CUDA 를 이용한 고속 자막 처리 기술

김현수, 김대열, 권승철, 손채봉
광운대학교 전자통신공학과
{ hyunsoo | wagon0004 | gwaja92 | cbsohn }@kw.ac.kr

High-speed caption processing technology using CUDA

Hyun soo Kim, Dae yeol Kim, Seung-Cheol Kwon and Chae-Bong Sohn
Department of Electronics and Communications Engineering, Kwangwoon University

요 약

본 논문에서는 멀티미디어 및 수학 관련 알고리즘 분야에서 각광 받고 있는 CUDA(Compute Unified Device Architecture) 기법을 실시간 동영상 자막 처리에 이용 한다. 실시간 고화질 동영상 자막 처리의 낮은 속도를 개선하기 위한 방법 제안으로 써, 이의 함수 구성을 제안한다. 기존의 자막 처리 방식에서는 하나의 코어만을 이용하였다. 이 방법 대신에 CUDA 를 적용 함으로써, 더 많은 코어를 이용해 실시간 자막 처리의 지각적인 성능을 향상하였다. 삽입하고자 하는 자막에 대해 비트맵 이미지를 형성하고, 이의 정보를 처리한다. Intel Core™ i7-4710 MQ , GTX870 환경에서 실험하였으며, 실험 결과 C code 만으로 연산을 한 것 보다 CUDA code 가 약 88% 정도의 속도 향상이 있음을 보였다.

1. 서론

HD(high-definition) 화질 시대를 지나 UHD(Ultra-high-definition) 4K, 8K 시대가 되었다. 하지만, 이를 이용해 실제로 방송을 하는 데에서는 준비가 되지 않았다. 일본 NHK 에서는 12Gbps 의 데이터 량을 갖는 8K-UHD 비디오(7,680X4,320, YUV 4:2:0, 8bits, 30fps)를 실시간으로 부복호화하기 위하여 MPEG-2 및 AVC 부호화 기술 기반으로 16 대 및 8 대의 인코더를 병렬로 처리하여 약 120Mbps 로 압축할 수 있는 코덱 시스템을 개발하였고, 최근에는 H.264|AVC 기술을 확장하여 약 72Mbps 로 압축할 수 있는 코덱시스템을 개발하였다. 현재 H/W 성능을 감안하면 8K-UHD 비디오의 단일 코덱에 의한 실시간 부복호화는 어려울 것으로 예상된다[1]. 따라, 이러한 실정에 CUDA 를 이용한 병렬처리 방법은 UHD 영상을 상용화 시키는데 박차를 가할 것으로 예상된다. 이에 본 논문에서는 실시간으로 자막 처리를 하는데 있어 CUDA 를 이용하여 고속 자막 처리에 기여 한다.

2. CUDA 병렬화 기술

CUDA 는 GPU 에서 C 언어 포함 그 이외의 언어를 사용하여 작성할 수 있도록 만든 GPGPU 기술이다[2]. CUDA 는 G8X GPU 로 구성된 GeForce8 시리즈 이상의 nVidia GPU 에서 동작한다. CUDA 는 GPU 내의 다중 코어를 이용하여 병렬 처리를 가능 하게 한다. GPU 를 이용함으로써 CPU 의 부하감소와 CUDA 를 이용한 커다란 성능 향상을 기대한다.

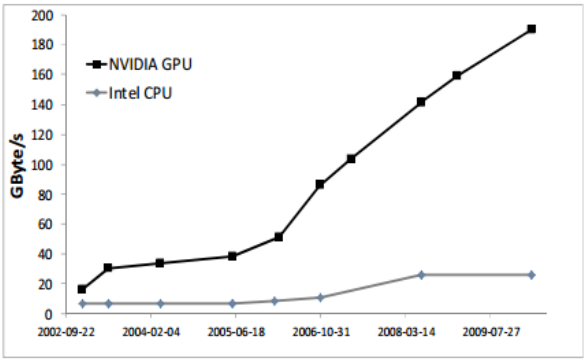

그림 1. Intel CPU 와 NVIDIA GPU 성능[2]

Fig 1. Intel CPU 와 NVIDIA GPU performance

3. 자막 파싱 방법

실시간으로 자막을 영상에 출력하기 위해서는 그림 3 에서 보이 듯 자막의 유무를 판단하고 이를 Bitmap 으로 생성하는 과정이 필요하다. 자막 Bitmap 에는 자막 정보와 자막 정보 주변으로 투명색(transparent color)이 존재한다.

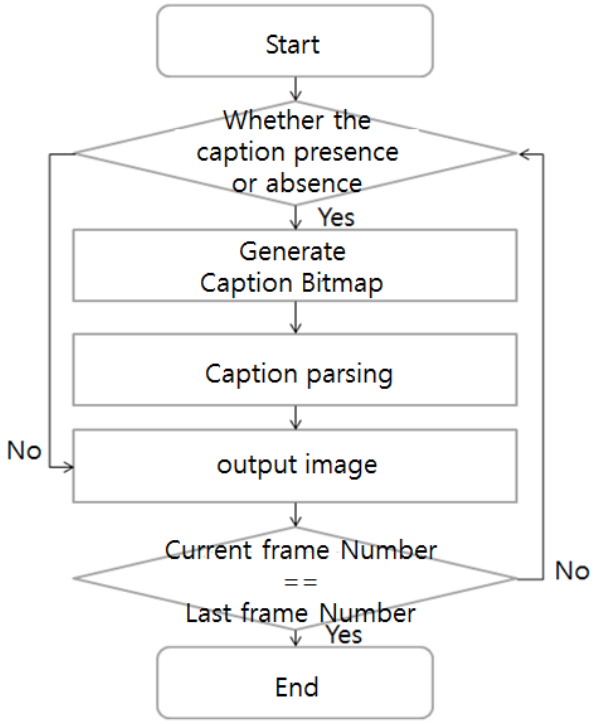

그림 3. 자막 데이터 파싱 순서도

Fig 3. Flow-chart of caption data passing

자막 Bitmap 을 이용하여 자막을 파싱 하는 방법은 표 1 과 같다.

표 1. 자막 삽입 수도 구문

Table 1. C pseudo code of set caption

Algorithm CaptionMask_C

Procedure CaptionMask_C(data)
set i , j to 1
 	
loop (i < HEIGHT)
loop(j< WIDTH)
set index to i * WIDTH + j
if any of adjacent pixels of data(index) is white then
set data(index) to black
else
do nothing
set j to j+1
end loop
set i to i+1
end loop

표 2. 자막 삽입 C 구문

Table 2. C syntax of set caption

Syntax

for(i=0; i<CAPTION_HEIGHT ; i++) {
for(j=0;j<CAPTION_WIDTH;j++) {
if(bitmap pixel data
== transparent color)
set caption
}
}

먼저 자막의 크기에 대응하는 자막 bitmap 을 만든다. 그 후 자막의 가로 세로 각각의 픽셀의 조건을 확인한다.

1. 픽셀의 색깔이 투명색인가?
2. 픽셀의 상하좌우 색이 하얀색 인가?

그 후 두 가지 조건을 모두 만족할 시 검은색의 자막 정보를 입력 하여 준다. 그렇지 않다면 투명색을 입력하여 기존의 데이터가 출력 가능하게 만든다.

4. 자막 파싱 병렬화

그림 4 와 같이 영상 한 프레임에 들어가는 자막을 병렬로 처리 할 수 있는 방법을 제안한다. 자막 파싱을 병렬 처리 하였을 경우 효율 및 속도를 측정 하는 것을 목표로 진행하였다.

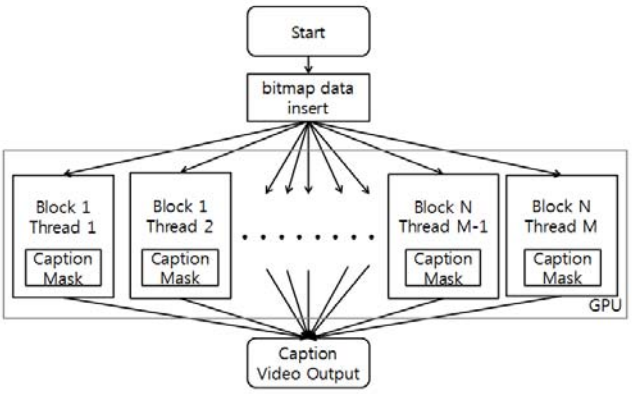

그림 4. CUDA 자막 데이터 파싱 블록도

Fig 4. Block diagram of CUDA caption data passing

그림 5 의 과정에서 볼 수 있는 것 처럼 먼저 자막의 크기에 대응하는 자막 bitmap 을 만든다. 그 후 cudaMalloc API 를 이용하여 CPU 에서 GPU 로 데이터를 넘겨준다. GPU 내에서는 CaptionMask_CUDA 연산을 진행한다.

각 각의 쓰레드가 담당하고 있는 픽셀 정보가 투명색이고 상하좌우의 정보가 하얀색이면 검은색을 출력하도록 데이터를 입력한다<표 3>. 즉, 자막을 출력한다. 만약 그렇지 않다면 투명색을 출력하도록 데이터를 입력한다. 모든 연산이 마친 후 CPU 로 데이터를 넘겨준 후, 이 데이터를 이용해 영상에 출력하도록 한다.

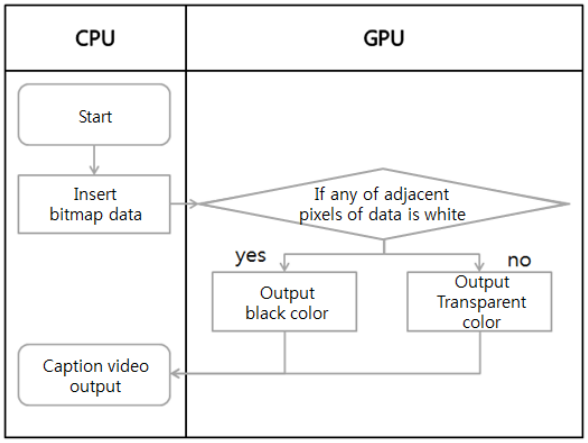

그림 5. Thread 내의 CaptionMask_CUDA 연산 과정

Fig 5. CaptionMask_CUDA operation process in Thread

표 3. 자막 삽입 CUDA 수도 구문

Table 3. C pseudo code of set caption

```
Algorithm CaptionMask_CUDA
Procedure CaptionMask_CUDA(data)

set i   to blockIdx.x
set j to blockDim.x * threadIdx.y + threadIdx.x
set width to blockDim.x * blockDim.y
set index to i * width + j
        if any of adjacent pixels of data(index) is white then
          set data(index) to black
          else
                set data(index) to black
             else
                do nothing
```

표 4. 자막 삽입 CUDA 구문

Table 4. CUDA syntax of set caption

```
Syntax
__global__
void CaptionMask(unsigned int* data) {

    int i = blockIdx.x;
    int j = blockDim.x * threadIdx.y + threadIdx.x;
    int width = block\Dim.x*blockDim.y;

    if (data[i * width + j] == 0)
                        data[i*width + j] = 0x00649601;
        else if (data[i * width + j] == 0x00649601) {
            if (data[(i - 1)*width + j] == 0x00ffffff ||
                data[(i + 1)*width + j] == 0x00ffffff ||
                data[(i)*width + j - 1] == 0x00ffffff ||
                data[(i)*width + j + 1] == 0x00ffffff)
                data[i+width+j] = 0;
    }
};
```

5. 실험 및 결과

CUDA 기반 자막 파싱 프로그램의 속도를 측정하기 위하여 QueryPerfomanceCounter API 함수를 이용하였다. 또한 1920x1080 해상도의 Full-HD 영상에 대한 실험을 진행하였다. 실험 환경은 표 2 와 같다.

표 5. 실험 환경

Table 5. Experimental environment

조건	사양
OS	Windows 7 (64bit)
CPU	Intel Core™ i7-4710 MQ 2.5GHz
GPU	GTX 870 (4G)
CUDA	CUDA toolkit 6.5

실험 결과는 표 3 과 같다.

표 6. 한 줄 자막에 대한 수행시간

Table 6. Process time for one sentence caption

	C code	CUDA
Delay (ms)	1.099	0.586

실험을 진행한 결과 CUDA 코드로의 자막 처리 시간이 0.586ms 로 기존의 C 코드로의 자막 처리 시간 1.099ms 에 비해 약 88%의 속도가 향상이 된다.

CUDA 코드로 처리함에 있어서 병렬 자막 처리 시간에는 크게 문제가 없었으나 CPU 에서 GPU 로 데이터를 복사하는, 메모리와 관련된 오버헤드가 매우 크게 작용 한다. 1920x1080 영상에 대한 메모리 할당 시간은 447ms 로 자막 처리에 약 750 배에 달한다. 하지만 CUDA 코드의 자막 처리 속도가 2 배에 달하므로 병렬 처리로 인해 메모리 오버헤드의 상당수를 상쇄한다.

6. 결론

본 논문에서는 CUDA 를 이용해 자막 처리 병렬화를 진행하였다. 실험 결과 자막 처리 시간에서 87%의 속도 향상을 기대 할 수 있다. 반면에 병렬 처리를 위한 메모리 할당 부분에서 오버헤드가 발생 했다. 이는 실제 파일이 가지고 있는 자막의 줄 수의 합이 872 개 이상 되어야 속도가 향상된다. 실시간 자막 처리에 있어서 메모리 오버헤드가 가지는 의미는 크지 않다. 따라, 앞으로의 실시간 자막 처리에 있어서 높은 속도 향상이 기대된다.

7. 참고문헌

[1] 조숙희 외 2 명, “UHDTV 기술 및 표준화 현황”, TTA Journal, 140 호, pp. 49-54, 2012

[2] Jason Sanders, Edward Kandrot, “CUDA by example” Addison-Wesley, pp. 1-11, 2011.

[3] “CUDA RUNTIME API”,v7.0, nVidia, 2015

[4] Rob Farber, ” CUDA application design and development”, 2012

[5] Shane Cook, “CUDA Programming: A developer’s guide to parallel computing with GPUs”, 2012

[6] Kirk, David B. “Programming Massively Parallel Processors : (A)Hands-on Approach, 2011