2014년 2월 전자공학회 논문지 제 51권 제 2 호
Journal of The Institute of Electronics and Information Engineers Vol. 51, NO. 2, February 2014 http://dx.doi.org/10.5573/ieie.2014.51.2.076

논문 2014-51-2-11

샷 경계 탐지 알고리즘의 병렬 설계와 구현

(Parallel Design and Implementation of Shot Boundary Detection Algorithm)

이 준 구*, 김 승 현*, 유 병 문**, 황 두 성***

(Joon-Goo Lee©, SeungHyun Kim, Byoung-Moon You, and DooSung Hwang)

요 약

최근 고화질 영상의 증가와 더불어 대용량 영상 데이터의 처리는 높은 연산이 요구되어 병렬 처리 설계가 선택되고 있다. 영상 처리에서 나타나는 많은 단순 연산이 병렬처리 가능한 경우, CPU 기반 병렬처리보다는 GPU 기반 병렬처리를 적용하는 것이 계산문제의 시간과 공간 계산 복잡도를 줄일 수 있다. 본 논문은 영상에서 샷 경계 탐지 알고리즘의 병렬 설계와 구현을 연구하였다. 제안하는 샷 경계 탐지 알고리즘은 프레임 간 지역 화소 밝기 비교와 전역 히스토그램 정보를 이용하는데, 이들 데이터의 계산은 대량의 데이터에 대한 높은 병렬성을 갖는다. 이들 연산의 병렬처리를 최대화하기 위해 화소 밝기와 히스토그램의 계산을 NVIDIA GPU에서 병렬 설계 하였다. GPU 기반 샷 탐지 방법은 국가기록원에서 선택된 10개의 비디오 데이터에 대한 성능 테스트를 수행하였다. 테스트에서 GPU 기반 알고리즘의 탐지율은 CPU 기반 알고리즘과 유사하였으나 약 10배의 연산 속도가 개선되었다.

Abstract

As the number of high-density videos increase, parallel processing approaches are necessary to process a large-scale of video data. When a processing method of video data requires thousands of simple operations, GPU-based parallel processing is preferred to CPU-based parallel processing by way of reducing the time and space complexities of a given computation problem. This paper studies the parallel design and implementation of a shot-boundary detection algorithm. The proposed shot-boundary detection algorithm uses pixel brightness comparisons and global histogram data among the blocks of frames, and the computation of these data is characterized with the high parallelism for the related operations. In order to maximize these operations in parallel, the computations of the pixel brightness and histogram are designed in parallel and implemented in NVIDIA GPU. The GPU-based shot detection method is tested with 10 videos from the set of videos in National Archive of Korea. In experiments, the detection rate is similar but the computation time is about 10 time faster to that of the CPU-based algorithm.

Keywords : CUDA, parallel design, GPU, shot-boundary detection

* 학생회원, *** 정회원, 단국대학교 컴퓨터과학과
(Dept. of Computer Science, Dan-kook University)
** 정회원, (주)엘엔와이비젼
(L&Y Vision Technologies, Inc.)
*** 정회원, 단국대학교 운동의과학과
(Dept. of Kinesiologic Medical Science, Dan-kook University)
© Corresponding Author(E-mail: leejg01679@gmail.com)
※ 이 논문은 행정안전부 국가기록원 재원으로 2013년 기록보존기술 연구개발사업의 지원을 받아 수행된 연구임
접수일자: 2013년9월24일, 수정완료일: 2014년2월3일

I. 서 론

디지털 기술 발전은 비디오 데이터의 빠른 생성과 더불어 컴퓨팅 자원의 활용을 증가시켰으나, 비디오 데이터의 통합 관리를 위한 저장, 검색, 분류 등 연관 서비스가 효과적으로 제공 되지 않은 실정이다. 샷 경계 탐지는 비디오 데이터의 저장, 색인, 검색 등의 서비스를 효율적으로 제공하기 위한 기반기술 이다[1].

한편, 멀티미디어 데이터의 크기와 그 수가 증가함에

따라 영상을 빠르게 분석하고 처리하기 위한 연구들이 활발히 진행되고 있다. 고화질 영상의 증가와 함께 대용량 데이터의 처리는 높은 연산이 요구된다. 샷 경계 탐지 역시 대량의 데이터를 연산함에 있어 시간적, 공간적으로 많은 비용이 소모된다. CPU(Central Processing Unit)는 복잡한 분기문과 캐시 등을 고려하고, 수용 가능한 스레드의 수가 적기 때문에 높은 병렬성을 효과적으로 처리하기에 한계가 있다. 따라서 병렬처리 연산에 집중적이고 대량의 코어를 탑재한 GPU(Graphics Processing Units, 그래픽스 처리 장치)를 이용해 제한된 병렬성을 극복하고 많은 연산을 효율적으로 처리할 수 있다[2]. 최근 GPU의 부동 소수점 연산 능력은 CPU를 능가하기 시작하며, 영상 처리는 물론 유체 해석, 구조 해석, 생명 과학 등의 일반 계산 문제 작업에 활용되고 있다[3].

본 논문에서는 프레임들 간의 블록 기반 화소 밝기 차이, 히스토그램의 변화 등 복합된 영상 정보를 이용하는 GPU 기반 샷 전환 탐지 병렬 알고리즘을 제안하였다. 연속 프레임의 지역 영역의 변화는 프레임간 밝기가 보상된 대응되는 블록의 화소 차로 탐지하며, 객체의 이동을 고려한 두 프레임의 전체적 변화는 히스토그램을 사용한다. 본 알고리즘은 CUDA 기반의 GPU를 사용하여 구현되었으며, 효율적인 병렬화를 위해 공유메모리를 사용하고, 불필요한 데이터의 이동을 최소화하였다. 제안된 병렬처리 설계 방법은 연산 속도는 향상시키면서 순차 처리 기반의 알고리즘과 유사한 탐지율을 보였다.

본 논문은 2장에서는 샷 경계 탐지와 CUDA를 이용한 병렬 처리의 관련 연구에 대해 살펴보고, 3장에서는 샷 경계 탐지 알고리즘과 병렬화 방법을 기술한다. 4장에서는 국가 기록원 소장 디지털화 영상을 사용하여 기존의 화소 또는 히스토그램 차를 이용한 알고리즘과 제안하는 알고리즘을 비교한다. 마지막으로 제안하는 알고리즘의 문제점 및 개선 방향을 제시한다.

Ⅱ. 관련 연구

1. 샷 경계 탐지 알고리즘

샷 경계 탐지 알고리즘은 비디오 데이터 $V=(F_0,F_1,F_2,...,F_{N-1})$ 에 대해 샷 전환 벡터 $D=(d_0,d_1,d_2,...,d_{N-1})$ 를 출력한다. V는 N개의 프레임으로 구성된 비디오 데이터, F_k는 크기가 $X\times Y$인 2차원 행렬, 그리고 $F_k(x,y)$는 k_{th} 프레임의 좌표 (x,y)의 화소 밝기 값이다. 선택된 비유사도 측정값 $D(k)=d_k$는 현재 프레임 F_k와 참조 프레임 F_{k+1}의 샷 경계 여부를 나타낸다.

Zhang 등[4]과 Xiaoquan Yi 등[5]은 샷 전환 탐지를 위해 두 연속 프레임사이에 서로 대응되는 화소 값의 차이를 이용한 프레임간의 평균 밝기 차이를 사용했다 (1).

$$d_k=\frac{1}{XY}\sum_{x=0}^{X-1}\sum_{y=0}^{Y-1}1(|F_k(x,y)-F_{k+\tau}(x,y)|<\epsilon) \quad (1)$$

여기서 $x>0$이면 $1(x)=1$ 그 외는 $1(x)=0$이다. ε은 화소 차에 대한 허용 임계값(tolerance threshold)이다. 만약 $\tau=1$이면 참조 프레임은 F_{k+1}가 된다. 샷 전환 탐지의 비유사도 측정에는 화소 정보의 히스토그램이 이용되었다[4, 6~10]. C개의 색과 L개의 빈(bin)을 고려 시 $H_k(c_{ij})$를 k-번째 프레임에서 색 c_{ij}의 히스토그램 값이라 할 때 식 (2)가 사용된다[4, 8~9].

$$d_k=\frac{1}{LC}\sum_{i=0}^{C-1}\sum_{j=0}^{L-1}1(|H_k(c_{ij})-H_{k+1}(c_{ij})|<\epsilon) \quad (2)$$

블록 기반 비교 방법에서는 프레임을 일정 크기의 중복이 없는 블록 영역으로 구분하고 대응 블록 영역의 화소 , 통계 값 , 히스토그램 등을 비교한다. $X_b\times Y_b$ 크기의 B개로 프레임을 블록화한 경우 F_k의 i-번째 블록 내 화소를 $b_{k,i}(x,y)$라 할 때, 블록 내 화소 차로부터 비유사도는 식 (3)과 같다.

$$d_k=\frac{1}{B}\sum_{i=0}^{B-1}[\frac{1}{X_bY_b}\sum_{x=0}^{X_b-1}\sum_{y=0}^{Y_b-1}1(|b_{k,i}(x,y)-b_{k+1,i}(x,y)|<\epsilon)] \quad (3)$$

Zabih 등은 연속적인 두 프레임에서 에지들의 생성 또는 소멸 정도를 측정하는 에지 변화율(edge change ratio)을 사용하여 두 연속 프레임사이의 샷 전환을 탐지했으며[11], Robert 등은 칼라 모멘트(color moment) 비교 방법을 이용했다[12]. Matsumoto 등은 연속된 프레임 간의 화소 차이, 칼라, 에지 변화율, 영역 분할을 사용하는 여러 SVM 학습 기반 샷 탐지를 수행하였다[13]. 연산량을 줄이기 위한 노력으로는 슬라이스 영상의 수직, 수평 중심 블록 영상만을 이용한 시공간 히스토그램과 매크로 블록 정보들을 이용하여 샷 탐지를 수행하였다[14].

기존의 샷 탐지 알고리즘들은 제안하는 특징 벡터의

설정과 유사도 측정, 임계값의 설정, 정확율(precision)과 재현율(recall)을 이용한 평가 등으로 요약된다. 두 연속 프레임사이의 동일한 화소 또는 선정된 블록을 비교하는 방법과 히스토그램을 이용한 방법은 계산이 간편하여 보편적으로 사용되나 객체의 이동이나 화소의 밝기 등의 변화에 취약하고, 에지변화율 특징을 이용한 방법은 밝기 변화나 객체 이동에는 강하나 탐지를 위한 계산 량이 많고 잡음에 민감한 문제점이 있다.

2. CUDA

GPU는 수 백개의 부동 소수점 계산 유닛과 독립된 메모리를 가지고 있다. 컴퓨터 그래픽스를 위한 GPU를 이용해 CPU가 처리하던 응용 프로그램들의 계산을 가능하게 하는 GPGPU(General-Purpose computing on Graphics Processing Units, 범용 그래픽 처리 장치)가 활발히 사용되고 있으며 CUDA(Compute Unified Device Architecture)와 OpenCL(Open Computing Language) 등은 GPU 기반 병렬처리 프로그램 환경을 제공한다. CUDA 병렬 컴퓨팅 아키텍처는 CPU와 GPU의 co-processing 처리 환경을 제공하며, GPU에서 수행하는 병렬 처리 알고리즘을 C 프로그래밍 언어를 사용해 작성할 수 있는 구조이다. OpenCL은 이종 플랫폼(CPU, GPU)에서 실행되는 프로그램을 작성할 수 있도록 해주는 C 기반의 언어이다. 기존 PC 환경에 GPGPU 프로그래밍이 가능한 그래픽 카드만 추가하면 개발 및 활용이 가능하며, 연산 능력이 뛰어나 활용 가능성이 높은 기술이다.

CUDA는 그림 1과 같이 여러 개의 SM(Streaming Multiprocessor)으로 구성되어 있고, 하나의 SM은 여러 개의 SP(Streaming Processor)로 구성되는 계층적 구조이다. 워프(warp)는 스레드의 스케줄링을 담당하는 단위로, 하나의 워프가 32개의 스레드를 관리한다. GTX580의 경우 32개의 SP가 모여 하나의 SM을 이루며, 각 SM은 최대 48개의 워프를 지원한다. 하나의 SM은 1,536(48X32)개의 스레드를 동시에 생성할 수 있으며, 16개의 SM을 가지므로 24,576개의 스레드를 동시에 생성할 수 있다. CUDA의 메모리 구조는 그림 2와 같이 구성되어 있다. 레지스터(register)는 각 스레드의 변수를 저장하며 가장 빠른 접근속도를 갖는다. 공유 메모리(Shared Memory)는 블록 내 스레드 들이 데이터를 공유할 수 있으며 캐시 기능이 있다. 전역 메모리(Global Memory)는 접근 속도는 느리지만 대량의 데이터를 저장할 수 있으며 호스트(host)와 통신이 가능하다. 마지막으로 상수 메모리(Constant Memory)와 텍스처 메모리(Texture Memory)는 각 스레드에서 읽기만 가능하며 캐시 기능이 있다.

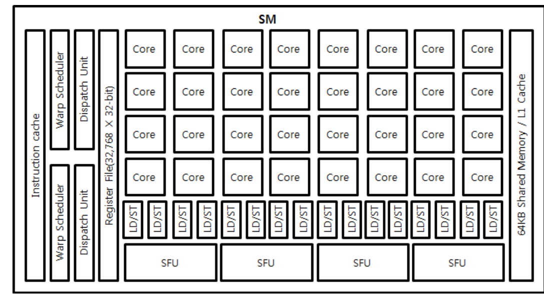

그림 1. CUDA의 SM 구조
Fig. 1. SM structure of CUDA.

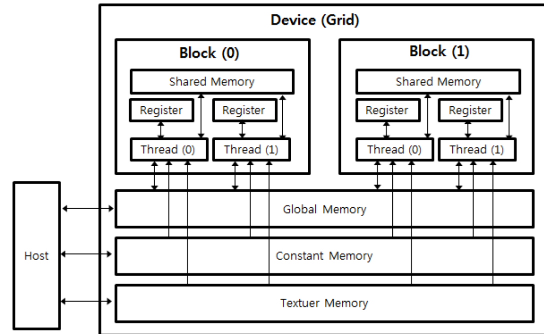

그림 2. CUDA의 계층적 메모리 구조
Fig. 2. Hierarchical memory architecture of CUDA.

GPU를 이용한 병렬처리는 SIMD(Single Instruction, Multiple Data) 방식의 처리를 이용한다. 반복적으로 나타나는 연산을 수 만개의 스레드로 분할하여 순차성이 없는 대량의 데이터에 대해 적용함으로써 GPU의 수 백개의 코어가 대량의 데이터 연산을 빠르게 처리할 수 있다. 대용량 행렬 곱셈[15], Sum Reduction[16~17], 제르니커 모멘트(Zernike moments)를 이용한 샷 경계 탐지[18] 등에서는 반복적으로 나타나는 연산들을 작은 스레드들로 분할하여 CUDA를 이용해 병렬 처리 함으로써 계산 효율을 높였다. 반복적으로 사용되는 데이터에 대해 공유 메모리를 사용하고, 접근 속도가 매우 느린 DRAM의 접근을 최소화하면 계산 효율을 더욱 향상시킬 수 있다. CUDA의 기술이 보편화되면서 많은 학술 논문과 함께 다양한 분야의 응용 프로그램들이 속도 향상의 결과를 보이고 있으며[3], OpenCL을 이용한 연구도 꾸준히 증가하고 있다.

Ⅲ. 본 론

본 논문에서는 샷 탐지를 위해 블록을 단위로 하는 지

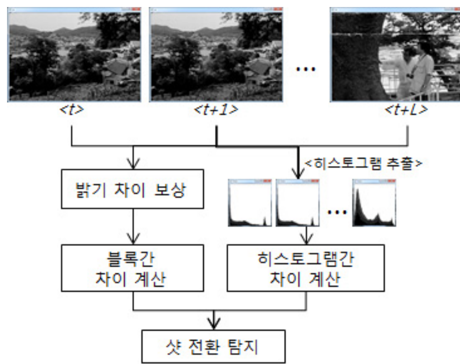

그림 3. 샷 경계 탐지 알고리즘
Fig. 3. A shot boundary detection algorithm.

역적 화소 기반 방법을 사용하며, 프레임간 급격한 밝기 차이로 인한 오탐지를 줄이기 위해 k_{th} 프레임과 $(k+1)_{th}$ 프레임간의 밝기를 보상하고, 객체의 이동으로 인한 오탐지를 줄이기 위해 k_{th} 프레임부터 $(k+L)_{th}$ 프레임 까지의 전역적 히스토그램 정보를 이용해 객체의 이동을 보상한다(그림 3). 각 방법들은 대량의 데이터에 대해 연산을 수행하기 때문에 처리 시간을 단축하기 위하여 CUDA를 이용해 GPU에서 수행 되도록 알고리즘을 병렬화 하였다.

1. 샷 탐지 알고리즘

제안하는 방법에서 화소 기반 샷 탐지 알고리즘은 프레임 간 급격한 밝기 변화에 대해서 오탐지를 유발할 수 있으며[19], 이러한 문제는 프레임 간 밝기를 보상하는 것으로 해결할 수 있다. 샷 탐지를 위한 기준 프레임이 F_k, 참조 프레임이 F_{k+1}일 때, 두 프레임 간의 밝기 차이 보상값인 $I(k,k+1)$의 계산은 식 (6)과 같다.

$$I(k,k+1)=\frac{1}{XY}\sum_{x=0}^{X-1}\sum_{y=0}^{Y-1}(F_k(x,y)-F_{k+1}(x,y)) \tag{6}$$

여기서 각 시그마는 프레임의 X축과 Y축의 화소 수를 나타낸다. 지역 정보를 이용하는 프레임 간의 유사도를 계산하기 위해 밝기가 보상된 연속된 두 프레임을 블록 단위로 분할하고, 전체 블록 수에 대한 유사 블록 수의 비율을 계산한다. F_k와 F_{k+1} 간의 유사 블록 수의 비율인 $n(k,k+1)$의 계산은 식 (7)과 같다.

$$n(k,k+1)=\frac{1}{B}\sum_{i=0}^{B-1}[\frac{1}{X_bY_b}\sum_{x=0}^{X_b-1}\sum_{y=0}^{Y_b-1}1(|b_{k,i}(x,y)-b_{k+1,i}(x,y)-I(k,k+1)|-\theta_1)] \tag{7}$$

여기서 B는 블록의 수, X_b와 Y_b는 블록의 가로와 세로의 길이, $b_{k,i}(x,y)$는 k_{th} 프레임의 i번째 블록의 좌표, θ_1은 샷 전환일 가능성이 있는 화소를 탐지하기 위한 임계값이며, $x>0$이면 $1(x)=1$ 그 외는 $1(x)=0$이다. $n(k,k+1)$가 1에 가까울수록 프레임 간의 유사도가 높음을 의미한다.

프레임 간 급격한 객체의 이동은 오탐지를 유발할 수 있으며, 이 문제는 기준 프레임 F_k로부터 F_{k+L} 프레임까지의 히스토그램을 전역 정보로 사용하여 해결할 수 있다. k로부터 연속하는 L개의 프레임까지의 히스토그램 차이 $h(k,k+L)$는 식 (8)과 같이 계산한다.

$$h(k,k+L)=\frac{1}{LC}\sum_{l=1}^{L}\sum_{c=0}^{C-1}1(H_k(c)-H_{k+l}(c)-\theta_2) \tag{8}$$

여기서 k는 기준 프레임, L은 전역 정보 계산에 필요한 프레임 개수, C는 컬러공간의 개수, θ_2는 두 프레임 사이의 히스토그램의 차를 탐지하기 위한 임계값이다. $h(k,k+L)$가 1에 가까울수록 프레임 간의 유사도가 높음을 의미한다.

마지막으로 지역 정보와 전역 정보를 이용해 샷 전환 여부를 탐지하는 d_{k+1}은 식 (9)를 이용하여 계산한다.

$$d_{k+1}=\begin{cases}1 & \text{if } z(k,k+1)>\theta_3\\ 0 & otherwise\end{cases} \tag{9}$$

여기서 $z(k,k+1)=\frac{n(k,k+1)+h(k,k+L)}{2}$이고, θ_3는 샷 전환 프레임을 탐지하기 위한 임계값이다.

2. GPU 기반 병렬 처리를 이용한 성능 향상

CUDA에서는 CPU와 GPU를 각각 호스트(host)와 디바이스(device)로 표기하며, 호스트와 디바이스간의 메모리 전송은 DRAM을 사용하기 때문에 상당한 시간을 소모한다. 효율적인 병렬 처리를 위해 불필요한 메모리 전송을 최소화 하고, 공유 메모리를 사용하여 DRAM으로의 접근을 줄이는 것이 필요하며, 대량의 데이터에 대해 동일한 연산을 수행하는 SIMD(Single Instruction, Multiple Data) 구조가 적합하다. 제안하는 샷 탐지 알고리즘(그림 3)의 병렬 설계는 밝기 차이 보상, 블록간 차이 계산, 히스토그램 계산을 SIMD 식으로 구조화 시킨다.

프레임간 밝기 차이를 보상하는 식 (6)은 하나의 스레드가 하나의 화소를 계산하도록 $X\times Y$개의 스레드를

구성하여 병렬로 처리할 수 있으며, 이때 각 스레드를 수행하기 위한 GPU 커널 *brightComp*은 다음과 같다.

```
kernel brightComp(Mat f1, Mat f2, Mat rst){
  Index i=blockIdx*blockDim+threadIdx;
  r[i]=f1[i]-f2[i];    }
```

여기서, *Mat*은 행렬을 다루는 객체이고, 기준 프레임 *f1*과 참조 프레임 *f2*를 이용해 동일 인덱스 간의 화소 차이를 나타내는 *rst* 행렬을 계산한다. 각 스레드가 참조할 행렬의 *Index*를 계산하기 위해 스레드의 블록 아이디 *blockIdx*, 블록의 크기 *blockDim*, 스레드의 아이디 *threadIdx*를 이용하며, *Index*를 이용해 해당 좌표의 차이를 계산한다. 각 스레드가 커널을 개시한 후의 결과 행렬에 대해 *Sum Reduction*[16~17]을 가로축으로 수행한 뒤 세로축으로 한번 더 수행하여 나온 총 합을 이용해 $I(k,k+1)$를 계산한다.

식 (7)은 블록 간의 차이를 계산하며 여기서 대괄호 내의 계산과 바깥쪽의 계산은 스레드의 구성이 다르므로 분할하여 구성한다. 대괄호 내의 식은 밝기 보상 식과 같은 방법으로 스레드를 구성하여 결과 맵을 계산할 수 있으며, 각 스레드가 수행하기 위한 GPU 커널 *PixComp*는 다음과 같다.

```
kernel PixComp(Mat f1, Mat f2, Num bc, Mat rst){
  Index i=blockIdx*blockDim+threadIdx;
  Num temp=absolute(f1[i]-f2[i]-bc);
  if(temp<thresh1) rst[i]=1;
  else             rst[i]=0;  }
```

여기서 *bc*는 $I(k,k+1)$이며 *f1*과 *f2*의 차이를 계산할 때 감산해준다. *thresh1*은 식 (7)의 θ_1이며, 화소 간 차이의 값이 샷 전환일 가능성이 있는 화소를 나타내면 *0*, 그렇지 않으면 *1*로 하여 *rst*에 저장한다. *PixComp* 커널 수행 후 *rst*를 이용하여 블록 간 차이를 계산한다. 스레드는 블록의 숫자만큼 구성하여 하나의 스레드가 하나의 블록 간 차이를 계산하도록 하며, 각 스레드가 수행하기 위한 GPU 커널 *compBlk*는 다음과 같다.

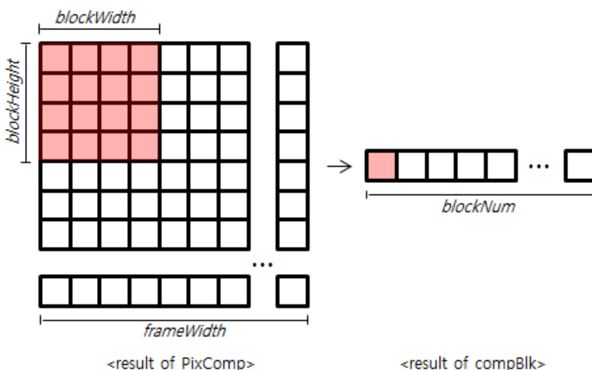

그림 4. $n(k,k+1)$을 계산하는 과정
Fig. 4. Process of calculating $n(k,k+1)$.

```
kernel compBlk(Mat src, Vec rst){
  Index h, w, y, x;
  Size frameWidth, blockWidth, blockHeight;
  initialize frameWidth, blockWidth, blockHeight;
  Num sum=0;
  h=blockIdx*blockHeight;
  loop y from h to (h+blockHeight) step 1
    w=(y*frameWidth)+(threadIdx*blockWidth);
    loop x from w to (w+blockWidth) step 1
      sum=sum+source[x];
  rst[i]=(sum/(blockWidth*blockHeight))*100;      }
```

여기서, *src*는 *PixComp*의 *rst*이며 *compBlk*의 *rst*는 각 블록에 대해 *thresh1*을 만족하는 화소의 개수를 저장하는 벡터이다. 하나의 스레드가 하나의 블록을 카운트 하기 위해서는 그림 4와 같이 프레임의 가로 길이 *frameWidth*, 블록의 가로 길이 *blockWidth*, 블록의 세로 길이 *blockHeight*의 정보가 필요하고, 블록의 개수인 *blockNum*의 크기로 할당된 *rst* 벡터에 계산 결과를 블록의 평균으로 하여 저장한다. 계산된 *rst* 벡터를 이용해 $n(k,k+1)$를 계산한다.

히스토그램 간의 차이를 계산하기 위해서는 먼저 *frame*의 정보가 디바이스로 입력될 때마다 CUDA에서 지원하는 원자값 합(atomic add)을 사용하여 히스토그램을 계산한다[20]. 중복되는 화소 값이 증가함에 따라 히스토그램의 각 빈에 접근하는 횟수도 증가하기 때문에 공유메모리를 사용하여 접근 속도를 향상시킨다. 스레드는 화소의 숫자만큼 구성하여 하나의 스레드가 하나의 화소 값에 해당하는 히스토그램의 빈(bin)을 증가시키도록 하며, 각 스레드가 수행하기 위한 GPU 커널 *histogram*은 다음과 같다.

```
kernel histogram(Mat src, Hist rst){
  shared_memory s_hist[256] = {0};
  Index idx = blockIdx*blockDim+threadIdx;
  atomicAdd(&(s_hist[src[idx]]), 1);
  rst = s_hist;      }
```

여기서, *src*는 원본 프레임, *rst*는 계산된 히스토그램,

*s_hist*는 히스토그램 계산에 사용될 공유 메모리이다. *atomicAdd()*는 첫 번째 인자 위치에 두 번째 인자의 수만큼 값을 증가시킨다.

히스토그램 간 차이를 계산하는 식 (8)은 병렬성을 나타내긴 하지만 연산을 적용할 데이터가 많지 않아 SIMD 구조로는 적합하지 않다. 이러한 경우 GPU에서 계산 속도를 줄이는 것보다 호스트와 디바이스간 데이터를 전송하는데 부하(overhead)가 따른다. 따라서 히스토그램 차이의 평균 $h(k,k+L)$은 커널 *histogram*의 결과를 호스트로 반환하여 순차적으로 계산한다. 마지막으로 식 (9) 역시 연산량이 적기 때문에 $n(k,k+1)$를 반환하여 호스트에서 처리한다.

Ⅳ. 실 험

제안한 알고리즘의 성능을 평가하기 위하여 국가기록원의 디지털화 영화 필름을 사용하였다. 실험에 사용된 플랫폼의 사양은 Intel Core i7 3.4GHz CPU, 8GB RAM, 64비트 시스템, GTX580 GPU를 사용하였으며, 실험 영상은 1960년대부터 1970년대 사이에 촬영된 것으로 텔레시네 장치를 사용하여 아날로그 영상을 디지털로 변화된 영상이다. 사용된 디지털화 영화 필름의 프레임 수, 샷 전환수는 표 1과 같다.

제안된 알고리즘의 평가에서 사용된 디지털화 비디오는 흑백 비디오로 변환시켜 프레임간의 비교에는 그레이-레벨의 화소 밝기와 히스토그램이 사용되었다. 샷 탐지 알고리즘의 성능을 평가하는 기준으로는 재현율 *rec*과 정확도 *pre*와 *F-measure*를 사용한다.

$$rec = \frac{D}{D+M} \tag{10}$$

$$pre = \frac{D}{D+N} \tag{11}$$

$$F-measure = 2 \times \frac{pre \times rec}{pre + rec} \tag{12}$$

여기서 D는 탐지한 샷 전환 수, M는 탐지하지 못한 샷 전환 수, 그리고 N은 실제로 샷 전환이 발생하지 않았으나 오 탐지한 수를 나타낸다. 식 (10)에서 재현율이 *1*이면 모든 샷 경계를 탐지한 것을 의미한다. 식 (11)에서 정확도의 값이 *1*이면 모든 샷이 예측된 것이다. 따라서 식 (12)에서 *F-measure*의 값이 *1*에 가까울수록 예측율이 높다는 것을 의미한다.

표 1. 실험 비디오
Table 1. Test Videos.

비디오	프레임 수	샷 전환 수
V1	5,000	21
V2	5,000	50
V3	5,000	47
V4	5,000	46
V5	5,000	36
V6	5,000	48
V7	5,000	43
V8	5,000	50
V9	5,000	40
V10	5,000	47

각 단계에서 사용한 임계값의 설정을 위해 컷의 발생 빈도가 높고 종류가 다양한 표본영상 V를 사용하였다. θ_1은 V에서 샷 전환이 발생하는 프레임들의 화소 밝기 차이의 평균, θ_2는 V에서 샷 전환이 발생하는 프레임들의 전역적 히스토그램 차이의 평균, 그리고 θ_3은 V에서 샷 전환이 발생하는 프레임들의 비율의 평균으로 사용한다. 임계값의 설정 방법에 따라 $\theta_1 = 30$, $\theta_2 = 50$, $\theta_3 = 60$으로 설정 했다. 객체 이동 거리 보상을 위한 L은 화소 기반의 알고리즘 보다 더 많은 객체의 움직임 정보를 갖기 위해 연속한 두 프레임보다 하나 더 많은 3 프레임으로 사용하였다. 그림 5는 설정한 임계값에 대한 샷 탐지의 예이다. x축과 y축은 각각 $h(k,k+L)$와 $n(k,k+1)$을 나타내며, 각 프레임들의 계산 결과를 좌표로 하여 그래프로 표시하였다. 이때, 직선 $y=-x+120$은 x와 y의 평균이 θ_3=60인 경우에 대해 아래쪽을 샷 경계로, 위쪽을 연속된 샷으로 구분한다.

효율적인 병렬 처리를 위해서는 적절한 블록과 스레드의 수를 설정해야 한다. 표 2는 블록과 스레드의 수에 따른 처리 시간을 실험한 결과이다. 스레드의 수가

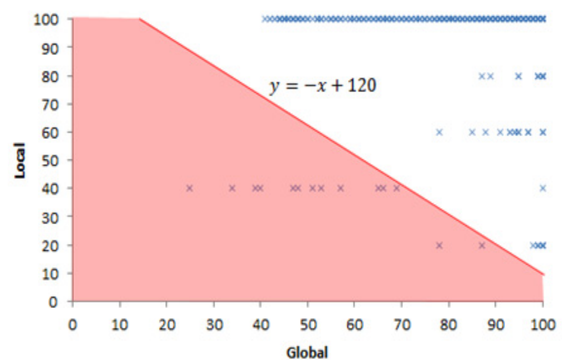

그림 5. θ_3=60으로 탐지된 컷의 예
Fig. 5. Example of shot boundary when θ_3=60.

표 2. 블록과 스레드의 크기에 따른 처리 시간 (단위 : s)
Table 2. Processing time by block and thread size.

블록	스레드	처리시간	CPU	GPU
57,600	6	39.62	27.35	12.27
23,040	15	33.21	27.50	5.71
11,520	30	29.66	24.33	5.33
5,760	60	28.44	26.28	2.16
2,880	120	28.33	26.24	2.09
1,440	240	28.27	26.19	2.08
480	720	28.21	26.71	1.50

CUDA의 스케줄 단위인 워프의 크기 32보다 작은 경우는 효율이 떨어지며, 60 이상부터는 큰 차이가 없었다. 따라서 인덱스의 계산을 간단히 하기 위해 화소간의 연산인 경우 스레드와 블록의 수를 각각 프레임의 가로와 세로 길이로 했다. 히스토그램을 계산하는 경우는 스레드와 블록을 각각 히스토그램의 빈의 수와 프레임의 세로 길이이다. 히스토그램 비교에서는 스레드와 블록을 각각 워프의 크기와 히스토그램의 빈의 수를 워프로 나눈 수로 지정했다. Sum Reduction에서는 가로 방향 수행의 경우 화소간의 연산과 동일하게 스레드를 구성하며, 세로 방향 수행의 경우 첫 번째 열만 계산하면 되므로 블록과 스레드를 각각 1과 세로 길이로 했다.

설정한 조건에 따라 제안 알고리즘을 순차 처리 방법과 병렬 처리 방법으로 각각 실험하였으며, 각 모듈간 시간을 측정해 표 3에 나타냈다. 준비 작업에는 CPU의 순차 처리 작업이 대부분이므로 큰 차이를 보이지 않았다. 하지만 화소간 연산을 기반으로 하는 밝기 보상 모듈과 블록간 차이 계산 모듈은 순차 처리 방법과 비교해 약 90%까지 속도가 향상되었다. 히스토그램 계산의 경우 대량의 스레드를 생성하였음에도 불구하고 화소값의 빈에 해당하는 메모리에 반복적으로 접근하기 때문에 병목현상이 발생하여 GPU를 최대로 활용하지 못해 성능이 크게 향상되지 못했다.

표 3. 병렬 처리와 순차 처리의 시간 차이 (단위 : s)
Table 3. Time difference of parallel and sequential processing.

구분	병렬처리	순차처리	비율
준비	26.62	28.35	1.07
밝기 보상	1.15	11.02	9.58
블록 간 차이	1.21	9.39	7.76
히스토그램	4.14	5.87	1.42
전체	33.66	54.90	1.63

탐지율 비교를 위해 기존의 알고리즘들과 순차 처리 제안 알고리즘, 병렬 처리 제안 알고리즘을 각각 실험하여 표 4에 나타내었다. 훼손이 심하거나 객체 또는 카메라의 이동이 큰 경우 화소 정보만 이용하면 급격한 화소차로 인해 정확율은 낮으면서도 재현율은 높게 나타났다. 동일한 경우에 대하여 전역적 히스토그램 정보만 사용하면 샷 경계가 발생해도 밝기가 비슷한 경우 재현율이 낮았다. 제안 알고리즘은 각 경우에 대하여 정확율과 재현율 모두 증가하였으며, 평균적으로 F-measure가 히스토그램 기반 보다는 약 10%, 화소 기반 보다는 약 40%가 향상되었다. 제안 알고리즘의 순차기반과 병렬기반의 비교에서는 미비한 차이를 보이는 V6를 제외한 모든 비디오에서 탐지율이 동일하다.

표 4. 탐지율의 비교
Table 4. Comparison of detection rate.

구분	히스토그램			화소			제안 순차			제안 병렬		
	pre	rec	F-m	pre	rec	F-m	pre	rec	F-m	pre	rec	F-m
V1	0.82	0.86	0.84	0.58	1.00	0.74	0.88	1.00	0.93	0.88	1.00	0.93
V2	0.92	0.90	0.91	0.37	0.94	0.53	0.87	0.94	0.90	0.87	0.94	0.90
V3	0.95	0.89	0.92	0.60	0.89	0.72	0.96	0.98	0.97	0.96	0.98	0.97
V4	0.85	0.76	0.80	0.41	0.89	0.56	0.84	0.89	0.86	0.84	0.89	0.86
V5	0.70	0.86	0.78	0.21	0.86	0.34	0.94	0.94	0.94	0.94	0.94	0.94
V6	0.95	0.83	0.89	0.39	0.92	0.55	0.92	0.98	0.95	0.92	0.96	0.94
V7	0.97	0.84	0.90	0.21	1.00	0.34	0.98	0.93	0.95	0.98	0.93	0.95
V8	0.93	0.74	0.82	0.51	0.96	0.67	0.88	0.88	0.88	0.88	0.88	0.88
V9	0.90	0.68	0.77	0.25	0.88	0.39	0.90	0.85	0.87	0.90	0.85	0.87
V10	0.78	0.83	0.80	0.47	0.98	0.63	0.86	1.00	0.92	0.86	1.00	0.92
평균	0.88	0.82	0.84	0.40	0.93	0.55	0.90	0.94	0.92	0.90	0.94	0.92

V. 결 론

제안하는 샷 경계 탐지 알고리즘은 대량의 데이터 연산을 필요로 하기 때문에 CUDA를 이용한 병렬 설계를 통해 처리 속도의 향상을 시도하였다. 효율적인 병렬화를 위해 수 많은 스레드로 작업을 분할 하는 SIMD 구조의 모듈을 선택하여 병렬화 하고, GPU와 CPU간의 데이터 이동을 최소화 하였으며, 공유 메모리의 사용을 통해 처리 시간을 단축시켰다. 실험에서 병렬 설계를 통해 제안된 방법은 기존의 알고리즘과 탐지 결과가 유사했으며, 최대 10배의 속도 향상을 보였다. 본 연구의 결과 대량의 연산 과정을 스레드 단위로 충분히 분할 할 수 있다면 다른 샷 경계 탐지 알고리즘에도 CUDA를 이용한 병렬처리 설계를 적용할 수 있을 것이다.

REFERENCES

[1] Jinhui Yuan, Huiyi Wang, Lan Xian, Wujie Zheng, Jianmin Li, Fuzong Li, Bo Zhang, “A Formal Study of Shot Boundary Detection,” IEEE Trans. on Circuits and Systems for Video Technology, Vol. 17, No. 2, pp. 168-186, Feb. 2007.
[2] J. G. Han, Y. S. Ko, S. H. Suh, S. H. Ha, “Performance Enhancement of Scaling Filter and Transcorder using CUDA”, Journal of Korean Institute of Information Scientists and Engineers, Vol. 16, No. 4, pp. 507-511, Apr. 2010.
[3] NVIDIA, NVIDIA Manufacturing Day 2013, [online]Available:https://registration.gputechconf.com/form/session-listing
[4] H. J. Jhang, A. Kankanhalli, S. W. Smoliar, “Automatic Partitioning of Full-Motion Video,” Multimedia Systems, Vol. 1, No. 1, pp. 10-28, June 1993.
[5] Xiaoquan Yi, Nam Ling, “Fast Pixel-Based Video Scene Change Detection,” IEEE Int. Symposium on Circuits and Systems, Vol. 4, pp. 3443-3446, May 2005
[6] A. Nagasaka, Y. Tanaka, “Automatic Video Indexing and Full-Video Search for Object Appearances,” IFIP Proceeding of Visual Database Systems, pp. 113-127, 1992.
[7] Y. W. Han, S. I. Cheong, S. J. Kim, S. Y. Lee, S. H. Kim, “Improving Histogram Scene Change Detection Method Using Motion Vector,” Proceedings of Fall Conf. on Korean Institute of Information Scientists and Engineers, Vol. 26, No. 2, pp. 410-412, Oct. 1999
[8] B. H. Shekar, K. Holla K. Raghurama, Kumari M. Sharmila, “Video Cut Detection Using Chromaticity Histogram,” Int. Journal of Machine Intelligence, Vol. 3, No. 4, pp. 371-375, Dec. 2011
[9] Z. Cernekova, C. Kotropoulos, I. Pitas, “Video Shot Segmentation Using Singular Value Decomposition,” Proc. 2003 IEEE Int. Conf. Multimedia and Expo, Baltimore, Maryland, Vol. 2, pp. 301 - 304, July 2003.
[10] Priyadarshinee Adhikari, Neeta Gargote, Jyothi Digge, B. G. Hogade, “Abrupt Scene Change Detection,” World Academy of S, cience, Engineering and Technology 42 2008, No. 18, pp. 711-716, June 2008.
[11] R. Zabih, J. Miller, K. Mai, “A Feature-based Algorithm for Detecting and Classification Production effects,” ACM Multimedia System, Vol. 7, No. 1, pp. 119 - 128, March 1999.
[12] Robert A. Joyce, “Temporal Segmentation of Video using Frame and Histogram-Space,” IEEE Trans. on Multimedia, Vol. 8, No. 1, pp. 130-140, Feb. 2006.
[13] Matsumoto, Kazunori, “SVM-Based Shot Boundary Detection with a Novel Feature,” IEEE Int. Conf. on Multimedia and Expo, pp. 1837-1840, July 2006.
[14] J. K. Jin, J. H. Cho, J. H. Jeong, “Fast Scene Change Detection Using Macro Block Information and Spatio-temporal Histogram,” The Institute of Electronics Engineers of Korea, Vol. 48, No. 1, pp. 141-148, Jan. 2011.
[15] NVIDIA, CUDA C Best Practices Guide, [online] Available:http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html
[16] David B. Kirk, Wenmei W. Hwu, Programming Massively Parallel Processors:A Hands-on Approach, Elsevier, pp. 99-103, 2010.
[17] Jason Sanders, Edward Kandrot, CUDA By Example An Introduction to General-Purpose GPU Programming, Addison Wesley, pp. 79-81, 2010.
[18] Pablo Toharia, Oscar D. Robles, Ricardo Suarez, Jose Luis Bosque, Luis Pastor, “Shot Boundary Detection Using Zernike Moments In Multi-GPU Multi-CPU Architectures,” Journal of Parallel and Distributed Computing, Vol. 72, No. 9, pp. 1127-1133, sep. 2012.
[19] I. S. Jeong, O. J. Kwon, “Video Shot Boundary Detection using Relative Difference between Frames,” Optical Engineering, Vol. 42, No. 3, pp.604-605, March 2003.

[20] Shane Cook, CUDA Programming A Developer's Guide to Parallel Computing with GPUs, Elsevier, pp. 97-103, 2013.

저 자 소 개

이 준 구(학생회원)
2012년 단국대학교
컴퓨터과학과(공학사)
2014년 단국대학교
전자계산학과(공학석사)

<주관심분야 : Image Processing, Parallel Processing, Machine Learning>

김 승 현(학생회원)
2013년 단국대학교
컴퓨터과학과(공학사)
2013년～현재 단국대학교
컴퓨터과학과(석사과정)

<주관심분야 : Image Processing>

유 병 문(정회원)
1987년 경북대학교
전자계산학과 공학사
1992년 충남대학교
전산과 이학석사
2000년 Wayne State University
컴퓨터공학과 공학박사
현재 (주)엘앤와이비젼 대표이사

<주관심분야 : Motion Analysis, 3D/2D image registration, (medical) image processing>

황 두 성(정회원)
1986년 충남대학교
계산통계학과 이학사
1990년 충남대학교
계산통계학과 이학석사
2003년 Wayne State University
컴퓨터공학과 공학박사
현재 단국대학교 컴퓨터과학과 부교수
현재 단국대학교 운동의과학과 부교수

<주관심분야 : Machine Learning, Parallel Processing, Semantic Web>