GPGPU 에서 쓰레드 구성을 위한 성능에 관한 연구

김현규 [1], 이효종 [1,2]
전북대학교 컴퓨터공학과 [1], 영상정보신기술연구센터 [2]
e-mail : hkskyp@gmail.com
e-mail : hlee@chonbuk.ac.kr

A Study on GPGPU Performance for the Configurations of Threads

Hyun Kyu Kim[1] , Hyo Jong Lee[1,2]
1. Dept. of Computer Science and Engineering, Chonbuk National University
2. Center for Advanced Image and Information Technology

요 약

최근 GPGPU 를 활용한 병렬처리가 각광을 받고 있는 가운데 GPU 의 구조적 특성인 매니코어(many-core)기반에서 쓰레드(thread)의 구성이 성능에 얼마나 영향을 미치는지에 관해 수치적 해답을 얻고자 하였다. 이는 멀티코어(multi-core)기반으로 작성된 프로그램을 GPGPU 로 변환하는 과정에서 쓰레드의 최대활용도를 빠르게 추측 할 수 있도록 도움을 얻고자 하는데 일차적인 목적이 있다. 현재 GPGPU 의 쓰레드 구성은 입력되는 데이터의 양을 고려하여 충분한 테스트를 거쳐 경험적인 최적화 수를 지정해 주워야 한다. 이번 연구를 통해 GPGPU 로 변환하는 과정에서 최적의 쓰레드 수 구성 방법을 추측 할 수 있으며 더 나아가 동적으로 최적의 수를 구할 수 있도록 하는데 목적이 있다.

1. 서론

최는 GPU 설계 구조를 활용한 병렬처리 시스템이 많은 연구에서 활용되고 있다. 여기서 활용되는 기술을 GPGPU(*General-Purpose computation on Graphics Processing Units)*라고 하는데 CPU 에서 처리하던 응용프로그램들을 GPU 에서 실행 할 수 있도록 하는 기술이다. 그 기술을 활용하는 방법에는 OpenCL, CUDA (Computer United Device Architecture)등이 있다.[1] GPU 는 전통적인 매니코어 기반 시스템이다. 이것은 CPU 에서 사용되는 멀티코어 시스템과는 차이를 보이는데 이 부분이 CPU 에서 실행되는 프로그램을 GPU 에 활용함에 있어 여러 가지 문제점을 발생한다.[2] 그 예로 멀티코어에서 최적의 쓰레드 수와 매니코어에서 최적의 쓰레드수가 서로 다르다는 것이다. 이는 해당 프로세서의 구조적 특징에 따른 것인데 GPU 에서는 보통 수백에서 수천 개의 쓰레드를 동시에 실행 시킬 수 있는 반면 CPU 에서는 수개의 쓰레드를 동시에 실행 시킬 수 있기 때문이다.

여기서 중요한 점은 GPU 에서 실행되는 쓰레드의 수가 최댓값을 가지도록 커널 함수를 호출하는 방법이 GPU 의 성능에 따라 다양한 구성이 존재하므로 GPU 에서 실행될 수 있는 쓰레드 수가 최댓값을 가질 수 있도록 구성만 한다면 최대 쓰레드 수를 가진 경우에는 모두 동일한 성능을 나타내는가 하는 것이다. 따라서 다양한 방법으로 최대 쓰레드 수를 구성하여 실행 하였을 경우 GPU 의 성능의 변화를 관찰하여보고 어떠한 방법으로 GPU 에서 실행 하였을 때 성능이 최대를 보이는지에 관한 문제를 해결하고 GPGPU 를 활용함에 있어 최대 성능을 얻기 위한 쓰레드 구성의 근사적 최적 값 알아내고자 연구를 하였다.

2. 영상처리에서의 병렬화 작업

CUDA 에서 동일한 쓰레드의 수에 따른 병렬화 성능을 고찰하기 위하여 의료영상에서 자주 활용되는 Binary Threshold Image Filter 를 사용하였다. 이 필터는 입력된 데이터의 상한 값과 하한 값으로 나누어주는 기능이다. 여기에 임계 값에 포함된 데이터와 그렇지 않는 데이터를 이진형식으로 분류 해주는 필터가 Binary Threshold Image Filter 이다.[3] 이를 간략히 표현하면 수식(1)과 같다.

$$Output(x_i) = \begin{cases} V_{in} & (Lower \le x_i \le Upper) \\ V_{out} & (x_i < Lower,\ Upper < x_i) \end{cases} \quad \cdots (1)$$

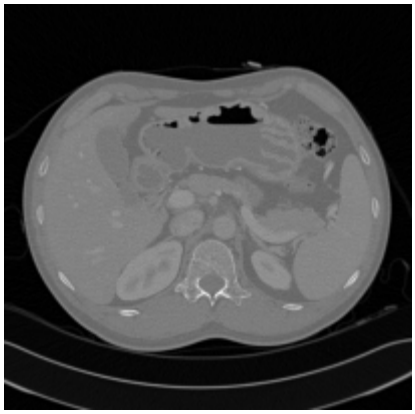
V_{in} 과 V_{out} 는 이진 형식으로 출력되는 값이며 Lower 는 하한값, Upper 는 상한값을 나타낸다..

그림 1 은 의료영상에서 사용되는 컴퓨터 단층 촬영 장치로 촬영된 이미지로써 512x512 화소에 -1024 ~ 1024 의 데이터 범위를 가진다.

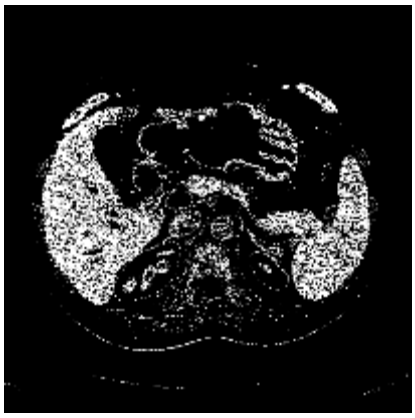
그림 2 는 그림 1 을 사용하여 하한 값과 상한 값을 각각 100, 150 으로 지정하고 임계영역에 포함되는 값과 포함되지 않는 값을 각각 255, 0 으로 지정하여 출력된 결과이다.

수식(1)에서 입력데이터와 출력데이터가 일대일 관

계를 가지고 의존적인 데이터를 가지지 않으므로 각 화소 별로 쓰레드를 생성하여 출력 값을 얻는 병렬화 작업이 가능하다.



(그림 1) 512x512 크기의 의료영상 이미지



(그림 2) 하한값과 상한값으로 이진화를 적용한 결과

3. CUDA 에서 최대 쓰레드 수 결정

CUDA 는 커널함수를 호출 함으로써 실행이 되는데 이때 실행될 그리드(Grid)공간과 블록(Block)공간의 크기를 지정해야 한다. 각 공간은 3 차원으로 구성되며 그리드의 각 차원에는 블록의 개수를 블록공간의 각 차원에는 쓰레드의 개수를 지정한다.

커널이 한번 호출되어 수행될 때 생성되는 쓰레드들을 모두 통칭하여 그리드라고 한다.

하나의 블록은 와프(Warp)라는 물리적 단위로 나뉘어 진다. 와프를 이용하여 전역메모리 접근시간 지연을 감출 수 있다. 블록은 스트림 멀티프로세서(Stream Multiprocessor)에서 실행되는 단위이다.[4]

GPU 에서 지원하는 CUDA 쓰레드를 최대로 활용하기 위해 필요한 계산식은 크게 두 단계로 나뉘어진다. 첫 번째 단계는 블록 공간의 크기를 지정하는 것이고 두 번째 단계는 그리드 공간의 크기를 지정하는 것이다. 블록 공간에서 크기는 지정 하는 방법 (2)와 같다.

$$T_B = \{x = T_{SM}/B_n \mid x \in N, x < max(T_B)\}$$
$$1 \le B_n \le TB_{SM} \quad \text{.....(2)}$$

여기서 T_{SM}은 하나의 스트림 멀티프로세서가 지원하는 최대 쓰레드 수 이고 TB_{SM}은 하나의 스트림 멀티프로세서가 지원하는 최대 블록의 개수이다.

$max(T_B)$는 단위 블록당 지정할 수 있는 쓰레드의 최대값이다. 그리드 공간의 크기 지정은 (3)과 같다.

$$B_G = \{\lceil G/T_B \rceil\} \quad \text{.....(3)}$$

G 는 그리드이며 입력데이터를 처리하기 위해 필요한 총 쓰레드 개수의 자연수배 이여야 한다. $\lceil x \rceil$는 자연수를 취하는 상계이다.

T_B는 블록공간에서 각 차원에 대입하고 B_G는 그리드 공간에서 각 차원에 대입해야 한다. 각 차원에 대입하기 위해서는 (4)를 만족해야 한다.

$$N = xyz \quad \text{.....(4)}$$

이 소인수분해 문제는 현재 NP 문제로 알려져 있지만 B_G 크기 정도의 소인수분해는 실용적인 알고리즘들이 많이 개발되었다.[5] 소인수분해를 하지 않을 경우 입력된 데이터를 처리하기 위해 필요한 쓰레드 수 보다 CUDA 에서 실행되는 그리드가 더 많아지게 되어 처리시간이 증가하게 된다.

T_B의 값이 CUDA 정의에 의해 2 의 배수가 나오므로 소인수분해 문제에 어려움이 발생되지 않으므로 필요한 경우에만 하면 된다.

실험에 사용된 GPU 는 GeForce GTX 550 Ti 이며 T_{SM} = 1536, TB_{SM} = 8, $max(T_B)$ = 1024 값을 가진다. 이 값들은 사용되는 GPU 에 따라 정해져 있다.[6]

4. 실험 및 고찰

이번 연구에서는 CPU 기반에서 높은 병렬 확장 성을 가진 Threshold Binary Image Filter 를 사용하여 CUDA 에서 활용하였을 때, 실행되는 CUDA 쓰레드의 수를 최댓값으로 구성 시키는 각각의 경우에 대해서 실행시간을 비교하였다.

Threshold Binary Image Filter 의 경우에는 레지스터 메모리, 공유 메모리, 전역 메모리 등의 메모리 접근 회수가 0~2 번에 불과하므로 메모리 접근 지연에 따른 성능 감쇠현상을 줄일 수 있으므로 CUDA 쓰레드의 경합으로 생기는 문제를 구체적으로 파악하기에 적당하다고 판단하였다.

입력데이터로 사용된 이미지의 한 장의 크기는 512 x 512 화소이며 컴퓨터 단층 촬영 장치를 이용하여 의료영상을 1mm 간격으로 촬영 할 경우 일반적으로 200 여장 이상의 이미지가 생성되는데 이번 실험에 사용된 이미지는 245 장 이다. 여기서 입력데이터 크기에 따른 속도를 비교하기 위해 임의로 35 장을 추출하였는데 이것은 7mm 로 촬영하였을 때 생성되는 이미지 수와 비슷하다.

실험을 위해 CUDA 에서 최대 쓰레드 수 결정은 245 장, 35 장 일 때 표 1 과 같다.

표 1 과 같이 구성하게 되면 입력데이터를 처리할 때 필요한 그리드 수가 192 일 때는 64 만큼 384, 768 일 때 256 만큼 보다 더 늘어나게 되는데 이 수치는 해당 T_B 보다 작은 값 이므로 블록 단위

로 실행되는 CUDA 에 영향을 미치지 않는다. 그리고 245 장 이미지에서 T_B 가 192 일 때 B_G 값이 334,507 이 되는데 그리드 공간의 축에 설정 할 수 있는 최대 개수보다 큰 소수가 되어 소인수분해가 되지 않았기에 실험결과에서 제외하였다.

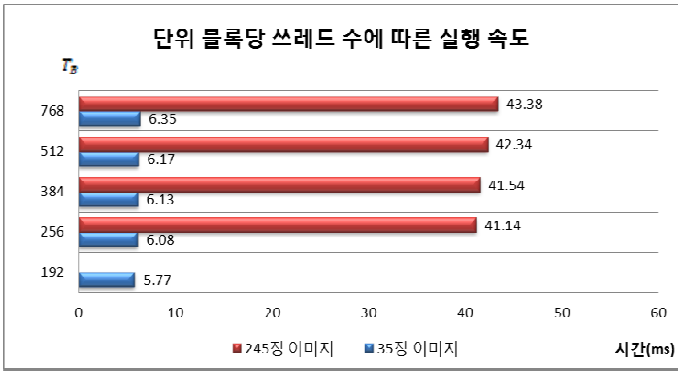
<표 1> 블록공간과 그리드공간

T_B	B_G	
	35 장	245 장
192	(937, 17, 3)	-
256	(2^{10}, 7, 5)	(2^{10}, 7^2, 5)
384	(937, 17, 2)	(347, 241, 2)
512	(2^9, 7, 5)	(2^9, 7^2, 5)
768	(937, 17, 1)	(347, 241, 1)

그림 3 은 T_B 의 각 원소를 35 장 이미지와 245 장 이미지를 GPU 를 이용하여 처리하였을 때 얻은 결과이다. 평균적인 결과값을 얻기 위해 각 원소를 항목별로 10,000 번씩 반복 실행하여 얻은 평균 값이다.

실험 결과 T_B 의 값이 작을 수록 가장 좋은 성능을 보여주는 것으로 나타났다. T_{SM} 의 값이 모두 동일한 값을 가지도록 한 실험에서 T_B 에 따른 성능이 약 4%~10%까지 차이를 보임을 알 수 있다. 결과에 따르면 B_n 의 값이 TB_{SM} 에 가깝게 그리드공간과 커널공간을 지정하고 커널 함수를 실행 시킬 때 최대 성능을 발휘한다는 것을 의미한다.

여기서 주목할 점은 메모리의 지연을 감추는데 활용하는 와프의 수가 많을수록 반드시 성능효과를 볼 수 없다는 것이다. 이번 연구에 사용된 커널 함수가 전역메모리에 접근하는 회수가 2 번뿐이기 때문에 와프 수 증가가 오히려 성능을 감소시키는 결과를 가져왔다. 만약 커널 함수에서 전역 메모리 접근이 빈번하다면 와프 수가 많을수록 성능이 향상 될 것이나 CUDA 에서는 전역 메모리 접근을 줄이기 위한 다양한 방법이 존재하므로 커널 함수에서 전역 메모리 접근이 빈번하다는 이유로 와프 수를 증가 시키기보다는 전역 메모리 접근을 줄이는 방법을 찾는 것이 올바른 방법이다.



(그림 3) 단위 블록당 쓰레드 수에 따른 실행 속도 비교

5. 결론

CUDA 에서 커널 함수를 호출하여 쓰레드 자원을 최대로 활용하도록 구성하는 방법은 다양하다. 이때 각 구성에 따른 성능의 차이를 알아보기 위해 컴퓨터 단층 촬영 장치에서 촬영된 의료영상을 입력데이터로 사용하여 임계 범위의 값을 이진화 하는 영상처리 기법을 적용하였다. 그리고 CUDA 에서 최대 쓰레드 수를 구성하도록 하는 방법으로는 소인수분해를 활용하여 그리드 공간을 효율적으로 구성 할 수 있도록 제시하였다. 실험 결과 전역 메모리 접근과 T_B 구성에 따라 GPU 의 성능에 영향을 미친다는 결론을 얻을 수 있었다. 전역 메모리 접근이 적을 경우에 T_B 의 크기가 작을수록 성능이 좋은 것으로 나타났다. 따라서 연구결과를 기준으로 다른 GPU 를 활용할 때 성능의 근사적 최적 값을 예측 할 수 있으며, 서로 다른 GPU 환경에서 최대 활용도를 가지고자 할 때 기준으로 제시 할 수 있다. 또한 CUDA 를 이용했을 때 메모리 접근 지연에 따른 성능감소가 발생하는지를 파악하는데 도움을 줄 수 있다.

참고문헌

[1] GPGPU. http://www.gpgpu.org.

[2] Krik DB, Hwu W-mW. "병렬 컴퓨터로써의 GPU", ' 대규모 병렬 프로세서 프로그래밍 : CUDA 를 이용한 실용적 접근', 비제이퍼블릭, 2011. p. 20-25.

[3] ITK. Binary Threshold Image Filter, http://www.itk.org/Doxygen/html/classitk_1_1BinaryThresholdImageFilter.html.

[4] NVIDA. CUDA_C_Programming_Guide, 2011, http://www.developer.nvidia.com/nvidia-gpu-computing-documentation.

[5] Knuth DE. "소인수분해", 'The Art of Computer Programming', 3th ED., 한빛미디어, 2007. p. 436-479.

[6] NVDIA. Compute Capability, 2011, http://developer.nvidia.com/category/zone/cuda-zone.