FPGA를 위한 NVMe 드라이버 구현 및 분석*

신준식[O1] 이재환[1] 김희훈[1] 이재진[1]
[1]서울대학교 컴퓨터공학부
junsik@aces.snu.ac.kr, jaehwan@aces.snu.ac.kr, heehoon@aces.snu.ac.kr, jaejin@snu.ac.kr

Implementation and Evaluation of NVMe Driver for FPGA

Junsik Shin[O1] Jaehwan Lee[1] Heehoon Kim[1] Jaejin Lee[1]
[1]Department of Computer Science and Engineering, Seoul National University

요 약

FPGA(Field-Programmable Gate Array)에 PCIe, DDR4, 이더넷 등 다양한 인터페이스를 활용해 직접 스토리지를 장착하여 높은 대역폭을 실현하는 near-storage computing이 시도되고 있다. 본 논문에서는 FPGA에 NVMe SSD를 OCuLink로 직접 연결하여 읽기와 쓰기를 호스트 CPU 및 FPGA에 존재하는 다른 CPU의 간섭없이 제어하였으며 기존에 존재하는 NVMe 드라이버 대비 최대 1.2배의 성능을 보였다.

1. 서 론

FPGA(Field-Programmable Gate Array)는 CPU, GPU 등의 ASIC과 달리 PCIe, DDR4, 이더넷 등 다양한 인터페이스를 가지도록 설계할 수 있다. 이러한 장점에 힘입어, 최근에는 FPGA에 NVMe SSD를 장착하여 낮은 지연시간과 높은 대역폭을 실현하는 near-storage computing이 시도되고 있다.

CPU의 간섭없이 FPGA에서의 계산을 지속하기 위해서는 NVMe SSD로의 읽기 및 쓰기를 제어하는 NVMe 드라이버를 FPGA내에 구현해야 한다. 그러나 제조사 제공 IP(Intellectual Property)를 포함, 하드웨어 수준 NVMe 드라이버는 고가로 판매되고 있고 코드가 비공개이므로 연구에 어려움이 있다[1]. 다른 방법으로, FPGA와 SoC로 묶인 CPU 또는 호스트 CPU에 소프트웨어 드라이버를 구동할 수 있다[2]. 이 경우, FPGA는 NVMe SSD로의 읽기와 쓰기를 매번 프로세서와 통신하여 진행해야 한다. 이러한 악영향을 줄이기 위해 본 논문에서는 FPGA의 OCuLink를 통해 연결된 NVMe SSD에 읽기와 쓰기를 호스트 CPU 및 FPGA에 존재하는 다른 CPU의 간섭없이 제어하였다.

본 논문에서 개발한 드라이버와 비슷한 기술로는 GPU Direct Storage가 있다. GPU Direct Storage는 FPGA가 아닌 GPU에서 직접 NVMe 저장장치를 읽고 쓰는 기술인데 다양한 분야의 딥러닝 애플리케이션에서 많이 사용되고 있다. 만약 GPU Direct Storage보다 throughput이 잘 나온다면 GPU와 FPGA 디바이스 메모리 이상의 데이터를 사용하고 메모리 집약적인 연산을 하는 딥 러닝 애플리케이션을 실행할 때 GPU보다 높은 성능을 기대할 수 있다. 그래서 본 논문에서는 FPGA에 NVMe 드라이버를 구현하고 최적화하여 GPU Direct Storage보다 높은 성능을 달성하였다. 실험 결과 NVIDIA Tesla V100을 사용한 GPU Direct Storage 대비 쓰기는 1.17배, 읽기는 약 1.2배 정도의 성능 향상을 보였다.

2. NVMe

NVMe는 Non-Volatile Memory Express의 약자로, SSD와 같은 플래시 메모리 기반의 저장장치를 컴퓨터 시스템에 연결하고 효율적으로 사용하기 위한 프로토콜이다. NVMe의 일반적인 읽기 및 쓰기 과정은 표 1에서 확인할 수 있다.

이러한 과정은 NVMe의 효율적인 데이터 처리 및 낮은 대기시간을 실현하는 데에 기여한다. 또한, Multi-Queue, 평행 처리 및 효율적인 인터럽트 처리 등의 NVMe의 특징이 기존의 SATA 인터페이스보다 훨씬 빠르고 효율적인 방식으로 데이터를 전송하는 데 사용된다.

표 2 NVMe의 읽기 및 쓰기 과정

① Doorbell Ring
Doorbell 레지스터는 메모리 공간으로, 호스트는 이를 업데이트하여 처리해야 할 command가 있음을 NVMe 디바이스에게 알린다.
② Doorbell 확인 및 command 처리
NVMe 디바이스는 doorbell값의 변화를 감지하고 호스트에 있는 command queue에서 새로운 command를 가져가 처리하기 시작한다.
③ 데이터 처리
읽기의 경우 디바이스는 데이터를 읽어와 호스트에게 반환하고 쓰기의 경우 호스트가 데이터를 디바이스에게 전송한다.
④ Command 완료
디바이스는 작업을 완료하고 호스트에게 완료되었다는 신호인 completion을 보낸다. 호스트는 completion queue를 확인하여 작업이 성공적으로 완료되었는지 확인한다.

3. NVMe SSD 드라이버 설계 및 구현

Xilinx Zynq UltraScale+ ZU19EG FFVD1760가 장착된 Bittware사의 250-SoC 및 삼성전자 970 EVO Plus 2TB를 사용하여 FPGA NVMe 드라이버를 구현하였다. Bittware 250-SoC에는 NVMe SSD를 연결할 수 있는 OCuLink 커넥터[4]가 있고 OCuLink를 통해 호스트 메모리를 거치지 않고 직접 NVMe SSD에 읽고 쓰기를 제어할 수 있다. OCuLink는 PCIe를 사용하므로 FPGA 제조사인 Xilinx에서 제공하는 DMA/Bridge Subsystem for PCI Express IP[5]를 사용하여 NVMe SSD에 있는 NVMe controller와 통신하였다. 해당 IP는 NVMe에서 OCuLink를 통해 오는 PCIe 신호를 AXI4-MM 인터페이스로 변환해준다. 본 논문

* 본 연구는 2023년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원(No. 2018-0-00581)과 한국연구재단의 지원(No. RS-2023-00222663)을 받아 수행된 연구입니다. 이 연구를 위해 연구 장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드립니다.

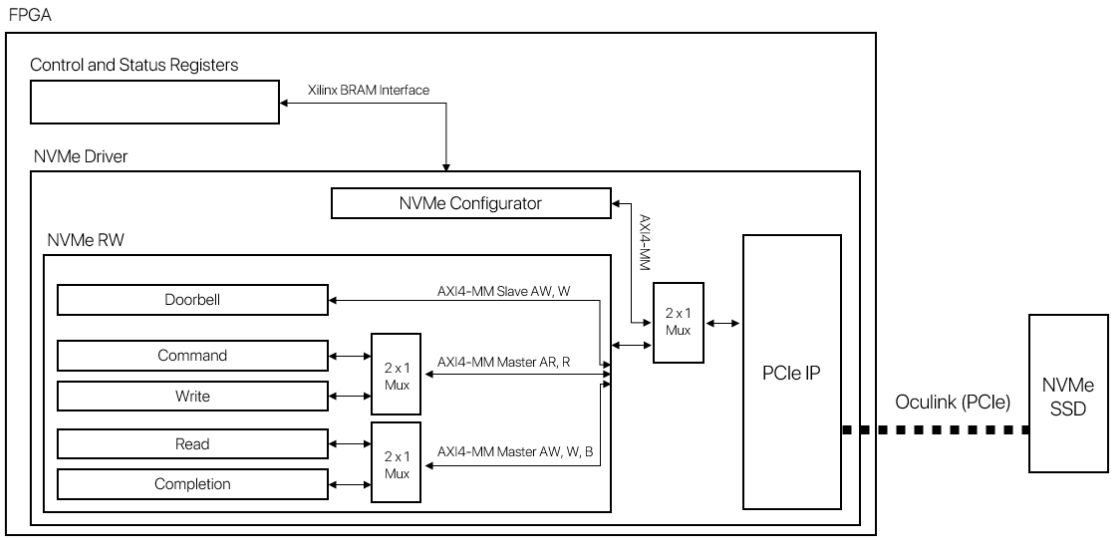

그림 1 FPGA NVMe 드라이버 Architecture

에서 구현한 NVMe 드라이버는 크게 PCIe enumeration 및 NVMe controller를 초기화하는 module인 NVMe Configurator와 읽기 및 쓰기 커맨드(Command)를 보낼 수 있는 module인 NVMe RW로 나뉜다. 드라이버의 전체적인 구성은 그림 1을 통해 확인할 수 있다.

3.1 NVMe Configurator Module

기존의 PCIe enumeration은 BIOS와 software로 되어있는 드라이버가 처리했었고 FPGA에서도 SoC로 묶인 CPU에서 enumeration을 할 수 있었다. 그러나 FPGA의 programming logic에 읽기와 쓰기 module을 구현해야 하므로 FPGA에서 직접 PCIe enumeration을 하였다.

PCIe enumeration 후에는 NVMe controller에 직접 접근이 가능하다. NVMe controller에 읽기와 쓰기 같은 I/O command를 보내기 위해서는, 먼저 admin submission queue와 completion queue를 생성해야 한다. 이를 위해 미리 만들어 둔 admin submission queue 및 completion queue의 크기와 base address를 NVMe controller의 register에 써야 한다. 이렇게 admin queue를 설정하면 NVMe controller가 admin submission queue 및 completion queue의 존재를 감지하고 admin command를 처리할 준비가 된다.

3.2 NVMe RW Module

Admin submission queue를 통해 NVMe controller에 admin command를 보내면 읽기와 쓰기에 필요한 I/O queue를 생성할 수 있다. 이를 위해 create I/O submission queue command와 create I/O completion queue command 두 가지 admin command를 사용한다. NVMe RW module을 통해 이 두 가지 admin command를 NVMe controller에게 전송하면, controller는 I/O submission queue와 I/O completion queue의 base address와 크기를 파악하게 된다. 이로써 controller는 I/O command를 처리할 준비를 완료하게 된다.

NVMe SSD와 같은 저장장치에 대한 읽기 및 쓰기 작업은 일반적으로 4단계의 프로세스를 거친다. ① Doorbell module에서 NVMe controller에 있는 doorbell register 값에 보낼 command 만큼 더한 값을 새로 쓴다. ② NVMe controller는 doorbell register 값의 변화를 인지해 FPGA에 I/O command를 요청한다. ③ Command module에서 I/O command 요청에 대한 응답으로 읽기/쓰기 command를 NVMe controller에 보낸다. ④ Write 및 Read module에서 command에 따라 쓸 데이터를 보내거나 읽을 데이터를 받는다. ⑤ 데이터 전송이 끝나면 Completion module에서는 NVMe controller에서 보내는 completion을 기다리고 다른 module은 다음 I/O command를 처리할 준비를 한다.

4. NVMe 드라이버 최적화

초기에 구현한 FPGA NVMe 드라이버는 3.2의 ①~⑤ 단계를 순차적으로 처리한다. 그러나 각 module 간에는 사전작업들이 있고 그 작업들은 다른 module이 동작 중인 상태에서 병렬로 실행되어도 문제가 없다. 이러한 특성을 고려해 각 module을 최적화하였고 높은 성능 향상을 보였다.

4.1 Doorbell Module과 Command Module 최적화

일반적인 NVMe 드라이버에서는 Doorbell module이 doorbell을 울리고 NVMe controller가 command를 요청하면 submission queue에서 요청 크기만큼의 command를 꺼내서 NVMe controller에 보내야 한다. 그러나 본 논문에서 구현한 NVMe 드라이버에서는 Doorbell module과 Command module은 서로 다른 port를 사용하고 독립적으로 동작할 수 있으므로 doorbell을 울릴 때 동시에 command를 생성하여 미리 준비해두고 NVMe controller가 command를 요청하면 생성했던 command를 바로 보내면 된다. 일반적인 드라이버보다 NVMe controller의 command 요청을 인지하고 submission queue에서 command를 꺼내서 보내는 과정을 없앨 수 있다.

4.2 Write Module 최적화

초기에 구현한 드라이버에서 쓰기는 NVMe SSD에서 쓸 주소를 보내면 FPGA에서 쓸 데이터를 보내고 다음 주소에 대한 쓰기를 처리한다. 쓸 데이터를 보내는 동안 새로운 쓰기 주소를 미리 받고 주소에 대응되는 데이터를 미리 꺼내는 프리패치(prefetch)를 사용하여 데이터 전송이 끝나면 바로 다음 데이터를 보내 쓰기 성능을 향상시킬 수 있다.

4.3 Read Module 최적화

초기에 구현한 드라이버에서는 읽기는 NVMe SSD에서 읽을 데이터를 보내면 AXI4-MM B 인터페이스를 통해 읽기에 대한 응답을 보내고 다음 읽기를 처리해야 한다. 읽기 응답에 관련

된 신호는 읽기 주소 및 읽을 데이터와는 다른 port를 사용하므로 새로운 주소 및 데이터를 받으면서 이전 읽기에 대한 응답을 병렬로 처리하면 읽기 성능을 향상시킬 수 있다.

5. 실험

본 논문에서 SystemVerilog로 구현한 드라이버의 성능을 확인하기 위해 NVIDIA사에서 제공하는 GPU Direct Storage의 성능과 비교하였다. FPGA와 NVMe SSD는 OCuLink로 연결되어있는데 NVMe SSD 1개당 PCIe 3.0 4레인을 사용하므로 동등한 실험을 위해 motherboard에 연결되어있는 V100에서 사용할 수 있는 PCIe 3.0 레인 수를 16레인에서 4레인으로 고정한 뒤에 실험하였다. 실험 결과에서 I/O크기는 한 command를 사용해 읽고 쓰는 데이터의 크기를 말한다.

표 3 System Configuration

CPU	AMD EPYC 7452
RAM	2 x DDR4 16GB 2133MHZ
FPGA	Xilinx Zynq ZU19EG
FPGA Board	Bittware 250-SoC
GPU	NVIDIA Tesla V100 16GB
Motherboard	Supermicro H12SSL-CT
NVMe	Samsung SSD EVO Plus 970 2TB

5.1 순차 쓰기 성능 비교

4.1, 4.2에서 최적화한 방법을 FPGA에 적용한 것과 GPU Direct Storage를 사용해 V100의 device memory에서 motherboard에 연결된 NVMe SSD에 쓰는 것의 throughput을 측정하여 비교하였다. 쓰기 I/O 크기를 256KB에서 4MB까지 증가시켜가며 10번씩 실행한 뒤에 평균값으로 throughput을 측정하였으며 latency도 같은 방식으로 측정하였다. 결과는 그림 2와 표 3에서 확인할 수 있다. 측정 결과 I/O size 4,096KB에서 GPU Direct Storage 대비 1.17배의 성능 향상이 있었다.

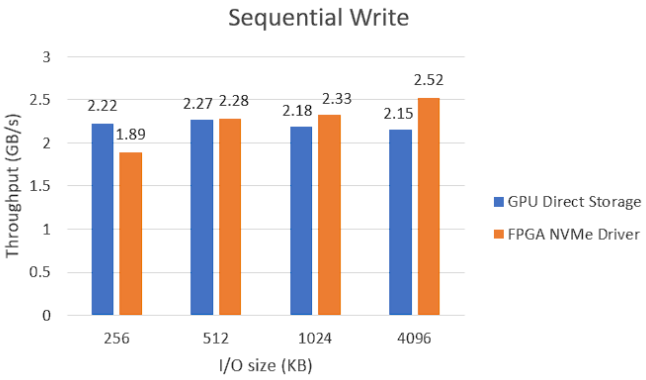

그림 2 I/O size에 따른 쓰기 throughput 측정

표 4 I/O size에 따른 쓰기 latency 측정 (단위: us)

I/O size (KB)	256	512	1024	4096
GPU Direct Storage	115.279	225.507	469.152	1906.42
FPGA NVMe Driver	135.517	224.561	439.485	1626.39

5.2. 순차 읽기 성능 비교

4.1, 4.3에서 최적화한 방법을 적용한 것과 GPU Direct Storage의 읽기 성능을 비교하였다. 쓰기와 같이 I/O size를 변화해가며 실험하였으며 결과는 그림 3과 표 4에서 확인할 수 있다. I/O size 4,096KB에서 GPU Direct Storage의 대비 1.2배의 성능 향상이 있었다.

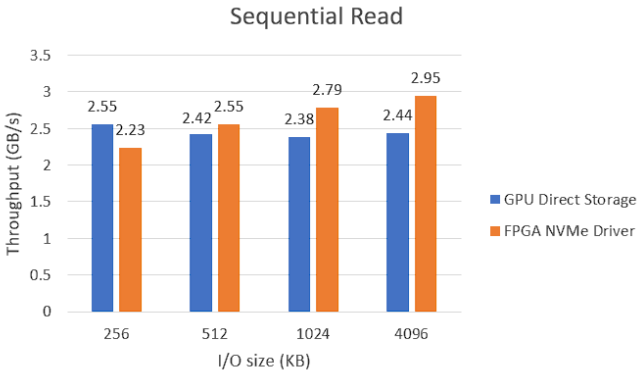

그림 3 I/O size에 따른 읽기 throughput 측정

표 5 I/O Size에 따른 읽기 latency 측정 (단위: us)

I/O Size (KB)	256	512	1024	4096
GPU Direct Storage	100.21	211.41	429.52	1676.1
FPGA NVMe Driver	114.56	200.44	367.16	1389.0

6. 결론 및 향후 연구

본 논문에서는 OCuLink를 통해 FPGA에 직접 연결된 NVMe 저장장치에 읽고 쓸 수 있는 드라이버를 구현하였다. NVIDIA GPU Direct Storage를 사용해 V100으로 NVMe SSD에 읽고 쓴 실험 결과와 비교했을 때, 쓰기 성능은 약 1.17배 높았으며 읽기 성능은 약 1.2배 높았다. 구현한 NVMe 드라이버는 딥 러닝 애플리케이션에 사용하여 LLM(Large Language Model)의 입력 및 가중치를 빠르게 읽고 쓰는 데 사용할 것이다.

참 고 문 헌

[1] Xilinx, “PG328 NVMe Host Accelerator,” 2021.

[2] A. Stratikopoulos, C. Kotselidis, J. Goodacre and M. Luján, “FastPath: Towards Wire-Speed NVMe SSDs,“ 2018 28th International Conference on Field Programmable Logic and Applications (FPL), Dublin, Ireland, pp. 170-1707, 2018, doi: 10.1109/FPL.2018.00036.

[3] NVIDIA, “GPUDirect Storage,” https://docs.nvidia.com/gpudirect-storage/index.html

[4] BittWare, “250-SoC Hardware Reference Guide,” 2021.

[5] Xilinx, “PG195 DMA/Bridge Subsystem for PCI Express Product Guide,” 2022.

[6] NVM Express, “NVMe over PCIe Transport Specification,” 2022.