Journal of The Korea Society of Computer and Information
Vol. 19, No. 8, August 2014
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2014.19.8.001

GPGPU를 이용한 비디오 기반 실시간 화재감지 알고리즘 구현

손 동 구*, 김 철 홍**, 김 종 면*

An Implementation of a Video-Equipped Real-Time Fire Detection Algorithm Using GPGPU

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

요 약

본 논문에서는 많은 양의 연산량을 요구하는 비디오 기반 4단계 화재감지 알고리즘의 실시간 처리를 위해 범용 그래픽 처리 장치 (general-purpose graphics processing unit, GPGPU)를 이용한 병렬 구현 방법을 제안한다. 또한 GPGPU 기반 화재감지 알고리즘의 효용성을 확인하기 위해 범용 고성능 CPU와의 성능을 비교하였다. SXGA(1400×1050) 해상도의 화재 비디오 5개를 이용해 모의실험 결과, GPGPU기반 화재감지 알고리즘은 CPU 구현보다 약 6.6배 더 높은 성능을 보였으며, 평균 프레임 당 30.53ms의 실행시간이 소요되어 실시간 처리(초당 30프레임)가 가능함을 보였다.

▸ Keywords : 분화재 감지 알고리즘; 범용 그래픽 처리 장치; 실시간 처리

Abstract

This paper proposes a parallel implementation of the video based 4-stage fire detection algorithm using a general-purpose graphics processing unit (GPGPU) to support real-time processing of the high computational algorithm. In addition, this paper compares the performance of the GPGPU based fire detection implementation with that of the CPU implementation to show the effectiveness of the proposed method. Experimental results using five fire included videos with an SXGA (1400×1050) resolution, the proposed GPGPU implementation achieves 6.6x better

•제1저자 : 손동구 •교신저자 : 김종면
•투고일 : 2014. 7. 1, 심사일 : 2014. 7. 27, 게재확정일 : 2014. 8. 19.
* 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).

performance that the CPU implementation, showing 30.53ms per frame which satisfies real-time processing (30 frames per second, 30fps) of the fire detection algorithm.

▸ Keywords : AFire detection algorithm; General-purpose graphics processing unit; Real-time processing

I. 서 론

유화재는 많은 인명과 막대한 재산의 피해를 가져 올 수 있기 때문에 조기에 화재를 검출하는 시스템 개발의 필요성이 증가하고 있다. 화재를 검출하는 방법으로 온•습도 센서, 연기 센서 등을 사용하는 센서 기반의 방법이 있다. 하지만 센서를 이용하였을 경우, 센서의 감지 범위 안에 화재가 존재해야하기 때문에 즉각적으로 화재의 여부를 확인할 수 없는 단점이 있다[1]. 이와 같은 단점을 극복하기 위해 CCTV 등의 영상정보를 이용한 화재감지 연구가 활발히 진행되고 있다[2-13].

본 논문에서는 영상 정보를 이용한 화재감지 알고리즘을 크게 4단계로 구성한다. 그림 1은 선행 연구에서 각 단계별로 사용했던 방법들을 보여준다. 첫 번째 단계인 색상 분할은 화재가 가지는 색상에 대한 정보를 이용하여 화재 영역을 분리하는 단계이며, RGB(R:빨강색, G:녹색, B:청색)[2-3], YCbCr (Y:휘도성분, CrCb: 색차성분)[1][4-5] 등의 색 공간에서 화재 색상을 분리하는 연구가 선행되어 왔다. 이 중 YCbCr 색 공간은 화재 색상뿐만 아니라 주변과의 조도 차이를 이용하기 때문에 화재가 가진 색상 특징을 분리하는 것이 용이하다. 움직임 추정 단계에서는 광류[6], 시간차[7], 가우시안 혼합 모델[8], 배경 감산[3][7][9] 등의 방법이 사용되어 왔다. 이 중 배경 감산 방법은 움직이는 물체를 빠른 시간 내에 분리 할 수 있기 때문에 널리 사용되는 방법 중의 하나이다. 움직임 추정 단계를 통해 얻은 화재 후보 영역은 화재 색상과 비슷한 빨간색 차량, 빨간색 옷을 입은 사람 등이 남아 있을 수 있다. 따라서 화재의 흔들림, 화재의 색상 및 형태, 질감 변화 등 색상 이외의 화재 특징을 추출하는 방법이 선행되어 왔다[3-4][7-8][10-13]. 마지막으로 화재의 분류 방법으로는 서포트 벡터 머신 (support vector machine, SVM)[3][11], 뉴럴 네트워크 (neural network, NN)[6] 등이 있다. 이 중 뉴럴 네트워크는 지역 최소 점에 빠질 수 있고, 높은 분류 성능을 위해서는 대량의 학습데이터가 필요하다. 반면에 SVM은 적은 학습데이터로 높은 성능을 보장할 수 있기 때문에 본 논문에서 요구하는 실시간 처리를 만족시키기에 용이하다. 이와 같은 4단계의 화재감지 알고리즘은 높은 정확도를 제공하지만 알고리즘의 복잡성으로 인해 많은 양의 연산을 요구하는 단점이 있다.

이러한 문제를 해결하기 위한 대안 중의 하나로 칩 내부에 다수의 코어를 탑재하여 높은 성능을 기대할 수 있는 멀티코어(multi-core)프로세서가 유망하다[14-15]. 멀티코어 프로세서 중 GPU (graphics processing unit)는 현재 널리 사용되고 있는 상용화된 프로세서 중의 하나이며, 기본적으로는 SIMD (single instruction multiple data) 방식으로 동작하지만 더욱 발달된 형태로써 하나의 명령어로 많은 쓰레드를 제어하는 SIMT (single instruction multiple thread) 방식으로 동작한다. GPU를 범용적인 용도로 사용하기 위해 GPGPU (general-purpose graphics processing unit)가 출시되었으며, CUDA (compute

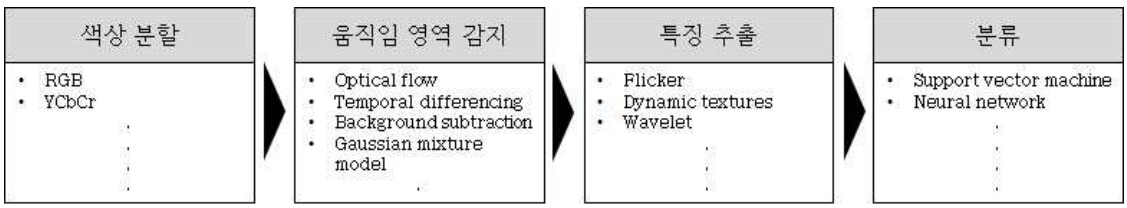

그림 1. 4단계 화제 감지 알고리즘
Fig. 1. 4-stage fire detection algorithm

unified device architecture)와 OpenCL (open computing language) 등의 프로그래밍 프레임워크를 이용한다. GPGPU는 CPU에서 빠르게 처리하기 힘든 대량의 기하 및 화소 데이터들을 병렬처리를 통해 고속으로 처리할 수 있다. 특히 많은 픽셀 데이터를 포함하고 있는 고 해상도의 영상을 이용하여 복잡한 연산을 반복적으로 수행하는 화재감지 알고리즘의 경우, GPGPU를 사용하여 화재감지 알고리즘의 입력으로 사용되는 영상의 픽셀 간 지역성과 규칙성을 이용함으로써 빠른 처리가 가능하다[16].

본 논문에서는 CUDA 프레임워크를 이용하여 GPGPU 상에서 4단계 화재감지 알고리즘을 병렬 구현한다. 실험환경으로는 3.40Ghz Intel i5-3570K CPU와 NVIDIA Geforce GTX560을 사용하며, SXGA(1400px× 1050px) 해상도의 비디오 영상 5개에 대해 커널별 실행시간을 분석한다. 또한 GPGPU를 사용한 화재감지 알고리즘의 성능 평가를 위해 동일한 화재 감지 알고리즘에 대해 CPU와 GPGPU의 실행시간을 비교한다.

본 논문의 구성은 다음과 같다. 2장에서는 구현한 화재감지 알고리즘을 설명하고, 3장에서는 병렬적으로 구현한 방법을 설명하며, 4장에서는 5가지 영상에 대한 알고리즘의 실행시간 분석 및 CPU 대비 제안한 GPGPU 방법에 대한 성능을 비교한다.

II. 관련 연구

1. 화재 감지 알고리즘

본 논문에서는 YCrCb 색 공간을 이용한 색상 분할 단계, 배경 차감 알고리즘을 이용한 움직임 추정단계, DWT (discrete wavelet transform)를 이용한 화재 특징 추출단계, SVM을 이용한 화재 분류 단계의 4단계 화재감지 알고리즘을 제안한다.

1.1 색상 분할

색상 분할 단계에서는 입력 영상의 색상을 비교하여 화재 색상을 분리한다. 본 논문에서는 화재의 색상 뿐만 아니라 주변과의 조도 차이를 이용하여 화재의 색상을 분할하기 위해 입력 RBG 영상을 YCbCr 영상으로 색 공간을 변환한다. 색 공간을 변환하는 식은 아래와 같다[9].

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

위 식에서 Y는 휘도를, Cb와 Cr은 각각 빨간색과 파란색의 색차를 이용한 색상 성분을 의미한다. YCbCr 색 공간으로 변환된 각 픽셀은 아래 식들을 이용하여 화재 후보영역을 결정한다 [9].

$$F_Y(x,y) = \begin{cases} 1 & Y(x,y) > y_{mean} \\ 0 & otherwise \end{cases}$$

$$F_{Cb}(x,y) = \begin{cases} 1 & Cb(x,y) < Cb_{mean} \\ 0 & otherwise \end{cases}$$

$$F_{Cr}(x,y) = \begin{cases} 1 & Cr(x,y) > Cr_{mean} \\ 0 & otherwise \end{cases}$$

$$F(x,y) = F_Y \times F_{Cb} \times F_{Cr} \quad (2)$$

화재 영역을 포함한 픽셀은 일반적으로 비 화재 픽셀보다 높은 휘도를 가지므로 식 (2)에서와 같이 현재 프레임의 평균보다 높은 휘도 Y를 가진 픽셀을 화재영역 후보로 둔다. 또한 화재 영역은 그렇지 않은 영역보다 더 높은 Cr성분을 가지고 있기 때문에 현재 프레임의 Cr성분 평균보다 높은 Cr성분을 가지고 있는 픽셀을 식 (2)에서는 화재영역 후보로 간주한다. Cb성분의 경우도 동일한 이유로 평균보다 낮은 픽셀을 화재영역으로 간주한다.

1.2. 움직임 영역 감지

화재는 정적인 배경에 비해 다양한 움직임을 보인다. 따라서 이전의 색상 분할 단계에서 화재 영역으로 간주된 부분을 대상으로 움직임이 있는 영역을 화재후보 영역으로 간주한다. 움직임이 있는 후보 영역을 검출하기 위해서 현재 프레임과 배경 영상의 차이를 비교한다. 다음의 수식 (3)을 만족하면 움직임이 있는 영역으로 구분하며 임계값(Th)은 모의실험을 통해 '3'으로 정하였다[3].

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

식 (3)에서 $I_{Current}$는 입력된 영상의 명도를, I_{BG}는 배경

영상의 명도를 나타낸다. 배경 영상은 매 프레임 갱신되며, 방법은 아래 식과 같다[3].

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

최초 배경은 첫 번째 프레임의 회색조 영상을 이용한다. 이후 각 프레임의 배경은 수식 (4)과 같이 이전 프레임까지 갱신된 배경영상에서 현재 프레임의 영상을 비교하여 갱신한다.

1.3. 특징 추출

화재가 발생한 영역은 화염의 질감 및 흔들림과 같은 특징으로 인해 주변의 다른 물체와 쉽게 구분할 수 있다. 이러한 특징을 이용하기 위해 화재감지 분야에서 많이 사용되는 알고리즘 중의 하나인 이산 웨이블릿 변환 (DWT)을 사용한다. 영상정보는 2차원 행렬의 형태로 입력되기 때문에 1차원 DWT를 확장한 2차원 DWT를 사용한다. 2차원 DWT는 1차원 DWT를 열 방향과 행 방향을 통해 구현된다. 2차원 DWT를 수행한 결과는 행과 열 방향으로 각각 저대역 필터와 고대역 필터를 적용한다. 적용하는 순서에 따라 저대역-저대역(LL), 저대역-고대역(LH), 고대역-저대역(HL), 고대역-고대역(HH)의 4가지 결과를 얻는다. 그림 2는 2차원 DWT를 수행하는 방법을 보여준다[13].

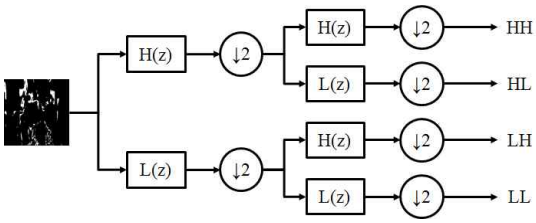

그림 2. 2차원 DWT 방법
Fig. 2. 2-Dimension DWT Method

1.4. 분류 알고리즘

화재 후보 프레임을 분류하기 위한 알고리즘으로 SVM을 사용한다. SVM은 적은 트레이닝 데이터로 높은 분류 성능을 보이며, 경험에 기반한 요소가 필요하지 않아 분류 속도가 빠르고, 단순한 연산만으로도 분류할 수 있는 장점이 있다. SVM 분류기는 다음과 같다[3].

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

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 웨이블릿 에너지 특징들을 분류하기 위한 서포트 벡터는 선형적으로 분리되지 않는다. 따라서 이러한 벡터들을 분리할 수 있는 최적 초평면을 찾기 위해 수식 (6)의 radial basis function (RBF) 커널을 사용한다.

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

여기서 x와 y는 입력 특징 벡터이고, σ는 분류의 정확도에 영향을 미치는 기저 함수의 폭을 결정하는 변수이다. 본 논문에서는 σ값으로 가장 높은 분류 성능을 보여주는 0.1을 실험을 통해 얻었다.

2. GPGPU 아키텍처

그림 3은 본 논문에서 사용한 7개의 스트리밍 멀티프로세서(streaming multiprocessor, SM)를 가지는 GTX 560의 아키텍처를 보여준다[17]. SM은 명령어를 인출(fetch)하여 각 SM에 포함된 스트리밍 프로세서(streaming processor, SP)에 전송하고, 각 SP들은 SIMD방식으로 동작한다. 하나의 SM은 다음과 같은 특징을 갖는다.

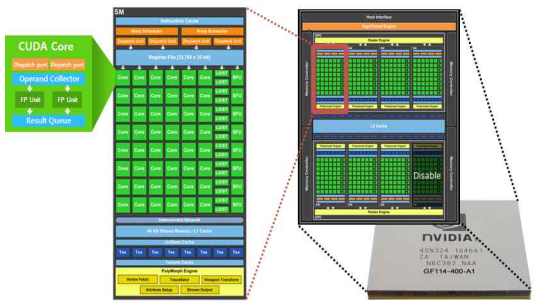

그림 3. Geforce GTX 560 아키텍처
Fig. 3. Geforce GTX 560 architecture

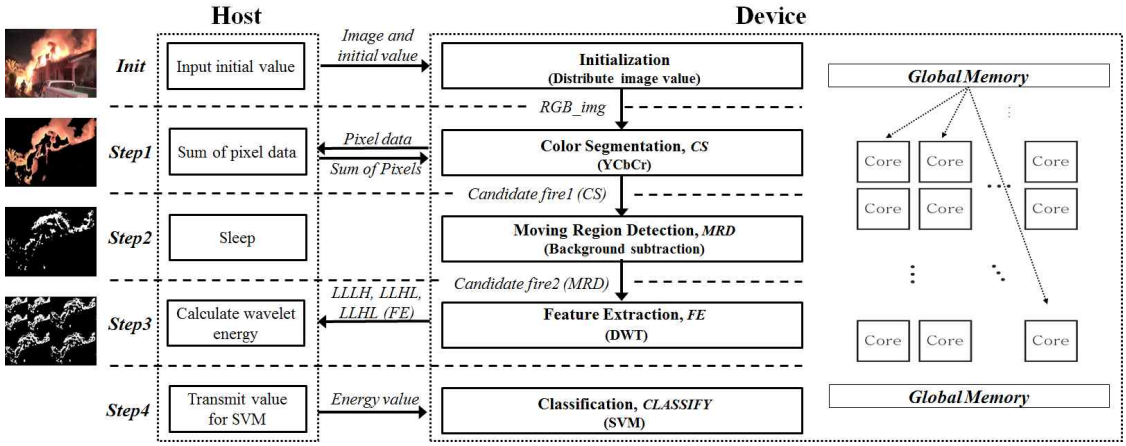

그림 4 화재 감지 알고리즘의 병렬 구현
Fig. 4 Parallel implementation of a fire detection algorithm

- FPU (floating point unit)와 ALU (arithmetic logic unit)로 구성된 48개의 CUDA core
- 메모리로부터 읽기/쓰기를 실행하는 16개의 LD/ST (lode/store unit)
- 초월함수를 처리하는 SFU (special function unit)
- 워프 (warp)라 불리는 병렬 쓰레드 그룹을 관리하는 warp scheduler
- 48KB의 공유 메모리와 L1 캐시 (cache)

NVIDIA사에서는 GPU를 범용으로 사용할 수 있게 하는 프로그래밍 프레임워크인 CUDA를 제공한다. CUDA에서 CPU는 호스트(host), GPU를 디바이스(device)라고 하는데 이들 사이에서 데이터를 주고받기 위해 전역 메모리를 사용한다. 디바이스에서 수행될 함수는 커널(kernel) 함수라고 하는데 커널 함수에서 사용될 데이터는 전역 메모리에 접근 가능한 상태로 복사되어 있어야 한다. 또한 디바이스에서 수행한 결과를 호스트에서 처리하기 위해서는 메인 메모리로 결과를 전송해야 한다. 한편 CUDA는 병렬 수행을 쓰레드와 블록 단위로 연산을 수행한다. 하드웨어에서 쓰레드는 CUDA core에, 블록은 SM에 매핑되어 연산을 수행한다. 각 쓰레드와 블록은 그래픽카드의 여러 메모리를 사용할 수 있다. 메모리의 종류에는 크게 레지스터, 공유 메모리, 전역 메모리로 나눌 수 있다. 레지스터와 공유 메모리는 GPU 코어 내부에 있는 메모리이며, 수 사이클 안에 데이터를 읽고 쓸 수 있다. 전역 메모리는 GPU 코어 외부에 있는 메모리이다. 비록 GPU와 전역 메모리 사이에 캐시가 존재하여 메모리 성능을 향상시킬 수 있지만 캐시 미스가 발생하는 경우에는 수백 사이클이 걸린다. 느린 전역 메모리를 사용하지 않고 레지스터와 공유 메모리만을 사용하여 프로그램을 수행한다면 매우 빠르게 연산을 수행할 수 있지만 다음과 같은 제한 사항이 있어 쉽지 않다. 1) 레지스터와 공유 메모리는 쓰레드 간에는 동기화가 가능하지만 블록 간 동기화가 이루어지지 않는다. 2) 또한 그 크기가 표 1에서와 같이 48KB로 한정되어있다. 3) 전역 메모리는 비록 접근속도가 느리지만, 블록간의 동기화가 가능하고, 크기가 수 GB로 매우 크다. 따라서 이들 메모리의 특성을 고려해서 알고리즘을 구현해야 한다.

III. 화재 감지 알고리즘의 병렬 구현

GPGPU를 이용한 영상처리는 영상의 지역성과 규칙성을 이용하여 영상처리 알고리즘의 병렬성을 최대한 이용하여 구현해야 최대의 효과를 얻을 수 있다. 본 논문의 4단계 화재감지 알고리즘에 대한 병렬 구현의 경우, 색상 분할과 움직임 영역 감지 단계에서는 다른 위치의 픽셀에 연관성 없이 독립적인 연산이 가능하기 때문에 최대한의 병렬성을 찾을 수 있다. 반면에 화재 특징 추출 단계 및 화재 분류 단계는 주변의 인접한 픽셀과의 연관성이 중요하기 때문에 효율적인 접근 방법이 필요하다. 그림 4는 4단계 화재감지 알고리즘의 단계별 호스트와 디바이스의 데이터 이동을 보여준다. 먼저 초기화 단계로는 입력받은 영상 정보 및 초기화 변수들을 전역 메모리에 저장한 후 GPU의 쓰레드를 이용해 각 코어에 전달한다. 본 논문에서는 하나의 영상을 그림 5와 같이 가로 175 세로 132의 블록으로 나눈 후, 블록에 할당된 픽셀 데이터를 다

시 단일 픽셀 단위로 8x8 총 64개로 나누어 각 쓰레드에 하나씩 할당된다. 할당된 쓰레드들은 (idx, idy)와 같이 좌표를 갖게 되며 픽셀 데이터를 코어로 불러와 동시에 수행함으로써 병렬화가 가능하다.

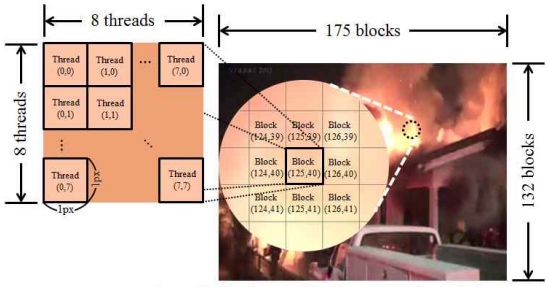

그림 5. 각 쓰레드의 이미지 처리 영역
Fig. 5. Image processing area for each thread

■ **Step 1:** 색상 분할 단계에서는 각 프레임의 YCbCr 색 공간의 채널별 평균이 필요하다. 그러나 평균을 구하는 과정에서, 픽셀들의 합을 반복문을 통해 계산하게 되면 GPU에서의 연산이 CPU에 비해 성능이 저하되게 된다. 따라서 이 부분은 그림 4와 같이 영상의 픽셀 데이터들을 CPU로 전송한 후, 그림 6와 같이 CPU에서 계산한다. 계산된 결과는 GPU로 전송하며 그림 7과 같이 RGB 값들을 YCbCr로 변환한 후, 주어진 조건을 만족하면 RGB 값을, 그렇지 않으면 '0'값을 1차 화재 후보 값으로 저장한다.

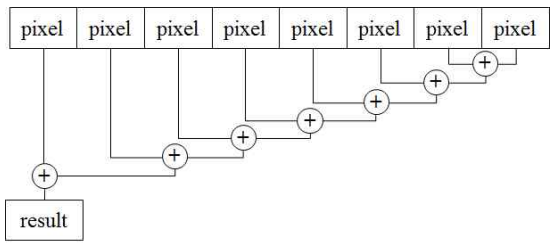

그림 6. 픽셀 합의 계산 방법
Fig. 6. Calculation method of pixel summation

```
//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 7. 색상 분할의 커널 함수
Fig. 7. Kernel function of color segmentation

■ **Step 2:** 움직임 추정 단계에서는 그림 4와 같이 다른 위치의 픽셀에 연관성 없이 독립적인 연산이 가능하기 때문에 GPU에서만 연산을 수행한다. 각 쓰레드는 각 픽셀들은 담당하여 처리하며, 그림 8과 같은 방법으로 연산을 수행한다. 먼저 1단계의 색상 분할 단계에서 추출한 1차 화재 후보 영역을 회색 영상으로 변환한 후, 만약 현재 영상인 $Gray_img[idx,idy]$와 배경 영상인 $Gray_bg_img[idx,idy]$와의 차가 기준치(threshold) 이상의 값을 갖는다면 원래 영상 값을, 그렇지 않으면 '0'의 값을 저장한다. 기준치 값의 경우 모의실험을 통해 '3'으로 정하였다. 또한 카메라의 경우 주변 환경에 의해 흔들림 현상이 발생할 수 있으므로 위와 같은 배경 감산 방법만으로는 완전한 결과를 출력할 수 없다. 따라서 본 논문에서는 위와 같은 단점을 보완하기 위해 만약 현재 영상이 배경 영상 보다 크면 배경영상에 '1'을 더하고, 그렇지 않으면 배경영상에 '1'을 감산하여 배경 영상을 지속적으로 보정한다.

```
//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 8. 움직임 영역 감지 커널 함수
Fig. 8. A kernel function of moving region detection

■ **Step 3:** DWT를 이용한 특징 추출의 모의실험 결과, 1단계 DWT에 비해 2단계 DWT에서 더욱 더 뚜렷한 화재의 특징을 추출하기 때문에 본 논문에서는 2단계 DWT의

movie1	movie2	movie3	movie4	movie5

그림 11 실험에 사용된 영상
Fig. 11 Movies used in the experiment

특징을 추출한다. 2단계 DWT 특징 추출 방법은 그림 2와 같이 1단계 DWT를 바탕으로 동일한 방법의 필터링을 통해 특징을 추출할 수 있다. 2단계 DWT의 병렬 수행에서 고대역 통과 필터(high-pass filter)와 저대역 통과 필터(low-pass filter)를 적용하기 위한 회선(convolution)연산은 다른 픽셀에 독립적으로 수행할 수 있기 때문에 쓰레드 간의 충돌이 발생하지 않는다. 하지만 필터를 적용하는 과정에서 행 방향 연산이 진행되는 동안 열 방향 연산이 수행되면 잘못된 결과 값을 얻을 수 있으므로, 행 방향 연산이 완료된 후 열 방향 연산이 수행되도록 동기화해야 한다. 따라서 그림 9와 같이 Kernel 1은 열 방향 필터 처리를, Kernel 2는 열 방향 다운 샘플링을, Kernel 3은 행 방향 필터 처리를, Kernel 4는 행 방향 필터 처리를 하는 커널로 구성하여 행 방향 연산과 열 방향 연산이 서로 간섭되지 않게 한다. 마지막으로 2단계 DWT의 결과 중 LLLH, LLHL, LLHH를 CPU로 전송하여 웨이블릿 에너지의 평균값을 구해 다음 단계인 화재 분류 알고리즘의 입력으로 사용한다.

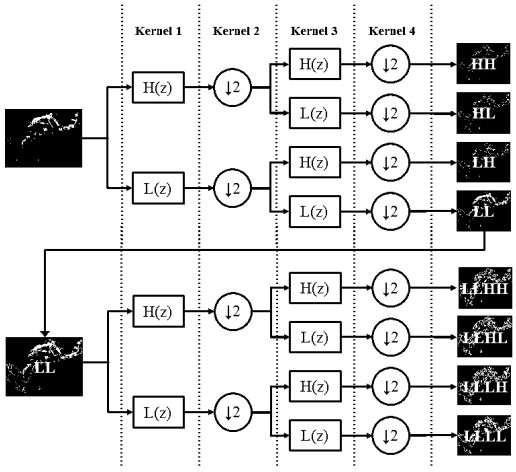

그림 9 특징 추출의 커널 함수
Fig. 9 A kernel function of feature extraction

■ **Step4:** 마지막 단계인 화재 분류 단계에서는 SVM을 이용하여 화재 유/무를 판단한다. SVM의 경우 화재 영상과 비 화재영상을 통한 학습이 필요하다. 따라서 본 논문에서는 각각 200개의 화재 영상과 비 화재 영상을 통해 총 150개의 서포트 벡터 데이터를 추출하였다. 그림 10과 같이 SVM의 분류 연산은 각각의 서포트 벡터에 대해 서로 영향을 주지 않고 독립적으로 연산이 가능하다. 따라서 색상 분할 및 움직임 영역 감지 단계와 동일하게 각 서포트 벡터에 대해 독립적인 쓰레드를 사용하여 병렬 구현하였다.

■

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

$$sign\left(\sum_{i=1}^{l} w_i \cdot k(x, x_i) + b\right)$$

그림 10 분류 알고리즘의 커널 함수
Fig. 10 A kernel function of the classification algorithm

IV. 실험 환경 및 결과

1. 실험 환경

표 1. GPGPU 프로세서 사양
Table 1. Specification of the GPU used in this study

항목	값
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	65535×65535×65535
Shared memory/SM	48KB
Warp size	32

본 논문에서는 3.40Ghz Intel i5-3570K CPU와 NVIDIA Geforce GTX560 GPU를 이용하여 실험하였으며, 표 1은 본 논문에서 사용한 GPGPU 프로세서의 사양을 보여준다. 또한 제안한 화재 감지 알고리즘을 검증하기 위해 그림 11과 같이 SVGA+(1400x1050) 해상도의 화재 및 비화재 비디오 영상 5개를 이용하였다.

2. 실행시간

그림 12는 5가지 비디오 영상에 대한 CPU와 GPU의 실행시간을 비교한 결과를 보여 주며, 여기서 실행시간은 각 비디오 전체 프레임에 대한 평균 실행시간을 나타낸다. 성능 비교 결과, 5가지의 비디오 영상에 대해 각 CPU의 평균 실행시간에 차이가 있음을 보이는데, 이는 화재의 분포도에 따라 알고리즘이 처리해야할 작업의 양이 달라지기 때문이다. 비디오 영상 3번의 경우 화재 후보 영역으로 판단되는 부분이 적어 그림 7과 같이 색상 분할 단계에서 색 공간을 변경하는 추가 연산이 없기 때문에 실행시간에서 가장 빠른 것을 알 수 있다. 반면에 비디오 영상 1과 5의 경우, 화재 후보로 판단되는 영역이 많아 위와 같은 추가 작업이 많아져 실행시간에서 가장 느린 것을 알 수 있다. GPGPU를 기준으로 성능을 분석한 결과, 그래프의 모양이 CPU에 비해 일정하게 유지되는 것을 알 수 있다. 이는 CPU의 순차적인 연산과 다르게, 픽셀마다 추가적인 연산이 수행되더라도 GPGPU의 모든 코어들을 이용하여 병렬적으로 처리하기 때문이다. 따라서 5가지의 서로 다른 영상들에 대해 비슷한 실행시간이 요구된다.

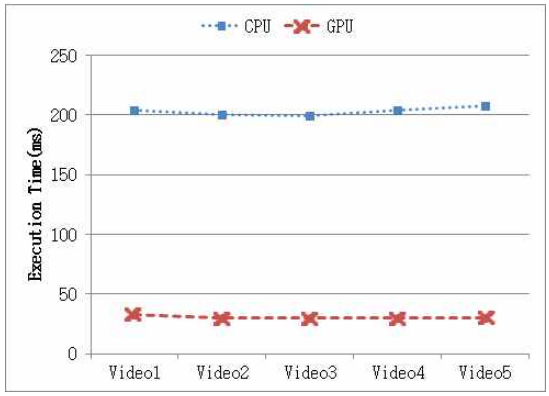

그림 12 CPU와 GPGPU를 이용한 비디오별 실행시간
Fig. 12 Execution times of CPU and GPGPU for each video

결과적으로, GPGPU 상에서 최소 200 프레임으로 구성된 5개의 비디오 영상에서 평균 실행 시간은 한 프레임 당 최대 30.53*ms*으로써 CPU상에서의 201.50*ms*보다 약 6.6배 향상되었으며, 실시간 영상처리에서 요구하는 초당 30 프레임의 처리(33.3*ms*) 속도를 모두 만족하였다.

3. 커널별 실행시간 비교

표 2는 각 커널별 실행시간을 보여주며, 그림 13은 CPU 대비 GPGPU의 각 커널별 실행시간 향상 비율을 보여준다. 여기서 MRD는 움직임 추정 단계, CS는 색상 분할 단계, FE는 특징 추출 단계, CLASSIFY는 분류 단계를 의미한다. MRD와 FE의 경우, CPU에 비해 GPGPU에서 더 높은 성능 향상을 보이는데 이는 해당 커널이 높은 병렬성을 가지고 있기 때문이다. FE의 경우에는 회선 연산을 위해 같은 작업을 반복 수행하며, MRD의 경우에는 현재 영상에 대한 픽셀 데이터들과 이전 영상에 대한 픽셀 데이터의 단순 비교를 반복하기 때문에 높은 병렬성을 갖는다. CS에서는 성능 향상폭이 앞의 두 커널과 비교하여 3배 정도 낮다. 이는 CPU에서 블록 내의 합과 각 픽셀의 합을 구하는 과정이 포함되어 비교적 낮은 병렬성을 가질 뿐만 아니라 CPU와 GPU간의 데이터 전송시간이 요구되기 때문이다. 반면에 CLASSIFY에서는 오히려 GPGPU의 실행시간이 CPU의 실행시간 보다 높은 것을 알 수 있다. 이는 CLASSIFY는 순차적인 연산이 요구되는 커널이며 GPGPU에서 수행되기 위해서는 호스트인 CPU에서 디바이스인 GPU의 공유 메모리로 데이터를 가져와야 하는 시간이 추가로 소요되기 때문이다. 표 3에서 보듯이 실제로 CLASSIFY 커널에서 수행되는 실행 시간은 적지만, 메모리 초기화 시간 및 데이터 복사 시간이 CPU에서 수행된 시간보다 더 많이 소요되는 것을 알 수 있다.

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

Execution time(ms)	MRD	CS	FE	CLASSIFY
CPU	27.2	58.7	117.3	0.008
GPGPU	0.6	19.5	10.4	0.032

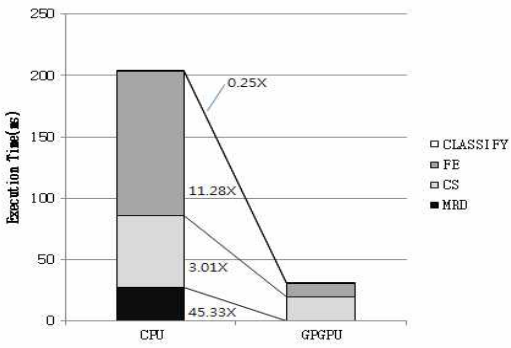

그림 13 커널별 CPU 대비 GPGPU의 성능 향상
Fig. 13 Improved performance of GPGPU over CPU for each Kernel

표 3. CLASSIFY단계에서의 세부항목 별 실행시간
Table 3. Execution time for each part of CLASSIFY

항목	실행시간(ms)
Memory initialization	0.007
Memory copy (HtoD)	0.014
Memory copy (DtoH)	0.006
Kernel execution time	0.005

V. 결 론

본 논문에서는 비디오 기반 실시간 화재감지 알고리즘을 위해 GPGPU를 사용하여 병렬 구현하였다. 또한 GPGPU 상에서 5개의 화재 또는 비 화재 비디오 영상에 따른 화재감지 알고리즘의 실행시간을 분석하였으며, GPGPU기반 화재감지 알고리즘의 효용성을 확인하기 위해 CPU와의 성능을 비교하였다. 5개의 HD급 비디오 영상을 이용하여 모의 실험한 결과, GPGPU는 CPU보다 6.6배의 성능을 향상을 보였으며, 각 프레임 당 30.53ms의 실행시간을 보였다. 이는 GPGPU기반 화재감지 알고리즘이 HD급의 높은 해상도인 SXGA 영상에서도 초당 30프레임의 실시간 처리가 가능함을 보였다.

참고문헌

〔1〕 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.

〔2〕 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.

〔3〕 B. C. Ko, K. H. Cheong, and J. Y. Nam, "Early Fire Detection Algorithm Based on Irregular Patterns of Flames and Hierarchical Bayesian Networks," Fire Safety J, vol. 45, no. 4, pp. 262-270, 2010.

〔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," Lect. Notes Comput. Sci., vol. 4681, pp. 1209-1220, 2007.

〔8〕 X. Qi and J. Ebert, "A Computer Vision Based Method for Fire Detection in Color Videos," Int. 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 Image Rep, 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 Trans. Instrum.

Meas., 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〕 G. Marbach, M. Loepfe, and T. Brupbacher, "An image processing technique for fire detection in video Images," Fire Safety Journal, vol. 41, no. 4, pp. 285-289, 2006.
〔13〕 O. Gunay, K. Tasdemir, B. U. Toreyin, and A. Enis, "Fire Detection in Video Using LMS Based Active Learning," Fire Technol, vol. 46, no. 3, pp. 551-577, 2010.
〔14〕 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.
〔15〕 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 Trans. Parallel Distrib. Syst., vol.22, no.1, pp.91-104, 2011.
〔16〕 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 Trans. Parallel Distrib. Syst., vol. 22, no. 1, pp. 91-104, 2011.
〔17〕 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
현 재: 울산대학교 전기공학부 교수
관심분야: 임베디드 SoC, 병렬처리.
Email : jmkim07@ulsan.ac.kr