GPGPU 를 이용한 양 방향성 필터의 병렬 구현 및 성능 평가

장기준*, 노원우*
*연세대학교 전기전자공학과
e-mail : kijoon901@yonsei.ac.kr, wro@yonsei.ac.kr

Efficient Parallel Bilateral Filter using GPGPU

Ki Joon Chang*
Won Woo Ro*
*Dept. of Electrical and Electronic Engineering, Yonsei University

요 약

양 방향성 필터는 이미지표면 평탄화와 잡음제거에 좋은 성능을 보이지만 특유의 연산 복잡도로 인하여 연산 시간이 오래 걸린다는 단점이 존재한다. 따라서 본 논문에서는 고도의 병렬수행을 바탕으로 하는 그래픽연산장치(GPU)에 적합하도록 수정된 효율적인 양 방향성 필터를 NVIDIA 의 CUDA 를 사용하여 GTX 285 GPU 에서 구현하였다. 영상의 전 영역을 참조하는 대신 인접하고 연속된 영역으로의 근사화, 적은 메모리 사용량, 빠른 접근속도를 가지며 충돌이 최소화된 공유메모리 버퍼, Warp 를 고려한 병합된 메모리 접근방법을 바탕으로 병렬화 하였다. 그 결과, 같은 방식의 순차실행 알고리즘 대비 최소 약 34 배에서 최대 약 76 배의 속도 개선과 30dB 내외의 PSNR 을 갖는 양 방향성 필터를 구현할 수 있었다.

1. 서론

영상이나 이미지의 잡음을 제거하는 방법들 중 비교적 간단하면서도 좋은 성능을 보이는 방법으로 양 방향성 필터(Bilateral Filter)가 제안 된 바 있다[1]. 양 방향성 필터는 이미지의 표면을 평탄화 한다는 점에서는 가우시안 필터와 유사하나 영상의 경계면을 보존한다는 점에서 차이가 있다. 이 필터는 픽셀간의 거리와 값에 가중치를 갖는 두 개의 가우시안 커널의 곱으로 구성되는데, 같은 필터 커널 윈도우 내에서라도 거리가 가깝고, 픽셀 값의 차이가 적을수록 더 많은 가중치를 부여 받고, 거리가 멀고 픽셀 값의 차이가 클수록 기준이 되는 픽셀의 값에 영향이 적다는 점을 이용한다.

$$k = \sum_{i_p=-\infty}^{\infty}\sum_{i_q=-\infty}^{\infty} G_I(I_p - I_q)G_s(S_p - S_q)$$

(Normalization factor)

$$I_p = \frac{1}{k}\sum_{i_p=-\infty}^{\infty}\sum_{i_q=-\infty}^{\infty} G_I(I_p - I_q)G_s(S_p - S_q)\, I_p$$

<수식 1> 양방향성 필터

이러한 양 방향성 필터는 비교적 간단하게 구현할 수 있기 때문에 많은 분야에 사용되고 있으나, 한 정점의 필터링 된 값을 구하기 위하여 이미지 내의 모든 정점과의 거리와 픽셀 값의 차이를 계산해야 하는 특성 때문에 처리 속도가 매우 느리다. 또한, 양방향성 필터를 순차수행을 특징으로 하는 기존의 CPU 기반으로 처리하면 특유의 연산복잡도와[2] 가우시안 함수로 인한 중첩된 초월함수 연산 때문에 실시간으로 양 방향성 필터의 적용이 필요할 경우에는 그 사용에 한계점을 지닌다.

반면, 그래픽연산장치(GPU, Graphic Processing Unit)는 CPU 와는 달리, 그래픽연산의 특성상 SIMD(Single Instruction, Multiple Data)기반의 고도로 병렬화된 영상처리에 적합한 특성을 갖고 있다. 이러한 성능을 일반적인 과학연산에 이용하려는 움직임이 2006 년말 NVIDIA 의 G80 아키텍처와 함께 소개된 CUDA 를 시작으로 현재의 OpenCL, Direct Compute 등과 함께 GPGPU(General Purpose Computing on GPU)라는 이름으로 꾸준히 발전하고 있다. 특히, NVIDIA 의 CUDA 는 발표된 이래 꾸준한 지원과 기존의 Shading Language 대비 사용이 매우 편리하다는 점 때문에 GPGPU 아키텍처들 중 가장 보편적으로 사용되고 있다. CUDA 는 SIMT(Single Instruction, Multiple Threads)라는 새로운 개념을 정립하였고, 다양한 메모리 계층, C/C++과 유사한 API 들을 제공하여 사용자가 컴퓨터 그래픽 파이프라인에 대한 지식 없이도 편리하게 이를 이용 할 수 있도록 하였다.

기존의 CPU 보다 압도적인 연산장치들이 집적되어 있는 GPU 이지만, 적합한 입력 데이터 병렬성의 확보와 GPGPU 아키텍처에 맞는 알고리즘 최적화, 그리고 CPU - GPU 간 데이터 전송의 문제 등을 해결하지 못

하면 충분한 성능향상을 이룩할 수 없다[3, 4]. 따라서 본 연구에서는 CPU - GPU 간 데이터 전송량과 메모리 사용을 최소로 하면서, 빠른 처리 속도를 갖는 효율적인 양 방향성 필터를 NVIDIA 의 CUDA 를 사용하여 구현하고자 한다.

2. 관련연구

서론에서 언급한 바와 같이, 많은 연산량은 양 방향성 필터의 속도를 저하시키는 주요한 원인 중 하나이기 때문에 많은 선행연구들이 이를 줄여나가는 방향으로 이루어져 왔다. 전체의 이미지를 참조하는 대신, 연산하려는 픽셀과 거리가 가까운 픽셀만으로 필터링하는 방법을 Pham 과 Weiss 가 제안하였다[5, 2]. 이는 인접하는 픽셀을 취하는 방법이 2 차원 또는 1 차원인 것으로, 멀리 떨어진 곳의 픽셀은 영향을 적게 끼친다는 사실을 이용한 일종의 근사치를 사용한 방법이다.

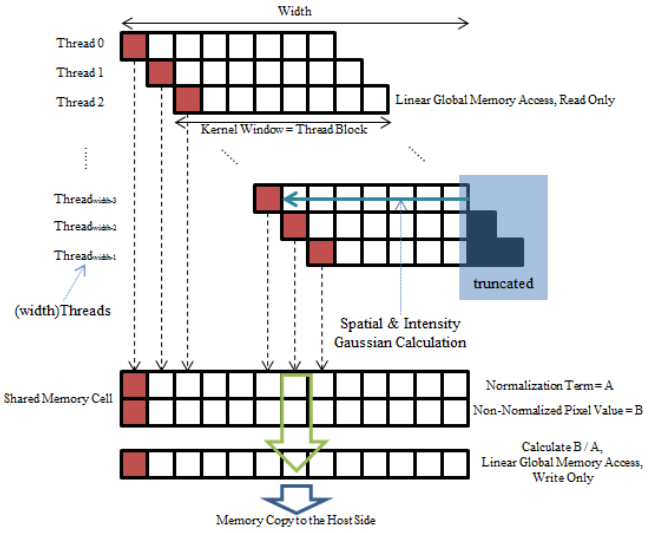
또한, 필터 커널 윈도우의 크기나 모양을 조정하는 것 이외에도 반복적인 초월함수 연산을 감소시키기 위한 Histogram 이나 Estimator 에 의한 방법[6], 양 방향성 필터의 연산 식을 일종의 Convolution 으로 취급하여 속도를 향상시키는 방안도 소개되었다[7].

3. 제안하는 방법

NVIDIA 의 CUDA 아키텍처는 CPU 와 마찬가지로 수 개의 메모리 계층을 지닌다. 그 중, 속도가 가장 느리지만 저장용량이 가장 큰 전역메모리(Global Memory)에의 접근은 32 개 스레드의 집합체인 Warp 를 기본단위로 한다[3]. Warp 는 다른 Warp 와 데이터 의존성이 없으며, 하나의 Streaming Multiprocessor(SM)에서 동시에 수행되는 스레드의 집합체이다. 따라서 전역메모리에의 접근도 이 단위를 기본으로 하게 된다. 32 개의 스레드가 연속적인 메모리의 영역에 접근할 때 그 속도가 가장 빠른 반면, 메모리 접근 패턴에 Stride 가 존재하거나 정렬되지 않은 메모리 접근을 하게 될 경우 그 성능이 저하 될 수 있다. 또한 CPU 에서 프로그램 제어권을 GPU 로 넘겨주기 전에 필터링에 필요한 데이터를 전역메모리의 영역에 옮겨야 하는데, 이 때 사용할 수 있는 CPU - GPU 간 통신의 대역폭에는 제한이 따른다.

CUDA 아키텍처에서는 CPU 에서 찾아볼 수 없는 Programmable SRAM 인 공유메모리(Shared Memory)를 제공하는데, 이를 연산 중간에 일종의 버퍼로 활용한다면 DRAM 인 전역메모리에 접근하는 것보다 더 빠른 속도를 얻어낼 수 있다. 공유메모리는 같은 스레드 블록과 그 데이터를 공유하며, 이를 이용하여 연산 중간결과를 저장해두거나 정규화에 필요한 값을 저장해 둘 수 있다. 그러나 동일 블록내의 다른 스레드가 같은 공유메모리 셀에 접근할 경우, 그 데이터 접근은 의도했던 바와 같이 병렬화되지 못하기에 이를 최소화 시키는 노력이 필요하다.

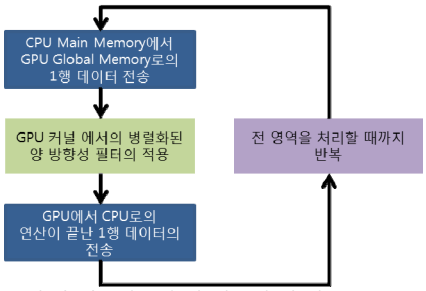
이러한 점을 바탕으로 할 때, 가능한 한 스레드들이 연속적인 메모리 영역에 접근하도록 하는 알고리즘, 경쟁상태를 최소로 하는 공유메모리의 사용, 그리고 전역메모리의 사용량을 최소화하는 것이 필요하다. 이 사실을 바탕으로, 본 논문에서 제안하는 GPU 에 적합한 양 방향성 필터의 알고리즘이 다음 (그림 1)에 표현되어 있다.



(그림 1) 제안된 양방향성 필터의 처리과정

제안된 구조에서는 하나의 픽셀을 계산하는데 하나의 스레드를 할당하였다. 각각의 스레드는 정규화에 필요한 값과 정규화되지 않은 픽셀의 값을 저장하는 공유 메모리 버퍼를 2 개씩 가지며, 다른 스레드에 의해 접근 받지 않는 고유한 영역이다. 이는 (그림 1)에서 B / A 로 표현된 연산의 최종 단계에서 공유메모리 경쟁상태를 최소로 하고, 나눗셈에 소요되는 비교적 긴 연산시간을 감내하도록 병렬성을 극대화하는 역할을 한다. 필터링에 필요한 영역의 크기(Pixel Window Size)는 스레드 블록의 크기와 동일하며, 연속적인 형태로 배치되어 있다. 또한 병합된 전역메모리 접근과 Warp 단위 명령어 수행을 감안하여 가급적 Warp 의 약수만큼의 크기를 갖도록 한다. 단, 커널 윈도우의 크기가 커질수록 필터링의 정확도는 높아지지만 계산 속도가 떨어지기 때문에 최적의 크기는 이미지의 해상도에 따라서 실험적으로 결정되어야 할 사항이다.

낮은 GPU 메모리 점유와 낮은 대역폭에서의 빠른 작동 보장을 위하여 CPU - GPU 간 데이터 전송은 선형적으로 배치된 행 단위로 이뤄지고, GPU 에서 진행되는 필터링 과정 또한 이 단위를 기본으로 한다. 따라서 GPU 에는 연속적으로 배치된 1 행의 픽셀 데이터를 저장할 메모리만 할당되어 병합된 메모리 접근, 데이터 전송량과 사용량의 최소화를 이루도록 하였다. 또한 (그림 2)에 나타난 것과 같이, 모든 영역에 대해 연산을 끝마치기 위해서는 이미지 높이만큼의 연산 반복이 필요하다.



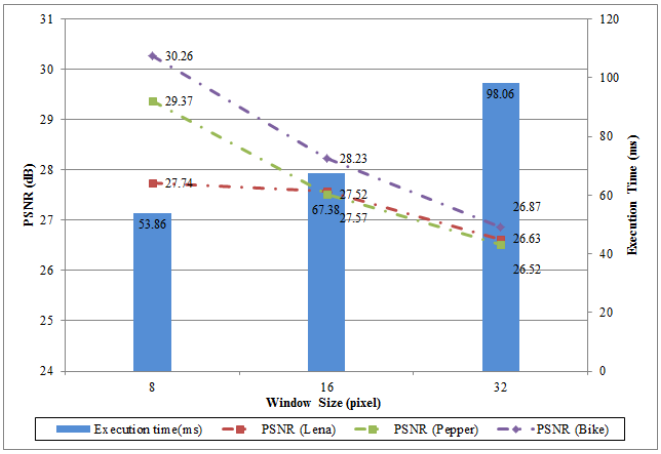
(그림 2) 제안된 양 방향성 필터의 프로그램 흐름

4. 성능분석

제안된 알고리즘의 성능을 분석하기 위해 3 가지 이미지를 가지고 소요시간과 PSNR(Peak Signal to Noise Ratio)을 분석하였다. (그림 3)은 가로와 세로 각 512 픽셀, 24bit RGB 비트맵 이미지를 커널 윈도우 크기를 바꿔가며 제안된 양 방향성 필터로 1 회 반복하여 측정한 결과이다.

<표 1> 실험환경

CPU	Intel Xeon® E5440 2.83GHz
RAM	16GB DDR3
OS	Fedora Linux 10 (64 bit)
Tool Kit, SDK	3.0
GPU	NVIDIA GTX285



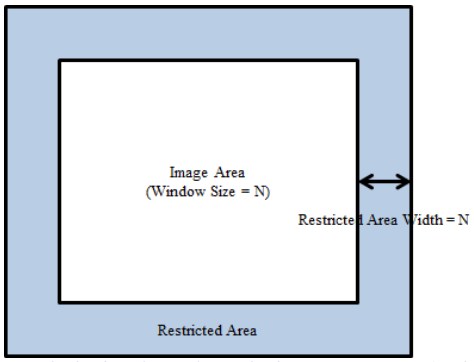
(그림 3) 제안된 양 방향성 필터의 소요시간과 PSNR [8]†

<표 2> 순차실행 CPU 대비 성능향상 폭††

Window Size	Relative Speed up(times)
8	33.98
16	53.59
32	75.75

커널 윈도우의 크기가 커질수록 복잡도가 기하급수적으로 증가하는 필터의 특성상 CPU 알고리즘 대비 속도 향상 폭은 점점 커지지만 유사한 정도의 필터링 효과를 가우시안의 표준편차를 조절함으로써 얻을 수 있기 때문에 제안된 알고리즘 에서는 커널 윈도우 크기가 8 인 것이 가장 적절하다고 판단된다. 이 값은 직선형의 커널 윈도우 모양을 갖는 Separable Kernel 방식에서 가장 효율적이라고 언급한 10 픽셀 이하라는 사실과 일치하기 때문에 충분히 타당한 결과로 여겨질 수 있다[5]. 이 보다 크기가 작아진다면 필터의 효과가 거의 없고, 더 커진다면 연산시간은 증가하지만 그 효율성이 떨어지기 때문이다.

윈도우의 크기가 커짐에 따라 PSNR 이 작아지는 것만 제외한다면 모든 설정 값에서 30dB 내외의 양호한 PSNR 값을 보여주고 있다. 이미지의 색 패턴에 따른 약간의 차이는 있지만 실시간 영상처리에 사용할 수 있는 유효한 필터링 성능을 보여주고 있음을 알 수 있다.



(그림 4) 제안된 알고리즘에서 커널 윈도우의 크기가 제한되는 영역

5. 제한사항 및 향후의 과제

이미지의 끝부분에서 스레드의 잘못된 메모리 영역 참조를 방지하기 위한 부분이 GPU 커널에 사용되었다. 이를 위하여, (그림 1)에서 잘려나가는 부분과 같이, 이미지의 경계에서 강제로 커널 윈도우 크기를 이미지 끝부분으로 제한하였다. (그림 4)에서 볼 수 있는 것과 같이 경계 면에서 참조하는 영역의 감소로 인한 제한적이고 부정확한 필터링이 일부 발생할 가능성이 있다. 또한 이를 위해 일부의 스레드는 제어 흐름이 다른 스레드와는 다른 분기를 사용해야만 했다. 이때, Warp 내부의 스레드 중 일부가 다른 스레드와 다른 코드를 실행하는 GPU 특유의 Warp Divergence 가 발생하여 잠재적인 성능하락을 불러올 가능성이 존재한다[9]. 이를 해결하기 위해 전역 메모리 영역 대신 이미지의 경계에 대한 고려가 불필요한 텍스처 메모리에 최적화된 알고리즘을 사용하는 것이 방법이 될 수 있다.

더 빠른 성능을 얻기 위해서는 소프트웨어 내부 파이프라인이 필요하지만 알고리즘 특성상 각 단계 사이에는 의존성이 존재하여 데이터 전송과 GPU 커널 실행을 중첩할 수 없다. 이는 복수의 GPU 를 사용하거나 CPU 와의 협업을 고려한 이종간 컴퓨팅(Heterogenerous Computing) 알고리즘에 대한 연구로써 해결될 수 있을 것으로 보인다.

참고문헌

[1]C. Tomasi and R. Manduchi, “Bilateral Filtering for Gray and Color Images”, *Proceedings of the 1998 IEEE International Conference on Computer Vision*

[2]Ben Weiss, “Fast Median and Bilateral Filtering”,

Proceedings of ACM SIGGRAPH 2006
[3]NVIDIA Corporation, "CUDA Programming Guide 4.0"
[4]NVIDIA Corporation, "CUDA C Best Practice Guide 4.0"
[5]Tuan Q. Pham and Lucas J. van Vliet, "Separable Bilateral Filtering for Fast Video Preprocessing", *IEEE International Conference on Multimedia & Expo (ICME'05)*
[6]Fr´edo Durand and Julie Dorsey, "Fast Bilateral Filtering for the Display of High-Dynamic-Range Images", *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2002*
[7]Sylvain Paris Fr´edo Durand, "A Fast Approximation of the Bilateral Filter using a Signal Processing Approach", *International Journal of Computer Vision archive, Volume 81 Issue 1, January 2009*
[8]www.compression.ru/video/quality_measure/info_en.html "MSU Quality Measurement Tool: Metrics information"
[9]W. W. L. Fung, Ivan Sham, George Yuan and Tor M. Aamodt, "Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow", *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture(MICRO '07)*

(그림 5) 처리 전의 원본 이미지

(그림 6) 제안된 방법으로 양 방향성 필터링 된 이미지, Window Size = 16, Gaussian Kernel 의 표준편차는 모두 공히 20

---

† [8]의 도구를 사용하여 PSNR 의 측정
†† 단일 스레드, 단일 코어, Intel SpeedStep®기능을 끈 상태에서 측정한 데이터를 사용하여 계산한 수치이다.