JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 18, No. 5 : 1177~1186 May. 2014

다중 GPGPU를 이용한 컴퓨터 생성 홀로그램의 병렬화 구현

서영호[1*] · 이윤혁[2] · 김동욱[2]

Implementation of Parallel Computer Generated Hologram Using Multi-GPGPU

Young-Ho Seo[1*] · Yoon-Hyuk Lee[2] · Dong-Wook Kim[2]

[1*]College of Liberal Arts, Kwangwoon University, Seoul 139-701, Korea
[2]Department of Electronic Materials Engineering, Kwangwoon University, Seoul 139-701, Korea

요 약

컴퓨터생성홀로그램은 수학적으로 모델링된 광학적인 현상을 컴퓨터로 연산한 것이다. 이때 방대한 량의 연산이 필요하기 때문에 실시간으로 고해상도의 홀로그램을 얻기 위해서는 고속 기법이 필요하다. 본 논문에서는 CGH를 위한 두 가지 병렬화를 제안한다. 첫 번째는 GPU 내에서 CGH 알고리즘을 병렬화하는 것이고, 두 번째는 다수의 GPU를 위한 병렬화이다. 제안한 알고리즘 구조는 CUDA를 이용하여 GTX780 Ti GPU에 구현하였다. 약 10K의 입체 정보를 이용하여 1,024×1,024의 컬러 홀로그램을 생성하는데 약 106ms가 소요된다.

ABSTRACT

Computer-generated hologram (CGH) is to mathematically model optical phenomenon with digital computer. Because it requires huge amount of computational power, a fast and high performance technique is needed. In this paper, we proposed two parallelizations for CGH calculation. The first is to parallelize CGH algorithm in a GPU (general processing unit) and the second is to parallelize multiple GPUs. The proposed algorithm was implemented in GTX780 Ti GPU. It calculates a 1,024×1,024 hologram with 10K object points for about 24ms.

키워드 : 디지털 홀로그램, 범용 그래픽 프로세싱 유닛, 컴퓨터 생성 홀로그램, 실시간 동작, 병렬화

Key word : Digital hologram, General purpose graphic processing unit (GPGPU), Computer generated hologram, Real-time operation, Parallelization

접수일자 : 2014. 04. 12 심사완료일자 : 2014. 05. 02 게재확정일자 : 2014. 05. 07

* **Corresponding Author** Young-Ho Seo(E-mail:dwkim@kw.ac.kr, Tel:+82-2-940-5167)
College of Liberal Arts, Kwangwoon University, Seoul 139-701, Korea

Open Access http://dx.doi.org/10.6109/jkiice.2014.18.5.1177 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

CGH를 이용하여 하나의 홀로그램을 생성하기 위해서는 많은 연산량이 필요하다[1]. 고속의 홀로그램 생성을 위한 연구는 크게 두 가지로 분류된다. 그중 하나로는 FPGA를 이용한 하드웨어로의 구현[2-6]한 것이고, 다른 한 가지는 GPU를 이용한 병렬 프로그램을 이용한 하는 것이다[7-12].

FPGA를 이용한 방법으로는 CGH를 연산하기 위한 전용 연산 시스템인 HORN-6가 연구되었다[3]. 100% 파이프라인 구조를 기반으로 하는 CGH 프로세서에 대한 연구도 진행되었다[4]. 이 연구에서는 프레넬 변환을 수행하기 위한 CGH 셀의 하드웨어 구조를 제안한 후에 이를 확장하여 CGH 커널을 구성하였고, 이를 다시 확장하여 CGH 프로세서를 구현하였다. [4]의 하드웨어는 [2]보다 최대 87.32%의 높은 성능을 갖는다. [5]의 논문에서 제안된 FPGA 기반의 하드웨어는 1,920×1,080 크기의 HD급의 홀로그램을 실시간으로 생성할 수 있다. [6]의 논문에서는 메모리 접근을 줄일 수 있는 하드웨어 구조 및 동작 방식을 제안하여 [5]에서 제안된 하드웨어에 비해서 메모리 접근 횟수를 약 2,000배 감소시켰다.

GPU를 이용한 방법으로 싱가폴대[9]는 CGH 수식을 복소 형태로 변환한 후에 연산을 분리하는 알고리즘을 제안하였다. 분리된 항을 각각 룩업 테이블(look-up table, LUT)로 만든 후에 연산을 고속화시키고, GPU로 구현하였다. 1K개의 광원으로 구성된 객체에 대해 1,024×768크기의 홀로그램을 0.3초당 한 장씩 생성할 수 있었다. 중국 Zhongshan 대학의 Wang[10]은 GPU를 이용하여 3D 메쉬 모델 기반의 CGH를 생성하였다. 또한 일본 Chiba대의 Shimobaba[11]는 AMD의 GPU를 기반의 GPU 프로그래밍 기법을 활용하여 CGH 생성을 고속화하였고, HD크기의 홀로그램을 0.31초당 한 장씩 생성할 수 있었다. 광운대는 광원 및 홀로그램의 블록화를 통해서 GPU를 최적으로 이용할 수 있는 연구와 GPU 내부의 다양한 메모리의 사용에 따른 CGH의 효율성을 분석하여 성능을 향상시키고자 시도하였다[12]. 최근에는 OpenMP 기반의 CPU 병렬화 기법을 이용하여 다수의 GPU를 병렬적으로 동작시켜서 CGH의 성능을 향상시키는 연구들이 수행되어 오고 있다. 한양대[13]에서는 OpenMP를 활용한 두 개의 GPU를 이용하여 병렬 동작의 최적화 방법을 제안하여 CPU대비 9,700배의 성능 개선을 할 수 있었다. 광운대[14]에서는 외부 확장 장치를 이용하여 4개의 고성능 GPU를 병렬화하여 컬러 홀로그램을 고속으로 생성하는 시스템을 제안하고 구현하였다. 본 논문에서는 고속의 CGH연산을 GPU를 이용할 때 적용시킬 수 있는 성능의 개선 방법과 GPU의 자원에 할당에 따른 최적의 연산방법에 대해 논의한다.

Ⅱ. GPGPU구조 및 동작

2.1. GPGPU의 구조

고성능 제어기와 적은 ALU(Arithmetic and Logic Unit)을 가지는 CPU의 구조와는 달리 GPGPU는 그림 1과 같이 작은 컨트롤 유닛과 다수의 ALU를 포함하고 있어 연산 능력에서 병렬화에 유리한 구조를 가진다. 그림 1과 같이 하나의 GPU내에는 하나의 전역메모리와 여러 개의 SM(streaming multiprocessor) 및 적재/저장 유닛 등을 가지고 있다. 하나의 SM은 여러 개의 SP(streaming processor)와 공유메모리 등으로 이루어져 있고, SP는 하나 혹은 다수의 쓰레드를 동작을 수행할 수 있다. GPU내의 전역메모리는 저장할 수 있는 데이터량은 크지만 여러 SM이 동시에 접근하기 어렵고 대기지연 시간도 비교적 길기 때문에 비교적 낮은 성능을 갖는다. 하지만 캐시 형태로 되어 있는 상수메모리(Constant Memory)나 SM 내부의 공유메모리는 저장할 수 있는 데이터량은 적지만 전역메모리에 비해서 접근 속도가 빠르다. 따라서 많이 호출되어 사용되는 변수들은 상수 메모리나 공유 메모리를 사용하는 것이 유리하다[15].

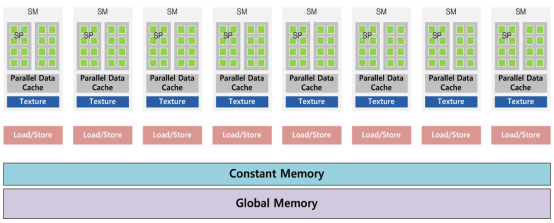

그림 1. GPGPU의 구조
Fig. 1 Structure of GPGPU

2.2. GPGPU의 동작

본 논문에서 사용한 병렬 프로그래밍 언어인 CUDA는 쓰레드와 블록, 및 그리드 단위로 GPU내의 연산 단위를 구분한다. 그림 2와 같이 하나의 블록은 3차원의 쓰레드로 구성할 수 있고, 하나의 그리드는 3차원의 블록으로 구성할 수 있다. 각 쓰레드는 GPU내의 SP에 맵핑되며 연산하는 최소 단위로 하나의 블록 내의 최대 쓰레드의 개수는 GPU의 성능에 따라 결정된다. 또한 블록 내에 구성된 쓰레드는 인덱스를 가지고, 그림 3과 같이 워프(warp) 단위로 실행된다. 따라서 모든 쓰레드를 효율적으로 동작시키기 위해서는 블록 당 쓰레드의 개수를 워프의 배수로 하는 것이 유리하다[15].

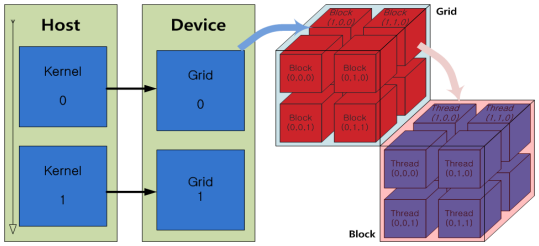

그림 2. CUDA의 연산 단위
Fig. 2 Calculation unit of CUDA

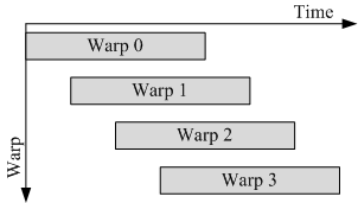

그림 3. 워프의 실행 스케줄
Fig. 3 Executing schedule of warps

각 블록은 블록내의 쓰레드에 따라서 하나 혹은 그 이상의 블록이 하나의 SM으로 맵핑되어 동작한다. 블록내의 모든 쓰레드 내에 할당된 레지스터의 개수가 SM내의 총 레지스터의 개수이하로 될 경우 최대의 블록으로 할당할 수 있다. 또한 블록도 쓰레드와 같이 인덱스를 가지게 되며 모든 SM에 할당된 블록의 개수가 그리드당 총 블록의 수보다 적을 경우 모든 할당된 블록의 동작이 끝난 뒤 남은 블록들이 동작이 이루어진다.

Ⅲ. CGH 알고리즘의 병렬화

3.1. CGH 알고리즘

홀로그램은 광학계를 이용하여도 취득할 수 있지만 광학계 자체를 수학적으로 모델링한 연산에 의해서 구할 수도 있다. 이러한 수학적인 연산을 통해 얻어진 홀로그램을 컴퓨터 생성 홀로그램(computer-generated hologram, CGH)이라고 한다[12,14].

CGH는 식 (1)과 같이 정의되는데 홀로그램의 위상으로부터 홀로그램의 강도(I_α)를 얻는 방법이다. 여기서 N은 3차원 객체의 광원수를 뜻한다. k는 참조파의 파수로 $2\pi/\lambda$로 정의되고 λ는 사용된 파의 파장을 나타낸다. x_α와 y_α는 홀로그램내의 위치를 뜻하고 x_j, y_j, 및 z_j는 3차원 객체의 위치를 나타낸다. a_j는 객체의 밝기 정보를 나타낸다. p는 홀로그램 평면에서 하나의 화소의 크기를 나타낸다.

$$I_\alpha = \sum_j^N a_j \exp\left[-j\,k\sqrt{(px_\alpha - px_j)^2 + (py_\alpha - py_j)^2 + z_j^2}\right] \quad (1)$$

$x_{\alpha j}$와 $y_{\alpha j}$는 각각 $x_{\alpha j} = x_\alpha - x_j$ 및 $y_{\alpha j} = y_\alpha - y_j$로 정의한다. 식 (1)에서 $|px_{\alpha j}, py_{\alpha j}| \ll z_j$의 조건인 경우에 Fresnel 근사를 통해서 식 (2)와 같이 근사될 수 있다.

$$I_\alpha = \sum_j^N a_j \exp\left[-j2\pi\left(\frac{\lambda}{z_j} + \frac{p^2}{2\lambda z_j}(x_{\alpha j}^2 + y_{\alpha j}^2)\right)\right] \quad (2)$$

3.2. CGH 병렬화

식 (2)와 같이 CGH 수식은 방대한 반복 연산이 이루어진다. 따라서 GPU를 이용하여 병렬 연산을 수행할 경우 속도를 향상 시킬 수 있다. 그림 5에 식 (2)를 이용하여 CGH 연산 방식에 대하여 나타내었다. 그림 5(a)는 하나의 홀로그램 화소를 모든 유효 광원에 대하여 병렬로 연산하는 방법으로 연산 결과를 누적 덧셈이 필요하다. 따라서 연산 결과를 다시 메모리에 저장해야 하기 때문에 메모리 접근 횟수가 늘어나게 되어 속도가 저하된다. 또한 많은 메모리 접근으로 인하여 메모리 병목현상이 생기게 된다. 그림 5(b)는 유효광원을 모든 홀로그램 화소에 대하여 병렬로 연산하는 방식으로 GPU의 하나의 레지스터에 모든 입력광원에 대하여 연산이 끝날 때까지 누적할 수 있기 때문에 메모리 접근이 줄게

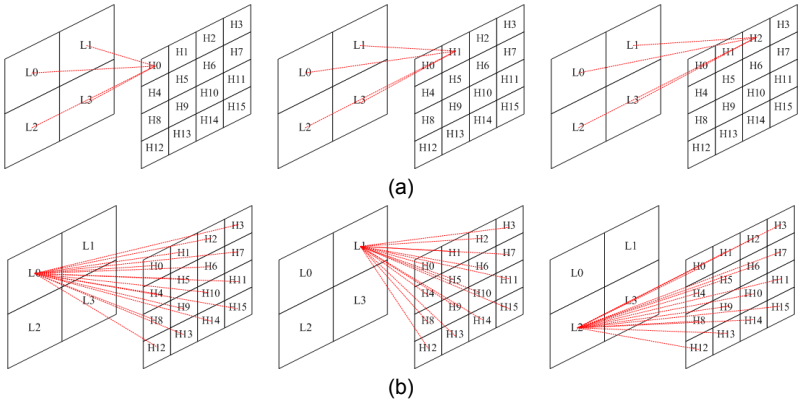

그림 4. CGH 연산 방식 (a) 광원의 병렬화 (b) 홀로그램 화소의 병렬화
Fig. 4 CGH calculation method (a) parallization of light sources (b) parallization of holographic pixels

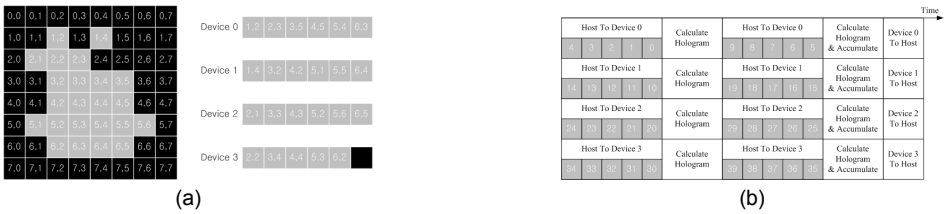

그림 5. 광원의 타일링 (a) GPU에 개수에 따른 타일링 (b) 메모리 크기에 따른 버퍼의 타일링
Fig. 5 Tiling of light sources (a) by the number of GPU (b) by memory size

된다. 따라서 모든 GPU의 쓰레드는 홀로그램 화소에 대하여 맵핑되어 연산을 하는 것이 유리하다.

식 (2)에서 입력되는 광원이 z_j값에 대한 변수 λ/z_j와 $p^2/2\lambda z_j$는 256레벨의 데이터 이므로 LUT (Look-up table) 형태로 구현 할 수 있다. 이 값에 대한 데이터는 화소의 크기 또는 입력 광원의 파장과 256레벨에 맵핑된 z_j값이 변하지 않을 경우 항상 같기 때문에 초기 한 번만 GPU에 메모리에 적재 시켜놓으면 된다. LUT의 데이터는 입력광원의 z_j값에 따라서 반복적으로 접근해야 한다. 따라서 GPU 커널이 개시되면 전역 메모리에 저장된 256개의 데이터를 접근 속도가 가장 빠른 공유 메모리에 적재 시킨다[12].

3.3. 물체 정보의 분리

식 (2)에서 홀로그램의 화소를 계산하기 위해서는 광원과 홀로그램 화소간의 위치에 관한 항과 광원의 세기의 곱을 누적덧셈을 통하여 구한다. 하지만 화소의 세기가 '0'일 경우 위치에 관한 연산에 독립적으로 누적값이 '0'이므로 이는 유효하지 않은 광원이 된다. 따라서 유효 광원만 분리하여 GPU에 데이터를 전송하고 연산함으로써 연산 속도를 향상 시킬 수 있다[12].

홀로그램 한 프레임에 대하여 연산할 때 광원에 대한 데이터는 많은 접근을 한다. 따라서 전역 메모리 보다는 캐시 형태로 되어있는 상수 메모리(constant memory) 또는 공유메모리(shared memory)를 통한 데이터 전송을 통하여 속도를 향상 시킬 수 있다. 접근 속도가 가장 빠른 것은 공유 메모리에 대한 접근이지만 광원에 대한 데이터는 각 쓰레드에서 한번만 접근하기 때문에 전역 메모리에서 한번 적재시켜 반복적으로 사용되는 공유 메모리의 사용은 오히려 전역 메모리에서 한번 적재시켜 연산하는 방법에 비하여 느려진다. 따라서 호스트에서 상수 메모리로 데이터를 전송하고 상수 메모리에서 한번만 적재하여 연산하는 방법을 사용하는 것이 속도 향상을 시킬 수 있다. 하지만 상수 메모리 또한 저장할 수 있는 데이터 량이 제한되어 있기 때문에 메모리 크

기와 GPU의 개수에 따라 유효 광원을 타일링하여 전송해야한다. 그림 5에서 입력 광원에서 유효광원을 타일링하는 방법에 대하여 나타내었다. 그림 5(a)는 유효 광원과 GPU의 개수에 따른 타일링 방법으로 광원의 세기가 '0'이 아닌 광원의 데이터를 GPU의 ID에 따라 나누어 준다. 그림 5(b)는 상수 메모리의 크기 또는 연산 속도에 따라 해당 ID의 광원 버퍼를 타일링하는 방법이다. 표 1 GPU의 메모리 종류에 따라 1K의 유효 광원에 따른 1024×]024의 홀로그램을 생성할 때의 시간을 측정한 결과이다. 초기 파라미터 LUT의 계산 시간과 데이터 전송시간을 제외한 20프레임의 홀로그램을 생성하였을 때 평균 시간에 해당한다. LUT와 광원의 메모리를 모두 상수 메모리에 저장을 하고 연산하는 방법이 31.137ms로 가장 효율 적으로 연산하는 결과를 보였다 [12].

표 1. 메모리 종류에 따른 평균 연산 시간
Table. 1 Average calculating times to memory types

Access Memory Test		
LUT	Object	Average Time (ms)
Global	Global	34.46924
Shared	Global	33.46924
Constant	Global	34.87168
Global	Shared	34.96569
Shared	Shared	33.41839
Constant	Shared	35.01628
Global	Constant	32.74693
Shared	Constant	31.68068
Constant	Constant	31.13706

3.4. 홀로그램의 분리

앞 절에서 홀로그램의 화소에 대하여 쓰레드를 맵핑하는 것이 유리하다는 것을 보였다. 하나의 쓰레드에서는 최소 상수 메모리에서 읽어온 데이터를 저장할 4개(x_j,y_j,z_j,a_j)의 레지스터와 홀로그램의 화소의 좌표(x_α,y_α)의 2개, 홀로그램 화소의 누적 덧셈의 결과($\mathcal{R}(I_\alpha)$,$\mathfrak{I}(I_\alpha)$)의 2개, 메모리 접근을 위한 주소의 1개의 레지스터 그리고 중간 값을 저장할 1개의 레지스터가 필요하므로 총 10개 이상의 레지스터가 필요하다. 만약 식 (3)의 두 조건을 만족할 경우 하나의 쓰레드에서 $n\times m$개의 홀로그램 화소를 생성할 수 있다. $N_{Reg/Thread}(Reg)$는 쓰레드당 레지스터이고 $N_{Totalthread}(Thread)$은 동시에 실행할 수 있는 총 쓰레드의 개수로 SM당 할당한 쓰레드와 GPU내의 총 SM의 개수의 곱으로 나타낼 수 있다. N은 생성할 홀로그램의 너비이고 M은 높이이다. 하나의 쓰레드에서 여러 개의 홀로그램 화소를 생성할 경우 좌표 항에 대하여 레지스터를 공유할 수 있어 속도를 향상시킬 수 있다. 하지만 한 번에 사용 가능한 쓰레드 보다 타일링된 홀로그램의 블록의 개수가 적을 경우 모든 자원을 쓰지 않는 현상이 생기므로 최소 타일링 개수는 최대 사용 쓰레드 이상으로 해야 한다[12].

$$N_{Reg/Threads}(Reg) > 8+(n+m)+2(n\times m) \qquad (3)$$
$$N_{Totalthread}(thread) < \lceil N/n \rceil \times \lceil M/m \rceil$$

그림 6에 쓰레드당 맵핑되는 홀로그램 화소를 나타내었다. 그림 6(a)는 각 쓰레드에 하나의 홀로그램 화소 하나씩 맵핑되고 (b)는 각 쓰레드에서 2×2블록을 연산하는 방법이고 (c)는 4x4 블록을 연산한다. (b)의 방법에서 쓰레드당 사용하는 레지스터는 20개이고 (c)의 방법은 48개의 레지스터를 사용한다.

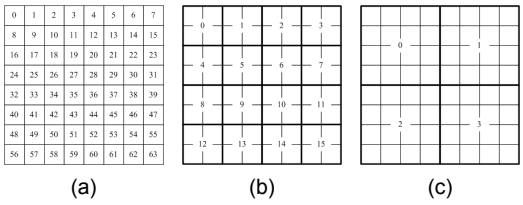

그림 6. 홀로그램 타일링 (a) 1×1, (b) 2×2, (c) 4×4
Fig. 6 Hologram tiling (a) 1×1, (b) 2×2, (c) 4×4

표 2는 쓰레드당 홀로그램 화소를 타일링하여 맵핑하였을 때 연산 시간에 대한 결과이다. 1K광원에 대하여 1,024×1,024 홀로그램을 생성하며 상수 메모리를 사용하고 블록당 512개의 쓰레드를 할당하였다. 블록당 최대 레지스터는 32,768이고 블록의 크기를 512개 이므로 쓰레드당 최대 64개의 레지스터를 할당 할 수 있다. 따라서 쓰레드당 4×4블록을 연산을 할 경우 48개의 레지스터를 사용한다. 또한 SM당 3개의 블록이고 총 16개의 SM이 존재하기 때문에 한 번에 연산할 수 있는 총 화소의 개수는 393,216개의 화소를 연산할 수 있다. 따라서 1,024×1,024의 홀로그램을 생성하기 위해서는 총 2.67번의 연산을 수행해야한다. 만약 한 번에 연산할 수

있는 총 화소의 개수가 홀로그램의 총 화소의 개수보다 적을 경우 최대 효율을 내기 어렵다. 따라서 최소의 화소의 개수 보다는 커야 한다[12].

표 2. 홀로그램 타일링에 따른 평균 연산 시간
Table. 2 Average calculation time according to hologram tiling

Hologram Tiling	
Hologram Tiling	Average Time (ms)
1	31.19869
2×2	24.83576
4×4	24.53093

3.5. GPU 블록 크기조절

표 3은 블록당 쓰레드의 개수에 따라 1K광원으로 1,024×1,024홀로그램을 상수 메모리를 이용하여 생성한 결과에 대한 연산 시간이다. GTX 780Ti은 SM당 2048개의 SP를 가지고 있기 때문에 하나의 블록당 1024개의 쓰레드로 구성을 할 경우 SM당 2개의 블록을 구성을 할 수 있고 SP 낭비를 최소화 할수 있어 효율이 좋다. 표 3에 쓰레드의 개수에 따른 생성 시간의 평균을 나타내었다. 메모리 접근 밴드 폭은 96.48 GB/s이고 쓰레드를 1024개로 하였을 경우 5,046 GFlops의 부동 소수점 연산 능력을 가진다. 광원과 LUT는 상수메모리를 사용하기 때문에 실제 CGMA의 비는 모든 커널이 종료될 때 까지 누적된 홀로그램의 세기를 저장하기위한 두 번의 메모리 접근이 있다. 따라서 1K의 유효 광원을 연산 할 때 10:1의 CGMA가 가능하다.

표 3. 블록당 쓰레드 개수에 따른 평균 연산 시간
Table. 3 Average calculation times according to the number of threads in a block

Threads Dimension			
X	Y	Total	Average Time (ms)
32	32	1024	31.58189
32	16	512	31.19869
32	8	256	31.4118
32	4	128	31.65344
16	16	256	31.48317
16	8	128	31.62211
16	4	64	32.11208
8	8	64	32.11853
8	4	32	37.04837
4	4	16	70.22265

Ⅳ. 다중 GPU를 이용한 병렬화

4.1. GPU를 기반으로 한 병렬 연산 방법

GPU 특성과 입력 포인트 클라우드를 고려하여 GPU 연산 방법에 대한 가이드라인을 제시한다. 먼저 가동할 수 있는 GPU의 특성을 파악하고, 최적의 블록과 쓰레드 할당 방법을 얻는다. 또한 적절한 입력 광원의 타일링 방법을 실험적으로 찾는다. 이때 가장 중요한 것은 GPU를 활성화시키고 동작시키는 전체적인 작업 순서를 결정하는 것이다. 그림 7에 제안하는 GPU 운용 방식에 대한 동작 순서를 나타냈다.

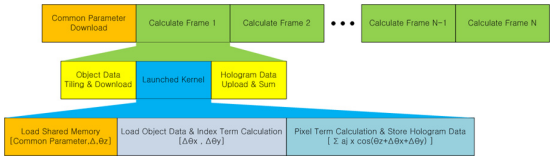

그림 7. GPU 동작 순서
Fig. 7 Operational sequence of GPU

4.2. 시스템 구성에 따른 병렬 컴퓨팅의 최적화

입력 광원에 따른 하나의 쓰레드에서 다중 GPU로 연산할 때 효율적인 연산을 위한 가이드라인을 제시하여 시스템의 구성 성분에 따른 최적화된 병렬화 성능을 도출해야 한다. 알고리즘의 병렬화는 현재 시스템이 어떻게 구성되어 있는지에 따라 많이 좌우되기 때문에 어떤 정답이 하나만 존재하는 것이 아니고 시스템 환경과 입력 데이터의 조건에 따른 환경변수에 맞추어 병렬 컴퓨팅의 성능을 결정한다. 그림 8에는 한양대[13]에서 수행된 GPU 병렬화 방식을 나타냈고, 그림 9에서는 광운대[14]에서 사용된 병렬화 방식을 나타냈다. 이 두 가지 방식은 어 병렬화된 알고리즘의 동작과 다양한 데이터 대역폭에 따라서 성능이 변할 수 있다. CPU와 GPU 사이에 큰 데이터 대역폭이 요구된다면 그림 8의 방법이 적당하다. 그렇지 않다면 불필요한 CPU 쓰레드의 점유로 인해서 CPU를 필요로 하는 동작의 성능을 저하시켜 전체 성능이 낮아진다. GPU는 단독으로 모든 동작을 감당할 수 없고 CPU의 도움을 필요로 할 뿐만 아니라 GPU에 의한 병렬 동작 이외에 CPU를 이용한 다른 동작들이 동시에 수행되고 있기 때문이다. CPU와 GPU 사이의 데이터 전송시간이 GPU에 의한 병렬화 동

작 시간에 비해서 작다면 많은 CPU 쓰레드를 데이터를 전송하는데 사용할 필요가 없다.

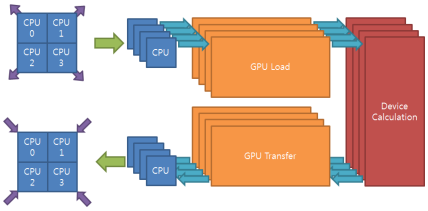

그림 8. 다중 CPU 쓰레드에 의한 다중 GPU 구동
Fig. 8 Multiple GPU execution by multiple CPU threads

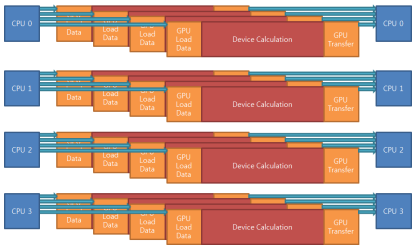

그림 9. 하나의 쓰레드에 의한 다중 GPU 구동
Fig. 9 Multiple GPU execution by a CPU thread

4.3. 다중 GPU 기반의 CGH

CGH를 이용하여 컬러 홀로그램을 생성한다면 식 (2)는 R, G, 및 B 성분에 대해서 세 번 수행해야 한다. R, G, 및 B 성분을 위한 홀로그램은 각각 다른 파장의 광원을 사용하기 때문에 동일한 연산으로 수행될 수 없다. 식 (2)에서 λ에 다른 값이 사용되어 $I_{\alpha j,R}$, $I_{\alpha j,G}$, 및 $I_{\alpha j,B}$를 따로 구한다. 공간적인 위치는 동일하기 때문에 $(p_\alpha x_\alpha - p_j x_j)^2 + (p_\alpha y_\alpha - p_j y_j)^2$는 R, G, 및 B 성분이 모두 동일한 연산 결과를 사용한다. $p_j z_j/\lambda$와 $1/2\lambda p_j z_j$의 연산 결과는 각 색차 성분 별로 각각 256개의 값을 가지기 때문에 매번 연산을 하는 것보다는 룩업테이블(look-up table, LUT)을 만들어서 사용하는 것이 좋다. 이와 같은 연산은 GPU 내부에서 수행되고, 그 동작을 그림 10에 도식적으로 나타냈다[14].

식 (2)의 CGH는 속도를 위하여 2개의 GPU를 이용하여 구현하였다[1. 깊이와 RGB로 구성된 광원 정보 중에서 유효한 광원만 2개의 GPU에 나누어 입력한다. 입력된 광원을 이용하여 각 GPU는 중간 홀로그램을 연산한다. 그 결과는 CPU에서 합쳐진 후에 정규화 과정을 거쳐서 최종적인 홀로그램으로 생성된다. 다중 GPU를 이용하여 병렬적으로 CGH를 수행하는 과정을 그림 11에 나타냈다.

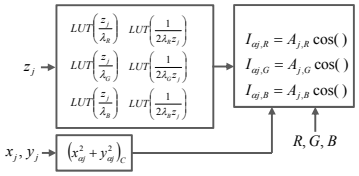

그림 10. GPU의 동작
Fig. 10 The operation of the GPU

구현한 GPU 병렬 동작은 CUDA API내의 비동기 동작 함수를 통하여 호스트에서 GPU내의 메모리로 전송하는 동작을 제외한 GPU 커널을 독립적으로 연산이 가능하도록 구현하였다.

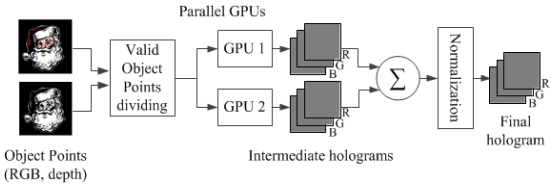

그림 11. 다중 GPU를 이용한 병렬처리 방식
Fig. 11 The parallel processing method using multiple GPUs

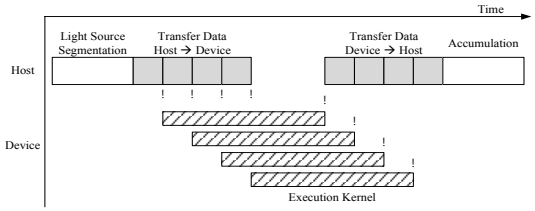

그림 12. 호스트와 디바이스 간의 동작 스케줄링
Fig. 12 Operational scheduling between host and device

그림 12에는 호스트 프로그램과 GPU 커널간의 스케쥴링을 나타내었다. 호스트에서 유효한 광원의 데이터를 GPU의 개수만큼 나누어서 GPU의 메모리로 각각 전송한 뒤 이벤트를 발생시킨다. 각 GPU에서는 호스트에서 이벤트를 확인하여 커널을 수행하고 커널이 모두 끝나면 호스트로의 이벤트를 발생 시킨다. 호스트에서는 각 GPU로부터 이벤트를 확인하여 데이터를 호스트의 메모리로 홀로그램 결과를 로드 시킨다. 모

든 GPU로부터 데이터를 로드 시킨 뒤 결과를 누적하여 홀로그램을 생성한다. 그림 12의 "!"는 이벤트 발생을 나타낸다.

Ⅴ. 실험 결과 및 구현

구현한 고속 홀로그램 생성 프로그램은 nVidia사의 CUDA를 이용하여 프로그램 하였고 사용된 GPGPU는 GTX 780Ti 4 대를 이용하여 실험 하였다. GTX 780Ti의 특성은 표 4에 나타냈다. 총 DRAM(전역 메모리)의 크기는 3,072MB이고 상수 메모리는 64KB, 블록당 공유 메모리는 48KB이고 SM당 65,536개의 32비트의 레지스터를 사용할 수 있다. 또한 블록당 최대 쓰레드의 크기는 1,024개이고 SM당 사용할 수 있는 최대 쓰레드(SM당 SP의 개수)는 2,048개이고 총 SM의 개수는 15개로 이루어져 있고 코어는 1.08GHz로 동작 한다. 또한 DRAM은 3.5GHz의 속도로 동작하고 384bit로 데이터를 주고받는다. 이러한 GPU를 이용하여 CGH를 생성하는데 약 10K의 입체 정보를 이용하여 1,024×1,024의 컬러 홀로그램을 약 106ms에 생성하였다.

표 4. nVidia GTX780Ti의 특성
Table. 4 Characteristics of nVidia GTX780Ti

Contents	GTX780Ti
Total Global Memory	3,072 MB
Total Constant Memory	64 KB
Memory Clock Frequency	3.5 GHz
Memory Bandwidth	384 bit
Total Streaming Processor(SM)	15 EA
Shared Memory / SM	2048 KB
Register / Block	65536 EA
SP / SM	2048 EA
Max Threads / Block	10224 EA
Core Clock Frequency	1.08 GHz
GFlops	5046 GFlops

보정 및 중간 시점 생성의 단계에서 사용되는 알고리즘은 C/C++ 및 OpenCV로 구현하였고, CGH와 S/W 방식의 복원은 CUDA 언어로 구현하였다. 이러한 S/W 엔진들을 라이브러리화한 후에 LabView에서 함수로 이식하여 통합하였다. 홀로그램 영상을 복원하기 위해서는 광학장치를 이용하였고, 또한 테스트를 위해서 S/W 방식으로 복원하였다. 이를 위해 표 5와 같은 파라미터들을 이용하여 실험하였다.

표 5. 실험을 위한 파라미터
Table. 5 Experimental parameter

Parameter		Specification
S/W	Distance	100cm
	Hologram Size	1,024×1,024
	Pixel Pitch (p)	10.4μm
	Wavelength (λ)	633nm (Red Laser)

S/W적인 복원 방법은 홀로그램을 2차원 영상으로 취급하여 확인하고 화질을 테스트하기 위한 것이다. 2차원으로 복원하는 것이라 특정 거리에서만 복원이 되기 때문에 이 경우는 복원될 거리를 지정해준 다음, 그 거리의 평면에 맺히는 2차원 영상을 복원하게 된다. 이를 위해 식 (4)에 정의된 Fresnel 변환을 이용하였다

$$F(x,y)=\frac{e^{ikz}}{i\lambda z}e^{\frac{i\pi}{\lambda z}(x^2+y^2)}\iint f(\xi,\eta)e^{\frac{i\pi}{\lambda z}(\xi^2+\eta^2)}e^{-\frac{i2\pi}{\lambda z}(\xi x+\eta y)}d\xi d\eta \quad (4)$$

여기서 $F(x,y)$와 $f(\xi,\eta)$는 각각 복원된 영상과 홀로그램 영상의 화소 값을 나타내며, $i=\sqrt{-1}$ 이다. z는 복원영상의 깊이 값을 나타내며, k와 λ는 사용한 광파의 파수(wave number)와 파장을 각각 나타낸다. 소프트웨어 복원 결과는 그림 13와 같다. 본 연구팀은 그림 14과 같은 GPU 구현 환경을 보유하고 있다. 그림 14는 8개의 GPU를 병렬로 장착하여 동작시킬 수 있는 Cubix의 외장 GPU 확장 데스크톱으로 본 논문에 이용되었다. GTX780Ti을 사용할 경우에 총 8개의 GPU를 병렬로 동작시킬 수 있다. 본 논문에서는 이 중에서 4개의 GPU를 이용하여 CGH를 구현하였다.

Ⅵ. 결 론

본 논문에서는 GPU의 자원할당에 따른 GPU의 동작에 대하여 소개하고 홀로그램 생성을 위한 최적화 기법에 대하여 가이드라인을 제시하였다. GPU의 커널이 실행 될 때 필요한 레지스터 및 실행 쓰레드 및 블록의 개수에 따른 GPU의 동작을 소개하고 최적화된 자원 할당 방법 대하여 제안하고 또한 고속의 홀로그램 생성을 위

한 타일링 기법과 사용하는 메모리 및 데이터 포맷에 대하여 최적화 하는 방법에 대하여 제안하였다. 실험을 통하여 구동 GPU의 특성을 파악하여 최대 쓰레드를 개수를 결정 하고 상수 메모리를 사용하여 CGMA의 비를 높이고 타일링을 통하여 최적의 GPU 동작 점을 확인하였다. 본 논문에서 제시하는 가이드라인을 통하여 구동 GPU의 타겟에 따른 홀로그램 생성의 최적화 할 수 있다.

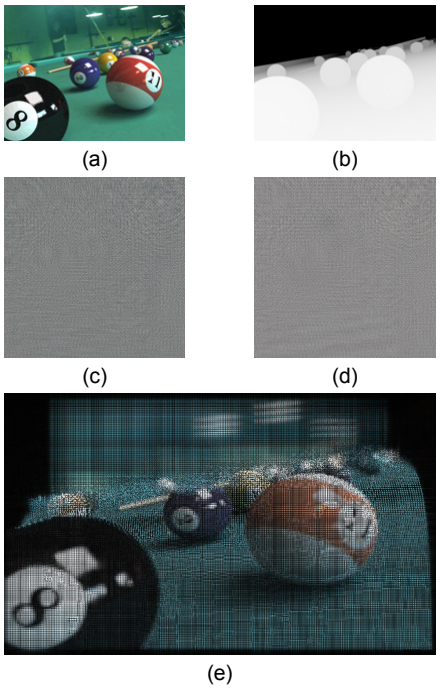

그림 13. S/W를 이용한 홀로그램 복원 결과
Fig. 13 Hologram reconstruction by using S/W

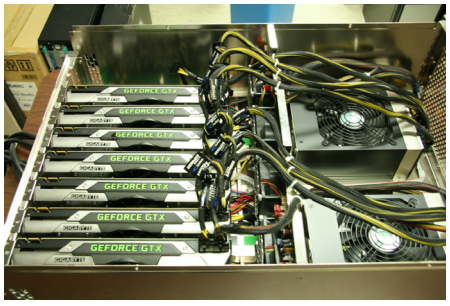

그림 14. GPU 병렬 확장을 위한 외장 데스크톱
Fig. 14 External desktop for parallel extension of GPU

감사의 글

이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (NRF-2012-2012S1A5A2A03034349).

REFERENCES

[1] T. Motoki, H. Isono, and I. Yuyama, “Present Status of Three-Dimensional Television Research,” Proc. IEEE 83(7): 1009-1021(July 1995).

[2] T. Ito, N. Masuda, K. Yoshimura, A. Shiraki, T. Shimobaba, and T. Sugie, “Special-Purpose computer HORN-5 for a real-time electroholography,” Optics Express, Vol. 13, No. 6, March 2005.

[3] Y. Ichihashi, H. Nakayama, T. Ito, N, Masuda, T. Shimobaba, A, Shiraki, and T. Sugie, "HORN-6 special-purpose clustered computing system for electroholography", Optics Express, vol. 17, no. 16, pp. 13895-13903, Aug, 2009.

[4] Y.-H. Seo, H.-J. Choi, J.-S. Yoo, and D.-W. Kim, "An architecture of a high-speed digital hologram generator based on FPGA", Journal of Systems Architecture, Vol. 56. pp. 27-37, Dec. 2009.

[5] Y.-H. Seo, H.-J. Choi, J.-S. Yoo, and D.-W. Kim, "A New Parallelizing Algorithm and Cell-based Hardware Architecture for High-speed Generation of Digital Hologram", Journal of Systems Architecture, Vol. 16. pp. 54-63, Jan. 2011.

[6] Y.-H. Lee, Y.-H. Seo, J.-S. Yoo, and D.-W. Kim, "Hardware architecture of high-performance digital hologram generator on the basis of a pixel-by-pixel calculation scheme", Applied Optices, Vol. 51. pp. 4003-4012, June. 2012.

[7] N. Masuda, T. Ito, T. Tanaka, A. Shiraki, and T. Sugie, “Computer generated holography using a graphics processing unit,” Optics Express, Vol. 14, No. 2, January 2006.

[8] L. Ahrenberg, P. Benzie, M. Magnor, and J. Watson, “Computer generated holography using parallel commodity graphics hardware,” Optics Express, Vol. 14, No. 17, August 2006.

[9] Y. Pan, X. Xu, S. Solanki, X. Liang, R. Bin, A. Tanjung, C. Tan, and T.-C. Chong, "Fast CGH computation using S-LUT on GPU", Optics Express, vol. 17, No. 21, pp. 18543-18555, Oct. 2009.

[10] Y.-Z. Liu, J.-W. Dong, Y.-Y. Pu, B.-C. Chen, H.-X. He, and H.-Z. Wang, "High-speed full analytical holographic computations for true-life scenes", Optics Express, vol. 18, no. 4, pp. 3345-3351, Feb. 2010.

[11] T. Shimobaba, T. Ito, N, Masuda, Y, Ichihashi, and N. Takada, "Fast calculation of computer-generated-hologram on AMD HD5000 series GPU and OpenCL", Optics Express, vol. 18, no. 10, pp. 9955-9960, May. 2010.

[12] Yoon-Hyuk Lee, Dong-Wook Kim, Young-Ho Seo, "High-Speed Generation Technique of Digital holographic Contents based on GPGPU", Journal of the Korea Society of Digital Industry and Information Management, Vol. 9, No. 1, pp.151-163, 2013.

[13] J.S. Song, J.S. Park, Y.H. Seo, J.I. Park “Fast Generation of Digital Hologram Based on Multi-GPU”, Journal of Korean Society of Broadcast Engineers, Vol. 16, no. 6, pp.1009-1017, Nov. 2011.

[14] Young-Ho Seo, Yoon-Hyuk Lee, Ja-Myong Goo, Yu-Yeul Kim, Bo-Ra Kim, and Dong-Wook Kim, "A New System Implementation for Generating Holographic Video using Natural Color Scene", The Korean Society of Broadcast Engineers, Journal of Broadcast Engineering, v.18, n.2, pp.149-158, 2013.

[15] Kirk, David, “Programming Massively Parallel Processor 1’st Edition”, Elsevier, 2010.

서영호 (Young-Ho Seo)

1999년 2월 : 광운대학교 전자재료공학과 졸업(공학사)
2001년 2월 : 광운대학교 일반대학원 졸업(공학석사)
2004년 8월 : 광운대학교 일반대학원 졸업(공학박사)
2005년 9월 ~ 2008년 2월 : 한성대학교 조교수
2008년 3월 ~ 현재 : 광운대학교 교양학부 부교수
※관심분야 : 실감미디어, 2D/3D 영상 신호처리, 디지털 홀로그램, SoC 설계

이윤혁(Yoon-Hyuk Lee)

2012 광운대학교 전자재료 공학과 공학사 졸업
2014 광운대학교 일반대학원 공학석사 졸업
※관심분야 : 디지털 홀로그램, SoC 설계

김동욱 (Dong-Wook Kim)

1983년 2월 한양대학교 전자공학과 졸업(공학사)
1985년 2월 한양대학교 공학석사
1991년 9월 Georgia공과대학 전기공학과(공학박사)
1992년 3월 ~ 현재 광운대학교 전자재료공학과 정교수
2009년 3월 ~ 현재 광운대학교 실감미디어 연구소 연구소장
※관심분야 : 3D 영상처리, 디지털 홀로그램, 디지털 VLSI Testability, VLSI CAD, DSP설계