Journal of The Korea Society of Computer and Information
Vol. 20, No. 1, January 2015
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2015.20.1.001

GPU를 이용한 다양한 해상도의 비디오기반 실시간 화재감지 방법 구현 및 성능평가

손 동 구*, 김 철 홍**, 김 종 면*

Implementation and Performance Evaluation of a Video-Equipped Real-Time Fire Detection Method at Different Resolutions using a GPU

Dong-Koo Shon*, Cheol-Hong Kim**, Jong-Myon Kim*

요 약

본 논문에서는 기존에 많이 사용되는 복잡한 4단계 화재 감지 알고리즘의 성능을 향상시키기 위해 그래픽스 처리 장치 (GPU)를 이용한 효율적인 병렬 구현 방법을 제안하였고 성능을 분석하였다. 또한 현재 많이 사용되고 있는 7가지 서로 다른 해상도 (QVGA, VGA, SVGA, XGA, SXGA+, UXGA, QXGA)의 비디오를 입력으로 하여 성능을 분석하였다. 더불어 각 해상도별 GPU 기반 실행시간과 고성능 CPU에서의 실행시간을 비교 분석하였다. 각 해상도의 5가지 화재 및 비 화재 비디오를 이용하여 모의 실험한 결과, GPU는 CPU보다 실행시간에서 우수한 성능을 보이는 동시에 FULL HD급의 높은 해상도인 UXGA 영상에서도 프레임 당 25.11ms의 실행시간이 소요되어 초당 30 프레임의 실시간 처리가 가능함을 보였다.

▸ Keywords : 화재 감지 방법, 그래픽 처리 장치, 실시간 처리, 비디오 해상도

Abstract

In this paper, we propose an efficient parallel implementation method of a widely used complex four-stage fire detection algorithm using a graphics processing unit (GPU) to improve the performance of the algorithm and analyze the performance of the parallel implementation method. In addition, we use

•제1저자 : 손동구 •교신저자 : 김종면
•투고일 : 2014. 9. 12, 심사일 : 2014. 10. 15, 게재확정일 : 2014. 12. 6.
* 울산대학교 전기전자컴퓨터공학과(School of Electrical, Electronics, and Computer Engineering, University of Ulsan)
** 전남대학교 전자컴퓨터공학부(School of Electronics and Computer Engineering, Chonnam National University)
※ 이 논문은 2013년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2013R1A2A2A05004566). 또한, 본 연구는 산업자원통상부의 광역경제권 선도산업 육성사업의 일환인 "동남광역경제권 선도산업 지원단"의 2014년 연구비 지원으로 수행되었음 (No. R0001220).

seven different resolution videos (QVGA, VGA, SVGA, XGA, SXGA+, UXGA, QXGA) as inputs of the four-stage fire detection algorithm. Moreover, we compare the performance of the GPU-based approach with that of the CPU implementation for each different resolution video. Experimental results using five different fire videos with seven different resolutions indicate that the execution time of the proposed GPU implementation outperforms that of the CPU implementation in terms of execution time and takes a 25.11ms per frame for the UXGA resolution video, satisfying real-time processing (30 frames per second, 30fps) of the fire detection algorithm.

▸ Keywords : Fire detection method, graphics processing unit, real-time processing, video resolution

I. 서 론

소방방재청의 통계[1]에 따르면 2013년 한 해 동안 국내에서 40,932건의 화재가 발생하여 4,343억 원의 재산과 2,184명의 인명피해를 가져왔다. 이처럼 화재는 막대한 재산과 큰 인명 피해를 가져 올 수 있기 때문에 화재를 조기에 감지하는 시스템 개발이 중요하다. 화재를 검출하는 방법으로는 센서 기반의 감지 방법이 있는데 주로 온·습도 센서, 연기 센서 등을 이용한다. 하지만 이러한 센서를 이용할 경우 감지 거리 안에 발화가 일어나야 감지할 수 있기 때문에 신속하게 화재의 여부를 판단할 수 없는 단점이 있다[2]. 이와 같은 단점을 극복하기 위해 CCTV장치 등을 이용한 비디오 기반의 화재감지 방법 연구가 활발히 진행되고 있다[3-14].

기존의 비디오 기반 화재감지 방법은 크게 4단계로 구성된다. 그림 1은 기존 연구에서 각 단계별로 사용했던 방법들을 보여준다. 첫 번째 단계인 색상 분할에서는 RGB(red, green, blue)[3], YCbCr (Y:luminance, Cb: chrominance-blue Cr: chrominance-red)[4-5] 등의 색 공간에서 화재의 특징적인 색상을 이용하여 화재 영역을 분리하는 연구가 선행되어 왔다. 이 중 YCbCr 색 공간을 이용한 색상 분할은 화재 색상뿐만 아니라 주변과의 휘도를 이용하기 때문에 화재가 가진 색상 특징을 분리하는데 유리하다. 움직임 추정 단계에서는 광류[6], 시간차[7], 가우시안 혼합 모델[8], 배경 감산[7][9] 등의 방법이 사용되어 왔다. 이 중 배경 감산(background subtraction) 방법은 움직이는 물체를 빠르게 분리하는 방법으로 널리 사용되고 있다. 색상 분할 단계와 움직임 추정 단계를 통해 얻은 화재 후보 영역은 빨간색 차량, 빨간색 옷을 입은 사람과 같이 화재 색상이 비슷한 객체가 남아 있을 수 있다. 따라서 화재의 흔들리는 모양, 화재의 형태 및 색상, 질감 변화 등을 이용한 화재특징 추출 방법이 선행되어 왔다[4][7-8][10-11]. 마지막으로 화재와 비화재 프레임의 분류 방법으로는 서포트 벡터 머신 (support vector machine, SVM)[11], 뉴럴 네트워크 (neural network, NN)[6] 등이 있다. 이 중 뉴럴 네트워크는 지역 최소점에 빠질 수 있고, 분류 성능을 높이기 위해서 대량의 학습데이터가 필요하다. 반면에 SVM은 적은 양의 학습데이터로 높은 성능을 보장할 수 있다. 따라서 본 논문에서는 비디오 기반 화재감지 알고리즘의 실시간 처리를 위해 SVM을 선택하였다. 이와 같은 4단계 화재감지 알고리즘은 높은 정확도를 보여 주지만 알고리즘의 복잡성 때문에 연산량이 상당하여 실시간 애플리케이션에 적용하는 것은 제한적이다.

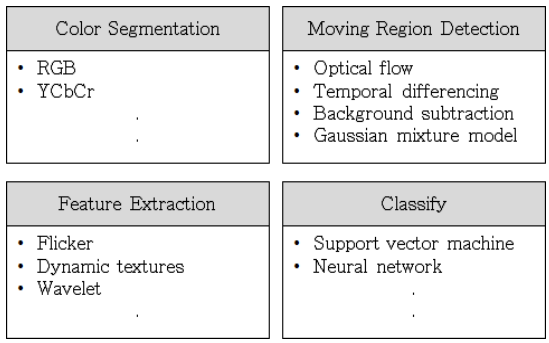

그림 1. 4단계 화재 감지 방법
Fig. 1. 4-stage fire detection method

이러한 문제를 해결하기 위한 대안 중의 하나로 칩 내부에 다수의 코어를 탑재하여 높은 성능을 기대할 수 있는 멀티코어(multi-core)프로세서가 유망하다[12-13]. 멀티코어 프로세서 중에서 GPU (graphics processing unit)는 현재 널리 사용되고 있는 상용화된 프로세서 중의 하나이며, 많은 스레드를 동시에 수행할 수 있는 SIMT (single instruction multiple thread) 방식으로 동작한다. 범용적인 용도로 GPU를 사용하기 위해 GPGPU (general-purpose computing on graphics processing unit)기술이 개발되었으며, 프로그램을 작성하기 위해 CUDA (compute unified device architecture)와 OpenCL (open computing language) 등의 프로그래밍 프레임워크를 이용한다. CPU에서는 대량의 기하 및 화소 데이터들을 처리하기 힘들지만, GPU는 병렬처리를 통해 고속으로 처리할 수 있다. 특히 수많은 화소를 가지는 고 해상도 영상을 이용하여 반복적으로 복잡한 연산을 수행하는 화재감지 알고리즘의 경우, GPU를 사용하여 영상의 픽셀 간 지역성과 규칙성을 병렬화하여 빠른 처리가 가능하다[15].

본 논문에서는 CUDA 프레임워크를 이용하여 GPU 상에서 4단계 화재감지 알고리즘을 병렬 구현하였다. 실험에 사용된 시스템은 3.40Ghz Intel i5-3570K CPU와 NVIDIA Geforce GTX560을 사용하였으며, QVGA, VGA, SVGA, XGA, SXGA+, UXGA, QXGA 등 7종류 해상도의 비디오 영상 5개에 대해 커널별 실행시간을 분석하였다. 또한 GPU를 사용한 화재감지 알고리즘의 성능 평가를 위해 동일한 화재 감지 알고리즘에 대하여 CPU와 GPU의 실행시간을 비교하였다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 4 단계 화재감지 알고리즘 및 실험에 사용된 환경을 설명한다. 3장에서는 화재감지 알고리즘의 병렬 구현 방법을 기술하며, 4장에서는 다양한 해상도의 5가지 영상에 대한 알고리즘의 실행시간을 분석하는 동시에 GPU기반 알고리즘 성능을 CPU 기반 알고리즘 성능과 비교 분석한다. 마지막으로 5장에서는 본 논문의 결론을 맺는다.

II. 관련 연구

본 논문에서는 YCbCr 색 공간을 이용한 색상 분할 단계, 배경 차감 알고리즘을 이용한 움직임 영역 감지단계, 이산 웨이블릿 변환(discrete wavelet transform, DWT)을 이용한 화재 특징 추출단계, SVM을 이용한 화재 분류 단계로 구성되는 기존 4단계 화재감지 알고리즘을 설명하고, 이를 실시간으로 처리하기 위해 GPU기반 구현 방법을 제시한다.

1. 색상 분할

색상 분할 단계에서는 입력 영상의 화소별 색상을 분석하여 화재로 추정되는 화소를 분리한다. 본 논문에서는 화재의 색상뿐만 아니라 주변과의 휘도 차이를 이용하여 화재의 색상을 분할하기 위해 입력 RBG 영상에서 YCbCr 영상으로 색 공간을 변환한다. 색 공간을 변환하는 식은 아래와 같다[9].

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.2568 & 0.5041 & 0.0979 \\ -0.1482 & -0.2910 & 0.4392 \\ 0.4392 & -0.3678 & -0.0714 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16.0 \\ 128.0 \\ 128.0 \end{bmatrix} \quad (1)$$

식 (1)에서 Y는 휘도를, Cb와 Cr은 각각 빨간색과 파란색의 색차를 이용한 색상 성분을 의미한다. YCbCr 색 공간으로 변환된 각 픽셀을 아래 식 (2)에 적용하여 화재 후보영역을 결정한다.

$$F_Y(x,y) = \begin{cases} 1 & Y(x,y) > y_{mean}, Cb(x,y) < Cb_{mean}, Cr(x,y) < Cr_{mean} \\ 0 & otherwise \end{cases} \quad (2)$$

식 (2)에서 Y_{mean}, Cb_{mean}, Cr_{mean}는 전체 이미지에 대한 채널별 평균을 계산한 결과이다. 대체로 비 화재 영역의 픽셀보다 높은 휘도를 가지는 화재 영역 픽셀의 특징을 이용하여 Y_{mean}보다 높은 Y를 가지는 픽셀을 화재 후보 영역으로 설정 한다. 또한 화재 영역은 그렇지 않은 영역보다 더 높은 Cr성분을 가지고 있기 때문에 현재 프레임의 Cr_{mean}보다 높은 Cr성분을 가지고 있는 픽셀을 화재 후보 영역으로 간주한다. Cb성분의 경우에도 동일한 이유로 Cb_{mean}보다 낮은 픽셀을 화재 후보 영역으로 간주한다.

2. 움직임 영역 감지

화재의 특징 중 하나는 화염이나 연소하고 있는 물체로 인하여 움직임이 발생한다는 점이다. 따라서 앞의 색상 분할 단계에서 선택된 화재 후보 영역 중에 움직임이 있는 영역을 화재 후보 영역으로 간주한다. 움직임 영역을 검출하기 위해여 배경 감산 알고리즘을 이용한다[7].

$$I_{Current}(x,y) - I_{BG}(x,y) > Th \quad (3)$$

수식 (3)에서 $I_{Current}$는 입력된 영상의 명도를, I_{BG}는 배경 영상의 명도를 나타낸다. 임계값(Th)은 모의실험을 통해 '3'으로 정하였으며 위 식을 만족하면 움직임이 있는 영역으로 구분하였다. I_{BG}는 아래 식 (4)와 같이 매 프레임 갱신된다.

$$I_{BG_{n+1}}(x,y)=\begin{cases} I_{BG_n}(x,y)+1 & \text{if } I_{Current}(x,y) > I_{BG_n}(x,y) \\ I_{BG_n}(x,y)-1 & \text{if } I_{Current}(x,y) < I_{BG_n}(x,y) \\ I_{BG_n}(x,y) & \text{if } I_{Current}(x,y) = I_{BG_n}(x,y) \end{cases} \quad (4)$$

각 프레임의 배경은 수식 (4)와 같이 이전 프레임까지 갱신된 배경 영상에서 현재 프레임의 영상을 비교하여 갱신한다. 다만 알고리즘이 처음 수행될 때는 I_{BG}를 가지고 있지 않기 때문에 첫 번째 프레임의 회색조 영상을 배경으로 사용한다.

3. 특징 추출

화재가 발생한 영역은 화재의 흔들리는 모양, 화재의 형태 및 색상, 질감 변화와 같은 특징을 이용하여 화재가 아닌 물체와 구별할 수 있다. 이러한 특징을 추출하기 위해서 널리 사용되는 있는 DWT 방법을 이용한다. 영상정보는 2차원의 형태로 입력되기 때문에 2차원 DWT를 이용한다. 2차원 DWT는 1차원 DWT를 확장하여, 열 방향과 행 방향으로 각각 수행되며, 고대역 통과 필터와 저대역 통과 필터를 적용하는 순서에 따라 저대역-저대역(LL), 저대역-고대역(LH), 고대역-저대역(HL), 고대역-고대역(HH)의 4가지 부대역(subband) 이미지를 얻을 수 있다. 그림 2는 2차원 DWT의 수행 방법을 보여준다[11].

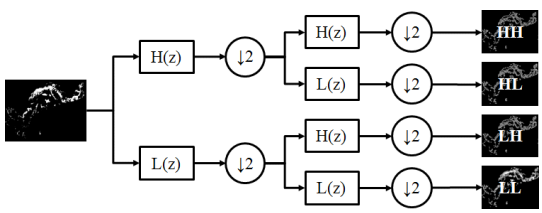

그림 2. 2차원 DWT 방법
Fig. 2. 2-dimensional DWT method

4. 분류 알고리즘

서포트 벡터 머신 (SVM)은 분류 속도가 빠르고, 적은 학습 데이터로 높은 분류 성능을 보이는 장점이 있기 때문에 화재 및 비화재 프레임을 판별하기 위한 알고리즘으로 사용한다. SVM 분류 함수는 다음과 같이 정의된다[11].

$$f(x) = sign\left(\Sigma_{i=1}^{l} w_i \bullet k(x,x_i) + b\right) \quad (5)$$

수식 (5)에서 w_i는 각 커널의 출력에 대한 가중치, $k()$는 커널 함수, b는 바이어스 항을 나타내며, l은 서포트 벡터의 개수이고, $sign()$은 x가 속한 클래스를 의미하며 1과 -1을 결정한다. 즉 SVM 분류기는 1과 -1로 분류하는 이진 분류기이다. 2.3절에서의 결과인 웨이블릿 에너지를 분류하기 위한 서포트 벡터는 선형적으로 분리되지 않기 때문에 이러한 벡터들을 분리할 수 있는 최적 초평면을 찾기 위해 식 (6)의 radial basis function (RBF) 커널을 사용한다.

$$k(x,y) = \exp\left(-\frac{||x-y||^2}{2\sigma^2}\right) \quad \text{for } \sigma > 0 \quad (6)$$

위 식에서 x와 y는 입력 특징 벡터이고, σ는 기저 함수의 폭을 결정하는 변수이다. σ값은 분류의 정확도에 영향을 미치는데, 본 논문에서는 실험을 통해 가장 높은 분류 성능을 보여주는 0.1을 σ값으로 선택하였다.

5. GPU 아키텍처

GPU는 그래픽 처리의 가속화를 위해 만들어 졌고, 이를 범용적인 용도로 사용하기 위해 GPGPU 기술이 개발됨으로써 멀티미디어 데이터의 병렬 처리를 통해 멀티미디어 애플리케이션을 빠르게 수행할 수 있다. 그림 3은 본 논문에서 사용한 GTX 560의 아키텍처를 보여준다. GTX 560은 7개의 스트리밍 멀티프로세서(streaming multiprocessor, SM)를 포함하고, 각 SM은 명령어를 인출(fetch)하여 각 SM이 가지고 있는 스트리밍 프로세서(streaming processor, SP)에 전달하여 SIMT 방식으로 동작한다[15].

그림 3. Geforce GTX 560 아키텍처
Fig. 3. Geforce GTX 560 architecture

그림 4 실험에 사용된 영상
Fig. 4 Movies used in the experiment

NVIDIA사는 GPU를 범용으로 사용하는 프로그램을 작성할 수 있도록 프로그래밍 프레임워크인 CUDA를 제공한다. CUDA에서 CPU는 호스트(host), GPU를 디바이스(device)라고 하며, 양쪽의 데이터를 주고받기 위해 그래픽카드의 전역 메모리를 사용한다. 디바이스에서 수행되는 함수는 커널(kernel) 함수라고 하며, 커널 함수에서 사용될 데이터는 CPU에서 그래픽카드의 전역 메모리에 복사되어 있어야 한다. 마찬가지로 GPU에서 수행한 결과를 호스트에서 처리하기 위해서는 다시 전역 메모리에서 메인 메모리로 결과를 전송해야 한다. 한편 CUDA는 스레드와 블록 단위로 연산을 수행한다. 각 스레드와 블록은 그래픽카드의 여러 고유한 메모리에 접근할 수 있다. GPGPU의 메모리 종류로는 크게 레지스터, 공유 메모리, 전역 메모리가 있다. 레지스터와 공유 메모리는 GPU 코어 내부의 메모리이며, 수 사이클 안에 데이터를 읽고 쓸 수 있다. 전역 메모리는 GPU 코어 외부의 그래픽카드에 있는 메모리이다. 비록 GPU와 GPU 전역 메모리 사이에 L2 캐시가 존재하여 메모리 접근성능을 향상시킬 수 있지만 캐시 미스가 발생하는 경우에는 접근하는데 수백 사이클이 걸린다. 그렇기 때문에 전역 메모리를 사용하는 대신 레지스터와 공유 메모리만을 사용한다면 매우 빠르게 연산을 수행할 수 있겠지만 다음과 같은 이유로 전역메모리를 이용한다. 첫째, 레지스터와 공유 메모리는 스레드 간에 동기화가 가능하지만 블록 간 동기화가 이루어지지 않는다. 둘째, 공유 메모리는 크기가 표 1에서와 같이 48KB로 한정되어있다. 셋째 전역 메모리는 비록 접근속도가 느리지만, 블록간의 동기화가 가능하고, 크기가 수 GB로 매우 크다. 따라서 알고리즘을 구현할 때, 이들 메모리간의 특성을 고려해야 한다.

본 논문에서는 3.40Ghz Intel i5-3570K CPU와 NVIDIA Geforce GTX560 GPU를 이용하여 실험하였으며, 표 1은 본 논문에서 사용한 GPU와 CPU 사양을 보여준다. 또한 GPU 기반 화재 감지 알고리즘의 성능을 비교하기 위해 그림 4와 같이 화재 및 비 화재 영상 5개를 각각 QVGA, VGA, SVGA, XGA, SXGA+, UXGA, QXGA해상도로 변환하여 이용하였다.

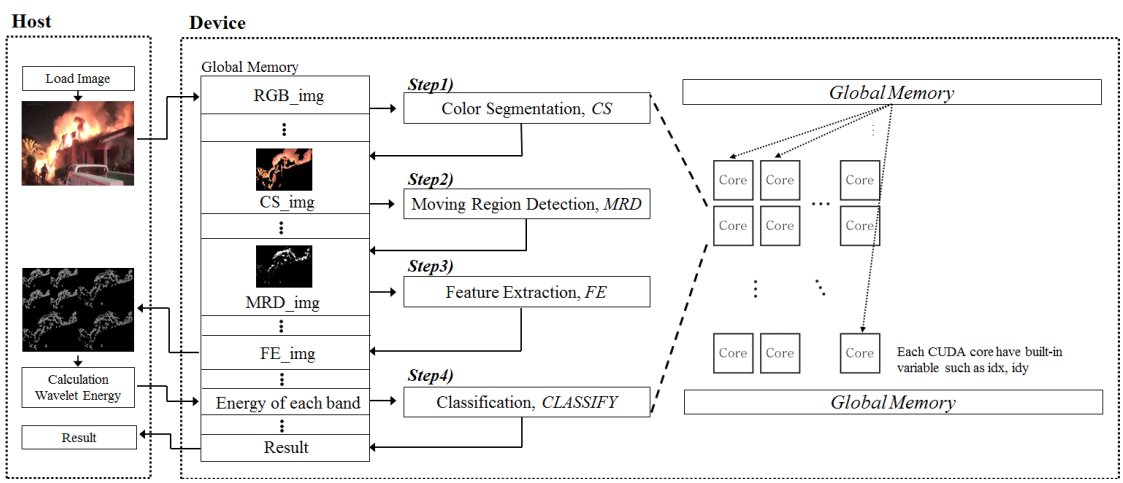

그림 5 화재 감지 알고리즘의 병렬 구현
Fig. 5 Parallel implementation of a fire detection algorithm

표 1. CPU와 GPU의 프로세서 사양
Table 1. Specification of the CPU and GPU

항목		값
GPU	Number of SM	7
	Number of CUDA cores	336
	GPU clock rate	1620Mhz
	Maximum threads/SM	1536
	Maximum sizes of a block	$1024 \times 1024 \times 64$
	Maximum size of a grid	$65535 \times 65535 \times 65535$
	Shared memory/SM	48KB
	Warp size	32
CPU	Processor	Intel Core i5-3570K
	CPU clock rate	3.40GHz
RAM		8.00GB

III. 화재 감지 알고리즘의 병렬구현

GPU를 이용한 영상처리는 영상의 지역성과 규칙성을 이용하여 영상처리 알고리즘의 병렬성을 최대한 이용하여 구현하면 최대의 효과를 얻을 수 있다. 본 논문의 4단계 화재감지 알고리즘에서, 색상 분할과 움직임 영역 감지 단계에서는 인접한 픽셀에 상관없이 각 픽셀에 대해 독립적인 연산이 가능하기 때문에 병렬성이 높다 볼 수 있다. 반면 화재 특징 추출 단계에서는 이웃한 픽셀과의 연관성이 중요하기 때문에 상대적으로 병렬성이 낮다. 그림 5는 4단계 화재감지 알고리즘의 호스트와 디바이스 사이의 단계별 데이터 이동을 보여준다. 먼저 입력된 영상 정보와 초기 변수들을 GPU 전역 메모리에 저장하고 이미지 데이터를 256개 픽셀로 이루어진 16x16 블록들로 분할하여 각 블록의 픽셀을 계산하는 256 스레드를 만들며, 이미지 크기에 맞게 블록의 개수를 설정한다. 할당된 스레드들은 특정 좌표를 갖게 되며 이를 이용하여 픽셀 데이터를 각 SP에서 동시에 수행하게 된다.

1. 색상 분할

색상 분할 단계에서는 각 프레임의 Y_{mean}, Cb_{mean}, Cr_{mean} 이 필요하다. 그러나 각 채널의 평균을 구하는 과정에서, 전체 프레임의 합을 구하기 위해 순차적으로 더해나가면 GPU가 CPU보다 처리속도가 느릴 수 있다. 멀티코어인 GPU의 장점을 활용하기 위해 그림 6과 같이 반복문에서 계산해야 할 픽셀의 개수를 절반으로 줄이면서 계산한다. 이때 GPU 하드웨어의 제약으로 모든 스레드가 동시에 동작하지 않기 때문에 데이터간의 비동기화가 발생할 수 있다. 각 반복이 끝날 때 마다 모든 스레드가 연산을 마치기 위해 __syncthread()를 이용한다. 실제 구현은 그림 7과 같으며, 각 스레드에서 계산된 결과는 CPU로 전송되는 동시에 각 블록이 가진 픽셀의 합을 다시 더한다. 그림 8의 커널 함수와 같이 주어진 조건을 만족하면 RGB로 변환한 값을 저장하여 화재 후보 값으로 설정한다.

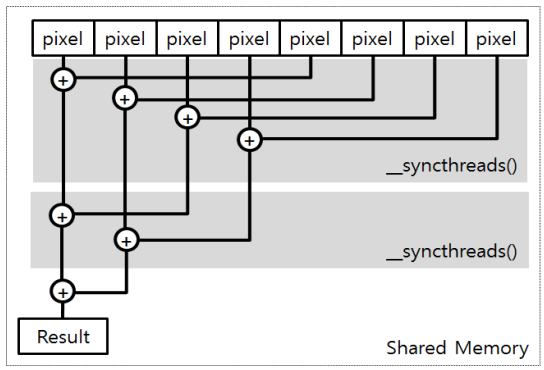

그림 6. 픽셀 합의 계산 방법
Fig. 6. Calculation method of pixel summation

```
// each Thread do in parallel
   threadSize = number of Thread
   remainThread = threadSize/2
   IF ThreadIdx ≤ remainThread
      share_ycbcr[ThreadIdx]
            = ycbcr[Block_i + ThreadIdx]
                + ycbcr[Block_i + ThreadIdx + remainThread]
   ENDIF
   __syncthreads()
   FOR each loop remainThread = remainThread/2 until remainThread >= 1
      share_ycbcr[ThreadIdx]
            = share_ycbcr[ThreadIdx]
                        + share_ycbcr[ThreadIdx+remainThread]
      __syncthreads()
   ENDFOR
   sum_of_Block_i(Y) = share_ycbcr(Y)
   sum_of_Block_i(Cb) = share_ycbcr(Cb)
   sum_of_Block_i(Cr) = share_ycbcr(Cr)
ENDFOR

FOR i = 0 to number of Block
   Y_sum  = Y_sum + sum_of_Block_i(Y)
   Cb_sum  = Cb_sum + sum_of_ Block_i(Cb)
   Cr_sum  = Cr_sum + sum_of_ Block_i(Cr)
ENDFOR
```

그림 7. 픽셀 합 계산 커널 함수
Fig. 7. Kernel function of pixel summation

```
//each pixel p_idx,idy do in parallel

   RGB convert to YCbCr
   IF ((Y>Ymean) & (Cb<Cbmean) & (Cr>Crmean))
      CS_img[idx, idy] = YCbCr convert to RGB
   ELSE
      CS_img[idx,idy] = 0
   ENDIF
```

그림 8. 색상 분할의 커널 함수
Fig. 8. Kernel function of color segmentation

2. 움직임 영역 감지

움직임 영역 감지 단계에서 각 스레드는 한 개 픽셀들을 담당하며, 그림 9와 같이 수행된다. 다른 위치의 픽셀과 연관성 없이 독립적인 연산이 가능하기 때문에 병렬성이 가장 높은 단계이다. 먼저 1단계의 색상 분할 단계에서 결정된 1차 화재 후보 영역을 그레이스케일(grayscale) 영상으로 변환한 후, 만약 배경 영상과 현재 영상사이 차이가 기준치(threshold) 이상이면 원래 픽셀 값을, 그렇지 않으면 '0'을 저장한다. 또한 주변 환경에 의해 영상의 흔들림 현상이 발생할 수 있으므로 위와 같은 방법으로는 완전한 결과를 출력할 수 없다. 따라서 본 논문에서는 위와 같은 단점을 보완하기 위해 배경 영상을 현재 영상의 상태와 비교하여 지속적으로 갱신한다. 본 논문에서는 기준치를 모의실험을 통해 '3'으로 정하였다.

```
//each pixel p_{idx,idy} do in parallel

Gray_img[idx,idy] = Candidate fire1 convert to Gray

IF Gray_img[idx,idy] - Gray_bg_img[idx,idy] > threshold
   MRD_img[idx,idy] = Gray_img[idx,idy]
ELSE
   MRD_img[idx,idy] = 0
ENDIF

IF Gray_img[idx,idy] > Gray_bg_img[idx,idy]
   Gray_bg_img[idx,idy]++
ELSE IF Gray_img[idx,idy] < Gray_bg_img[idx,idy]
   Gray_bg_img[idx,idy]--
ENDIF
```

그림 9. 움직임 영역 감지 커널 함수
Fig. 9. Kernel function of moving region detection

3. 특징 추출

DWT를 이용한 특징 추출 단계의 모의실험 결과, 1단계 DWT에 비해 2단계 DWT에서 더욱 좋은 성능을 보였기 때문에 본 논문에서는 2단계 DWT를 사용하여 특징을 추출하였다. 2단계 DWT 특징 추출 방법은 그림 2와 같이 1단계 DWT를 바탕으로 LL대역의 서브밴드에 다시 동일한 방법의 DWT을 적용하여 계산한다. DWT를 수행하면서 여러 분기문을 거치게 되는데 이 때 발생할 수 있는 GPU의 비효율적인 특성을 최소화하기 위해 그림 10과 같이 연산 순서를 변경하였다. 2단계 DWT의 병렬 처리 과정 중에 고대역 통과 필터와 저대역 통과 필터를 적용하기 위한 컨볼루션 연산 자체는 다른 픽셀에 독립적으로 수행할 수 있기 때문에 스레드 간의 충돌이 발생하지 않는다. 하지만 각 필터를 적용하는 순간에 다른 필터가 적용되면 잘못된 결과를 얻을 수 있다. 따라서 행 방향 연산이 진행되는 동안 열 방향 연산이 수행되면 잘못된 결과 값을 얻을 수 있으므로, 행 방향 연산을 완료한 후 열 방향 연산이 수행되기 위해 블록간의 동기화가 필요하다. 또한 병렬성을 최대한으로 활용하기 위해 고대역 통과필터와 저대역 통과필터는 분기문 없이 전체이미지에 각각 적용한 뒤, 열 방향과 행 방향의 다운 샘플링 커널함수에서 동시 처리한다. 이러한 기능들은 서로 분리된 커널로 구성하여 서로 간의 연산이 간섭되지 않게 하였다. 마지막으로 2단계 DWT의 결과 중 LLLH, LLHL, LLHH를 CPU로 전송하여 웨이블릿 에너지의 평균값을 계산한 후에 다음 단계인 화재 분류 알고리즘의 입력으로 사용하였다.

4. 분류 알고리즘

화재 분류 단계에서는 SVM을 이용하여 현재 프레임의 화재포함 여부를 판별한다. SVM의 경우 사전 학습이 필요하다. 따라서 본 논문에서는 각각 200개의 화재 영상 및 비 화재 영상을 이용하여 150개의 서포트 벡터 데이터를 추출하였다. 그러나 구현 결과 GPU에서 병렬로 수행한 방법이 CPU보다 시간이 더 소요되었는데, 이는 구현한 분류 알고리즘의 순차적인 연산이 많아 병렬성이 떨어지고, 알고리즘의 수행시간 보다 데이터를 GPU로 전송하는데 걸리는 시간이 더 많이 소요되었기 때문이다. 따라서 이 부분은 CPU로 처리하였다.

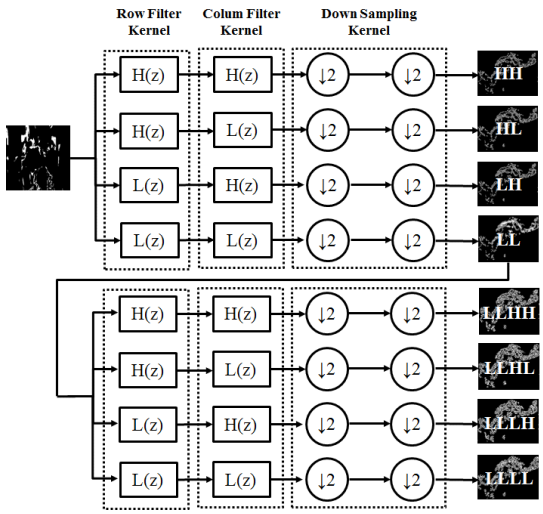

그림 10 특징 추출의 커널 함수
Fig. 10 Kernel function of feature extraction

IV. 실험 결과

1. 실행시간

실행시간은 각 비디오 프레임에 대한 평균 실행시간을 나타낸다. 그림 11은 여러 가지 해상도의 5가지 비디오 영상에 대한 GPU와 CPU의 실행시간을 비교한 결과를 보여 준다. 비교 결과 전체 해상도에서 GPU는 CPU보다 실행시간에서 우수한 성능을 보여주며, 해상도가 높아짐에 따라 실행시간이 오래 걸리는 것을 알 수 있다. 실행시간 증가의 형태는 전체 화소수 증가 추이와 비슷하게 증가하는 것을 알 수 있다. CPU는 실험에서 쓰인 가장 낮은 해상도인 VGA에서도 40.32ms이 소요되는 것에 비해 GPU는 실험에서 가장 높은 해상도인 QXGA에서도 40.50ms의 실행시간 만이 요구되었다. 결과적으로 GPU에서는 QXGA를 제외한 QVGA, VGA, SVGA, XGA, SXGA+, UXGA해상도에서 각각 2.28ms, 5.51ms, 7.98ms, 11.58ms, 20.31ms, 25.11ms의 실행시간이 소요되어 실시간 영상처리에서 요구되는 초당 30 프레임 처리(30 fps)를 모두 만족하였다.

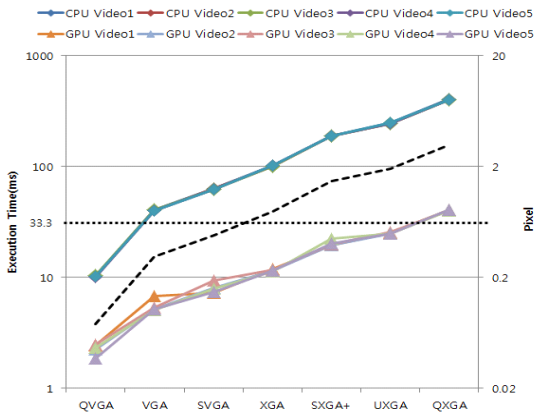

그림 11 CPU와 GPU를 이용한 비디오별 실행시간
Fig. 11 Execution times of CPU and GPU for each video

2. 픽셀 처리시간 비교

그림 12는 5가지 비디오 영상의 픽셀 당 처리시간을 보여준다. 픽셀 당 처리시간은 한 프레임의 한 개 픽셀을 처리하는데 소요되는 평균 실행 시간을 나타낸다. CPU와 GPU 모두 해상도가 증가할수록 하나의 픽셀을 처리하는데 소요되는 실행 시간이 적게 소요되는데, 이는 각 프레임을 처리할 때 메모리 할당 시간은 고정적인 반면, 해상도에 따른 처리 시간은 비례하기 때문이다. 하지만 비교 그래프에서 CPU는 픽셀 처리시간에서 가장 느린 경우와 빠른 경우의 비율이 1.07로

표 2. 커널별 CPU와 GPU의 실행시간 비교
Table 2. Execution time comparison of CPU and GPU for each Kernel

Resolution	System	CS	Speed up	MRD	Speed up	FE	Speed up
QVGA	CPU	3.03	1.726	1.55	35.046	5.663	12.663
	GPU	1.76		0.04		0.447	
VGA	CPU	11.8	3.116	5.95	43.549	22.56	14.474
	GPU	3.79		0.14		1.559	
SVGA	CPU	18.3	3.303	9.19	45.950	35.22	15.932
	GPU	5.54		0.20		2.211	
XGA	CPU	29.5	3.993	14.8	43.368	57.37	15.012
	GPU	7.39		0.34		3.822	
SXGA+	CPU	54.6	4.098	27.0	47.249	106.6	16.695
	GPU	13.3		0.57		6.388	
UXGA	CPU	71.0	4.454	35.1	41.917	139.2	16.783
	GPU	15.9		0.84		8.295	
QXGA	CPU	115	4.579	57.0	41.214	227.7	16.472
	GPU	25.3		1.38		13.82	

차이가 적은 것에 비해 GPU는 2.55로 처리시간의 차이가 큰 것을 알 수 있다. GPU연산을 위해서는 시스템 메모리에서 GPU 전역 메모리로 데이터 전송이 필요한데, 전송에 요구되는 시간이 전체 처리시간에서 차지하는 비율이 커지기 때문이다. 또한 저해상도에서는 블록 당 스레드의 개수가 적기 때문에 활성블록의 비율이 낮아지고, 따라서 GPU의 자원을 효율적으로 사용하지 못하기 때문이다.

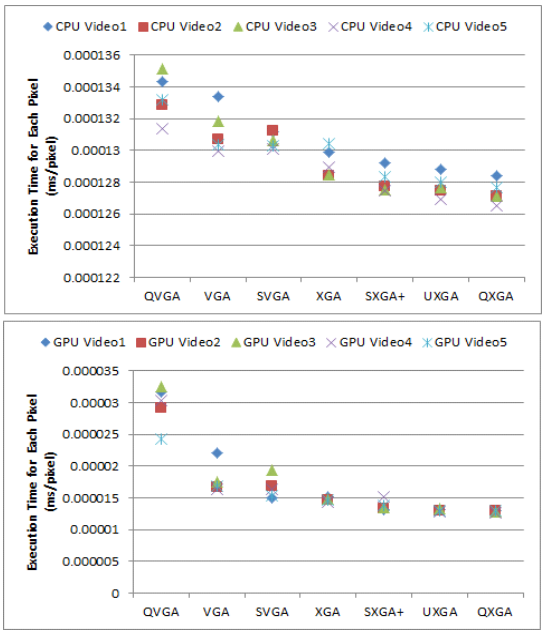

그림 12 서로 다른 비디오 해상도별 픽셀 처리시간 비교
Fig. 12 Comparison of execution time per pixel for different video resolutions

3. 커널별 실행시간 비교

표 2는 해상도별 각 커널의 실행시간 및 성능향상을 보여준다. 여기서 CS는 색상 분할 단계, MRD는 움직임 추정 단계, FE는 특징 추출 단계를 의미한다. CS단계에서는 성능향상 폭이 1.7에서 4.5 정도인데, 이는 각 픽셀의 합을 구하는 과정이 포함되어 비교적 낮은 병렬성을 가지고 있기 때문이다. MRD단계에서는 현재 영상과 이전 영상에 대한 픽셀 데이터들의 단순 비교를 반복하기 때문에 높은 병렬성으로 성능 향상 폭이 크다. 마찬가지로 FE단계에서도 단순 반복연산인 회선 연산이 많은 부분을 차지하여 높은 병렬성을 보이기 때문에 비교적 성능 향상 폭이 크다.

V. 결 론

본 논문에서는 비디오 기반 화재감지 알고리즘을 다양한 해상도에서 실시간 처리가 가능하도록 GPU기반 병렬 구현 방법을 제시하였다. GPU상에서 QVGA, VGA, SVGA, XGA, SXGA+, UXGA, QXGA 등 7가지 해상도의 5개 화재 또는 비 화재 비디오 영상에 따른 화재감지 알고리즘의 실행시간을 분석하였으며, CPU 구현과의 성능을 비교 분석하였다. 모의실험 결과, CPU는 VGA급 해상도에서도 40.32ms의 실행시간이 소요되어 실시간 처리에서 요구되는 초당 30 프레임 (30fps 혹은 프레임 당 33ms)을 만족하지 못하였으나, GPU는 2백만 화소에 가까운 UXGA해상도에서 각 프레임 당 25.11ms의 실행시간을 보여 초당 30프레임의 실시간 처리가 가능함을 보였다.

참고문헌

[1] National Emergency Management Statistics, National Emergency Management Agency, 2013.

[2] S. M. Kang, J. M. Kim, "Survey for Early Detection Techniques of Smoke and Flame using Camera Images," Journal of the Korea society of computer and information, vol.16, no.4, pp.43-52, 2011.

[3] Z, Zhang, T. Shen, J. Zou, "An Improved Probabilistic Approach for Fire Detection in Videos," Fire Technology, vol. 50, no. 3, pp.745-752, 2014.

[4] D. C. Wang, X. Cui, E. Park, C. Jin, H. Kim, "Adaptive flame detection using randomness testing and robust features," Fire Safety Journal, vol. 55, pp. 116-125, 2013.

[5] D, H, Lee, J, W, Yoo, K. H. Lee, Y. Kim, "Real Time Flame and Smoke Detection Algorithm Based on Conditional Test in YCbCr Color Model and Adaptive Differential Image," Journal of the Korea society of computer and information, Vol. 15, no. 5, pp. 57-65, 2010.

[6] I. kolesov, P. Karasev, A. Tannenbaum, and E.

Haber, "Fire and Smoke Detection in Video with Optimal Mass Transport Based Optical Flow and Neural Networks," in Proc. 2010 IEEE International Conference Image Processing, Hong Kong, pp. 761-764, 2010.

〔7〕 B. Lee and D. Han, "Real-Time Fire Detection Using Camera Sequence Image in Tunnel Environment," Lecture Notes in Computer Science, vol. 4681, pp. 1209-1220, 2007.

〔8〕 X. Qi and J. Ebert, "A Computer Vision Based Method for Fire Detection in Color Videos," International Journal of imaging and Robotics, vol. 2, no. 9, pp. 22-34, 2009.

〔9〕 T. Celik, H. Demirel, H. Ozkaramanli, and M. Uyguroglu, "Fire Detection Using Statistical Color Model in Video Sequences," Journal of Visual Communication and Image Representation, vol. 18, no. 2, pp. 176-185, 2007.

〔10〕 T. Qiu, Y. Yan, and G. Lu, "An Autoadaptive Edge-Detection Algorithm for Flame and Fire Image Processing," IEEE Transactions on Instrumentation and Measurement, vol. 61, no. 5, pp. 1486-1493, 2012.

〔11〕 B. C. Ko, K. H. Cheong, and J. Y. Nam, "Fire Detection Based on Vision Sensor and Support Vector Machines," Fire Safety Journal, Vol. 41, no. 3, pp. 322-329, 2009.

〔12〕 S. M. Kang, J. M. Kim, "Multimedia Extension Instructions and Optimal Many-core Processor Architecture Exploration for Portable Ultrasonic Image Processing," Journal of the Korea society of computer and information, vol.17, no.8, pp.1-10, 2012.

〔13〕 I. K. Park, N. Singhal, M. H. Lee, S. Cho, and C. W. Kim, "Design and Performance Evaluation of Image Processing Algorithm on GPUs," IEEE Transactions on Parallel and Distributed Systems, vol.22, no.1, pp.91-104, 2011.

〔14〕 I. K. Park, N. Singhal, M. H. Lee, S. Cho, and C. W. Kim, "Design and Performance Evaluation of Image Processing Algorithms on GPUs," IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 1, pp. 91-104, 2011.

〔15〕 P. N. Glaskowsky, "NVIDIA's Fermi: the first complete GPU computing architecture." White paper, 2009.

저 자 소 개

손 동 구

2013: 울산대학교
컴퓨터정보통신공학부 공학사.
현 재: 울산대학교
전기전자컴퓨터공학과 석사과정.
관심분야: 병렬프로세서 구조,
GPGPU 프로그래밍,
임베디드시스템
Email : dongkoo88@gmail.com

김 철 홍

1998 : 서울대학교 컴퓨터공학사.
2000 : 서울대학교 컴퓨터공학부 석사.
2006 : 서울대학교
전기컴퓨터공학부 박사
2005-2007년 : 삼성전자 반도체총괄
책임연구원
2007-현재 : 전남대학교
전자컴퓨터공학부 교수
관심분야 : 임베디드시스템,
컴퓨터구조, SoC설계,
저전력 설계
Email: cheolhong@gmail.com

김 종 면

1995: 명지대학교 전기공학과 공학사.
2000: University of Florida
전기컴퓨터공학과 공학석사.
2005: Georgia Tech
전기컴퓨터공학과 공학박사
2007-현 재: 울산대학교
전기공학부 교수
관심분야: 임베디드 SoC, 병렬처리.
Email : jmkim07@ulsan.ac.kr