초고해상도 홀로그램 생성을 위한 GPU 기반 Shift-FFT 처리 구현

*이재홍, **강호민, ***염한주, ****전상훈, *****박중기, ******김덕수
*, **, ******한국기술교육대학교
, *, *****한국전자통신연구원
*oorange31@koreatech.ac.kr, **0lines0@koreatech.ac.kr, ***hj.yeom@etri.re.kr,
****sh.cheon@etri.re.kr, *****jkp@etri.re.kr , ******bluekds@koreatech.ac.kr

GPU-based Shift-FFT Implementation for Ultra-High Resolution Hologram Generation

*Jaehong Lee, **Homin Kang, ***Han-ju Yeom, ****Sanghoon Cheon,
*****Joongki Park, ******Duksu Kim
*, **, ******Korea University of Technology and Education (KOREATECH)
, *, *****Electronics and Telecommunications Research Institute (ETRI)

요 약

본 논문은 초고해상도 컴퓨터 홀로그램 생성을 위한 GPU 기반 2D Shift-FFT 의 효율적인 구현 방법을 제안한다. 본 연구가 제안하는 알고리즘은 기존에 여섯 단계로 이루어진 처리과정을 다섯 단계로 줄임으로서, 병렬처리에서 비효율적인 메모리 접근 과정을 줄인다. 또한, 핀드(pinned) 메모리 기반의 CPU-GPU 데이터 통신 통로인 핀드 버퍼(pinned buffer)를 사용하고 다중 스트림을 채용함으로써, GPU 활용의 주요 병목원인이 되는 데이터 통신의 부하를 줄이고 GPU 활용 효율을 높인다. 본 연구는 제안하는 알고리즘의 효용성을 증명하기 위해 서로 다른 두 시스템에 알고리즘을 구현하고, 다양한 크기의 행렬에 대한 2D-FFT 처리에 대한 성능을 측정하였다. 그 결과, CPU 기반의 FFTW 라이브러리 대비 최대 3 배, 동일한 GPU 를 사용하는 cuFFT 라이브러리 대비 최대 1.5 배 높은 성능을 달성하였다. 이러한 결과는, 본 연구가 제안하는 알고리즘의 효용성을 보여주는 결과다.

1. 서론

홀로그래피(holography)는 1947 년 데니스 가보르(Dennis Gabor)에 의해 처음 제안되었으며[1], 별도의 장치없이 3 차원 영상을 볼 수 있다는 점에서 궁극의 영상 기술이라고 불린다. 홀로그래피를 기록하는 매체를 홀로그램(hologram)이라고 부른다. CGH(Computer Generated Hologram)는 홀로그램을 생성하는 대표적인 방법들 중 하나로, 가상공간 내 3 차원 물체를 홀로그램에 기록한다[2]. CGH 는 컴퓨터 연산만으로 다양한 물체에 대한 홀로그램을 생성할 수 있다는 장점을 가지며, 컴퓨팅 성능의 비약적 향상에 힘입어 최근 많은 관심을 받고 있는 기술이다.

홀로그램 디스플레이의 시야각은 픽셀 간격과 반비례하며, 일반적인 2 차원 디스플레이에 비해 높은 해상도를 요구한다. 예를 들어 시야각 40°를 만족하는 5cm × 5cm 크기의 홀로그램은 픽셀간격 500nm, 100K × 100K 수준의 초고해상도를 요구한다. CGH 의 핵심 모듈 중 하나는 광파의 파면 전파(wave-front propagation, diffraction)다. 파면 전파 계산의 주요 연산 중 하나는 Shift-FFT (Shift->FFT->Shift) 연산이다. FFT 는 고속 푸리에 변환(Fast Fourier Transform)으로[3], 초고해상도 홀로그램 생성을 위해서는 거대 2 차원 행렬에 대한 Shift-FFT 연산을 수행해야 한다.

FFT 의 성능을 높이기 위해 다양한 연구들이 진행되어 왔으며[4], 최근에는 GPU(Graphics Processing Unit)의 대규모 병렬처리 성능을 활용한 FFT 처리 기술들(예, cuFFT 라이브러리[5])이 개발되고 있다. 대부분의 GPU 기반 FFT 기술들은 전체 데이터를 메모리에 올려 놓고 FFT 연산을 수행한다. 하지만 초고해상도 홀로그램 생성을 위한 거대 2 차원 행렬에 대한 FFT 수행은 연산양이 많음은 물론, 메모리 공간 요구도 매우 크다. 따라서, 기존의 라이브러리를 직접 활용하는데 한계를 가진다.

본 연구는, 높은 병렬처리 성능을 보여주지만 제한적인 메모리 공간을 가진 GPU 를 활용하여 거대 2 차원 행렬에 대한 Shift-FFT 연산을 효율적으로 계산할 수 있는 알고리즘을 제안한다. 제안하는 알고리즘은 2D-FFT 연산을 1D-FFT 로 나누어 처리하는 방법으로 GPU 메모리 공간 문제를 해결한다. 또한, 다중 스트림(stream)과 핀드 버퍼(pinned buffer) 기법을 활용하여 CPU 와 GPU 사이의 통신 효율을 높이고, GPU 의 성능을 최대한 활용할 수 있도록 한다(3 장).

본 연구는 제안하는 방법의 성능을 검증하기위해, 기존의 FFT 라이브러리들과 성능을 비교하였다. 그 결과 CPU 기반 라이브러리(FFTW)와 GPU 기반 라이브러리(cuFFT) 대비 Shift-FFT 연산에 대해 각각 최대 3 배와 1.5 배 높은 성능을 달성하였다. 이러한 결과는 본 연구가 제안하는 GPU 기반 Shift-FFT 알고리즘의 효용성을 증명하는 결과다.

2. 관련 연구

CGH 연산 가속을 위해 Shimobaba 등은 Multi-core CPU 기반으로 성능을 높여왔다[6,7]. FFT 연산은 Intel Math Kernel Library(MKL)를 사용하고 FFT 를 제외한 연산에 OpenMP 를 사용하여 가속하였으며, 다양한 아키텍처(Xeon Phi 등)에서 그

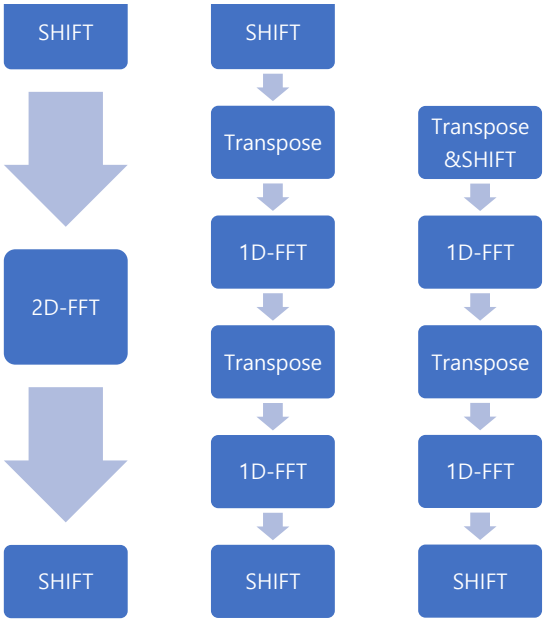

그림 1. (a) 기존의 2D Shift-FFT 처리 절차 (b) 1D-FFT 를 이용한 2D Shift-FFT 처리 절차 (c) 본 연구가 제안하는 Transpose & Shift 를 이용하는 처리 절차

성능을 검증하였다[7]. Matteo Frigo 와 Steven G. Johnson 은 고성능 FFT 라이브러리인 FFTW3 발표하였다[8]. FFTW3 는 CPU 의 SIMD(Single Instruction, Multiple Data) 연산을 사용하며 FFT 처리 성능을 높였으며, 다중 스레드를 사용을 통한 병렬처리를 지원한다. FFTW3 는 대표적인 CPU 기반 고성능 FFT 라이브러리 중 하나이기도 하다.

최근 GPU 의 높은 병렬처리 성능을 활용하는 FFT 알고리즘들이 제안되고 있다[5,9]. cuFFT 는 NVIDIA 가 제공하는 FFT 라이브러리로, CPU 에 비해 수배~수십 배 높은 성능을 보여준다[5,10]. 하지만, GPU 는 상대적으로 작은 크기의 메모리공간을 가지며, cuFFT 가 한번에 처리할 수 있는 데이터 크기가 제한되는 한계를 가진다. Gu 등은 2D-FFT 연산을 일련의 1D-FFT 수행으로 대체하는 방법으로 GPU 메모리 크기를 넘어가는 경우에 2D-FFT 를 수행하였다[11]. 그들은 또한 다중 스트림을 사용하여 데이터 통신을 최적화하였다. Ogata 등 및 S.Chen 과 Li 는 2D-FFT 를 처리하기 위해 CPU 와 GPU 를 동시에 사용했다[12,13]. 그들은 각 장치마다 데이터 크기에 따른 연산시간을 수식화하고, 실험데이터로 계수를 결정함으로써 CPU 와 GPU 에게 일을 효율적으로 분배하고 2D-FFT 처리 성능을 높였다.

본 연구는 Gu 등과 같이 1D-FFT 를 사용하여 2D-FFT 를 처리함으로써, GPU 메모리 크기 문제를 해결한다. 하지만, 단순한 2D-FFT 가 아닌 Shift-FFT 연산을 위한 최적의 구현을 제시한다는 점에서 Gu 등의 방법과 구별된다. 또한, 핀드 버퍼 메모리 사용으로 거대 2D 행렬에 대해서도 다중 스트림을 사용할 수 있다는 차별성을 갖는다.

3. GPU 기반 Shift-FFT 알고리즘

현재 일반적인 GPU 의 메모리(device memory) 크기는 2~11GB 수준인 반면, 초고해상도 홀로그램 생성을 위한 2 차원 행렬의 크기는 수십~수백 GB(예, 배정밀도 100K * 100K 해상도

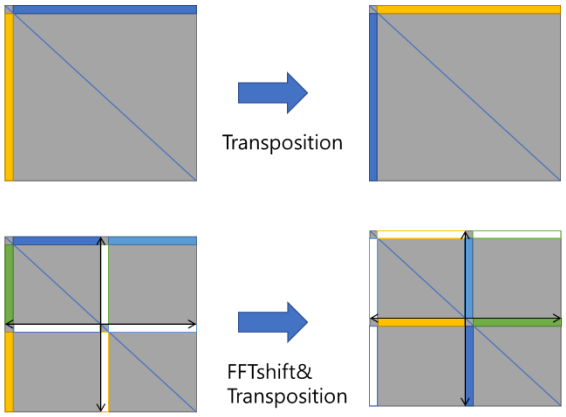

그림 2. 행렬 전치와 shift&전치

시, 149GB)에 달한다. 본 연구는 이와 같은 GPU 메모리 크기 제한의 문제를 해결하기 위해서, 2D-FFT 가 행렬의 각 행과 열에 대해서 행 방향 및 열 방향으로 각각 1D-FFT 를 한 것과 같다는 특성을 이용한다[11]. 하지만, 열 방향 FFT 수행 시의 메모리 접근 패턴은 GPU 가 선호하는 메모리 접근 패턴과 다르기 때문에 처리 효율이 떨어지는 문제점이 있다. 본 연구는 열 방향 FFT 를 수행 전, GPU 메모리에 행렬 데이터를 전치(transposition)하여 전송하는 방식으로 이러한 문제를 해결한다. GPU 메모리에 전송된 행렬은 전치된 상태로, 행 방향 FFT 를 수행하면 원본 행렬에서 열 방향 FFT 를 한 것과 같은 결과를 얻는다. 그 결과는 CPU 메모리(host memory)로 전송된 후, 다시 전치되어 열 방향 FFT 결과로 저장된다.

CGH 의 파면전파에서 사용하는 2D-FFT 는 일반적인 2D-FFT 전과 후에 shift 단계를 거친다[14](그림 1-(a)). 정사각행렬에 대한 FFT 를 위한 shift 는 행렬의 중심을 기준으로 하는 직교 좌표계에서, 1 사분면과 3 사분면, 2 사분면과 4 사분면의 원소를 서로 맞바꾸는 연산이다. 따라서 GPU 를 이용하여 파면전파를 계산하기 위해서는 그림 1-(b)와 같이 여섯 단계가 필요하다. 하지만, 본 연구는 shift 와 열 방향 FFT 를 위한 첫번째 행렬 전치 과정이 동시에 처리 가능하다는 것을 발견하였으며, 제안하는 알고리즘은 그림 1-(c)와 같이 다섯 단계로 연산을 수행한다. Shift 와 행렬 전치 연산은 메모리 접근이 주가 되는 연산으로 병렬처리 효율이 낮은 부분이기도 하다. 두 단계를 하나로 합침으로써 첫번째 shift 와 행렬 전치에 소요되는 시간을 기존의 70% 수준으로 줄일 수 있었다. 그 결과, 기존 Shift-FFT 대비 약 4~13%의 성능 향상을 얻을 수 있었다.

GPU 의 메모리는 초고해상도 홀로그램 생성을 위한 행렬 전체를 담을 수 없기 때문에, 행렬 전치와 shift 는 CPU 메모리(host memory)에서 수행되고, GPU 의 메모리(device memory)가 허용하는 크기만큼 GPU 로 전송하여 FFT 연산을 수행한다. 이러한 CPU-GPU 메모리 사이의 통신은 잘 알려진 GPU 활용의 주요 병목지점으로, 이를 해결하기 위해 본 연구는 다중 스트림을 사용하여 데이터 전송과 CPU 및 GPU 연산을 중첩하는 방법을 사용한다. 다중 스트림을 사용하여 데이터 전송과 연산을 중첩하기 위해서는 전송할 데이터가 항상 물리 메모리 공간에 적재되어 있음이 보장되어야 한다. 가상 메모리(virtual memory)로 내려가지 않고, 물리 메모리(physical memory) 공간에 유지되는 메모리 영역을 페이지-잠김(page-locked) 메모리 또는 핀드(pinned) 메모리라고 부른다. 일반적인 PC 의 시스템 메모리(CPU 메모리, DRAM)는 수십 GB 수준(최대 128 GB)으로, 초고해상도 홀로그램 처리에 필요한 메모리 크기에 미치지 못한다. 그리고 핀드 메모리로 잡을 수 있는 크기는

	Machine 1	Machine2
CPU	i5-8400	i7-9700k
RAM	DDR4 32GB	DDR4 128GB
GPU	RTX2080 8GB	GTX1060 6GB
운영체제	Windows 10	
CUDA	CUDA 10.2	
FFTW	FFTW 3.3.5	

표 1. 실험에 사용된 시스템 정보

물리적 메모리(DRAM) 중 일부로 제한된다. 따라서, 우리가 처리해야 할 행렬 전체를 핀드 메모리로 잡을 수 없으며, 다중 스트림 또한 사용할 수 없다는 문제가 발생한다. 본 연구는 작은 크기의 버퍼만 핀드 메모리 공간에 생성(핀드 버퍼)하고 CPU-GPU 사이의 데이터 통신을 해당 버퍼를 통해서 수행하게 함으로써 큰 크기의 행렬에 대해서도 다중 스트림을 이용한 FFT 처리가 가능하도록 하였다.

4. 결과 및 분석

본 논문에서 제시한 GPU 기반 Shift-FFT 알고리즘의 성능을 확인하기 위해, 서로 다른 네 가지 알고리즘을 구현하고 그 성능을 비교하였다. 각각의 알고리즘은 아래와 같다

- **FFTW**: FFTW 라이브러리를 이용하여 2D-FFT 를 처리하며 2D FFT 전&후에 shift 연산을 수행(그림 1-(a))
- **cuFFT/S**: cuFFT 라이브러리의 1D FFT 와 행렬 전치를 이용하여 2D FFT 를 연산하며, 2D FFT 전&후에 shift 연산수수행(그림 1-(b))
- **Ours**: cuFFT/S 에서 첫번째 shift 와 행렬 전치를 동시에 처리하는 본 연구가 제안하는 알고리즘(그림 1-(c))
- **Ours-PinBuf**: 핀드 버퍼를 이용하여 Ours 를 다중 스트림을 이용하도록 확장한 알고리즘으로, 실험에서는 총 네 개의 스트림을 사용함.

표 1 은 실험에 사용된 두 가지 시스템의 정보를 보여준다. 실험에 사용된 데이터는 2 차원 배정밀도(double-precision) 행렬을 임의의 값으로 생성하여 사용하였다. 각 행렬의 크기는 20K × 20K(6.0GB), 40K × 40K(23.8GB), 60K × 60K(53.6GB), 80K × 80K(95.4GB)이며 각 테스트 시스템의 주 메모리를 넘지 않는 행렬들을 이용하여 성능을 측정하였다. 즉, Machine 1 의 주메모리는 32GB 으로 20K × 20K 와 40K × 40K 행렬을 사용하였다. 20K × 20K(6.0GB)의 경우, Machine 1 의 GPU 메모리 크기 보다 작지만, cuFFT 의 2D-FFT 연산은 추가적인 GPU 의 메모리를 요구하기에 cuFFT 의 2D-FFT 를 바로 적용할 수 없음을 확인하였다. 따라서, cuFFT/S 를 이용하여 2D-FFT 를 수행하였다. Machine 2 의 주메모리는 128GB 으로, 네 종류의 서로 다른 크기를 가진 행렬들에 대해서 실험을 진행하였다.

그림 3 은 Machine 1 에서 네 가지 알고리즘의 성능을 보여준다. 높은 병렬처리 능력을 가진 GPU 를 사용하는 cuFFT/S 는 CPU 기반의 FFTW 대비 20K × 20K 와 40K × 40K 에서 1.72 배, 2.19 배 높은 성능을 보여준다. 같은 GPU 를 사용하는 Ours 는 cuFFT/S 대비 20K × 20K 및 40K × 40K 행렬에 대해 각각 1.13 배, 1.08 배 높은 성능을 보여준다. 이는 첫번째 shift 와 행렬전치 두 단계를 하나의 단계로 처리함으로써 얻어진 결과다. 또한, Our-PinBuf 는 Ours 대비 20K × 20K 와 40K × 40K 행렬에 대해 1.33 배, 1.29 배 높은 성능을 보여주었다. cuFFT/S 와 비교해서는 최대 1.5 배 높은 성능을 보여주었다.

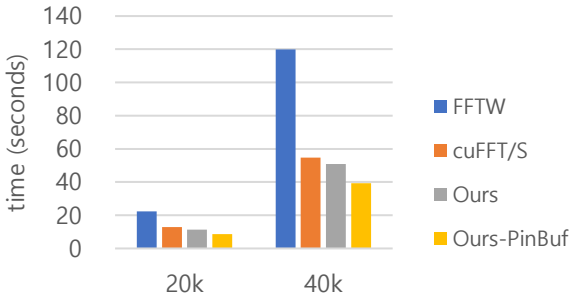

그림 3. Machine1 의 Shift-FFT 성능비교

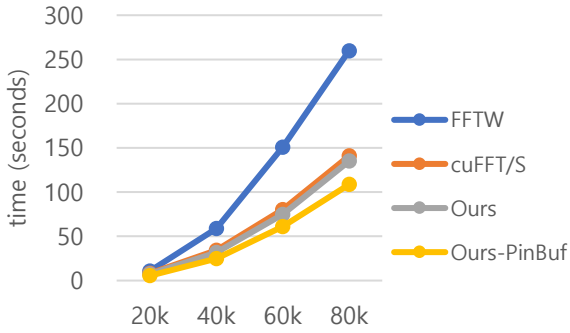

그림 4. Machine2 의 Shift-FFT 성능비교

그림 4 는 Machine 2 에서 서로 다른 알고리즘들의 2D Shift-FFT 처리 성능을 보여준다. Our-PinBuf 는 cuFFT/S 대비 20K × 20K, 40K × 40K, 60K × 60K, 그리고 80K × 80K 에 대해 각각 1.48 배, 1.38 배, 1.31 배, 1.30 배 높은 성능을 보여주는 등 Machine 1 에서와 유사한 성능 향상 경향을 보여주었다.

핀드 버퍼 사용 효과: 그림 5 는 Machine 1 에서 40K × 40K 행렬을 처리할 때, 1D-FFT 연산과 다른 연산들에 소요된 시간을 보여준다. 결과에서 볼 수 있듯, 핀드 버퍼를 사용함으로써 1D-FFT 성능이 크게(예, 2.2 배) 향상된 것을 확인할 수 있었다. 본 연구는 핀드 버퍼가 이러한 높은 성능향상을 이끄는 원인을 분석하기 위해서 Nsight Systems[15]를 이용하여, 각 GPU 스트림에서의 데이터 통신과 연산 시간의 비율을 분석하였다. Ours 의 경우 스트림 사용의 10%가 커널 연산, 나머지 90%는 데이터 통신인 반면, Ours-PinBuf 의 경우 한 스트림에서 커널 연산에 대한 비율이 17%로 높아 졌다. 핀드 메모리를 사용할 경우, 페이지-잠김이 아닌 경우 대비 2 배 이상 통신 속도가 빠르며[16], 이에 따라 데이터 통신에 의한 발생하는 병목 현상이 줄어든 결과다. 여기서 주목해야할 것은 Ours-PinBuf 는 다중 스트림을 이용하며, 서로 다른 스트림을 통해 이루어지는 커널 연산과 데이터 통신이 동시에 이루어질 수 있다는 것이다. Ours-PinBuf 의 경우 네 개의 스트림을 사용함으로, 데이터 통신 시간 중 최대 82%가 커널 연산과 중첩될 수 있음을 의미한다. 실제 Nsight Systems 를 이용한 분석결과, Ours 에서 90%의 시간이 데이터 통신에 의해 GPU 를 활용하지 못한 반면, Ours-Pin-Buf 에서는 그 비율이 60%로 낮아진 것을 확인할 수 있었다. 그 결과, 1D-FFT 수행 과정에서 GPU 를 더 효율적으로 활용하고, Ours 대비 높은 성능향상을 얻을 수 있었다.

5. 한계점 및 향후 연구

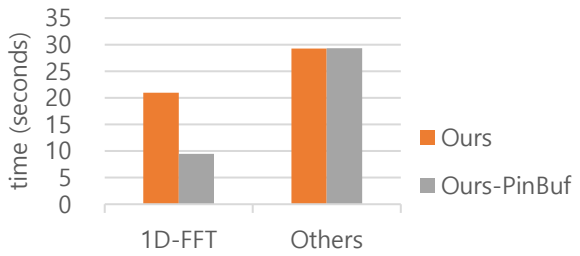

그림 5. Machine1 의 cuFFT 알고리즘별 40K × 40K 행렬 FFT 연산요소 비교

본 논문은 초고해상도 홀로그램 생성을 위한 GPU 기반 2D Shift-FFT 알고리즘을 제안하였다. 그 결과 CPU 기반의 FFTW 대비 최대 3 배, 동일한 GPU 를 사용하는 cuFFT 라이브러리 대비 최대 1.5 배 높은 성능을 달성하였다. 이는 본 연구가 제안하는 알고리즘이 Shift-FFT 에 필요한 연산 단계를 한단계 줄이고, 핀드 버퍼와 다중 스트림을 이용하여 GPU 의 활용 효율을 높임으로써 얻은 결과다.

본 연구는 1D-FFT 수행 과정에서 GPU 를 효율적으로 활용하였지만, 전체 연산 과정에서는 아직 CPU 의존도가 높다. 이는 2D Shift-FFT 의 성능을 극대화하는데 한계점으로 작용한다. 또한, GPU 의 다중 스트림을 사용하여 1D- FFT 를 연산을 수행하지만, 데이터 통신이 여전히 병목지점으로 작용한다는 문제점을 발견할 수 있었다. 따라서 향후 연구에서는 2D Shift-FFT 연산 전체 과정에 GPU 를 사용하고, CPU 와 GPU 사이의 데이터 통신을 줄일 수 있는 방법에 대해 연구하고자 한다.

감사의 글

이 논문은 2020 년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2019-0-00001, 홀로그램 영상 서비스를 위한 Holo-TV 핵심 기술 개발)

참고 문헌

[1] Gabor, D. (1947). A space-charge lens for the focusing of ion beams. *Nature*, 160(4055), 89-90.

[2] Slinger, C., Cameron, C., & Stanley, M. (2005). Computer-generated holography as a generic display technology. *Computer*, 38(8), 46-53.

[3] Goodman, Joseph W. "Introduction to Fourier Optics, Roberts and Company." *United States*, (2004).

[4] MIT: FFTW libraries. http://www.fftw.org/

[5] NVIDIA: CUFFT libraries. https://docs.nvidia.com/cuda/cufft/index.html

[6] Shimobaba, T., Weng, J., Sakurai, T., Okada, N., Nishitsuji, T., Takada, N., ... & Ito, T. (2012). Computational wave optics library for C++: CWO++ library. *Computer Physics Communications*, 183(5), 1124-1138.

[7] Murano, K., Shimobaba, T., Sugiyama, A., Takada, N., Kakue, T., Oikawa, M., & Ito, T. (2014). Fast computation of computer-generated hologram using Xeon Phi coprocessor. *Computer Physics Communications*, 185(10), 2742-2757.

[8] Frigo, M., & Johnson, S. G. (2005). The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2), 216-231.

[9] Moreland, Kenneth, and Edward Angel. "The FFT on a GPU." *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. Eurographics Association, 2003.

[10] Zhao, Z., & Zhao, Y. (2018, October). The Optimization of FFT Algorithm Based with Parallel Computing on GPU. *In 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)* (pp. 2003-2007). IEEE.

[11] GU, Liang; SIEGEL, Jakob; LI, Xiaoming. Using GPUs to compute large out-of-card FFTs. *In: Proceedings of the international conference on Supercomputing*. 2011. p. 255-264.

[12] S. Chen and X. Li, "A hybrid GPU/CPU FFT library for large FFT problems," 2013 IEEE 32nd International Performance Computing and Communications Conference (IPCCC), San Diego, CA, 2013, pp. 1-10.

[13] Ogata, Y., Endo, T., Maruyama, N., & Matsuoka, S. (2008, April). An efficient, model-based CPU-GPU heterogeneous FFT library. *In 2008 IEEE International Symposium on Parallel and Distributed Processing* (pp. 1-10). IEEE.

[14] MATHWORKS: MATLAB fftshift, https://kr.mathworks.com/help/matlab/ref/fftshift.html

[15] NVIDIA: Nsight Systems, https://developer.nvidia.com/nsight-systems

[16] NVIDIA: How to Optimize Data Transfers in CUDA C/C++, https://devblogs.nvidia.com/how-optimize-data-transfers-cuda-cc/