OpenCL을 이용한 JPEG2000 4K 초고화질 영상처리의 병렬고속화 구현

박대승*, 김정길**, 종신회원
남서울대학교
eosonox@gmail.com* cgkim@nsu.ac.kr**

A Parallel Implementation of JPEG2000 4K Ultra High Definition Image using OpenCL

Daeseung Park*, Cheong Ghil Kim**, *Lifetime Members*
Namseoul University

요 약

멀티미디어 기술의 급속한 발전과 사용자의 대형 화면에 대한 선호도가 높아지는 가운데 새로운 영상 압축 기술인 HEVC(High Efficiency Video Coding) 고화질 영상 압축 표준을 탄생시켰으며, 그 결과 기존의 HD급 영상보다 4배 이상, 16배까지 선명한 초고화질 UHD(Ultra High Definition) 영상 서비스가 새롭게 주목받고 있다. 또한 JPEG 2000 압축도 기존 처리되던 픽셀 이미지를 넘어 초고화질 해상도 이미지(4K : 3,840 x 2,160 또는 8K : 7680 x 4320)를 처리 지원을 하고 있다. 따라서 초고화질 이미지의 획득 및 저장을 위해서는 고속의 처리 기술이 필요하다. 이에 본 논문은 초고화질 해상도 이미지의 고속 처리를 위한 병렬처리 기술에 대한 연구를 위하여, JPEG 2000의 처리 과정을 살펴보고 전처리 단계인 색공간 변환 알고리즘 적용을 위하여 GPU환경에서 병렬 컴퓨팅을 통해 처리속도를 향상시키는 방법을 제안한다. 병렬화한 알고리즘의 구현은 OpenCL(Open Computing Language)을 이용하였다. 실험 결과 사용자 정의 쓰레드 기반 고속 처리와 비교하여 초고화질 해상도 이미지(UHD 4K : 3,840 x 2,160)를 기준으로 최대 5배의 성능 향상의 결과를 보여주었다.

Key Words : OpenCL, JPEG 2000, Thread, UHD, Color Conversioin

ABSTRACT

With the help of fast growing multimedia technology and high preference for users of large screens, the newest video coding standard, HEVC (High Efficiency Video Coding) high-quality video compression), has been introduced. Therefore, the high definition image services which are four times more clear than conventional HD video, are getting popular. JPEG 2000 also has stated to support 4K and 8K UHD. As a result, it requires fast processing technology to read and write UHD images. This paper introduces a study on fast parallel processing technology for UHD images. For this purpose, first, JPEG 2000 is reviewed and a GPU based parallel implementation is proposed for a preprocessing of color conversion stage. The parallelled algorithm is implemented with OpenCL (Open Computing Language). The simulation results show that the proposed method shows 5 times performance improvements on processing speed for 4K UHD over the method using threads.

I. 서 론

최근 HDTV(Full-HD급) 영상 콘텐츠가 대중화되고, 시청자들이 고화질, 고해상도의 영상에 익숙해지면서 HDTV 이후의 차세대 영상 서비스에 대한 관심이 증가하고 있는 가운데, UHD 콘텐츠는 Post-HD 미디어 시대의 주류로 부상하고 있으며, 점차로 콘텐츠 제작 환경도 UHD급으로 진화하고 있다. 그 결과 그림 1의 영상 데이터양 비교에서 확인하

※ 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 정보통신·방송 연구개발사업의 일환으로 수행하였음. [2014(I5501-14-1007), 3D 스마트미디어/증강현실 기술 한중일러 공조 국제표준화]
**남서울대학교 컴퓨터학과 (cgkim@nsu.ac.kr)
접수일자 : 2015년 1월 4일, 수정완료일자 : 2015년 2월 11일, 최종게재확정일자 : 2015년 2월 23일

듯이 현재의 HDTV 서비스에서 콘텐츠 획득과 제작 분야에서 주로 사용되는 영상 데이터양은 약 2.5Gbps 이상 필요하지만, 초고품질 4K UHDTV 서비스에 대응하기 위해서는 기존 Full-HD 데이터에 비해 최소 4배 정도인 10Gbps 데이터양 이상을 처리해야 하고, 8K UHDTV 서비스에 대응하기 위해서는 기존 Full-HD 데이터에 비해 약 29배 또는 58배 정도의 데이터 처리 기술이 필요하다[1,2,3].

이러한 상황의 극복은 컴퓨터의 성능을 향상시키려는 연구와 병행되어 꾸준히 계속되고 있으며, CPU의 처리 속도를 높이거나, CPU의 개수를 늘려 병렬로 작업을 처리하여 컴퓨터의 성능을 향상시키려는 방법들이 대표적인 예이다. 그 결과 최근 멀티코어 프로세서들이 범용 PC 뿐만 아니라 임베디드 시스템에서도 탑재될 만큼 그 사용이 보편화되고 있는 상황에서, 많은 멀티미디어 응용 프로그램이 이들을 활용하여 병렬화 되고 있다[4]. 이러한 하드웨어적인 향상과 더불어 병렬처리 소프트웨어도 이용되고 있다. 가장 일반적인 병렬처리 방법은 여러 개의 쓰레드(thread)를 생성하여, 각자의 프로세스에 역할을 할당하는 방법이다[3,5].

최근에는 GPU가 고성능화 되고 연산 코어 개수가 점차 많아짐에 따라, 해당 코어를 일반적인 병렬 처리 프로그램으로 적용하고자 GPGPU(General Purpose GPU) 라는 처리 방법이 제안되었다. GPGPU는 GPU 연산코어를 그래픽 처리 프로그램이 아닌, 일반 응용 프로그램에서 사용 할 수 있도록 라이브러리를 제공하는데, 응용 프로그램에서 적게는 수십 개에서 많게는 수백 개의 연산 코어를 활용할 수 있게 되어, 고성능의 병렬처리가 가능해 진다[6, 7].

	해상도(WxH)	프레임율(fps)	컬러포맷(YUV)	비트심도(bits)	데이터량(Mbps)
SD	720x480	30	4:2:0	8	124Mbps
HD	1920x1080	30	4:2:2	8	996Mbps
	1920x1080	30	4:2:2	10	1.25Gbps
	1920x1080	60	4:2:2	10	2.5Gbps
4K UHD	3480x2160	30	4:2:2	8	4Gbps
	3480x2160	60	4:2:2	10	10Gbps
	3480x2160	60	4:4:4	12	18Gbps
8K UHD	7680x4320	60	4:2:2	10	40Gbps
	7680x4320	60	4:4:4	10	60Gbps
	7680x4320	60	4:4:4	12	72Gbps
	7680x4320	120	4:4:4	12	144Gbps

그림 2. UHD콘텐츠용 영상 데이터

본 논문은 초고화질 해상도 이미지의 고속 처리를 위한 병렬처리 기술에 대한 연구를 위하여, JPEG 2000의 처리 과정을 살펴보고 전처리 단계인 색공간 변환 알고리즘 적용을 위하여 GPU환경에서 병렬 컴퓨팅을 통해 처리속도를 향상시키는 방법을 제안한다. 병렬화한 알고리즘의 구현은 OpenCL(Open Computing Language)을 이용하였다. 본 논문의 구성은 2장에서는 JPEG 2000 및 OpenCL에 대해 소개하고, 3장에서는 병렬처리 적용을 기술한다. 4장에서는 제안하는 실험 환경 및 결과를 기술하고, 마지막으로 5장에서 결론을 맺는다.

II. 관련 연구

1992년 JPEG(Joint Photographic Expert Group)이 국제표준으로 채택된 이후 다양한 멀티 미디어응용분야에 사용되고 있다. ISO/IEC 산하의 JTC1/SC29/WG1 그룹에서는 2000년 JPEG 2000이라 하는 새로운 표준을 발표하였다[4]. JPEG 2000 인코더/디코더의 블록 그림은 그림 2와 같다. 이산 웨이블렛 변환을 소스 영상 데이터에 적용되고 변환된 계수는 양자화를 거쳐 코드열을 생성하기 전에 엔트로피 부호화 과정을 거친다. 전처리 단계는 색변환 단계로서 RFB에서 YUV 변환을 실행한다[6],

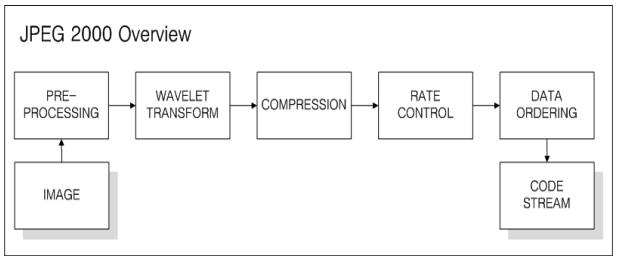

그림 3. JPEG 2000 블럭도

OpenCL은 Apple이 제안하고 AMD, Intel, NVIDIA등 그래픽 등 프로세서 제조사가 개발한 개방형 범용 병렬 컴퓨팅 프레임워크이다. OpenCL은 CPU, GPU등의 프로세서로 이루어진 이종 플랫폼에서 실행되는 프로그램을 작성할 수 있게 해 주며, 다양한 GPU에서 동작할 뿐만 아니라 GPU가 없는 환경에서는 CPU에서도 동작이 가능해 이식성이 뛰어나다[6].

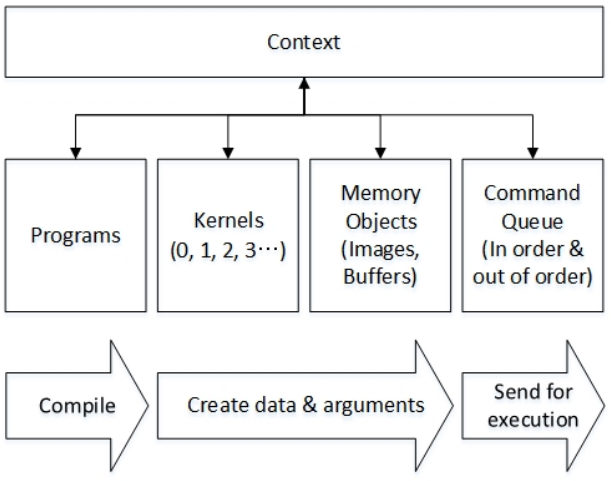

그림 4. OpenCL 컴파일 및 실행 모델

그림 4는 작성된 OpenCL 코드의 동작 순서를 나타내고 있다. (1) 작성된 OpenCL 코드를 컴파일해서 실행파일을 OpenCL 디바이스에 전달하고, (2) OpenCL에서는 1차원 배열만 받아들일 수 있기 때문에, 작업을 하기 위해 필요한 데이터를 직렬화 하여 매개변수로 전달하며, 실행 대기상태를

유지한다. (3) 최종적으로 OpenCL에 작업 시작을 요청하게 되고 명령을 수신한 Device는 작업을 완료한 후 결과를 OpenCL 버퍼로 반환하게 된다.

OpenCL은 API를 기반으로 동작하는데, 각 동작 과정에서 사용되는 API는 다음과 같다. OpenCL 작업이 시작되는 경우 초기화 작업에 clGetPlatformIDs, clGetDeviceIDs, clCreateContext, clCreateCommandQueue API가 사용되고, 초기화가 완료되면 GPU에서 구동될 OpenCL 커널을 준비하게 되고, clCreateProgramWithSource, clBuildProgram, clCreateKernel API가 사용된다. 매개변수를 전달하는 과정에서는 clEnqueueWriteBuffer API가 사용되고, 작업 실행을 요청할 때에는 clSetKernelArg, clEnqueueNDRangeKernel API가 사용된다. 작업이 완료되면 결과를 반환받을 때 clEnqueueReadBuffer API가 사용되고 마지막으로 자원을 해제하는 경우 clRelease API가 사용된다.

III. 병렬처리

RGB는 일반적으로 사용하는 색상 구조로서 3색상의 조합으로 밝기는 없고 모두 색상으로 구성되 된다, YUV는 Video Data 형식으로 사용하며 TV에서 컬러페이스로 전송하는 컬러 형식이다. YUV는 색과 빛이 별도로 구성되어 있는 형식으로 RGB방식에 비하여 작은 대역폭으로 전송이 가능한 장점이 있다. RGB 색상구조를 YUV 색상구조로 변환하는 방법은 수식 (1)을 이용하고 다시 환원하는 방법은 수식 (2)를 이용한다.

$$
\begin{aligned}
Y' &= W_R R + W_G G + W_B B \\
&= 0.299R + 0.587G + 0.114B \\
U &= U_{Max}\frac{B - Y'}{1 - W_B} \approx 0.492(B - Y') \\
V &= V_{Max}\frac{R - Y'}{1 - W_R} \approx 0.877(R - Y')
\end{aligned}
\tag{1}
$$

$$
\begin{aligned}
R &= Y' + V\frac{1 - W_R}{V_{Max}} = Y' + \frac{V}{0.877} = Y' + 1.14V \\
G &= Y' - U\frac{W_B(1 - W_B)}{U_{Max}W_G} - V\frac{W_R(1 - W_R)}{V_{Max}W_G} \\
&= Y' - \frac{0.232U}{0.587} - \frac{0.341V}{0.587} \\
&= Y' - 0.395U - 0.581V \\
B &= Y' + U\frac{1 - W_B}{U_{Max}} = Y' + \frac{U}{0.492} = Y' + 2.033U
\end{aligned}
\tag{2}
$$

병렬처리에서는 쓰레드가 가장 대표적으로 사용되는데, 쓰레드는 쓰기 쉽고 편하게 API를 제공하기 때문이다. 쓰레드란 그림 5에서 보듯이 한 개의 프로그램 또는 프로세스 내에서 실행되는 흐름의 단위로서, 일반적으로 한 프로그램은 하나의 쓰레드를 가지고 있으며, 이 쓰레드를 Main Thread (주 쓰레드)라고 한다. 이 Main Thread에서 Main Routine이 호출된다. 프로그램 환경에 따라 둘 이상의 스레드를 동시에 실행 가능한 멀티스레드(multithread) 방식도 가능한데, 멀티스레드는 프로세스 내의 메모리를 공유하며 따라서 고속의 프로세스 간의 전환이 가능하고, 이것으로 인해 흐름이 동시에 진행될 수 있다. 특히, 멀티 코아 환경에서의 멀티스레드는 각각의 CPU가 스레드 하나씩을 담당하는 방법으로 속도를 높일 수 있다. 이러한 시스템에서는 여러 스레드가 실제 시간상으로 동시에 수행될 수 있기 때문이다. 사용자 스레드는 커널 영역의 상위에서 지원되며 일반적으로 사용자 레벨의 라이브러리를 통해 구현되며, 라이브러리는 스레드의 생성 및 스케줄링 등에 관한 관리 기능을 제공한다.

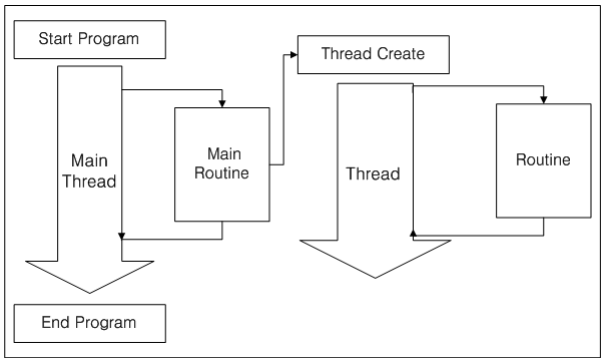

그림 5. 쓰레드 병렬처리 블럭도

그림 7는 DibColorSplitYUV() 함수로서 읽은 bmp 이미지를 RGB 색공간에서 YUV 색공간으로 변환하는 함수이다. 그림의 1번 박스는 읽어들인 bmp 이미지의 크기를 변수 w,h에 저장하고, 그레이스케일하여 색공간 정보를 각가의 포인터 변수에 저장한다. 마지막으로 구조체 struct argment에 각종 정보를 저장 한다. 2번 박스는 쓰레드 생성 및 실행 그리고 3번 박스는 쓰레드 대기 및 종료를 수행하는 블록이다.

그림 6은 여러 개의 쓰레드가 겹치지 않게 작업하기 위하여, 각 쓰레드별로 작업할 범위를 지정하는 구조체이다. 2차원 배열로 이루어진 이미지에서 시작 좌표 X,Y값과 종료 좌표 X,Y값을 지정하며, 해당 작업 영역에 대한 연산이 모두 완료되었는지를 확인하는 플래그함수를 포함한다.

```
//Thread Arg
struct threadposition{
    int threadnum;
    int startx;
    int starty;
    int endx;
    int endy;
};
```

그림 6. 쓰레드에 작업할 범위를 지정하는 구조체

그림 7은 여러 개의 쓰레드가 각각 어느 부분을 맡아야 하는지 계산하는 부분으로, 픽셀수/쓰레드개수로 계산하고, 계산된 영역을 매개변수로 Y, U, V 변환 함수를 각각 호출해 픽셀변환을 수행하는 그림 8 쓰레드 함수를 호출 한다.

```
//Data Convert [THREAD]
for(int i = 0; i < THREAD_VALUE; i++)
{
    struct threadposition *tp = (struct threadposition*)malloc( sizeof(struct threadposition) );

    tp->threadnum = i;
    tp->startx = 0;
    tp->starty = (SCREEN_HEIGHT / THREAD_VALUE) * i;
    tp->endx = SCREEN_WIDTH;
    tp->endy = (SCREEN_HEIGHT / THREAD_VALUE) * (i+1);

    HANDLE hThread = CreateThread(NULL, 0, threadwork, (LPVOID)tp, 0, NULL);
}
```

그림 7. 쓰레드가 작업할 범위를 계산하는 코드

```
//Windows Thread
DWORD WINAPI threadwork(LPVOID arg)
{
    struct threadposition *tp = (struct threadposition *)arg;

    for(int i = tp->starty; i < tp->endy; i++){
        for(int j = tp->startx; j < tp->endx; j++){
            d_y[i][j] = int2y(i_r[i][j], i_g[i][j], i_b[i][j]);
            d_u[i][j] = int2u(i_b[i][j], y);
            d_v[i][j] = int2v(i_r[i][j], y);
        }
    }

    threadStatus[tp->threadnum] = true;

    return 0;
}
```

그림 8. RGB를 YUV로 변환하는 쓰레드 처리 코드

그림 9는 그림 10의 OpenCL 코드를 컴파일하고 작업할 데이터를 전달한 뒤 OpenCL 작업 시작을 요청하는 역할을 하는 코드이다.

```
//OpenCL C로 작성된 코드로 프로그램 작성
program_gpu = clCreateProgramWithSource(context_gpu, 1, (const char **)&source_str, (const size_t *)
    &source_size, &ret);

//프로그램을 빌드
ret = clBuildProgram(program_gpu, 1, &device_gpu, NULL, NULL, NULL);

//커널 생성
kernel_gpu = clCreateKernel(program_gpu, "hello", &ret);

//파라미터 작성 및 버퍼 쓰기 실행
ret = clSetKernelArg(kernel_gpu, 0, sizeof(cl_mem), (void *)&gpu_ir);
ret |= clSetKernelArg(kernel_gpu, 1, sizeof(cl_mem), (void *)&gpu_ig);
ret |= clSetKernelArg(kernel_gpu, 2, sizeof(cl_mem), (void *)&gpu_ib);
ret |= clSetKernelArg(kernel_gpu, 3, sizeof(cl_mem), (void *)&gpu_as);

ret = clEnqueueWriteBuffer(command_queue_gpu, gpu_ir, CL_FALSE, 0, sizeof(i_r_v), i_r_v, 0, NULL, NULL);
ret |= clEnqueueWriteBuffer(command_queue_gpu, gpu_ig, CL_FALSE, 0, sizeof(i_g_v), i_g_v, 0, NULL, NULL);
ret |= clEnqueueWriteBuffer(command_queue_gpu, gpu_ib, CL_FALSE, 0, sizeof(i_b_v), i_b_v, 0, NULL, NULL);
ret |= clEnqueueWriteBuffer(command_queue_gpu, gpu_as, CL_FALSE, 0, sizeof(iNumElements), iNumElements, 0,
    NULL, NULL);

//작업수행 요청
ret = clEnqueueTask(command_queue_gpu, kernel_gpu, 0, NULL, NULL);

//작업 결과 반환
ret = clEnqueueReadBuffer(command_queue_gpu, gpu_ir, CL_TRUE, 0, sizeof(i_r_v), i_r_v, 0, NULL, NULL);
ret |= clEnqueueReadBuffer(command_queue_gpu, gpu_ig, CL_TRUE, 0, sizeof(i_g_v), i_g_v, 0, NULL, NULL);
ret |= clEnqueueReadBuffer(command_queue_gpu, gpu_ib, CL_TRUE, 0, sizeof(i_b_v), i_b_v, 0, NULL, NULL);
```

그림 9. OpenCL C로 작성된 코드를 컴파일하고 파라미터를 전달해 작업을 요청하는 코드

```
__kernel void hello(__global double* i_r_v, __global double* i_g_v, __global double* i_b_v,
                    __global int* arraysize)
{
    double y = 0, u = 0, v = 0;

    for(long i = 0; i < arraysize[0] * arraysize[1]; i++){
        y = (0.299*i_r_v[i])+(0.587*i_g_v[i])+(0.114*i_b_v[i]);
        u = (i_b_v[i]-y)*0.492;
        v = (i_r_v[i]-y)*0.877;

        i_r_v[i] = y;
        i_g_v[i] = u;
        i_b_v[i] = v;
    }
}
```

그림 10. OpenCL C언어로 작성된 OpenCL 처리 코드

IV. 실험 및 결과

본 장에서는 실험 환경 및 방법을 소개하고 결과를 기술한다. 표 1은 실험 환경을 요약하고 있으며 실험 이미지는 BMP 파일 포맷을 사용하였다. 실험은 해당 이미지에 대하여 RGB에서 YUV로 변환 연산에 소요되는 시간을 측정하였으며 최종 결과 값은 100회 반복 측정하여 평균값을 구하였다. 연산 속도 측정 구간은 이미지의 모든 픽셀에 대한 컬러 변환 연산이 완료될 때까지의 시간을 측정하였다. 초기화, 이미지로드, 결과출력 과정은 동일한 프로시저를 사용함으로 연산 속도 측정 부분에서 제외하였고, 속도 측정 방법은 YUV 변환 연산 함수를 호출해 반환되는 데까지 걸리는 시간을 마이크로초(microsecond) 단위로 측정 및 하였다. 쓰레드를 사용한 경우 4K의 RGB to YUV 변환의 연산 시간은 각각 평균 약 887,036μs 소요되었으며, OpenCL을 사용한 RGB to YUV 변환의 연산시간은 4K에서 각각 평균 172694 μs 소요되어 5배 이상의 속도 향상을 가져왔다. 그림 11은 컬러 공관 변환의 속도 성능 비교를 보여준다.

표 1. 실험환경

시스템 사양	구성
CPU	Intel Core i3-3240@3.40Ghz
RAM	16.00GB
System	Window 8.1 Pro K(64bit)
Compiler	Microsoft Visual Studio 2008
Image Sizes	UHD 4k 4096x4096
Image Type	bmp

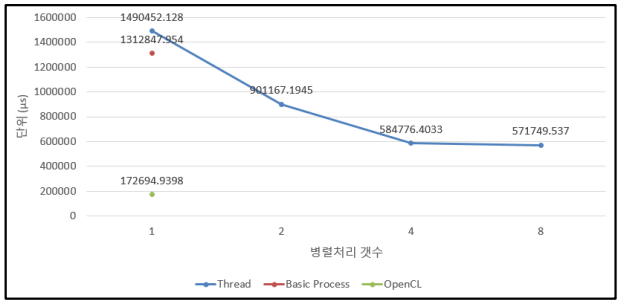

그림 11. 4K UHD의 단일작업, 스레드, OpenCL간 처리속도 비교

V. 결론

본 연구에서는 초고화질 해상도 3,840 x 2,160 사이즈인 UHD 4K 이미지 처리 지원을 위한 고속 JPEG 2000의 처리 과정을 살펴보고 전처리 단계인 색공간 변환 알고리즘 적용을 위하여 GPU환경에서 병렬 컴퓨팅을 통해 처리속도를 향상시키는 방법을 구현하였다. 병렬화한 알고리즘의 구현은 OpenCL을 이용하였고, 실험 결과 쓰레드 기반 처리와 비교하여 UHD 4K를 기준으로 최대 5배의 성능 향상의 결과를 보여주었다. 본 연구는 전체 JPEG 2000 과정으로의 적용과 더불어, TBB 및 SSE등의 다양한 병렬처리 기법을 본 연구의 OpenCL연산과 결합한 하이브리드 형태의 성능 비교를 수행 계획이다.

참 고 문 헌

[1] 김제우, 김동순, 신화선, 최병호, "UHD(Ultra High Definition) 콘텐츠용 실시간 획득, 저장 및 편집 시스템 기술, 방송공학회지 제17권 제4호, pp. 69-80, 2012년 10월.

[2] 배성호, 하광성, 김문철, 조숙희, 최진수, HEVC 기반 초고선명(4K-UHD) 비디오 시청품질 특성 분석, 2012년도 한국방송공학회 하계 학술대회, pp. 446-449, 2012년 7월.
[3] 백봉진, 류준호, 김정길, 이상운, 초고화질 해상도용 4K/8K JPEG 2000 고속 압축 처리를 위한 연구, 2014년 한국방송공학회 추계학술대회, 2014, 11. 07
[4] Cheong Ghil Kim, Do Hyun Lee, JeomGu Kim, "Optimizing Image Processing on Multi-core CPUs with Intel Parallel Programming Technologies," Multimedia Tools and Applications January 2014, Vol. 68, Issue 2, pp. 237-251, Jan. 2014.
[5] Cheong Ghil Kim and Bong Jin Beak, "Fast JPEG Color Space Conversion on Shared Memory", International Conference onInformation Science and Applications (ICISA), pp. 1-3, 2013
[6] 박세진, 마정현, 박찬익, "GPGPU 응용의 고성능 원격수행을 위한 OpenCL 기반 오프로딩 프레임워크", 한국차세대컴퓨팅학회 논문지 Vol.9 No.2, pp. 44-53, 2013년 4월
[7] Cheong Ghil Kim Yong Soo Choi, "A high performance parallel DCT with OpenCL on heterogeneous computing environment Multimedia Tools and Applications, Vol. 64(2), pp. 475-489, May 2013
[8] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG2000 Still Image Compression Standard," IEEE Signal Processing Magazine, Vol. 18, Issue. 5, pp. 36-58, Sep.2001.
[9] 김재준, 홍진우, JPEG2000 정지영상 부호화 기술 개요, 전자통신동향분석 제17권 제4호, pp. 65-74, 2002년 8월.

저자

박 대 승(Dae Seung Park)

・2015년 2월 : 남서울대학교 컴퓨터학과 학사졸업
・2015년 3월 : 남서울대학교 멀티미디어 방송연구센터

<관심분야> : 운영체제, 모바일 플랫폼

김 정 길(Cheong Ghil Kim) **종신회원**

・1987년 8월 : Univ. of Redlands, USA 컴퓨터과학과 학사졸업
・2003년 8월 : 연세대학교 컴퓨터과학과 공학석사 졸업
・2006년 8월 : 연세대학교 컴퓨터과학과 공학박사 졸업
・2006년 ~ 2007년 : 연세대학교 컴퓨터과학과 박사후 연구원
・2007년 ~ 2008년 : 연세대학교 컴퓨터과학과 연구교수
・2008년 ~ 현재 : 남서울대학교 컴퓨터학과교수

<관심분야> : 멀티미디어 임베디드 시스템, 이기종 컴퓨팅, 모바일 AR, 3D Contents