Journal of Digital Contents Society Vol. 9 No. 3 Sept. 2008(pp. 371-377)

GPU 기반의 MPEG-2 디코더의 구현

김경수*, 김홍식**, 김정길***, 박우찬****

요 약

최근 GPU 성능은 CPU 성장 속도에 비하여 급속도로 증가하고 있으며 계산이 많이 요구되는 다양한 응용 프로그램에서 GPU를 이용하려는 시도가 활발히 진행되고 있다. 본 논문에서는 GPU 프로그래밍 언어인 CG를 이용하여 MPEG-2 디코더를 구현하였다. 제안된 방법은 텍스쳐 데이터를 사용하여 비디오 표준에 맞춘 블록 렌더링을 하는 방식이며, 이는 스트림 프로세싱 구조인 GPU의 파이프라인을 이용하여 높은 병렬성을 가지고 실행된다. 또한 시스템 메모리와 GPU 사이의 데이터 대역폭을 줄이기 위해 그래픽 카드의 지역 메모리를 사용한다. 제안한 방법을 적용한 결과 CPU 보다 2배 이상의 성능 향상을 볼 수 있었다.

Implementation of GPU based MPEG-2 Decoder

Kyung Su Kim*, Hong Sik Kim**, Cheong Ghil Kim***, Woo Chan Park****

Abstract

Recently the performance of GPU is increasing much faster compared to GPU and GPU is used for various application programs. In this paper, MPEG-2 Decoder is implemented based on a GPU programming language, CG. The proposed methodology is to perform block rendering with texture data according to video standard with very high parallelism by using the pipeline of GPU which is a stream processing structure. To reduce the data bandwidth between system memory and GPU, local memory is used for graphic card. According to the experiment, the proposed scheme shows performance improvement by more than 2 times compared to CPU based scheme.

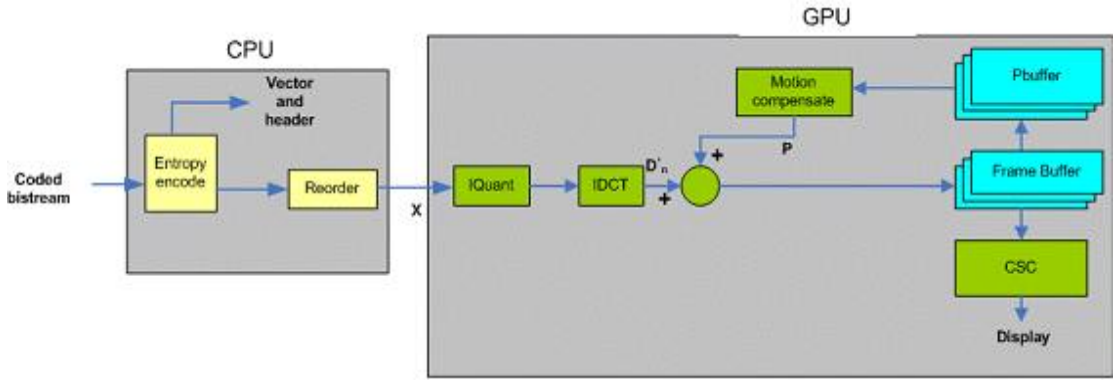
Keyword : GPU, MPEG, Parallelism

1. 서론

그래픽 카드는 지난 10년간 비디오 디코딩을 보조하는 용도로 사용 되어왔다. 일례로 비디오 오버레이는 리소스가 많이 드는 색상 공간 변환(color space conversion)을 조작하는데 이용 되고 있다. 최근에는 디코딩의 일부 특정기능에 대한 가속을 제공하는 방식에 대한 연구 결과들이 논문을 통해 발표되고 있고, DirectX 비디오 가속(DXVA)의 명세서에 정의되어 오늘날 상업적 그래픽 칩에 적용되고 있다. 하지만 이러한 그래픽 칩들은 대부분 특정한 비디오 표준만을 위한 고정된 하드웨어 회로로 구성되어 있다.

최근의 그래픽 칩에서는 프로그램 가능한 비디오 처리 엔진을 제공하기 시작하였으며, 전용 하드웨어를 사용하는 방법과 하드웨어의 추가 쉐이터(shader)를 이용한 2가지로 구분될 수 있다. 첫 번째 방법은 GPU(graphic processor unit) 내부에 전용 블록을 내장하고 이들을 프로그램을 통하여 제어하는 방식이다. 그 예로서 그래픽 카드회사 nVidia의 PureVideo와 ATI의 Avivo가 있다. nVida PureVideo 관련하여 기술문서[1]가 공개 되어 있으며, ATI의 Avivo는 GPU 내에 프로그램 가능한 병렬 쉐이더 하드웨어를 사용하는 방식이다. 두 번째 방법으로써 최신의 그래픽 칩은 128 비트 스트림 프로세서를 내장하고 있고, 이와 같은 내장 프로세서를 이용한 그래픽 칩의 프로그래머블 기능에 비디오 디코

※ 제일저자(First Author) : 김경수
접수일자:2008년07월10일, 심사완료:2008년09월03일
* 한국전자통신 연구원,
kimks0326@etri.re.kr
** 연세대학교, 전기전자공학과
*** 남서울대학교, 컴퓨터학과
**** 세종대학교, 컴퓨터공학과



(그림 1) 구현된 비디오 디코더의 전체구조

딩에 대한 연구가 활발하게 이루어지고 있다 [2][3][4].

[2]에서는 쉐이더 기반의 비디어 디코딩에 대한 결과를 최초로 발표하였다. 이는 비디오 디코딩의 기능 블록들 중 제한적인 MC(motion compensation)와 CSC(color space conversion) 만을 GPU에 구현하였다. [3]은 H.263의 기능 블록 중에 VLD를 제외한 모든 것을 GPU에 구현하였다. 하지만, 이 논문은 쉐이더 기반의 구현에 초점을 두었으며, CPU에서 비디오 디코딩을 처리할 경우 보다 성능이 더 낮다. [2][3]은 비디오 디코딩에 사용할 데이터가 CPU에서 CPU로 전송될 때 텍스쳐를 사용함에 반하여, [4]에서는 포인트 프리미티브를 사용하는 방법을 발표하였다. 이로 인하여 CPU에서 GPU로 이동하는 데이터의 량이 다소 줄어들었으며, 이로 인하여 성능 향상이 다소 있었다. 하지만, MPEG 표준을 지원하기에는 다소 어려움이 있다.

비디오 디코더를 위하여 쉐이더 프로그램을 이용한 대부분의 기존 논문들에서는 구현 방안에 대한 구체적인 설명이 부족하여 실제 시스템에서 이를 구현하기가 매우 어렵다. 또한, MC 및 CSC와 같은 기능 블록은 MPEG 표준을 만족하는 처리 방식에 대한 기술이 되어있지 않고 있다. 특히, GPU 프로그래밍의 경우 최신 기술이기 때문에 알려진 구현방법 및 사례가 매우 부족하며 플랫폼 의존적 병렬 프로그램(platform dependent parallel program) 이기 때문에 구현이 매우 어렵다는 문제점을 가지고 있다.

본 논문에서는 현재 발표된 쉐이더 프로그램을 이용한 비디오 디코더 관련 논문들 중 가장 좋은 성능을 가진 알고리즘을 조합하고 본 논문에서 구축한 환경에 맞도록 수정하여 MPEG-2 디코더를 구현하였다. 또한, MPEG 표준을 만족하는 MC 와 CSC에 대해서는 구현 방법이 구체적으로 제시된 자료를 발견하지 못했으며, 본 논문에서는 쉐이더에 적합한 방식을 제시하고 이를 구현하였다. GPU의 로컬 메모리를 사용하기 위하여 핑퐁(pingpong)소스[5]를 이용하여 내부 메모리 버퍼인 P-버퍼(P-buffer)를 이용할 수 있는 GPU 프로그램 환경을 구축하였다. 비디오 디코딩의 기본 알고리즘은 JM11.0 참조 소프트웨어 소스를 따랐다. 데이터 전송 방법은 GPU Gems와 같은 서적들을 참조하였으며, A&A 알고리즘은 nVidia의 JPEG 소스를 참조하였다[1][6].

논문의 구성은 다음과 같다. 2장에서는 제안하는 소프트웨어 디코더의 구조와 동작을 설명한다. 3장에서는 실험 환경 및 실험 결과에 대해 기술하였고, 마지막으로 4장에서 결론 및 향후 계획을 제시 하였다.

2. 제안하는 디코더 시스템 구성

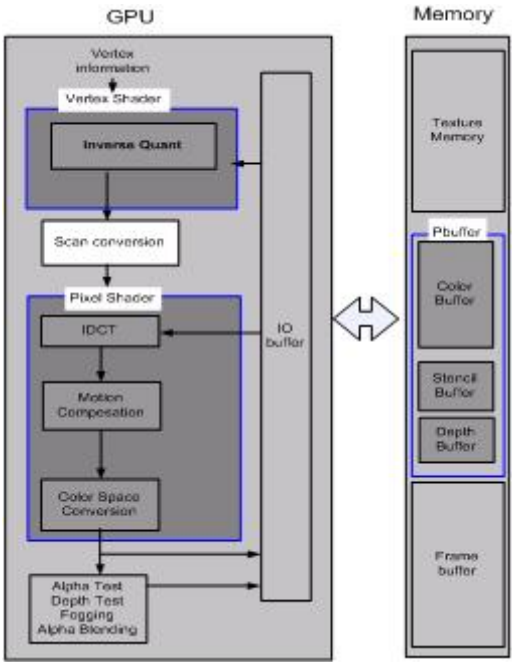
2.1 전체 구조 및 설계 방법

(그림 1)은 구현된 비디오 디코더의 전체 구조를 보여주고 있다. CPU를 이용하여 엔트로피 디코딩(entropy decoding)을 수행 하고 GPU에 적합하도록 데이터를 정렬 하였으며, GPU를 이용하여 역양자화(IQUAN), IDCT, MC, CSC작업

을 수행 하였다. 또한 GPU의 로컬 메모리를 효과적으로 사용하기 위해 P-버퍼를 이용하였다.

제안하는 구조에서는 엔트로피 디코딩이 끝난 데이터 가운데 0이 아닌 데이터를 이동하기 위해서 정점의 색상을 입력하는 함수를 이용하였다. 쉐이더를 이용한 비디오 디코더를 구현하기 위하여 OpenGL API를 사용 하였으며, 이때 가장 쉽게 이용될 수 있는 함수로는 glColor()와 glVertex()가 있다. glColor()함수를 사용하여 데이터를 정렬하였으며, 이때 R,G,B,A 4개의 채널을 이용 가능하다. 이것은 glVertex()를 사용하는 것에 비해 CPU와 GPU사이에 메모리 대역폭을 줄일 수 있는 장점을 가지고 있다.

MPEG-2 비디오 디코더의 각 기능 블록들은 GPU의 쉐이더 프로그램을 이용하여 구현 하였다. 기능 블럭들은 역양자화, IDCT, MC, CSC로 구성되어 있고 역양자화와 IDCT를 구현하였고, MC와 CSC는 현재 발표되어 있는 알고리즘을 이용하여 본 비디오 디코더에 최적화하여 구현하였다. 또한 GPU와 GPU 지역 메모리 사이의 메모리 대역폭을 줄이기 위해 P-버퍼를 이용하였으며, 이를 위하여 OpenGL Extension들을 사용하여 환경을 구축 하였다.

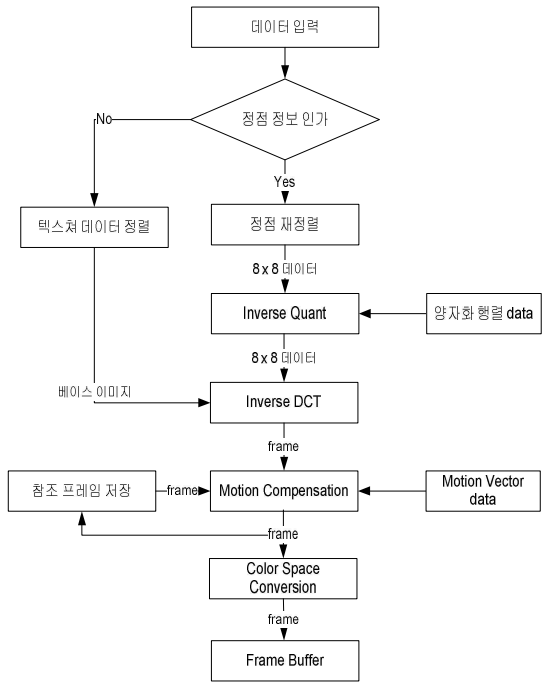


(그림 2) GPU를 이용한 비디오 디코더의 구조도

(그림 2)는 GPU의 버텍스 쉐이더(vertex shader)와 픽셀 쉐이더(pixel shader)에서 각각 분리되어 수행되는 작업을 보여주고 있다. 제안하는 구조에서는 버텍스 쉐이더를 이용하여 역양자화를 수행 하였다. 역양자화는 0이 아닌 데이터를 정점의 색상 속성을 이용하여 전달하기 때문에 버텍스 쉐이더를 이용한 프로그램에 최적화 된다. 하지만 IDCT, MC, CSC와 같은 작업들은 전체 해상도만큼의 작업이 필요하기 때문에 픽셀 쉐이더를 이용하여야한다.

2.2 전체 데이터의 흐름도

(그림 3)은 MPEG-2 디코더의 데이터 흐름을 나타내고 있다. 각 블록은 8x8개 픽셀들로 구성되며, 전체 데이터들의 이동은 기본 단위의 블록이다. CPU에서 엔트로피 디코딩을 수행한 다음 데이터는 GPU로 이동된다. GPU에 들어온 데이터는 역양자화, IDCT, MC, CSC 등이 차례로 수행 된다.



전체 데이터의 이동은 아래의 순서에 따라 진행된다.

1) GPU에서 VLD를 수행한 데이터가 이동된다.
2) 이 데이터가 정점의 정보인지를 비교한다. 정점 정보일 경우 3)번으로 이동하고 아닐 경우 4)번으로 이동한다.
3) 정점 데이터들을 정점 하나당 4개의 채널에 차례로 저장하고 데이터를 GPU로 보낸다.
4) 텍스쳐 데이터를 P-버퍼에 저장한다.
5) CPU로부터 이동된 정점들의 정보를 가지고 역양자화를 수행한다. 데이터 전송은 8x8 블록 형태로 진행된다.
6) 8x8 블록 형태로 입력된 데이터와 텍스쳐 이미지에 저장된 베이스 이미지를 이용하여 IDCT를 수행한다. 이러한 과정을 프레임내의 모든 블록에 대해서 수행한다.
7) 현재 프레임과 참조 프레임 그리고 모션 벡터를 이용하여 동작보정을 수행한다.
8) 동작보정을 수행한 데이터를 CSC(컬러 공간 전환)을 수행하고 이를 프레임 버퍼에 저장한다.

2.3 세부 기능들의 설계

2.3.1 역양자화(IQUANT)

역양자화에 대한 처리 방법은 현재 발표된 논문 중 가장 좋은 성능을 가지고 있는 Peking 대학의 처리방법 [4]를 참고하여 설계 구현하였다. 이는 포인트를 이용하여 데이터를 이동시키는 방법으로서 CPU와 GPU사이에 메모리 대역폭을 감소시키는 가장 효과적인 방법이다. 역양자화는 정의되어 있는 양자화 파라미터를 곱하여 수행되며, 양자화 기본적인 계산은 식(1)에 나타나 있다.

$$X_{iq}(u,v) = qp \times QM(u,v) \times X_q(u,v) \quad \text{(식1)}$$

여기서 u, v는 행렬의 인덱스이며, qp는 양자화 파라메터, QM(u,v)는 양자화 행렬이고 Xq는 현재 입력된 값을 의미한다[4].

역양자화 블록을 처리하기 위하여 버텍스 쉐이더에서 쉐이더 프로그램을 이용하여 처리한다. 정점의 컬러 속성을 이용한 데이터 이동 방식은 하나의 정점에 4개의 값을 저장하고 있다. 이 정점 정보들은 버텍스 쉐이더를 이용하여 데이터의 값을 변경할 수 있으며, 텍스쳐의 좌표변환을 이용하여 데이터를 텍스쳐에 저장 할 수 있다.

버텍스 쉐이더를 이용하여 프로그램을 하는 방법은 정점들의 수가 일정 하지 않기 때문에 쉐이더 프로그램을 정점의 수에 맞추어 프로그램밍 하여야 하며, CPU는 생성된 정점의 수에 따라 쉐이더 프로그램의 실행 시켜야 하고 P-버퍼를 이용하여 지역메모리에서 프레임 크기의 텍스쳐를 메모리에 준비하여야한다. 쉐이더 프로그램은 정점의 개수에 따라 내부 임시 레지스터를 이용하여 데이터들을 저장하고 상수 레지스터를 이용해 양자화 행렬의 값과 곱셈 연산을 수행한다. 곱셈 연산의 결과로 생성된 값은 텍스쳐의 좌표를 이용하여 미리 준비된 P-버퍼의 텍스쳐에 저장한다. 역양자화의 수행 단계는 다음과 같다.

1) 정점의 어트리뷰트 중 컬러 정보를 이용하여 CPU에서 정렬한다. 이때 0이 아닌 데이터만 컬러 정보에 저장한다.
2) 정점 정보를 정점 쉐이더를 이용하여 프로그램한다. 이때 정점의 개수가 고정 되어 있지 않기 때문에 많이 사용 되어지는 개수에 맞추어 프로그램을 작성한다.
3) 계산된 정보를 텍스쳐의 좌표를 이용하여 저장한다.

2.3.2 IDCT(Inverse DCT)

IDCT에 대한 처리 방법은 [4]를 참고하여 설계 구현하였다. 이 방법은 IDCT를 위하여 기본 베이스 이미지를 저장하고 저장된 데이터와 현재 프레임의 값을 내적계산(dot-product)을 하여 생성되는 결과 값을 텍스쳐에 저장하는 방식이다. 하지만 이 연구 방법은 현재 프레임의 데이터들이 이동되는 방법에 대한 구체적인 언급을 하고 있지 않고 있다.

제안하는 구조에서는 이 데이터의 이동 방식을 P-버퍼를 이용한 텍스쳐 메모리의 형태로 지역 메모리를 할당 하였다. 이는 각 쉐이더들이 데이터의 인덱싱 없이도 텍스쳐 메모리의 좌표를 이용하는 것과 같은 방식을 이용할 수 있는 장점을 가지고 있으며, 쉐이더는 텍스쳐 좌표의 값을 이용하여 병렬로 작업을 수행 할 수 있다.

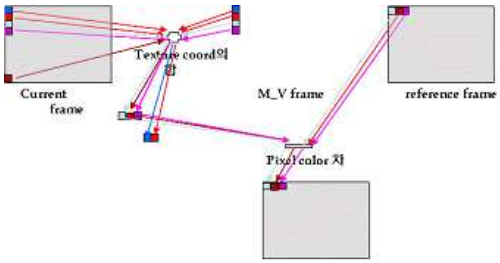
또한 역양자화와 IDCT가 단일 패스 렌더링으로 처리할 수 있다는 장점을 가지고 있다.[4][7].

CPU는 IDCT에 사용 되어질 기본 베이스 IDCT 이미지를 텍스쳐 만들어 텍스쳐 메모리를 이용하여 저장한다. 기본 베이스 IDCT 이미지는 사용 빈도가 높기 때문에 내부 지역 메모리에 P-버퍼를 할당하여 저장하게 된다. IDCT에 대한 쉐이더 프로그램은 2개의 텍스쳐 이미지를 호출하여 사용하며, 2개의 텍스쳐 이미지는 현재 프레임 이미지와 기본 베이스 IDCT 이미지이다. 쉐이더 프로그램은 for문을 사용 할 수 있으며 임시 레지스터를 사용 할 수 있기 때문에 이를 이용하여 IDCT의 기본 알고리즘 구현할 수 있다. 다음은 IDCT에 대한 수행 절차에 대한 설명이다.

1) CPU는 IDCT의 계산을 빨리 하기 위해서 기본 IDCT 베이스 이미지를 만든다. 기본 블록의 크기는 8x8이고 이동 단위는 64x64의 텍스쳐 이미지를 만든다.
2) 기본 IDCT 베이스 이미지는 2D-텍스쳐 이미지 형태로 GPU의 로컬 메모리에 저장된다. 이때 P-버퍼를 이용해서 이 데이터를 저장한다.
3) 역양자화 계산 후 만들어진 2D-텍스쳐 이미지와 기본 IDCT 베이스 이미지를 픽셀 쉐이더에서 호출한다.
4) 현재 프레임 이미지의 텍스쳐를 쉐이더 프로그램안의 for문을 이용하여 기본 베이스 이미지와의 도트 프로덕트를 수행한다.
5) 4)번에서 수행된 결과를 현재 프레임의 텍스쳐 좌표와 동일한 좌표를 가진 새로운 텍스쳐에 저장한다.

2.3.3 동작 보정(Motion Compensation)

이 절에서는 IDCT의 결과로 저장되어 있는 2D-텍스쳐를 이용한 동작 보정방법에 대해서 설명한다. 동작보정은 모션벡터를 이용하여 현재 블록과 참조 영상에 대한 덧셈을 계산하여 최종 영상을 구하는 방법이다. CPU는 픽셀 쉐이더의 상수 레지스터에 모션 벡터를 저장한다. 픽셀 쉐이더는 이 모션 벡터를 이용해서 입력된 8x8블럭의 텍스쳐에 좌표에 모션 벡터의 값을 더해서 참조 영상과의 계산을 수행하는데 이때 텍스쳐 좌표는 픽셀 좌표와의 계산을 고려하여야한다 [8]. 동작 보정은 다음의 단계를 통해 이루어진다. (그림 4)는 제안하는 구조에서의 동작보정의 예를 보여주고 있다.



(그림 4) 동작보정의 예

2.3.4 색상 공간 변환 (CSC: Color Space Conversion)

CSC 연산은 YCbCr 데이터를 그래픽 카드의 display 버퍼에서 출력 할 수 있도록 RBG로 변환하는 작업을 수행한다. 다음의 식을 이용하여 RGB의 값을 구할 수 있다.

```
R =  (Y + ((1.402) * cr));
G =  (Y - ((0.344 * cb) - ((0.714) * cr)));
B =  (Y + ((1.772) * cb));
```

제안하는 구조에서는 픽셀 쉐이더를 이용하여 CSC연산을 수행한다. CSC연산을 수행하기 위해서는 Y 프레임과 Cb, Cr 프레임이 필요하고 Cb와 Cr 프레임의 경우는 Y 프레임의 1/4의 크기를 갖는다. Y, Cb, Cr 프레임에 대해서 P-버퍼를 이용한 내부 메모리를 할당하여 저장 하였다. P-버퍼는 텍스쳐 좌표를 이용하여 해상도만큼의 픽셀들을 저장하며, 이는 데이터에 접근할 경우 인덱싱에 사용 할 수 있다.

3. 실험환경 및 결과

3.1 실험환경

정확한 실험 결과를 얻기 위하여 현재 비디오에 관련 된 논문에서 채택하는 일반 적인 데이터를 사용 하였다. 실험에 사용된 환경은 Windo wXP 운영체제에서 OpenGL API들을 사용하였고, CG 쉐이더 프로그램 언어를 이용하여 GPU에서 동작하는 소프트웨어 코덱을 구현하였다.

성능 비교를 위하여 3개의 다른 영상을 사용하였다.

실험은 3GHz의 인텔 프로세서와 3GB 시스템 메모리와 nVidia의 GeForce 8800GTS 그래픽 카드를 장착한 데스크 탑 컴퓨터에서 수행하였다. 쉐이더 언어인 CG를 이용하기 위해서 nVidia에서 제공하는 SDK 통합 라이브러리를 이용하였다[9]. 비디오 디코딩을 위한 각각의 모듈에 대한 실험을 CPU와 GPU에서 실행하여 그 수행 시간을 비교하였다.

3.2 실험 결과

3.2.1 비디오 디코더 각 모듈의 속도

<표 1> 모듈별 실험 결과 (단위: sec)

	CPU	GPU	Speed-up
DCT	0.0034	0.00125	2.8
IDCT	0.0035	0.00133	2.6
QUANT	0.0025	0.0008	3.1
IQUANT	0.0020	0.0009	2.2
MC	0.0021	0.0010	2
CSC	0.0070	0.0008	8.75

구현 된 비디오 코덱 모듈 중 시스템 구현 시 가장 좋은 성능을 발휘하는 모듈을 판단하기 위해 실험을 수행하였다. <표 1>은 320x240 영상에 대한 각 모듈의 실험 결과를 나타낸 것이다. 모듈에 따라 CPU와 GPU의 속도 차를 비교 하였다. 비교결과 각 모듈에 대해서 CPU보다 GPU로 구현한 것이 약 2배 ~ 8배 정도의 성능 향상을 가져오는 것을 알 수 있다.

3.2.2 영상에 따른 비디오 디코딩 속도

3가지 다른 영상을 비교함으로써 영상에 따른 성능 변화에 대한 실험을 수행하였다. <표 2>의 결과에 의하면, 동작의 변화가 큰 영상 일수록 연산량이 증가하는 것을 알 수 있다. 모든 영상에 대해서 GPU를 이용한 방법이 2-3배 정도 좋은 성능을 보이고 있다. 이는 [2][3]의 결과보다 성능면에서 우월하며, [4]와는 거의 대등한 결과이다. [4]는 포인트 프리미티브를 사용하여 CPU에서 GPU로의 전송량이 상대적으로 적으나 MPEG 표준을 지원하는 데는 한계가 있기 때문에 본 논문의 결과는 경쟁력이 있다고 생각된다.

<표 2> 영상에 따른 성능비교 (단위:sec)

비교영상	CPU	GPU
Car	0.035	0.016
Golf	0.031	0.012
foreman	0.033	0.011

4. 결론 및 향후 계획

GPU 프로그래밍 언어인 CG를 이용하여 MPEG-2 디코더를 구현하였다. 이는 스트림 프로세싱 구조인 GPU의 파이프라인을 이용하여 높은 병렬성을 가지고 실행 가능하며 시스템 메모리와 GPU 사이의 데이터 대역폭을 줄이기 위하여 그래픽 카드의 지역 메모리를 사용하였다. 실험 결과는 GPU 사용하여 디코더를 구현한 결과 CPU를 이용한 방법에 비해 약 2-3배 정도의 성능 향상을 보여주고 있다. 본 연구의 향후 과제로는 최신의 GPU에 대해서 H.264 및 MPEG-4의 모든 기능을 구현할 예정이다. 이를 위하여 현재 쉐이더 구조를 H.264나 MPEG-4의 기능 등을 지원할 수 있도록 개선하는 연구를 진행할 예정이다.

참 고 문 헌

[1] http://www.nvidia.com/Products/Geforce256.nsf

[2] Guobin Shen,Guang-Ping Gao, Shipeng Li, "Accelerate Video Decoding With Generic GPU", IEEE Transactions on Circuits and Systems for Video Technology, VOL 15, NO 5, pp.685-693, MAY 2005.

[3] Ari Hirvonen, Tapani Lepanen, "H.263 Video Decoding on Programmable Graphics Hardware", IEEE International Symposium on Signal Processing and Information Technology, pp.902-907, MAY 2005.

[4] Bo Han and Bingfeng Zhou, "Efficient Video Decoding on GPUs by Point Based Rendering," Procs. of the 21st ACM SIGGRAPH/Eurographics Symposiums on Graphics Hardware, pp.79-86, 2006

[5] http://www.gpgpu.org

[6] Thilaka Sumanaweera, Donald Liu: Medical Image Reconstruction with the FFT, GPU Gems 2, Matt Pharr(ed.), Addison-Wesley, 2005.

[7] Fang B., Shen G., Li S., Chen H.: "Techniques for efficeitne dct/idct implementation on generic GPU".

In Proceedings of IEEE International Symposium on Circuit and Systems, pp.1126-1129, 2005
[8] Francis Kelly and Anil Kokaram, "A Fast image interpolation for motion estimation using graphics hardware". Procs. of the IS&T/SPIE Electronic Imaging, pp. 184-194. May 2004
[9] nVidia SDK ver 9.1: Code samples: Discrete cosine transform, Available at http://developer.nvidia.com/object/sdk home.html

김 경 수

2007년 : 세종대학교 대학원 (컴퓨터공학 석사)
2007년~현 재 : 학국전자통신연구원(ETRI) 연구원
관심분야 : 3D Graphics(shader), Multi-Core processor

김 홍 식

1992년 : 연세대학교 전기공학과 학사
1999년 : 연세대학교 전기및컴퓨터공학과 석사
2004년 : 연세대학교 전기전자공학과 박사
2004년~2005년 : Virginia Tech Post Doctorial Research Fellow
2005년~2006년 : 삼성전자 반도체총괄 시스템 LSI 사업부 CAE 팀 책임연구원
2007년~현재 : 연세대학교 TMS 사업단 BK 연구교수
관심분야 : VLSI 설계 및 테스트, 영상 이미지 압축, 저전력 설계 및 테스트

김 정 길

2003년 : 연세대학교 대학원 (컴퓨터과학과 석사)
2006년 : 연세대학교 대학원 (컴퓨터과학과 박사)
2006년~2007년 : 연세대학교 컴퓨터과학과 박사후 연구원
2007년~2008년 : 연세대학교 컴퓨터과학과 연구교수
2008년~현 재 : 남서울대학교 컴퓨터학과 전임강사
관심분야 : 멀티미디어 임베디드 시스템, 컴퓨터구조, 병렬처리

박 우 찬

1995년 : 연세대학교 대학원 (컴퓨터과학과 석사)
2000년 : 연세대학교 대학원 (컴퓨터과학과 박사-컴퓨터구조)
2001년~2003년 : 연세대학교 아식설계공동연구소 연구교수
2003년~2007년 : 세종대학교 컴퓨터공학과 조교수
2008년~현 재 : 세종대학교 컴퓨터공학과 부교수
관심분야 : 3D Graphics Processor, Real-time Ray Tracer