GPU기반 블록 정합의 단계적 최적화

최호열*, 박인규*, 김일목**, 김성훈 **
*인하대학교 로봇공학과/정보통신공학과, **LG전자 생산기술연구원

Step-by-Step Optimization of GPU-based Block Matching

Ho Yeol Choi*, In Kyu Park*, Ill Moek Kim**, Sung Hoon Kim**
*Inha University, **LG Electronics
{ *chlghduf314@gmail.com, *pik@inha.ac.kr }

Abstract: 영상간의 지역적인 유사도 판별에 사용되는 블록 정합 알고리즘은 기준 블록과 탐색영역 사이의 정합을 수행하여 지역적 유사도를 구한다. 알고리즘의 특성상 대용량의 영상데이터를 고속으로 처리해야 할 경우 모든 탐색영역에 대해 정합을 수행하므로 상당한 수행시간이 소요된다. 이러한 문제를 해결하기 위하여 GPU (graphics processing unit)의 대용량 병렬처리를 이용한 블록 정합 알고리즘의 단계적 최적화 방법을 제안한다.

Keywords: Template Matching, GPU, Optimization, Fermi

Ⅰ. 서론

컴퓨터 비전 분야에서 블록 정합 알고리즘은 지역적 유사도를 이용하는 물체 인식 및 추적[3], 변이지도 추출[4] 등의 여러 분야에서 사용되는 알고리즘이다. 블록 정합 수행 시 기준 블록과 탐색 영역간의 유사도를 판단하는 방법으로 여러 가지가 있지만 비교적 간단하며 많이 쓰이는 방식은 SAD (sum of absolute difference), SSD (sum of squared difference), NCC (normalized cross-correlation)등이 있다. 블록 정합 알고리즘은 정합 수행 시 인접 픽셀간의 데이터 의존성이 없어 GPU를 이용한 SIMD(single instruction multiple data)방식의 병렬처리 구조에 적합하다. 본 논문에서는 NVIDIA 페르미(Fermi) 아키텍처에 대한 이해를 바탕으로 알고리즘의 병렬성 증대, 전역 메모리의 데이터 캐쉥과 효율적 데이터 구조, loop unrolling 등을 통하여 CUDA (compute unified device architecture) 기술을 이용한 단계적 최적화 방법을 제안한다.

Ⅱ. 알고리즘

본 논문은 기준 블록과 탐색 영역간의 유사도를 판단하기 위해 SAD방식의 정합을 수행하였다. 입력 영상은 다수의 블록들로 나뉘며 각 블록은 주변 영역에 대해 정합을 수행한다.

1. 유사성 계산

입력 영상을 일정크기의 블록으로 나눈 뒤 각 블록과 주변의 일정 탐색 범위에 대해서 블록 정합을 수행하게 된다. 이 때 정합 대상이 되는 영상정보와 블록간의 유사성을 판단하는 방법은 식(1)과 같다.

$$\sum_{(i,j)\in W}\left|I_1(i,j)-I_2(x+i,y+j)\right| \quad (1)$$

2. 최대 유사성 탐색

식(1)에 의해서 계산된 유사도가 최대가 되는 위치를 탐색한다. 이 때 유사도 값들의 정렬 형태는 탐색 순서를 고려하여 데이터 캐쉥이 효율적으로 수행될 수 있는 구조로 재배치 한다.

Ⅲ. 단계별 최적화

블록 정합의 경우 유사도를 계산하는 과정에서 인접 픽셀간의 의존성이 없으므로 GPU의 특성을 이용한 대용량 병렬처리에 적합하다. 기준블록과 탐색영역의 정합수행 시 GPU의 SIMD 방식의 병렬처리 모델을 적용하면 CPU로 작성된 알고리즘 대비 많은 속도향상을 얻을 수 있을 것이다. 하지만 단순히 GPU의 SIMD구조에 적합하게 알고리즘을 작성하는 것 만으로는 최대의 속도향상을 얻을 수 없다. GPU의 물리적인 구조, 데이터 접근순서에 따른 최적화된 메모리 구조, 적절한 메모리 사용, 데이터 캐쉥 등에 대한 이해를 바탕으로 최적화를 수행 한다면 GPU를 이용하여 최대의 속도향상을 얻을 수 있을 것이다. 본 논문에서는 페르미 아키텍처에 대한 이해를 바탕으로 블록 정합 알고리즘의 GPU기반 병렬처리 구현의 단계별 최적화 방법을 제안하였다.

1. 병렬성 확장 (1단계)

입력 영상을 여러 개의 블록으로 나눈 뒤 각 블록마다 수행되는 블록정합 알고리즘은 다음과 같이 구성된다. 기준 블록과 주변의 탐색 범위내의 데이터에 대해 SAD방식으로 유사도를 계산한 뒤 이 값들 중에 최대값을 가지는 위치를 탐색하는 과정은 GPU의 한 개의 쓰레드가 담당을 하게 된다. 이와 같은 구현은 영상 내의 나뉘어진 블록의 개수만큼의 쓰레드를 발생시켜 병렬로 수행한다. CUDA 커널 함수 내부의 2차원 블록의 유사성 계산의 경우 다중 반복문을 포함하게 된다. 이러한 구현은 대상 블록의 개수가 충분히 많지 않은 경우 GPU내부의 다수의 스트리밍 멀티프로세서 중 일부분만 사용하게 되는 현상을 초래하게 된다. 보다 많은 병렬성을 확보하기 위해서 본 논문에서는 커널 내부

에 존재하는 반복문을 커널 외부로 확장하여 병렬성을 증대시키는 방법을 제안한다. 제안한 방법은 두 가지 이점을 가지고 있다.

첫째, 동시에 발생되는 쓰레드의 수가 증가하므로 더 많은 병렬성을 확보할 수 있다. 이는 알고리즘이 수행되는 동안 GPU의 자원을 최대로 이용할 수 있도록 한다.

둘째, GPU의 경우 개별적인 CUDA 코어의 처리속도는 일반적인 CPU에 비해 약 20% 정도의 정도이며 반복문, 분기문 등의 제어문에 대한 처리능력이 상대적으로 부족하므로 커널에서 이러한 논리연산의 부담을 줄여 대용량 병렬 처리에 적합한 구조로 변경함으로써 향상된 속도를 얻을 수 있다.

2. 전역 메모리와 데이터 캐싱 (2단계)

페르미 아키텍처 이전에는 데이터 캐싱에 대한 이점 때문에 입력 데이터 형태로 텍스처 메모리를 많이 사용하였다. 하지만 페르미 아키텍처의 경우 새로 추가된 L1, L2 캐쉬를 이용하여 전역 메모리에 대한 캐싱을 지원한다. L1 캐쉬의 경우 스트리밍 멀티프로세서 내부에 존재한다. 64KB의 크기를 공유메모리와 공유하며 128byte씩 캐싱을 수행한다. L2 캐쉬의 경우 768KB의 크기로 스트리밍 멀티프로세서 외부에 존재하며 32byte씩 캐싱을 하여 L1캐쉬에 비해 Over fetch를 줄일 수 있는 장점을 가진다. 이러한 특성을 이용하여 전역 메모리의 L1, L2캐쉬의 기능을 적극 활용하는 것이 속도향상에 많은 도움이 된다. 또한 데이터 캐싱이 효율적으로 발생할 수 있는 최적화된 데이터 배열도 필요하다.

3. 공유메모리 사용 (3단계)

On-chip memory를 이용하여 사용자가 자유롭게 사용할 수 있는 캐쉬의 한 종류인 공유메모리의 경우 페르미 아키텍처 에서는 최대 48KB까지 사용 가능하다. 전역 메모리의 접근속도에 비해 수백 배 빠른 접근속도를 가진 공유 메모리를 적극 이용하여 블록과 탐색영역의 데이터를 모두 공유메모리로 복사 한 뒤 지속적인 공유메모리로의 접근을 통하여 블록 정합을 수행한다.

4. Loop Unrolling (4단계)

반복문의 경우 반복 횟수가 고정되어 있거나 예상이 가능한 경우 unrolling을 수행 한다. 이는 개별 쓰레드(thread) 내의 논리연산에 대한 부담을 줄이며 cache hit ratio를 증가하게 함으로써 속도향상을 가능하게 한다. 하지만 모든 반복문을 unrolling한다고 해서 최적의 속도향상을 기대할 수 있는 것은 아니다. 접근하는 데이터의 배열, 캐쉬 구조 등을 고려하여 unrolling을 수행하여야 한다 또한 레지스터 사용량이 증가하기 때문에 스트리밍 멀티프로세서당 최대 레지스터 허용치를 고려하여야 한다.

Ⅳ. 실험결과

실험에 사용된 GPU는 NVIDIA GTX580이며 총 512개의 CUDA 프로세서를 가지고 있다. 개별 프로세서의 클럭 속도는 832 MHz 이며 초당 1,581 GFLOPS의 연산성능을 가진다. 비교에 사용된 CPU는 Xeon X5680 2개로 구성되어있으며 이는 160 GFLOPS의 연산성능을 가진다. GPU의 활용도를 분석하기 위하여 디버깅 툴인 Parallel Nsight™를 사용하였다. 실험에 사용된 영상은 단일채널 12,000×1,024 크기의 영상이며 블록의 크기는 32×32이다. 정합 범위는 기준블록 주변 32×32범위 이다.

최적화되지 않았을 경우 커널 수행시간과 메모리 전송의 비율은 약 19대 1의 비율을 가지며 12,000개의 쓰레드를 발생시킨다. 최적화 방법을 순서대로 적용한 결과 커널 수행시간과 메모리 전송의 비율은 약 11대 9로 커널 수행시간의 비율이 줄어들었으며 발생된 쓰레드는 12,288,000개로 초기 구현에 비해 1,024배의 쓰레드가 발생되었다. 초기수행시간에 비해 최적화된 수행시간의 경우 약 27배의 속도 향상을 얻을 수 있었다. 비교를 위한 CPU알고리즘의 경우 16개의 쓰레드를 발생시켜 병렬 처리 하였으며 그 결과 최적화된 GPU 알고리즘이 약 23배 빠른 속도를 보였다.

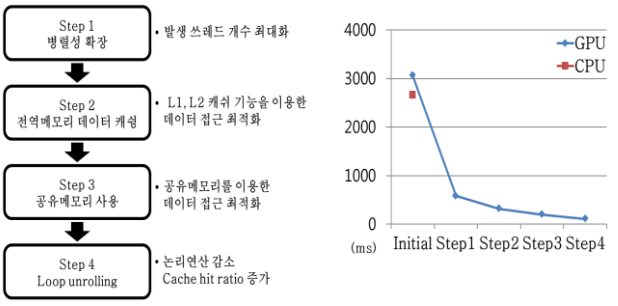

그림 1. 최적화 과정 및 수행 결과. (a) 단계별 최적화 과정. (b) 최적화에 따른 수행시간 변화.

Ⅴ. 결론

본 논문의 GPU를 이용한 단계별 최적화 방법을 통하여 최적화를 수행한 결과 최적화되지 않은 알고리즘에 비해 약 27배의 속도향상을 확인하였다. GPU를 이용하여 병렬 알고리즘 작성 시 알고리즘의 특성을 잘 파악한 뒤 본 논문에서 제시하는 순서에 따라 최적화 방법을 진행한다면 체계적이고 쉽게 GPU의 특성을 이용한 최적화된 알고리즘을 작성할 수 있을 것이다.

참고문헌

[1] NVIDIA Corporation, *CUDA C Programming Guide 3.2*, October 2010.

[2] NVIDIA Corporation, *Tuning CUDA Applications for Fermi 3.2*, August 2010.

[3] K. Hariharakrishnan and D. Schonfeld, "Fast object tracking using adaptive block matching," *IEEE Trans. on Multimedia*, vol. 7, no. 5, pp. 853-859, October 2005.

[4] T. Tao, J. C. Koo, and H. R. Choi, "A fast block matching algorithm for stereo correspondence," *Proc. IEEE Conf. on Cybernetics and Intelligent Systems*, pp.38-41, September 2008