The Journal of GIS Association of Korea *Vol.17, No.3, pp.371-379, November 2009*

GPU를 사용한 효율적인 공간 데이터 처리[1)]

이 재 일* · 오 병 우**

An Efficient Technique for Processing of Spatial Data Using GPU

Jae-Il Lee* · Byoung-Woo Oh**

요 약

최근 그래픽 프로세서(GPU)의 발전에 따라 대량의 프로세서를 탑재한 고성능 그래픽 카드가 개인 컴퓨터에서 널리 사용되고 있다. GPU를 사용하여 CPU의 부하를 줄이면서도 성능을 향상시킬 수 있어서 복잡한 연산을 처리해야 하는 다양한 응용 프로그램에 적용하는 연구가 활발히 진행되고 있다.
본 논문에서는 복잡한 연산이 필요한 공간 데이터 처리의 성능을 향상시키기 위하여 GPU의 병렬 처리 기술을 활용하는 방법을 제안하였다. 원본 공간 데이터를 화면에 출력하기 위해서는 그래픽 처리 연산이 필요하며 같은 종류의 연산을 모든 데이터에 적용해야 하므로 GPU의 SIMD 병렬 처리를 사용하여 성능을 향상시킬 수 있다.

주요어 : GPU, 공간 데이터, 병렬 처리

ABSTRACT : Recently, GPU (Graphics Processing Unit) has been improved rapidly on the need of speed for gaming. As a result, GPU contains multiple ALU (Arithmetic Logic Unit) for parallel processing of a lot of graphics data, such as transform, ray tracing, etc.
Therefore, this paper proposed a technique for parallel processing of spatial data using GPU. Spatial data consists of multiple coordinates, and each coordinate contains value of x and y axis. To display spatial data graphics operations have to be processed to large amount of coordinates. Because the graphics operation is identical and coordinates are multiple data, SIMD (Single Instruction Multiple Data) parallel processing of GPU can be used for processing of spatial data to improve performance.
This paper implemented SIMD parallel processing of spatial data using two kinds of SDK (Software Development Kit). CUDA and ATI Stream are used for NVIDIA and ATI GPU respectively. Experiments that measure time of calculation for graphics operations are carried out to observe enhancement of performance. Experimental result is reported that proposed method can enhance performance up to 1,162% for graphics operations. The proposed method that uses parallel processing with GPU for spatial data can be generally used to enhance performance for applications which deal with large amount of spatial data.

Keywords : GPU, Spatial data, Parallel processing

1. 서 론

최근 그래픽 프로세서(GPU : Graphics Processing Unit)의 발전에 따라 대량의 프로세서를 탑재한 고성능 그래픽 카드가 개인 컴퓨터에서 널리 사용되고 있다. 고성능 그래픽 카드 제조 회사에서는 GPU를 사용한 효율적인 병렬 처리가 가능하도록 GPU 라이브러리를 제공하고 있는 추세이다(AMD, 2009; NVIDIA, 2009).

GPU 라이브러리를 통해 CPU의 부하를 줄이면서도

*금오공과대학교 컴퓨터공학과 석사과정(jaeil@kumoh.ac.kr)
**교신저자, 금오공과대학교 컴퓨터공학부 교수(bwoh@kumoh.ac.kr)
1) 본 연구는 금오공과대학교 학술연구비에 의하여 연구된 논문임.

GPU의 병렬 처리를 프로그래밍에 이용할 수 있게 되어 복잡한 연산을 처리해야 하는 다양한 응용 프로그램에 적용하는 연구가 활발히 진행되고 있다(Buck et al., 2004; John et al., 2007; Halfhill, 2008; Jang et al., 2009).

현재 개인 컴퓨터에서 사용하는 그래픽 카드는 주로 게임 등에서 활용하고 있고 일반적인 응용 프로그램에서는 활용하지 못하는 실정이다. 본 논문에서는 공간 데이터 연산을 처리하기 위하여 그래픽 카드를 사용하는 방법을 제안하였다.

공간 데이터는 일반적으로 실수 좌표로 구성되어 있어서 기하 연산이 필요하며, 복잡한 지형을 표현하므로 일반적으로 대용량 데이터이다.

복잡한 공간 데이터를 효율적으로 처리하기 위한 다양한 방법이 연구되고 있다. 공간 데이터 처리를 위한 성능 향상을 위해서는 공간 인덱스를 구성하여 처리하는 데이터를 필터링하는 방법(오병우, 2008) 및 경량화하는 방법(윤근정·김혜영외, 2008)에 대해 주로 연구되고 있다. 필터링 및 경량화된 데이터가 대용량일 때는 여전히 기존의 CPU만으로 처리하기에는 복잡하므로 공간 데이터 자체를 효율적으로 처리하기 위한 방법에 대한 연구가 필요하다.

원본 공간 데이터를 화면에 출력하기 위해서는 실수 좌표에 대해 다양한 기하 연산 처리가 필요하다. 기하 연산은 모든 공간 데이터에 대하여 동일하게 적용하므로 병렬 처리를 활용하는 것이 가능하다. 특히, 여러 데이터에 대해 단일 연산을 병렬로 처리하기 위한 SIMD (Single Instruction Multiple Data) 병렬 처리 방법이 유효하므로 현재까지는 연구 사례가 없지만 GPU를 공간 데이터 처리에 적용하는 것이 가능할 것으로 판단되었다.

본 논문에서는 기존의 CPU만을 사용하여 공간 데이터를 처리하던 방법의 성능을 향상시키기 위하여 GPU의 병렬 처리 방법을 제안하였다. 또한, 병렬 처리는 동시에 실행되는 스레드의 개수가 성능 향상 정도에 영향을 주므로 스레드 개수에 따른 성능을 실험하였다.

본 논문에서 제안한 방법의 성능 향상 정도를 검증하기 위해서는 NVIDIA의 CUDA(Compute Unified Device Architecture) 및 AMD의 ATI 스트림 라이브러리를 사용하여 CPU만으로 처리하는 방법과 비교하였다.

본 논문의 구성은 다음과 같다. 2장에서는 GPU를 활용한 관련 연구에 대해서 살펴보며, 3장에서는 본 논문에서 제안하는 GPU를 사용한 공간 데이터 처리 방법에 대해서 설명한다. 4장에서는 CPU 및 GPU의 성능 향상 정도를 분석한다. 마지막으로, 5장에서는 논문의 결론을 맺는다.

2. 관련 연구

본 장에서는 GPU를 활용한 다양한 연구사례를 살펴보며, GPU의 구조 및 GPU를 사용할 수 있도록 해주는 라이브러리에 대해 살펴본다. ATI를 인수한 AMD와 NVIDIA가 현재 그래픽 카드 시장의 대부분을 장악하고 있으므로 두 가지 그래픽 카드에 내장된 GPU에 대해 살펴본다.

최근 그래픽 카드에서 연산 처리를 담당하는 GPU는 화려한 그래픽 효과를 위해서 많은 수의 내부 프로세서(ALU: Arithmetic Logic Unit)를 탑재하고 있으며, SIMD 병렬 처리 구조로 구성되어있다(Lefohn et al., 2005).

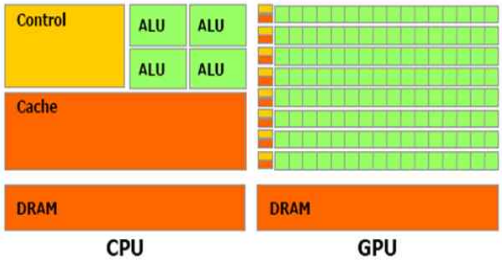
SIMD를 통하여 동일한 계산을 다수의 데이터에 대하여 수행하는 경우 성능 향상이 가능하므로 다양한 분야에서 GPU를 사용한 병렬 처리 기술이 연구되고 있다. 알고리즘분야에서는 Quicksort 알고리즘을 GPU를 사용하여 고성능의 sorting을 연구하고 있으며(Cederman and Tsigas, 2009), 물리학에서는 분자에 대한 시뮬레이션을 연구가 이루어지고 있다(Molemaker et al., 2008).

영상처리분야에서는 이미지 프로세싱과 동영상 코딩을 위해 GPU를 사용한 연구가 진행되고 있다(Bui and Brockman, 2009; Han and Zhou, 2006).

본 논문에서는 공간 데이터를 효율적으로 처리하기 위하여 GPU가 가지고 있는 다수의 프로세서를 활용하는 연구를 제안하였다.

GPU는 [그림 1]과 같이 CPU에 비해 많은 ALU(Arithmetic Logic Unit)를 가지고 있으며 그래픽 가속을 위하여 실수 연산에 특화되어 있다(NVIDIA, 2009). GPU는 많은 ALU를 가지고 있으므로 병렬 처리를 통해 수치 연산에서 우수한 성능을 발휘할 수 있다.

NVIDIA에서 제공하는 CUDA의 병렬 처리는 연산을 수행하는 가장 작은 단위인 스레드와 스레드들의 집합인 블록, 그리고 블록들의 집합인 그리드로 이루어진다. 각각의 스레드는 GPU에서 실행되는 함수인 커널을 SIMD 방식으로 병렬 처리할 수 있다.



[그림 1] CPU와 GPU의 구조

AMD(ATI)에서는 GPU 병렬 처리 기술인 ATI 스트림을 제공한다. 스트림에서는 CUDA와는 달리 계층 구조가 없고 동일 레벨의 스레드만을 구성한다. 스트림에서는 스레드를 2차원으로 구성할 수 있으며, 각 차원당 8,192개까지 스레드를 구성할 수 있다. 본 논문에서는 스트림 라이브러리 중에서 brook+를 사용하였다(Buck et al., 2004).

3. 공간 데이터 처리

3.1 좌표 데이터 처리 방법

본 장에서는 본 논문에서 제안하는 GPU를 사용한 공간 데이터 처리 방법에 대해 설명한다.

공간 데이터를 화면에 출력하기 위해서는 원본 공간 데이터를 디스크로부터 메모리로 로딩하고 사용자의 요구에 따라 패닝(panning), 확대/축소(zooming), 방향 전환(orientation) 처리를 수행하면서 지도를 디스플레이 장치에 출력한다. 공간 데이터를 로딩하는 단계와 디스플레이 장치에 출력하는 단계는 공통적으로 필요한 부분이므로 성능 향상 측면에서 가장 중요한 단계는 패닝, 확대/축소, 방향 전환을 수행하는공간 데이터의 좌표 처리 단계이다.

공간 데이터의 좌표 처리 단계에서 패닝은 그래픽 변환(transform) 연산 중에서 이동(translation)에 해당하고, 확대/축소는 신축(scale)에 해당하며, 방향 전환은 회전(rotation)에 해당한다.

그래픽 변환 연산은 출력하고자 하는 모든 공간 데이터의 각 좌표 값들에 대해 동일하게 반복적으로 수행된다. 처리할 공간 데이터의 양이 커질수록 연산의 반복 횟수가 증가되어 성능이 저하되는 요인이 된다. 특히, 공간 데이터를 구성하는 좌표는 일반적으로 부동 소수점 실수(floating point real number)로 표현되므로 CPU로 처리하기에는 성능에 한계가 있다.

본 논문에서는 CPU의 한계를 극복하고 효율적으로 공간 데이터를 처리하기 위하여 GPU의 병렬 처리를 사용하는 방법을 제안한다.

본 논문에서 공간 데이터 처리는 크게 좌표 연산과 메모리 복사로 나눈다. 좌표 연산 부분에서는 좌표에 대하여 이동, 신축, 회전의 좌표 변환 연산을 수행한다. 메모리 복사 부분에서는 계산된 좌표를 GPU의 메모리로부터 메인보드의 메모리로 복사한다. 좌표 연산 부분에서 수행하는 좌표 변환 수식은 다음과 같다.

$$P' = \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

여기서, x와 y는 기존 좌표를 나타내며 s_x와 s_y는 신축 값을, θ는 회전 각도, t_x와 t_y는 이동값을 나타낸다.

CPU는 출력하고자 하는 공간 데이터의 모든 좌표에 대해 좌표 변환 연산을 순차적으로 처리하는데 비해, GPU는 좌표 변환 연산을 다수의 ALU를 통해 스레드 개수만큼 SIMD 병렬 처리로 좌표 변환 연산을 동시에 수행하므로 성능을 향상시킬 수 있다.

GPU의 병렬 처리 표준(OpenCL)은 발표되었지만, 아직 공통적으로 활용할 수 있지 않다. 본 논문에서는 현재 그래픽 카드의 전세계 시장을 장악하고 있는 NVIDIA와 AMD(ATI)에서 제공하는 GPU 병렬 처리 라이브러리인 CUDA와 스트림 사용하여 각각 구현하고 실험한다.

3.2 CUDA 좌표 연산

NVIDIA의 GPU 라이브러리인 CUDA를 사용하기 위해서는 CPU에서 실행되는 호스트 코드와 GPU에서 실행되는 커널 코드가 필요하다.

호스트 코드는 GPU의 변수 선언과 초기화 등의 전체적인 처리와 GPU에서 사용될 커널 코드를 실행시키는 역할을 담당한다. 커널 코드는 GPU 내에서 스레드별로 동시에 실행되는 코드로서 본 논문에서는 공간 데이터 좌표 변환 연산을 처리한다.

호스트 코드에서는 좌표 연산 커널인 calc_cuda( ) 함수를 호출한다. 이 때 병렬 처리를 위하여 그리드와 블록 변수를 사용한다. 커널의 실행이 완료된 후에는 GPU의 메모리로부터 메인보드의 메모리로 연산 결과를 복사한다. [그림 2]는 CUDA의 호스트 코드를 의사(pseudo) 코드 형태로 보여준다. InData는 x, y를 갖는 행렬 구조체로서 연산대상인 좌표를 저장하고 있다. OutData는 계산된 결과값을 저장하고 있는 GPU 메모리이다.

CUDA의 커널은 스레드에서 병렬로 실행되는 코드이다. 커널 내부에서는 해당 스레드의 위치를 인식하기 위하여 인덱스를 계산한다. 인덱스는 CUDA에서 제공

```
// CUDA 호스트 코드
void process_cude( )
{
  // 좌표 변환 연산 커널 실행
  calc_cuda<<<그리드, 블록>>>(InData,
            이동, 신축, 회전 값, OutData)

  // GPU로부터 메인보드 메모리로 결과 값 복사
  cudaMemcpy(메인보드 메모리, OutData)
}
```

[그림 2] CUDA 호스트 코드

```
// CUDA 커널
void calc_cuda( )
{
  // 스레드 인덱스 계산
  i = blockIdx.x * blockDim.x + threadIdx.x
  j = blockIdx.y * blockDim.y + threadIdx.y
  index = I + j * 스레드 크기

  // 좌표 변환 연산
  OutData[index].x = InData[index].x 좌표의 변환
  OutData[index].y = InData[index].y 좌표의 변환
}
```

[그림 3] CUDA 커널 코드

되는 내장 멤버 변수인 blockIdx, blockDim, threadIdx를 사용하여 구하며, 인덱스를 통하여 해당 위치의 좌표 데이터에 접근하여 이동, 신축, 회전으로 구성되는 좌표 변환 연산을 수행한다. [그림 3]은 CUDA 커널 코드를 보여준다.

3.3 스트림 좌표 연산

ATI 스트림의 수행 루틴도 CUDA와 마찬가지로 호스트 코드와 GPU에서 실행되는 커널 코드로 구성된다. 본 논문에서는 brook+를 사용하여 구현한다.

호스트 코드에서는 좌표 연산 커널인 calc_stream( ) 함수를 호출한다. 커널의 실행이 완료된 후에는 GPU의 메모리로부터 메인보드의 메모리로 연산 결과를 복사한다.

[그림 4]는 스트림의 호스트 코드를 보여준다.

```
// 스트림 호스트 코드
void process_stream( )
{
  // 좌표 변환 연산 커널 실행
  calc_stream(*InData, 이동, 신축, 회전 값,
              *OutData)

  // GPU로부터 메인보드 메모리로 결과 값 복사
  OutData->write(메인보드의 메모리)
}
```

[그림 4] 스트림 호스트 코드

```
// 스트림 커널
kernel void calc_stream(input< >, ..., output< >)
{
  // 좌표 변환 연산
  output.x = input.x 좌표의 변환
  output.y = input.y 좌표의 변환
}
```

[그림 5] 스트림 커널 코드

스트림에서는 그리드, 블록과 같은 계층 구조가 없고 스레드만으로 구성되므로 CUDA와는 다른 방식으로 작동한다. CUDA에서는 스레드의 위치를 인식하기 위해서 내장 멤버 변수를 사용하지만 스트림에서는 커널 함수가 호출될 때 매개변수로 특정 스레드에 해당하는 데이터만이 전달된다. 이를 위해서 brook+에서는 <> 연산자를 사용한다.

input<>과 output<>은 각각 InData와 OutData로부터 해당 스레드에 대한 데이터만을 전달받으므로 인덱스 계산 없이 해당 데이터만을 접근할 수 있다. [그림 5]는 스트림 커널 코드를 보여준다.

4. 성능 평가

4.1 성능 평가 개요

본 장에서는 본 논문에서 제안한 GPU를 사용한 공간 데이터 처리 방법의 성능 향상 정도를 측정하기 위하여 성능 평가를 수행하였다. GPU를 사용할 경우에는 병렬 처리의 특성상 블록이나 스레드 개수의 구성에 따라 성능에 차이가 있으므로 최적화를 위한 실험이 필요하다. 또한, GPU의 메모리와 메인보드 메모리간의 복사도 고려해야 한다. 성능 평가를 위한 개발 환경은 <표 1>과 같다.

대용량 좌표 연산 실험에 사용되는 공간 데이터는 난수를 사용하여 임의의 좌표를 대량으로 생성하는 방법과 실제 좌표 데이터로 이루어진 공간 데이터를 사용하는 방법이 있다. 본 논문에서는 연산의 결과를 확인하기 위한 가시성을 위하여 실제 좌표 데이터를 샘플 데이터로 활용하였다.

<표 1> 개발 환경

구분	사양
운영체제	Windows XP Pro SP3
	Vista Ultimate K
CPU	Intel Core2 Duo E8400 Wolfdale(3.0GHz)
Ram	2GB
그래픽 칩셋	NVIDIA Geforce 9800GT
	NVIDIA Geforce GTX285
	ATI Radeon 3850
	ATI Radeon 4890
개발도구	Visual Studio 2008 C++ CUDA 2.2 Stream Brook+ 1.4.0

실험을 위한 공간 데이터로는 [그림 6]과 같은 서울 지역 도로 데이터를 사용하였다. 공간 데이터는 30,001개의 폴리곤 타입 객체이고 총 816,106개의(x, y) 좌표로 구성되어 있으므로 대용량 좌표 연산을 위한 실험에 적합하다.

실험은 <표 1>에서와 같이 NVIDIA사 및 ATI사의 GPU에 대해서 병렬 처리 요소인 스레드의 개수를 변화시키면서, 공간 데이터 출력에 필요한 GPU를 통한 좌표 변환 연산과 GPU 메모리로부터 메인보드 메모리로 데이터를 복사하는데 소요되는 시간을 반복 측정하여 평균값을 계산하였다. 그리고, 성능 향상 정도를 가늠하기 위해 CPU를 사용하여 공간 데이터 출력을 위한 좌표 변환 연산을 반복 수행하고 평균값을 측정하였다. 정확한 시간을 측정하기 위하여 100회 반복을 통하여 실험하였다.

실험에서는 최적의 성능 향상을 도출하기 위하여 GPU에서 동시에 실행이 되는 병렬 스레드의 개수를 6,400, 10,000, 19,600, 40,000, 57,600, 102,400, 160,000, 640,000, 2,560,000개로 변경하면서 실행 시간을 측정하였다.

[그림 6] 실험을 위한 공간 데이터

CUDA에서는 병렬 스레드가 그리드와 블록의 계층 구조로 구성되므로 블록의 크기를 5×5, 10×10, 20×20개의 스레드로 구성하고 그에 따른 그리드 크기를 계산하여 사용한다.

본 논문에서는 Windows XP 운영체제에서 세부적으로 측정된 성능을 기준으로 실험 내용을 서술하였으며, 최근 일반적으로 사용되고 있는 Vista 운영체제에서의 실험은 4.5절 결과 분석에서 GPU를 사용한 공간 데이터의 연산 성능 평가를 부가적으로 설명한다.

좌표 연산의 시간을 측정하는 방법으로 CUDA에서는 전용 컴파일러를 사용하기 때문에 CUDA 전용의 타이머인 cutStartTimer( )를 사용하였으며, 스트림 및 CPU의 시간 측정은 운영체제에서 제공하는 QueryPerformanceCounter( )를 사용하여 측정하였다.

4.2 CPU 성능 분석

실험에 사용된 PC의 CPU는 인텔의 코어2 듀오 E8400으로서 3.0GHz 클럭으로 동작하고 2개의 프로세서를 가지고 있다. 이 CPU를 사용한 실험에서 총 816,106개의 2차원 좌표를 순차적으로 변환하는데 평균 77.01ms가 소요되는 것으로 측정되었다. 이를 기준으로 그래픽 카드 및 스레드 개수에 따른 좌표 변환 연산 시간 및 메인보드 메모리로의 복사 시간의 합계를 비교한다.

4.3 CUDA 성능 분석

9800GT는 NVIDIA사의 그래픽 카드 중에서 고급형에 속하며, GTX285는 최고급형 모델급에 속하는 우수한 성능을 가진 제품이다. 측정된 결과는 <표 2>와 같다.

<표 2> CUDA를 사용한 공간 데이터 처리 평균 시간(ms) (일부 발췌)

GPU (프로세서 개수)	그리드	블럭	스레드 개수	좌표 변환 연산 시간	메모리 복사 시간	합계 시간
9800GT (112개)	16×16	5×5	6,400	6.25	7.25	13.50
	14×14	10×10	19,600	4.72	6.13	10.85
	64×64	**5×5**	**102,400**	**3.78**	**5.82**	**9.60**
	20×20	20×20	160,000	4.61	6.26	10.87
	100×100	20×20	2,560,000	11.55	11.98	23.53
GTX285 (240개)	16×16	5×5	6,400	3.96	7.38	11.34
	7×7	20×20	19,600	1.72	6.14	7.86
	16×16	**20×20**	**102,400**	**1.05**	**5.58**	**6.63**
	160×160	5×5	640,000	1.43	6.71	8.14
	100×100	20×20	2,560,000	1.49	12.58	14.07

지면 관계상 스레드 개수별로 가장 빠른 경우에 대한 블록과 그리드만을 표기하였다. 메모리 복사 시간은 비슷하지만, 스트림 프로세서가 112개인 9800GT에 비해 240개인 GTX285가 병렬 처리에 유리하여 연산 처리 능력이 뛰어난 것으로 측정되었다. 이를 통해 GPU를 사용하여 공간 데이터를 병렬로 처리하면 성능이 향상되고, GPU의 병렬 처리 능력에 따라서 향상의 정도에 차이가 있음을 알 수 있다.

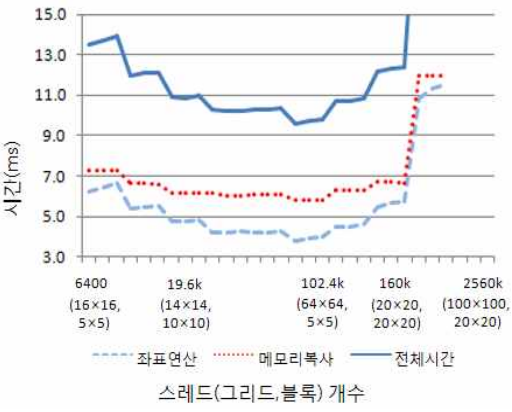
CUDA에서는 그리드와 블록을 몇 개로 구성하느냐에 따라서 성능에 영향을 준다. 이를 분석하기 위하여 스레드 개수를 증가시키면서 블록과 그리드 구성 변화에 따른 성능 변화를 살펴본다.

CUDA를 사용한 9800GT의 스레드 개수에 따른 처리 시간의 변화는 [그림 7]의 그래프와 같다. 그래프에서 보는 바와 같이 스레드 개수가 늘어나면 좌표 변환 연산 속도는 줄어들지만 메모리 복사 시간이 증가하여 합계 시간이 U자 형태로 증가하게 된다. 가장 빠른 성능을 나타내는 스레드 구성은 <표 2>와 같이 그리드를 64×64개의 블록으로 구성하고, 각 블록을 5×5개의 스레드로 구성하여 총 102,400개의 스레드를 사용하는 것으로 9.60ms가 소요되었다.

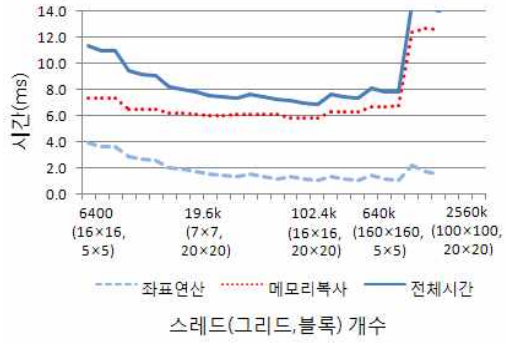
CUDA를 사용한 GTX285의 스레드 개수에 따른 공간 데이터 처리 시간의 변화는 [그림 8]의 그래프와 같다.

9800GT와 마찬가지로 U자 형태로 증가함을 알 수 있고, 16×16 그리드와 20×20 블록으로 총 102,400개의 스레드를 사용할 때 6.63ms로 가장 우수한 성능을 나타내는 것을 알 수 있다.

고가인 GTX285의 성능도 우수하지만, 저가의 9800GT 또한 공간 데이터 연산에 있어서 효율적임을 알 수 있다.



[그림 7] 9800GT 스레드별 시간



[그림 8] GTX285 스레드별 시간

4.4 스트림 성능 분석

본 논문에서는 AMD(ATI) 계열의 GPU를 위한 스트림 라이브러리를 사용하여 Radeon 3850 및 4890 그래픽 카드에 대해서 공간 데이터 처리 시간을 측정하였다. 3850은 AMD 계열의 그래픽 카드 중에서 중급형에 속하며, 4890은 고급형에 속하는 제품이다. 측정된 결과는 <표 3>과 같다.

<표 3> 스트림을 사용한 공간 데이터 처리 평균 시간(ms) (일부 발췌)

GPU(프로세서 개수)	스레드 개수	좌표 변환 연산 시간	메모리 복사 시간	합계 시간
Radeon 3850 (320개)	6,400	6.70	24.86	31.56
	19,600	2.40	20.18	22.58
	102,400	**0.49**	**16.76**	**17.25**
	640,000	0.32	25.85	26.17
	2,560,000	0.21	85.76	85.97
Radeon 4890 (800개)	6,400	6.67	28.49	35.16
	19,600	2.23	24.13	26.36
	102,400	**0.67**	**21.17**	**21.84**
	640,000	0.33	32.09	32.42
	2,560,000	0.22	105.57	105.79

Radeon GPU의 스트림 프로세서는 3850이 320개, 4890이 800개로 4890이 병렬 처리에 유리하게 생각되지만 실제 처리 시간에서는 3850과 크게 차이가 나지 않는 것으로 측정되었다. 이는 병렬 처리를 위한 스트림 프로세서가 충분히 많아서 둘 모두 1ms 보다 적은 시간이 걸렸기 때문으로 유추할 수 있다.

AMD 계열의 GPU를 NVIDIA 계열의 GPU와 비교하면 병렬로 처리하는 스레드 수가 많기 때문에 연산 속도는 매우 빠른 반면, 메모리 처리 속도가 상대적으로 느려서 9800GT 및 GTX285에 비하여 전체적인 성능이 낮은 것으로 측정되었다. 그리고, Radeon 4890은 3850의 후속 그래픽 카드이고 고가이어서 성능이 높을 것으로 기대되지만 성능 평가에서는 3850보다 오히려 낮게 측정되었다.

Radeon 3850의 스레드 개수에 따른 처리 시간의 변화는 [그림 9]의 그래프와 같다. NVIDIA 계열 GPU의 그래프와 마찬가지로 스레드가 증가할수록 좌표 연산 시간은 줄어들지만 메모리 복사 시간이 증가하여 합계 시간은 U자 형태로 증가함을 알 수 있으며, 스레드가 102,400개 인 경우에 17.25ms로 가장 성능이 우수함을 알 수 있다.

[그림 9] Radeon 3850 스레드별 시간

[그림 10] Radeon 4890 스레드별 시간

스트림을 사용한 Radeon 4890의 스레드 개수에 따른 처리 시간의 변화는 [그림 10]의 그래프와 같다. 다른 그래픽 카드와 마찬가지로 U자 형태로 증가하며, 102,400개의 스레드를 사용할 때 21.84ms로 가장 우수한 성능을 나타내는 것을 알 수 있다.

4.5 결과 분석

실험을 통해 측정된 Windows XP 및 Vista 운영체제에서의 CPU, CUDA, 스트림의 최우수 성능을 정리하면 <표 4>와 같다. 표에서 시간은 CPU의 경우에는 좌표 연산 시간이고, GPU의 경우에는 좌표 연산 시간과 GPU 메모리로부터 메인보드 메모리로 좌표를 복사하는데 추가로 소요되는 시간의 합계이다.

<표 4>의 성능은 CPU를 100%로 간주하여 성능 향상 정도를 백분율로 표시한 수치이다.

<표 4>의 (a)는 Windows XP에서 실험하였으며, 최소 353%에서 최대 1,162%까지 성능이 향상된 것으로 측정되었다. Windows XP와 더불어 일반적으로 사용되고 있는 Vista 운영체제에서의 GPU를 활용한 최대 성능은 <표 4>의 (b)와 같다. Vista 운영체제에서는 Windows XP에 비하여 CPU 및 GPU의 성능이 대략 1.6배 낮게 측정되었지만, GPU의 성능 향상 비율은 Windows XP 운영체제와 비슷하게 측정이 되었다.

[그림 11]은 <표 4>를 그래프로 표현한 것이다.

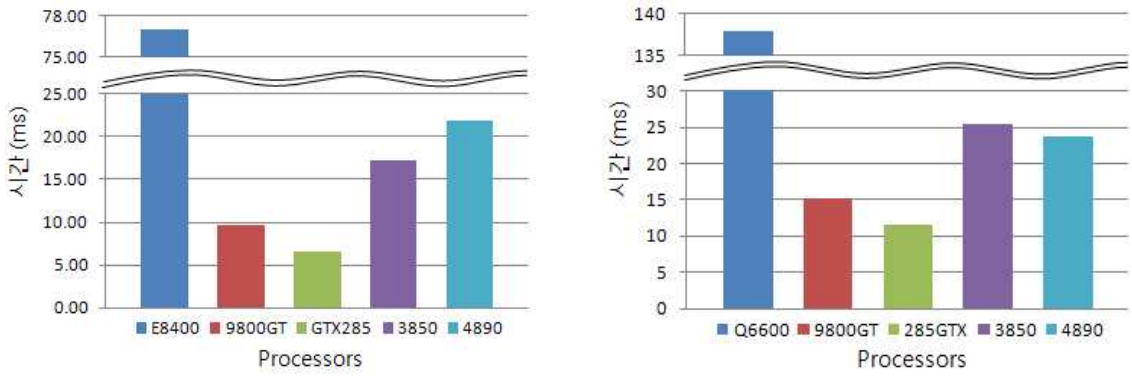
<표 4> 공간 데이터 처리 시간

프로세서	프로세서 개수	스레드	시간 (ms)	성능 (%)
E8400(CPU)	2	·	77.01	100
9800GT(CUDA)	112	102,400	9.60	802
GTX285(CUDA)	240	102,400	6.63	1,162
3850(스트림)	320	102,400	17.25	446
4890(스트림)	800	102,400	21.84	353

(a) Windows XP 운영체제

프로세서	프로세서 개수	스레드	시간 (ms)	성능 (%)
E8400(CPU)	2	·	137.86	100
9800GT(CUDA)	112	102,400	15.07	914
GTX285(CUDA)	240	102,400	11.61	1187
3850(스트림)	320	102,400	25.44	514
4890(스트림)	800	102,400	23.73	580

(b) Vista 운영체제



(a) Windows XP 운영체제 (b) Vista 운영체제

[그림 11] CPU와 GPU의 성능 비교

NVIDIA의 GTX285가 실험에 사용된 그래픽 카드 중에서 공간 데이터 처리에 가장 성능이 우수한 것으로 측정되었고, AMD 계열의 그래픽 카드는 다량의 스트림 프로세서로 GPU 연산 자체는 우수하지만 메인보드의 메모리와 데이터 전달시에 속도가 저하되는 특징이 있었다.

GTX285는 가장 고가의 그래픽 카드로 일반적으로 사용되기 힘들지만, 본 논문에서는 GPU 성능의 향상의 정도를 일반적으로 사용하는 그래픽 카드와 비교하기 위하여 성능을 평가하였다. 저가인 9800GT과 Radeon 3850, 약간 고가인 Radeon 4890은 가격면에서 GTX285에 비해 저렴하지만 CPU에 비하여 성능 향상이 있으므로, 저가의 그래픽 카드를 사용하여도 공간 데이터 좌표 연산에 있어 효율적이다. GTX285는 9800GT보다 가격이 4~6배 비싸지만 실험에 의한 성능에서는 2배에도 미치지 못하므로 가격대 성능비는 낮게 측정되었다.

GPU를 사용한 공간 데이터의 처리에서 좌표 변환 연산시 NVIDIA 계열의 그래픽 카드는 블록과 스레드의 구성에 따라 성능 향상의 정도가 다르게 나타났고, AMD 계열의 그래픽 카드는 스레드의 개수에 따라 성능 향상 정도가 다른 것으로 측정되었다.

공통적으로 병렬 처리를 위한 스레드가 증가하면 좌표 변환 연산에 걸리는 시간은 감소하지만 메인보드의 메모리로 데이터를 복사하는 시간이 증가되므로 최적의 스레드 개수를 찾는 것이 중요하다. 본 논문에서 사용된 환경에서는 스레드 개수가 102,400개인 경우 최적의 성능 효과를 발휘하는 것으로 측정되었다.

본 논문에서는 GPU를 사용한 효율적인 공간 데이터 처리 방법을 제안하였고, 실험을 통해 기존의 방법보다 최대 1,162%까지 성능을 향상시킬 수 있음을 증명하였다. GPU를 응용 프로그램에 사용하여 응용 프로그램의 성능 향상을 도모하는 방법이 공간 데이터 처리에도 적용됨을 알 수 있다.

본 논문에서 실험한 결과는 GPU를 공간 데이터 처리에 활용한 성능이며 실험에 사용된 그래픽 칩셋의 본래 목적인 그래픽 성능과는 무관하다.

5. 결 론

일반적으로 공간 데이터는 복잡한 지형을 표현하므로 대용량이며, 좌표를 처리하기 위하여 동일한 좌표 연산이 수행된다. 여러 데이터에 대해 단일 연산을 처리하기 해서는 병렬 처리 방법이 유효하며, 본 논문에서는 병렬 처리를 위한 GPU 라이브러리를 사용하여 효율적으로 공간 데이터 처리 방법을 제안하였다. 현재 고성능의 그래픽 카드가 일반 PC에 범용적으로 사용되고 있는 상황에서 GPU를 활용한 병렬 공간 데이터 처리는 성능을 현저하게 향상시킬 수 있는 중요한 기술이다.

CPU도 2개 또는 4개의 ALU를 구성하여 병렬 처리를 지원하고 있는 추세이지만, GPU의 100개 이상의 부동소수점 실수 전용 ALU와는 성능에서 차이가 있다.

그래픽 카드의 성능 및 공간 데이터의 용량에 따라 다소 차이는 있겠지만 실험에 사용된 그래픽 카드로는 최대 1,162%까지 성능이 향상된 것으로 측정되었다. 이를 통해 CPU만으로 공간 데이터를 처리하는 것보다 GPU를 사용하여 처리하는 것이 확연히 효율적임을 성능 평가를 통해 확인할 수 있었다.

성능 평가를 통하여 대용량의 공간 데이터를 처리하기 위한 연산을 일반적으로 사용되는 그래픽 카드에 탑재된 GPU를 통해 병렬로 처리할 수 있음을 입증하였다. GPU를 사용하여 병렬로 대용량의 공간 데이터를 처리

하면 메모리 복사 등의 추가 시간에도 불구하고 일반적으로 CPU를 사용하는 것보다 현저히 성능이 향상됨을 확인하였으며, 공간 데이터를 화면에 출력할 때 활용할 수 있음을 실험을 통해 증명하였다.

향후 연구 과제로는 현재 본 논문에서 공간 데이터에 대한 연산 결과를 화면에 렌더링할 때 운영체제의 시스템 함수를 사용하는 방법을 개선하는 연구 즉, GPU를 통해 계산된 좌표 변환 결과를 메인보드의 메모리로 전송하는 시간을 줄이기 위하여 DirectX 또는 OpenGL과 같은 그래픽 API를 사용하여 그래픽 카드 내에서 직접 렌더링함으로써 출력의 성능도 함께 향상시키는 연구가 가능하다.

참고문헌

오병우, 2008, "모바일 소프트웨어를 위한 효율적인 공간 인덱스", 한국GIS학회지, 16(1), pp.113-127.

윤근정·김혜영·전철민, 2008, "모바일 GIS를 위한 벡터 데이터 경량화 기법", 한국GIS학회지, 16(2), pp.207-218.

AMD, 2009, ATI Stream Computing User Guide ver 1.4.0.

Buck, I., T., Foley., D., Horn, J., Sugerman, K., Fatahalian., M., Houston, P., Hanrahan, 2004, "Brook for GPUs: Stream Computing on Graphics Hardware", In Proceedings of International Conference on Computer Graphics and Interactive Techniques SIGGRAPH 2004 Aug, pp.777-786.

Bui, P., J., Brockman, 2009, "Performance Analysis of Accelerated Image Registration Using GPGPU", ACM International Conference Proceeding Series, 383, pp.38-45.

Cederman, D., P., Tsigas, 2009, "GPU-Quicksort: A Practical Quicksort Algorithm for Graphics Processors", Journal of Experimental Algorithmics, 14, pp.1-24.

Halfhill, T.R., 2008, Parallel Processing with CUDA, Microprocessor Report Volume 22.

Lefohn, A., J., Kniss., J., Owens, 2005, "Implementing Efficient Parallel Data Structure on GPUs", GPU Gems2, Addison Wesley, pp.521-545.

Jang, B., S., Do, H., Pien, D., Kaeli, 2009, "Architecture-aware Optimization Targeting Multithreaded Stream Computing", ACM International Conference Proceeding Series, 384, pp.62-70.

Mark, W.R., R.S., Glanville, K., Akeley, M.J., Kilgard, 2003, "Cg: a System for Programming Graphics Hardware in a C-like Language", ACM SIGGRAPH, pp.896-907.

Nickolls, J., I., Buck, M., Garland, K., Skadron, 2008, Scalable Parallel Programming with CUDA, ACM Queue.

NVIDIA, 2009, CUDA Programming Guide ver 2.2.

John, D.O., D., Luebke, N., Govindaraju, 2007, "A Servey of Gerneral-Purpose Computation on Graphics Hardware", Computer Graphics Forum, 26, pp.80-113.

Stratton, J.A., S.S., Sam, W.W., Hwu, 2008, "MCUDA: An Efficient Implementation of CUDA Kernels for Multi-core CPUs", Languages and Compilers for Parallel Computing, pp.16-30.

Molemaker, J., J.M., Cohen, S., Patel, J., Noh, 2008, "Low Viscosity Flow Simulations for Animation", SCA 2008, pp.9-18.

Han, B., B., Zhou, 2006, "Efficient Video Decoding on GPUs by Point Based Rendering", ACM SIGGRAPH/ EUROGRAPHICS Symposium on Graphics Hardware, pp.79-86.

접수일 (2009년 10월 22일)
최종수정일 (2009년 11월 24일)
게재확정일 (2009년 11월 24일)