論 文
57-6-26

GPU를 이용한 JPEG2000 병렬 알고리즘

Parallel Processing Algorithm of JPEG2000 Using GPU

李 東 河* · 曺 時 元** · 李 東 旭†
(Dongha Lee · Shiwon Cho · Dong-Wook Lee)

Abstract - Most modern computers or game consoles are well equipped with powerful graphics processing units (GPUs) to accelerate graphics operations. However, since the graphics engines in these GPUs are specially designed for graphics operations, we could not take advantage of their computing power for more general nongraphic operations. In this paper, we studied the GPUs graphics engine in order to accelerate the image processing capability. Specifically, we implemented a JPEC2000 decoding/encoding framework that involves both OpenMP and GPU. Initial experimental results show that significant speed-up can be achieved by utilizing the GPU power.

Key Words : GPU, GPGPU, JPEG2000, OpenMP, Parallel Processing

1. 서 론

과거의 그래픽 처리 장치(Graphics Processing Units: GPU)는 3차원 그래픽과 같은 특수한 용도를 목적으로 설계되어, CPU가 처리하는 범용적인 목적에 사용하기에는 한계가 있었다. 그러나 최근의 GPU는 상당한 메모리 대역폭과 빠른 부동소수점 계산과 같은 산술 연산 능력을 보유하여, 중요한 계산 자원으로 주목받고 있다. 컴퓨터 그래픽 분야를 넘어 GPU를 범용 계산에 활용하여 다양한 분야에 응용하려는 연구 활동들은 GPGPU(General Purpose computation on GPU)라는 분야로 알려지게 된다. 그 결과로 GPU는 렌더링과 시각화처럼 컴퓨터 그래픽 관련 분야뿐만 아니라 미분 방정식의 선형시스템 풀이를 위한 행렬 계산, 시뮬레이션, 영상 처리, 데이터베이스 연산에 이르기 까지 다양한 분야에서 이용되고 있다. 많은 개발자들은 GPU가 CPU에서 독자적으로 수행되던 작업들을 대행할 수 있기를 기대하고 있으며 다양한 분야에서 상당한 성과를 보여주고 있다[1].

최근의 GPU는 정점 셰이더(vertex shader)와 프래그먼트 셰이더(fragment shader)를 지원하여 사용자들이 GPU를 손쉽게 사용할 수 있도록 했으며, 점차 GPU의 활용범위가 넓어지고 있다. 또한 복잡한 소수점 연산에 대해서도 점차 범용 CPU의 성능을 능가하며, 프로그램 기술의 향상으로 시뮬레이션 등 다른 분야에의 활용 가치도 높아지고 있다. 최근에는 높은 사양의 GPU 성능을 요구하는 3D 그래픽스 응용기술이 많이 개발되면서 GPU 성능 향상의 구심점이 되고 있다. PC와 게임콘솔 등에 사용되는 GPU에는 가장 최신의 기술이 동원되어 그래픽 성능을 극대화시키고 있으며 여기에 사용되었던 기술이 휴대용 모바일 기기에도 점차 적용되고 있다[2][3].

컴퓨터의 활용 분야가 다양해지면서, 더욱더 복잡한 문제들을 해결하기 위해 컴퓨터의 성능을 향상시키려는 연구가 꾸준히 계속되고 있다. CPU의 처리 속도를 높이거나, CPU의 개수를 늘려 병렬로 작업을 처리하여 컴퓨터의 성능을 향상시키려는 방법들이 대표적인 예이다. 그러나 하드웨어적인 향상만으로 컴퓨터의 병렬 처리가 불가능하며, 이를 지원하기 위한 병렬처리 소프트웨어도 반드시 필요하다[4]. 기존의 일반적인 프로그램을 개발하는 방법은 주로 단일 코더 CPU를 기준으로 하는 구현 방법이다. 따라서 멀티 코어를 지원하는 CPU의 성능을 충분히 활용하기 위해서는 새로운 프로그램 모델이 필요하다. 가장 일반적인 병렬처리 방법은 여러 개의 프로세스나 쓰레드(thread)를 생성하여, 각각의 프로세스에 역할을 할당하는 방법이다. 하지만, 각각의 프로세서는 독립적으로 실행되지만, 각각의 프로세스에서 계산된 결과를 하나의 데이터로 만들거나 다른 프로세스로 전달하는 경우도 생기기 때문에, 프로세스들의 데이터 교환 방식과 동기화가 반드시 필요하다. 이러한 프로세스들의 데이터 교환, 동기화 방법에 따라 각각 다른 병렬 프로그래밍 방법을 적용해야 한다. 현재 표준화된 병렬 프로그래밍 모델은 메시지 전달 방식(Message Parsing Interface: MPI)과 OpenMP라는 공유 메모리 방식이 있다[5].

본 논문에서는 GPU의 장점을 활용하는 JPEG2000 코덱을 구현하였다. 프로그래밍이 쉽고 일반 프로그램을 병렬 프로그램으로 변환하기도 쉬운 OpenMP를 이용한 병렬 프로그래밍 모델을 적용하여, 멀티 코어 CPU와 GPU의 병렬처리가 가능하도록 했다. 병렬처리를 함으로써 코드 특성상 많은 반복적인 연산이 필요한 JPEG2000 코덱의 처리 속도

* 學生會員 : 東國大 工大 電氣工學科 碩士課程
** 正 會 員 : 東國大 工大 電氣工學科 博士課程
† 교신저자, 正會員 : 東國大 工大 電氣工學科 敎授 · 工博
E-mail : dlee@dongguk.edu
接受日字 : 2008年 1月 31日
最終完了 : 2008年 4月 4日

를 향상하였다.

본 논문의 구성은 다음과 같다. 2장에서는 GPU, JPEG2000, OpenMP에 대하여 간략히 설명하고, GPU를 이용한 JPEG2000의 병렬 처리 알고리즘을 제시한다. 3장에서는 실험 결과를 보이고, 4장에서는 결과를 제시한다.

2. 본 론

2.1. GPU

PC, 워크스테이션, 게임 콘솔 등에 많이 사용되는 GPU는 3차원 장면을 그리는 등의 그래픽 작업에 대해서 매우 빠르게 동작한다. 그 이유는 각 단계의 연산이 동시에 처리가 가능한 그래픽의 특성을 잘 활용하도록 설계되고 그래픽 하드웨어의 성능이 빠른 속도로 향상되고 있기 때문이다. GPU의 발전은 고도로 병렬화된 컴퓨터 그래픽스 처리과정에서 최고의 성능을 얻어내기 위해 만들어졌기 때문에, 일반적인 응용분야에 그대로 적용하면 단순히 계산능력만큼의 성능향상을 기대하기는 어렵다. 또한, 하드웨어 구조가 급격히 변하는 경우가 많고, 대부분 외부에 공개되지 않으며 프로그래밍 모델이 기존과 다르고 프로그래밍 환경에 많은 제한 사항이 있다. 최근에 발표된 DirectX 9.0과 OpenGL 2.0은 HLSL과 GLSL을 지원하여, 기존의 불편함을 극복하였다. 이에 따라, 관련 하드웨어 및 응용 프로그램의 개발이 급속히 가속화될 전망이다[6].

일반적으로 3D 그래픽스의 처리과정은 크게 응용프로그램 단계(application stage), 도형 단계(geometry stage), 그리고, 벡터 그래픽을 그에 대응하는 픽셀 이미지로 변환하는 래스터화 단계(rasterization stage)의 세 단계로 구분된다[그림 1]. 응용프로그램 단계는 CPU에서 수행되며 사용자 입력처리와 3D 오브젝트간 충돌 등과 같은 물리적 연산을 담당하고 도형 단계에 정점 데이터를 제공하는 역할을 한다. 정점은 3D 오브젝트를 이루고 있는 점, 선, 삼각형 또는 다각형의 꼭지점이다. 정점 데이터는 정점 좌표, 색깔, 텍스처 등의 정보를 포함하고 있다. 도형 단계에서는 응용프로그램 단계로부터 입력된 각각의 정점에 대해 도형을 변환하는 연산과 광원에 의한 색깔 값이 계산된다. 변환 연산은 변환 행렬과 정점 벡터의 곱셈에 해당하며 빛에 대한 계산은 광원의 특성에 따라 연산이 달라진다. 변환 연산을 거친 정점은 내부를 채우고 있는 각 픽셀의 색을 결정하기 위해 래스터화 단계로 보내지게 된다. 래스터화 단계에서 각 픽셀 이미지는 여러 개로 나눈 프래그먼트(fragment)를 만든다, 프래그먼트의 크기가 작을수록 사실감이 높은 3차원 이미지가 만들어 진다. 래스터화 단계에서는 로딩되어 있는 텍스쳐 이미지를 외부 메모리로부터 읽어 텍스쳐 매핑을 하고 투명도(alpha) 채널 등의 나머지 픽셀도 연산을 통해 각 픽셀의 색깔 값을 결정하여 최종 결과를 프레임 버퍼에 저장한다.

GPU에서 가장 높은 연산능력을 필요로 하는 부분은 픽셀 세이더이다. 그 이유는 한 화면을 구성하는 프래그먼트의 수가 정점의 수보다 많기 때문이다. 따라서, 3D 그래픽, 게임과 같이 GPU를 사용하는 프로그램은 주로 픽셀 세이더를 최대한 활용하도록 작성되어 있다. GPU를 이용하는 프로그램은 먼저 데이터가 서로 독립적이며 병렬로 수행될 수 있도록 여러 부분으로 나누어 각각을 픽셀 세이더 프로그램으로 작성해야 한다. 프로그램의 입출력 데이터는 배열 형태로 텍스쳐 메모리에 저장할 수 있다. 픽셀 세이더는 각각의 프래그먼트에 대해 수행이 되고 프래그먼트의 수는 다각형(polygon)의 크기에 의해 결정되므로 출력하려는 데이터 범위는 정점의 좌표값을 조절하는 것으로 가능하다. 일반적으로 정점 좌표와 다각형은 화면과 동일한 사각형을 많이 이용한다.

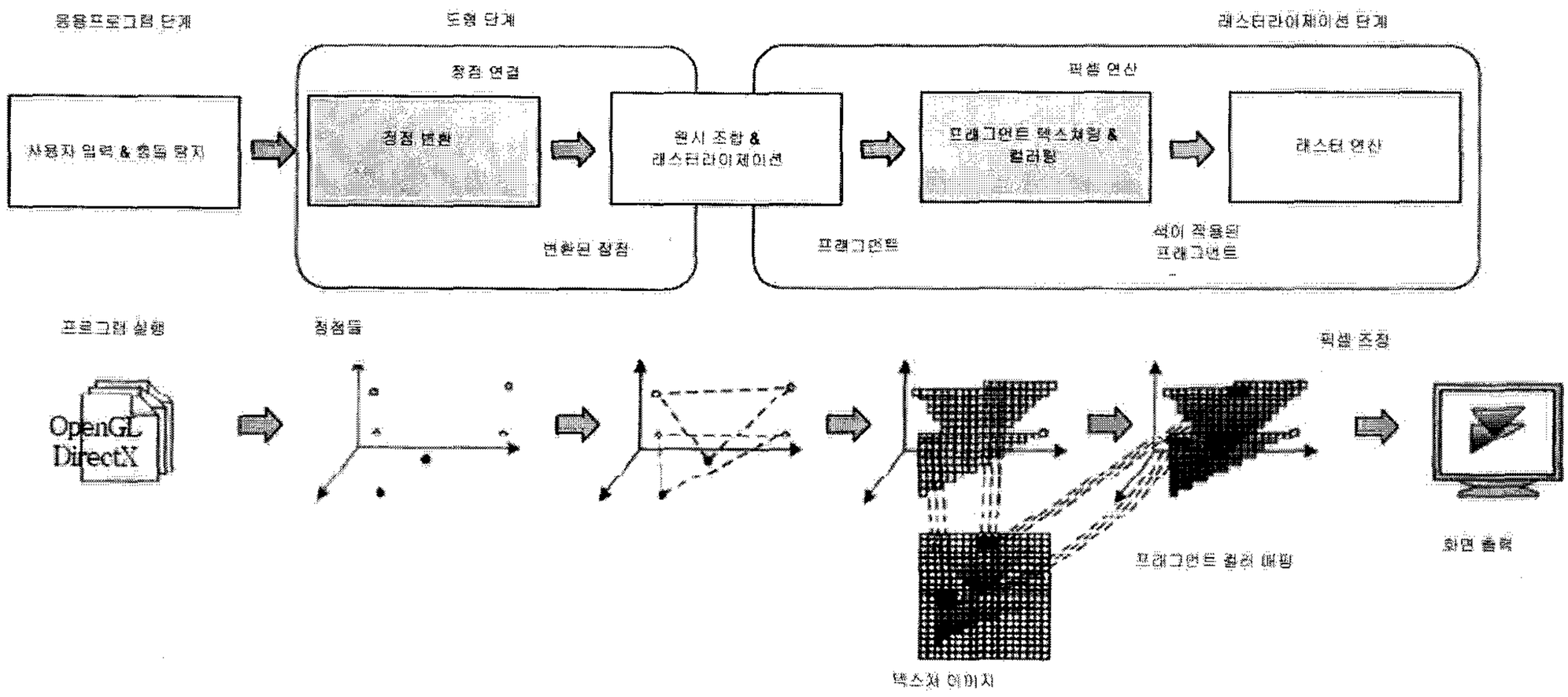


그림 1 GPU의 일반적인 처리과정

Fig. 1 General Processing of GPU

2.2. OpenMP

1997년에 산업표준으로 채택된 공유메모리 병렬프로그래밍 모델인 OpenMP는 표준 C/C++와 Fortran 77/90을 확장하는 병렬화 디렉티브와 라이브러리들의 집합 및 SPMD (Single Program Multiple Data) 수행 환경을 명시한 것이다. OpenMP의 특징은 크게 내포병렬성과 락킹(locking)을 통한 임계구역 동기화 구조로서, 특히 병렬성의 구현을 위한 디렉티브 개념을 지원하여 병렬 프로그래밍의 확장성을 높여준다[4]. 디렉티브 기반이란 것은 병렬 처리를 생각하지 않고 순차적으로 작성된 프로그램에 디렉티브들을 추가하는 것으로 원하는 부분만을 병렬적으로 처리하는 것이 가능하다.

```
int main (int argc, char **argv) {
  int i;
  large_array_t v;
  /* 병렬처리 하고 싶은 부분 */
  for (i = 0; i < N; i++)
    v[i] = 0;
  printf ("initialized array₩n");
  /* 병렬 처리하고 싶은 두 함수*/
  foo();
  bar();
}
```

```
int main (int argc, char **argv) {
  int i;
  large_array_t v;
  #pragma omp parallel for
  for (i = 0; i < N; i++)
    v[i] = 0;
  printf ("initialized array₩n");
  #pragma omp parallel sections {
    #pragma omp section
      foo();
    #pragma omp section
      bar();
  }
}
```

그림 2 병렬 처리 영역에 디렉티브 추가

Fig. 2 Adding Directives to Parallel Processing Region

2.1.1. OpenMP 수행 모델

OpenMP의 수행 모델은 포크/조인(fork/join) 모델이다. 마스터 쓰레드(master thread)는 OpenMP 디렉티브로 처리되지 않는 영역에서는 단일 쓰레드(single thread)연산을 수행하고, 디렉티브가 정의된 곳은, 자식 쓰레드(child thread)를 생성하여 쓰레드 별로 독립적으로 수행되게 된다.

각 실행을 마친 쓰레드는 다시 하나의 마스터 쓰레드로 합쳐져 순차적 코드를 수행하게 된다[그림 3].

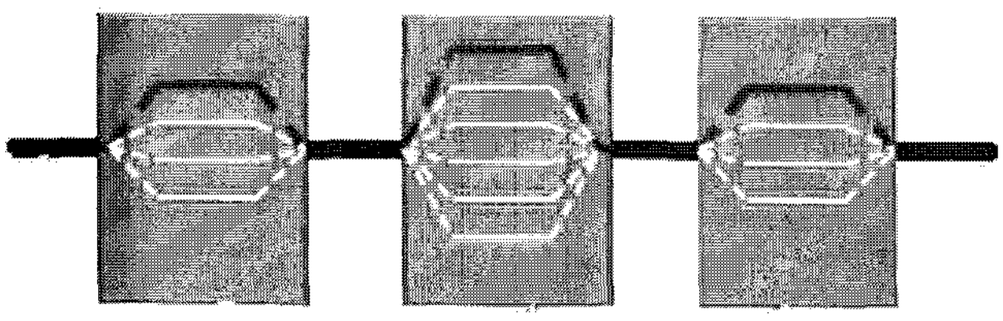


그림 3 포크/조인 수행 모델

Fig. 3 Fork/Join Model

2.2.2. OpenMP의 특징

OpenMP는 SGI(Silicon Graphics Inc.)가 Shared Memory 시스템의 확장성을 연구하는 것에서 시작되었다. 이 연구는 SGI 가 Cray를 인수하면서 본격적으로 가속화 되어 현재는 SGI 외에 intel, DEC, HP, IBM, Sun 등 많은 유명회사들이 참여하여 업계 표준으로 자리 잡게 된 것이다. 많은 회사가 참여한 표준이기 때문에 OpenMP API에 따라 작성하기만 하면 이를 지원하는 모든 시스템에서 사용 가능하다는 장점을 가지게 된다. OpenMP는 디렉티브 기반의 API이기 때문에 OpenMP를 사용하기 위해서는 컴파일러가 OpenMP 디렉티브를 인식해야 한다[5].

2.3. JPEG2000

JPEG 2000은 DCT를 사용하는 JPEG표준과 달리 DWT(Discrete Wavelet Transform)를 사용한다. JPEG에서 이용하는 DCT는 삼각 함수가 주기를 가지고 무한히 반복하는 함수이기 때문에 주파수 영역에서 시간 정보를 표현하지 못한다. 시간에 따라 주파수가 변동하는 신호를 푸리에 변환으로 나타내면 무한개에 가까운 삼각 함수를 더해야만 제대로 된 신호를 재생할 수가 있다. 실제로는 무한개의 삼각함수 즉, 무한개의 주파수로 원래 신호를 표현할 수는 없기 때문에 주로 사용되는 범위의 주파수만을 이용하여 표현을 하게 된다. 이 경우 시간에 따라 변화가 심한 신호는 원래 모양에 비해서 왜곡이 많이 발생하게 된다. JPEG에서는 이미지를 블록 단위로 압축을 하였기 때문에 사진을 확대하면 작은 블록 단위로 영상이 손상되는 것을 볼 수 있다. 그러나, JPEG 2000에서는 이미지를 웨이브렛 변환을 이용하여 압축하였기 때문에, 확대를 하여도 자연스럽게 연속적으로 매끄러운 선으로 표현될 수 있다. 웨이브렛 변환을 이용하여 이미지를 압축하면서 전체 이미지를 1/4, 1/16, 1/64 등의 낮은 해상도의 이미지로 줄일 수가 있다. 일반적인 비트맵 이미지는 축소할 때 원래의 정보가 제거되었기 때문에, 원래 크기로 복원하면 단순히 블록이 확대되는 블록화 현상이 생기지만, 웨이브렛 변환의 경우는 축소할 때 고주파 성분을 따로 저장하여, 확대할 때 다시 그 정보를 이용해서 원래 이미지를 복원한다. 원본 이미지를 작은 이미지로 축소하면서, 다시 확대할 때 사용할 고주파 성분을 ROI(Region of Interest), EBCOT 등의 압축 기술로 압축한 것이 JPEG 2000이다. 고주파 성분은 같은 값들이 반복해서 나타나는 경우가 많기 때문에, 기존 JPEG보다 더 효율적으로 압축을 할 수 있다[7].

JPEG 2000의 적용 분야로 관심을 받고 있는 분야는 디지털 카메라와 디지털 카메라 폰 등의 임베디드(embedded) 기기 응용 분야이다. 현재 디지털 카메라와 카메라 폰의 경우는 새로운 문화 조류를 형성할 정도로 보급이 확대되고 있으며, 작은 저장 메모리를 가지고 있는 여러 종류의 임베디드 기기들이 JPEG 2000 코덱을 많이 요구할 것으로 예상된다. JPEG2000 인코더/디코더의 블록 그림은 그림 4와 같다. 이산 웨이블렛 변환을 소스 영상 데이터에 적용하고 변환된 계수는 양자화를 거쳐 코드열을 생성하기 전에 엔트로피 부호화 과정을 거친다.

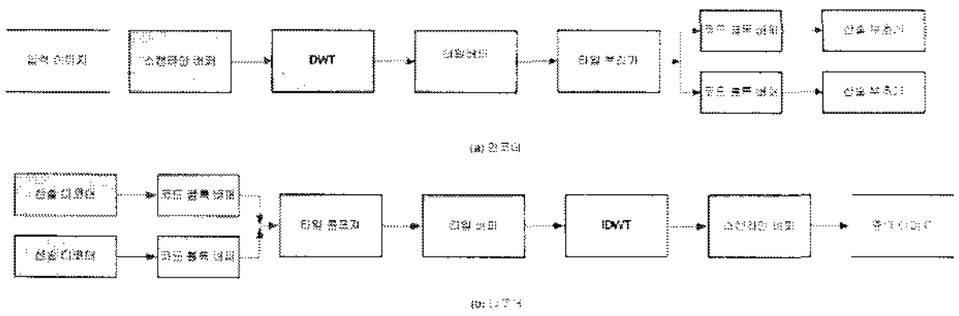


그림 4 JPEG2000의 블록도

Fig. 4 Block Diagram of JPEG2000

기존 CPU를 사용하는 코덱은 DWT와 같이 반복적인 연산과 실수 연산, 데이터 병목 현상 등으로 처리가 지연되는 경우가 발생하였다. 또한, 데이터 동기화 과정에서는 실제 코어의 개수보다 쓰레드의 개수가 많은 경우, 완전한 병렬처리가 불가능하였다. 본 논문에서는 GPU가 고정소수점 연산보다 부동소수점 연산에 뛰어난 성능을 보이는 것을 이용하기 위하여 JPEG2000코덱에서 사용하는 데이터들을 실수형 변수로 정의하여 연산 효율을 높였다. 입출력 이미지와 같은 배열 형태의 데이터는 텍스쳐 메모리에 저장하도록 하여 CPU와 GPU사이의 데이터 전송을 최소화 하였다. DWT 연산과 행렬 연산, 실수 연산은 GPU의 파이프라인(pipe-line)을 활용하여 빠른 연산 처리와 병렬 처리를 시도하였다[그림 5].

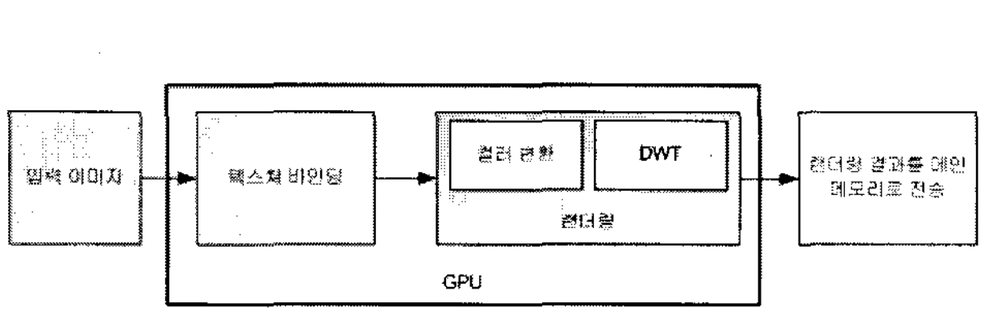


그림 5 GPU에서의 DWT 연산 순서도

Fig. 5 Block Diagram of DWT Operation on GPU

3. 실험 및 결과

실험을 위하여 3072x2304 사이즈의 BMP 이미지를 사용하였다. 이번 연구에서 사용한 GPU는 NVIDIA사의 GeForce 8800 GTS, 320MB를 사용하였다. CPU는 AMD Athlon 64 X2 Dual Core Processor 4200+, 2.2GHz를 사용하였다. 개발툴은 OpenMP를 지원하는 Microsoft Visual Studio 2005를 사용하고, Microsoft Windows XP 상에서 실험을 하였다.

실험은 GPU를 사용하지 않고 CPU만을 이용하여 JPEG2000 인코딩을 실행한 경우, (CPU) GPU를 이용한 JPEG2000 인코딩을 실행한 경우(CPU+GPU), GPU와 OpenMP를 적용하여 JPEG2000 인코딩(CPU+GPU+OpenMP)을 실행한 경우, 세 가지 경우에 대하여 실험을 하였다. 각 실험 별로 실행 시간과 CPU 점유율을 측정하여 비교하였다. OpenMP를 적용한 실험에서는 [그림 6]와 같이 CPU와 GPU가 메모리를 공유하면서 실수 연산이 많은 부분은 GPU가 연산을 하고, 반복적인 연산이 많은 DWT 연산과 텍스쳐 메모리와 프레임 버퍼 영역의 연산은 OpenMP를 이용하여, 병렬로 처리하도록 하였다.

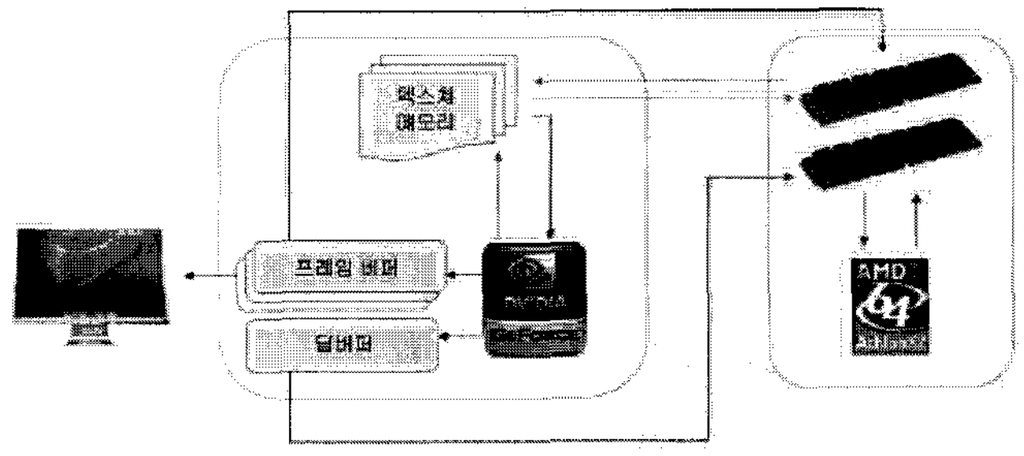


그림 6 시스템 구성도

Fig. 6 Construction of the System

[표 1]과 [그림 7]에서 보는 바와 같이 CPU만을 사용한 JPEG200 인코딩 실험에서는 평균 22.7초가 소요되었지만, GPU를 이용한 실험에서는 평균적으로 2.478초가 소요되었다. 이런 실험 결과를 비교해 보면 실행 시간에서 약 10배의 성능 개선 효과를 얻을 수 있음을 알 수 있다. 그리고, OpenMP를 이용하여 병렬 처리를 했을 때는 평균적으로 1.355초가 소요됨으로써 더욱 속도가 개선이 된 것을 알 수 있다.

또한, [표 2]와 [그림 8]에서 보는 바와 같이 CPU만을 이용한 경우, CPU 점유율이 평균 50%이지만, GPU를 같이 적용한 경우 10%대 초반으로 CPU의 점유율이 낮아진 것을 확인할 수 있다. CPU의 점유율이 낮아지는 만큼 GPU가 다른 작업을 수행한 결과이다. 그리고 OpenMP를 사용하여 병렬 처리를 적용한 경우, CPU 점유율이 평균 15%로 OpenMP를 적용하지 않은 경우보다 CPU점유율이 조금 상승하였다. 그 이유는 CPU에서 병렬 처리를 위해 여러 개의 쓰레드를 생성하여 처리하기 때문이다.

표 1 실행시간 (sec)

Table 1 Running Time (sec)

	CPU+GPU +OpenMP	GPU	CPU
1	1.359	2.484	25.759
2	1.360	2.468	21.500
3	1.344	2.468	24.264
4	1.360	2.468	19.344
5	1.359	2.500	25.173
6	1.360	2.484	20.173
7	1.344	2.469	24.264
8	1.344	2.468	18.488
9	1.360	2.485	23.203
10	1.359	2.485	25.156
평균	1.355	2.478	22.732

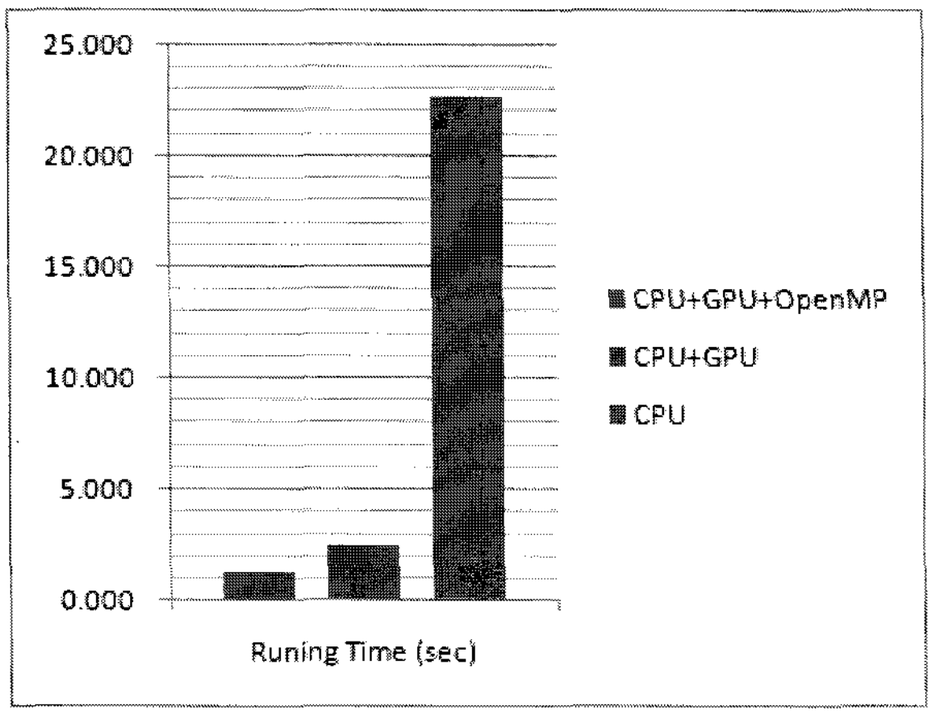


그림 7 실행시간 (sec)
Fig. 7 Running Time (sec)

표 2 CPU 점유율 (%)
Table 2 CPU Resources Occupation Rate (%)

	CPU+GPU +OpenMP	GPU	CPU
1	15	12	50
2	15	11	50
3	16	13	50
4	15	12	49
5	15	12	50
6	15	11	50
7	15	12	49
8	16	13	50
9	15	12	50
10	15	10	50
평균	15	12	50

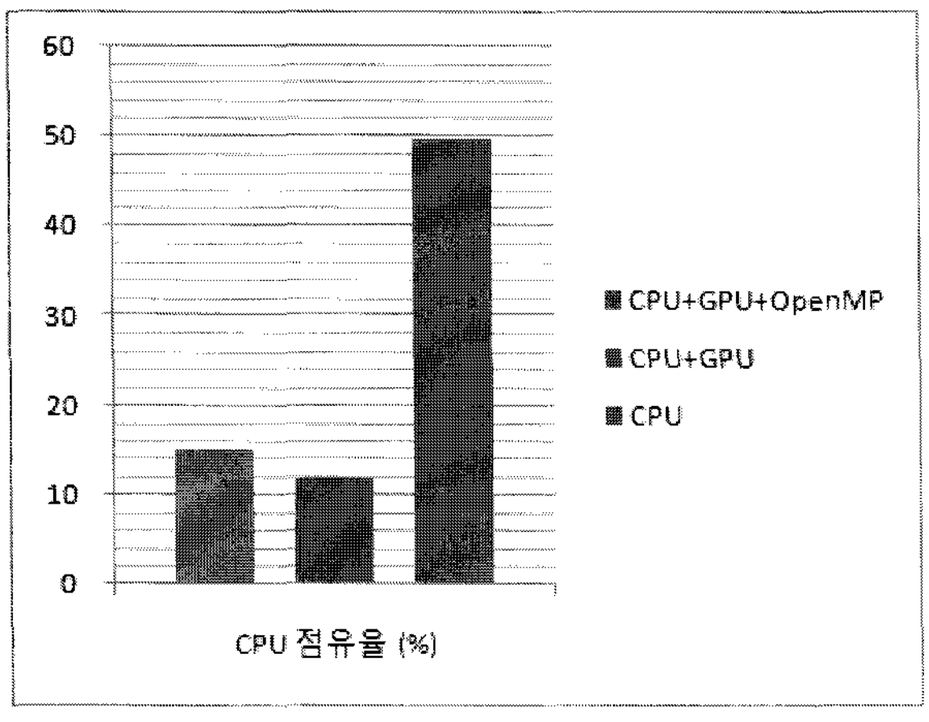


그림 8 CPU 점유율 (%)
Fig. 8 CPU Resources Occupation Rate (%)

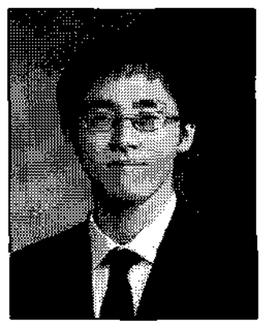
4. 결 론

본 논문에서는 OpenMP를 이용한 병렬 처리 방법과 GPU를 이용하여 상대적으로 높은 CPU의 부하율을 개선하고, JPEG2000코덱을 빠르게 처리하는 방법을 제안하였다. 실험결과에서 처리 속도에서는 약 10배의 속도 향상과 약 25% 수준의 CPU 점유율을 보여주었다. OpenMP를 사용하였을 경우, CPU에서 병렬 처리를 처리하게 때문에 CPU 점유율이 약간 상승하였지만, 빠른 처리 속도가 이를 보완해 주었다. 향후 연구로서는 멀티 코어를 지원하는 CPU와 GPU의 장점을 이용하여 MPEG4와 같은 동영상 분야에 적용할 수 있도록 할 계획이다.

참 고 문 헌

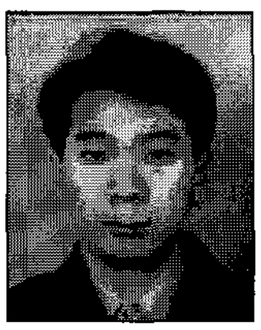
[1] J. Kruger and R. Westermann, "Linear Algebra Operators for GPU Implementation of Numerical Algorithms," ACM Transactions on Graphics 22(3), pp. 908-916, 2003.
[2] M. Harris, "Fast fluid dynamics simulation on the GPUs," GPU Gems, pp. 637-665, Addition Wesley, 2004.
[3] J. Bolz, I. Farmer, E. Grinspun, and P. Schroder, "Sparce Matrix Solvers on the GPU: conjugate gradients and multigrid," ACM Transactions on Graphics 22(3), pp. 917-924, 2003.
[4] The OpenMP Forum. OpenMP C and C++ Application Program Interface, Version 1.0. http://www.openmp.org. Oct. 1998.
[5] J. Throop, "OpenMP: shared-memory parallelism from the ashes", Computer, Volume: 32 Issue: 5, pp. 108-109, May 1999.
[6] R. Rost, "OpenGL® Shading Language", Addison Wesley, 2004.
[7] A. Skodras, C. A. Christopoulos, and T. Ebrahimi, "The JPEG 2000 still image compression standard," IEEE Signal Processing Magazine, pp. 36-58, 2001.

저 자 소 개



이 동 하 (李 東 河)

2006년 동국대학교 전기공학과 (공학사).
2008년 동국대학교 대학원 전기공학과 (공학석사).



조 시 원 (曺 時 元)

1994년 동국대학교 전기공학과 (공학사).
1998년 동국대학교 대학원 전기공학과 (공학석사).
2003년~현재 동국대학교 대학원 전기공학과 박사과정
Tel : 02-2260-3350
E-mail : stsolaris@gmail.com

이 동 욱 (李 東 旭)

1983년 서울대학교 전기공학과 (공학사).
1985년 서울대학교 대학원 전기공학과 (공학석사).
1992년 Georgia Tech.대 (공학박사).
1993년~ 현재 동국대학교 전기공학과 교수
Tel : 02-2260-3350
E-mail : dlee@dongguk.edu