 기능적 단위인 Streaming Multiprocessor (SM)에서 글로벌 메모리의 데이터를 읽어올 때, 메모리 대역폭을 최대로 이용하는 방법이며, 이를 위해선 몇 가지 제약 조건을 만족시켜야 한다. 최신 GPU에서는 제약조건이 많이 완화되었지만, 제약 조건을 만족시키도록 하는 것이 어렵지 않았기에 이를 적용하였다. 이는 데이터가 32, 64 또는 128 bit형이어야 하고, 스레드(thread)에서 접근하는 데이터의 주소가 순서대로 연결되어야 하는 것이다. 만약 결합전송 가정이 성립되지 않으면, 한 번에 옮길 수 있는 데이터를 여러 번에 걸쳐서 옮겨야 한다. 글로벌 메모리에 접근 하는 속도가 매우 느린 것을 생각하면, 이를 방지하는 것은 매우 중요하다. 따라서 이에 맞추어 데이터 구조를 변경하고, 알고리즘을 만들었다.



(그림 2) 가우시안 모델 데이터 구조

그림 2에는 각 모델의 데이터 구조가 어떤 식으로 이루어져 있는지를 보여준다. 각각의 데이터는 32-bit 부동소수점 형식으로 표현되었다. 모델에는 학습과정에서 미리 계산된 $Gconst$가 32개, 평균 μ와 Ivar는 32개씩 총 36번, 차례대로 이어져 있으며, 앞의 index는 차수, 뒤의 index는 가우시안 함수의 index이다. 이렇게 구조를 변경함으로써, 32개의 스레드에서 모델의 데이터에 접근할 때, 한번에 32개 데이터씩 읽어올 수 있어서, 메모리 대역폭을 최대로 활용할 수 있다.



(그림 3) 알고리즘 개요

그림 3에 나타난 것처럼, 32개의 스레드에서 각각 점수(score)를 계산하여 공유메모리(shared memory)에 저장한다. 이 이유는 계산된 점수들을 하나의 스레드에서 다시 읽어 들인 후, 최적점수(Best Score)를 계산하여야 하기 때문으로, 상대적으로 접근 속도가 빠른 공유 메모리에 저장을 하였다. 결과적으로 1개의 스레드에서 저장된 32개 score 중 최솟값을 찾아내게 된다.

2.3.3. 하드웨어 사양 비교

연구 초기에 적용한 모델인 GeForce GTX 550Ti와, 이후 개선된 사양인 GeForce GTX 680을 비교하였다. GTX 550Ti는 CUDA core개수가 192개이며, SM개수는 4개이다. 이에 비해 GTX 680은 CUDA core가 1536개, SM 개수가 8개로, 동일 알고리즘에서, 성능이 CUDA core 개수에 비례해서 (x8) 상승될 것으로 예상되었다. 하지만 실제 성능 차이는 약 2배로 (30s -> 15s), 처음 예상했던 8배에 크게 못 미쳤다. 이는 동일 알고리즘에서 하드웨어만 바꿀 때 영향을 미치는 것이 CUDA core 수가 아니라 SM 수라는 것을 의미한다.

<표 1> GTX 550Ti와 GTX 680 spec 비교

	GTX 550Ti	GTX 680
CUDA core 수	192	1536
SM 개수	4	8
GPU Clock(MHz)	900	1006
Memory 밴드폭(GB/s)	98.4	192.2

2.3.4. 지연시간 숨기기

GPU자체는 CPU와는 분리된 개체이기 때문에, GPGPU 사용 시, 이는 하나의 하이브리드 시스템을 형성한다. 일반적으로 CPU와 GPU사이에 데이터를 주고받는 것을 제외하면 각자 독립적으로 프로세스를 수행할 수 있다. 즉, GPU에서 계산중일 때에 CPU에서 다른 프로세스를 비동기적으로 처리할 수 있다. 따라서 GPU에서 GMM 확률 계산을 진행 하는 동안, CPU에서는 MFCC 특성 벡터를 읽어 들이도록 하였다. 이를 통해서, 특성 벡터를 읽어 들이는 시간을 숨겨 성능을 향상시킬 수 있었다.

2.3.5. 루프 언롤링(loop unrolling)

루프 언롤링은 최신의 컴파일러에서 내부적으로 구현되어 있다. 하지만 현재 알고리즘의 루프 구문이 복잡하고, 여러 변수를 사용하기 때문에 직접 루프를 풀어주는 것이 좋았다. 가장 큰 루프는 2군데로, 각 모델의 score를 계산하는 부분과 이를 통합해 하나의 Best Score를 계산하는 부분이다.

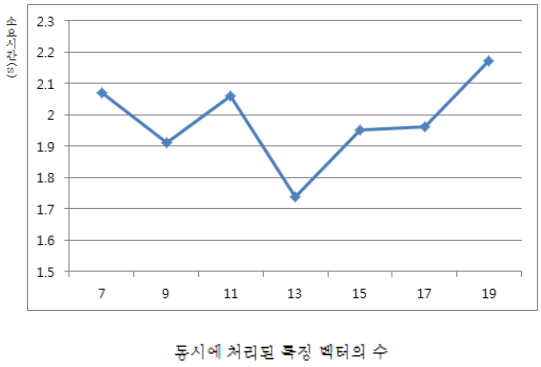
2.3.6. 글로벌 메모리 접근 줄이기

GMM 확률 계산 과정은 계산 자체에 걸리는 시간보다는 글로벌 메모리에 저장되어있는 데이터들을 읽어 들이는 데 시간이 매우 많이 걸린다. 때문에 글로벌 메모리에 접근하는 수를 줄이는 것은 성능 향상 측면에서 아주 중요하다. 이를 달성하기 위한 방법은 두 가지가 있는데, 먼저, 블록 당 처리하는 모델 수를 증가시켜 글로벌 메모리에서 특징 벡터를 읽어 들이는 수를 줄이는 것과, 다음으로, 다수의 특징 벡터를 한번에 처리하여 글로벌 메모리에 존재하는 모델에 접근을 줄이는 것이다.

기본적으로 NVIDIA의 Guide에서는 블록 당 스레드의 개수를 32 혹은 64의 배수로 만드는 것을 추천한다. GTX 680의 경우, 하나의 SM당 CUDA core의 수는 192

개로, 이를 최대한으로 활용하기 위해서는 6개의 모델을 하나의 블록에서 동시에 처리하는 것이 이상적이다. 하지만 블록 당 처리하는 모델 개수가 많아질수록 뱅크충돌이 많이 일어나 성능이 저하되었다. 따라서 5개의 모델만 하나의 블록에서 처리하도록 하였다.

5개의 모델을 하나의 블록에서 처리하면 총 160개의 스레드에서 연산이 일어나게 된다. 이에 추가적으로 여러 개의 특징 벡터를 동시에 처리하는 데, 스레드의 개수는 늘리지 않고 스레드에서 수행하는 연산을 늘렸다. 결과적으로 각 스레드에서 동시 처리하는 특징 벡터의 개수가 13개라고 하면, 공유메모리에 스레드마다 score를 13개씩 순서대로 저장하고, 65개의 스레드에서 이를 읽어서 각 특징 벡터와 각 모델 당 최솟값을 찾아낸다. 즉, 하나의 블록에서 65개의 Best Score를 출력하는 것이다. 공유메모리에 저장되는 데이터의 순서를 고려하면, 동시에 처리하는 특징 벡터의 수가 홀수일 때, 자동적으로 뱅크충돌이 약간 해제되는 효과가 있어 성능이 더 좋았다.



(그래프 1) 동시에 처리된 특징 벡터 수에 따른 계산 소요시간

그래프 1에 나타난 소요시간은 동일 알고리즘을 10번 내지 30번 반복하여 나온 값들의 평균이다. 동시에 처리되는 특징 벡터의 수는 모두 홀수만 사용하였다. 15개 혹은 17개에서 최대 성능을 예상했었지만, 실제 실험 결과 13개에서 소요시간이 1.72초 정도로 가장 좋은 성능을 보였다.

<표 2> 기존 CPU 대비 개선된 CUDA 알고리즘의 소요시간 및 성능 향상도

	기존 CPU Alg.	CUDA Alg.
소요시간(sec)	22.1	1.72
성능 향상도	x1	x12.85

결과적으로 표 2에 나타난 것처럼, 기존 CPU 알고리즘 대비 12.85배 성능 향상을 얻었다.

3. 결론

CUDA로 결합전송, 글로벌 메모리 접근 줄이기 등을 이용한 알고리즘을 구현함으로써, 기존 CPU 알고리즘 대비 약 13배 정도의 성능을 개선하였다. 이 외에도 성능에 큰 부분을 차지하는 것은 아니지만, Best Score 계산과정에서 하나의 스레드가 아닌, 여러 개의 스레드를 이용한 Reduction 알고리즘을 사용한다면 추가적인 성능 향상을 기대할 수 있다[4][5].

본 연구는 GMM 확률 계산 부분에만 집중하여 성능을 향상시켰다. 하지만 실제 음성인식 어플리케이션의 경우, 매 프레임마다 10ms 정도의 지연이 발생하기 때문에, 이를 고려하여 최고의 성능을 내기 위해서는 추가적인 연구가 필요하다.

감사의 글

본 연구는 한국전자통신연구원(ETRI) 자동통역지식처리연구센터의 Open R&D 프로그램의 일환으로 수행되었습니다.

참고문헌

[1] “CUDA C Programming Guide”, NVIDIA, 2012
[2] “CUDA C Best Practice Guide”, NVIDIA, 2012
[3] “GeForce GTX 680 Whitepaper”, NVIDIA, 2012
[4] P. Cardinal, P. Dumouchel, G. Boulianne, M. Comeau, “GPU Accelerated Acoustic Likelihood Computations”, Interspeech, 2008
[5] M. Harris et al., “Optimizing parallel reduction in CUDA”, NVIDIA, Tech. Rep., 2007