빅데이터 분석 커널들에 대한 컴퓨팅 스토리지 드라이브의 성능 평가 및 최적화

변홍수[1], 한정욱[1], 이명철[2], 김창수[2], 김영재[1]
[1]서강대학교 컴퓨터공학과, [2]한국전자통신연구원 스마트데이터연구실
{byhs, immerhjw, youkim}@sogang.ac.kr, {mclee, cskim7}@etri.re.kr

Evaluating and Optimizing the Performance of Computing Storage Drives for Big Data Analytics Kernels

Hongsu Byun[1], Jungwook Han[1], Myungcheol Lee[2], Changsoo Kim[2], Youngjae Kim[1]
[1]Department of Computer Science and Engineering, Sogang University, Seoul, Republic of Korea
[2]Smart Data Research Section, ETRI, Daejeon, Republic of Korea

요 약

최근 Samsung과 Xilinx에서 출시한 SmartSSD는 대표적인 Computational Storage Drive(CSD)이다. 본 연구에서는 SmartSSD를 활용하기에 앞서 SmartSSD 내부 FPGA의 성능을 평가하고 최적화하여 비교 분석했다. 이를 위해 Xilinx에서 제공하는 Vitis 플랫폼 문서에 소개된 대표적인 최적화 기법들을 SmartSSD가 사용될 수 있는 여러 빅데이터 처리 워크로드에 적용하여 기존 대비 어느 정도 성능이 향상되는지 확인했다. SmartSSD에서 최대 11.2배 성능이 향상되는 기법이 있는 반면 1% 정도로 매우 낮은 성능 향상을 보이는 기법이 있었다. 우리가 소개한 모든 최적화를 SmartSSD에 적용하였을 때 Vector Addition 워크로드에서 기존 대비 최대 28.6배 성능이 향상되었고, Host CPU와 비교하여 Host CPU 클럭이 제한적일 때 Average 워크로드에서 최대 10.1배 높은 성능을 보였다.

1 서론

In-Storage Processing(ISP)는 SSD와 같은 저장장치 내부에서 내부 하드웨어 자원들(CPU와 Memory)을 사용하여 Host의 응용 전체 또는 일부의 작업을 SSD 내부에서 실행하는 기술이다 [1, 2]. ISP는 Host의 CPU 및 메모리 자원 사용을 경감시킬 뿐만 아니라 Host와 SSD간에 데이터 이동 비용을 줄인다. ISP는 주로 SSD 내부에 FPGA 가속기 연산 장치를 내장하는 방식으로 상용 제품이 개발되고 있다. 이러한 연산 가능 저장 장치를 컴퓨팅 스토리지 드라이브 (Computing Storage Drive)라고 부른다. 최근에는 Samsung과 Xilinx가 공동으로 SmartSSD를 출시하였다 [3]. SmartSSD는 SSD 내부에 FPGA 가속기를 탑재하고 있으며 Xilinx에서 제공하는 Vitis 플랫폼을 통한 OpenCL 커널 프로그래밍을 지원한다. 따라서 런타임시 미리 프로그래밍된 커널 파일을 실행하면 FPGA 가속기를 통해 데이터 처리의 수행이 가능하다.

ISP를 사용하여 Host의 데이터베이스 또는 빅데이터 처리 응용을 SSD 내부에서 실행하는 연구들이 있었다 [1, 2, 4, 5]. FCAccel [4]는 FPGA 가속기를 이용해서 계산 가능 SSD를 개발하고 열 지향 DBMS를 통합하여 SQL 연산자를 SSD로 오프로드하는 기술을 제안했으며, Aquoman [5]은 FPGA 가속기를 이용해서 계산 가능 SSD의 프로토타입을 개발하고 JOIN과 같은 일반 분석 질의를 처리하는 엔진을 제안했다. POLARDB [1]는 테이블 스캔 워크로드를 SSD에서 실행하는 기술을 연구하였으며 Cognitive SSD [2]는 딥 러닝 기반 비정형 데이터 검색을 위한 계산 엔진을 SSD에 탑재하여 에너지 효율성을 증대하는 검색 기술을 연구하였다.

Vitis 플랫폼 문서 [6, 7]에는 FPGA 가속기 성능을 극대화 하기 위한 다양한 최적화 프로그래밍 기법이 소개되어 있다. 본 연구에서는 그 중 SmartSSD 내부 FPGA에 적용 가능한 대표적인 3가지 기법(로컬 메모리 버퍼링, Loop Pipelining, Multiple CUs)을 고찰하고 다양한 빅데이터 분석 커널들에 대하여 SmartSSD의 성능을 평가 분석한다. 대표적인 실험 결과로 Vector Addition 커널에서 최적화 기법을 통해 기존 대비 최대 28.6배 성능이 향상되었고, Host CPU와 비교하여 Host CPU 클럭이 제한적일 때 Average 워크로드에서 최대 10.1배 높은 성능을 보였다.

이 논문은 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2021-0-00136, 다양한 산업 분야 활용성 증대를 위한 대규모/대용량 블록체인 데이터 고확장성 분산 저장 기술 개발)

2 배경 지식

2.1 SmartSSD 구조

그림 1은 SmartSSD의 아키텍처를 묘사한다. 2.5” 폼팩터의 SmartSSD는 크게 세가지 컴포넌트 (SSD 디바이스, FPGA 가속기, PCIe 스위치)으로 나눌 수 있다.

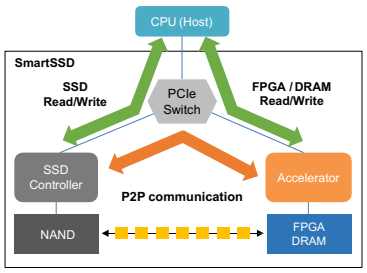

그림 1: SmartSSD Architecture [8]

첫째, SSD 디바이스는 SSD 컨트롤러와 NAND Flash를 포함한 디바이스이다. 일반적인 SSD와 동일하게 저장 장치로서의 역할을 수행한다.

둘째, FPGA 가속기는 직접 연결된 4 GB FPGA DRAM을 가지며 Host에서 오프로드 되는 응용을 수행한다. Xillinx에서 제조한 FPGA 가속기는 약 114.3만개의 Logic Cell로 구성되어 있고 300 MHz의 클럭 출력을 갖는다. FPGA 가속기에서 연산에 필요한 데이터는 FPGA 가속기의 DRAM에 적재된다. FPGA는 일반 ASIC (Application-Specific Integrated Circuit)과 비교해 개발시간이 짧고 재수정할 수 있어, 여러 응용에 최적화시킬 수 있다는 장점이 있다.

마지막은 PCIe 스위치이다. PCIe 스위치는 Host와 디바이스 사이의 데이터 이동을 담당한다. 또한 ISP 수행을 위해 NAND Flash와 FPGA 가속기 사이에 데이터 전송 시, Host를 거치지 않고 직접 P2P (Peer-to-Peer) 데이터 전송이 가능하다. 이때 Host는 FPGA에서 NAND에 저장된 데이터를 가지고 연산을 수행할 수 있는 명령만 FPGA에 내려주면 된다.

2.2 SmartSSD 프로그래밍

SmartSSD는 Xilinx에서 제공하는 Vitis 플랫폼을 통한 OpenCL 커널 프로그래밍을 지원한다. SmartSSD 프로그램 개발자는 OpenCL을 사용하여 FPGA에서 수행할 원하는 코드를 작성하고 Vitis 플랫폼에서 컴파일하여 FPGA 가속기에서 수행할 커널 바이너리파일을 생성한다. 응용은 런타임시 해당 커널 바이너리파일을 FPGA 가속기로 로드하고 실행 명령만 내려주면 SmartSSD 내 FPGA 가속기가 작업을 처리할 수 있다. 이는 사용자가 Verilog와 같은 하드웨어 수준의 언어를 알 필요가 없고, 응용의 실행 흐름에 맞춰 FPGA의 수행을 쉽게 조작할 수 있는 장점이 있다.

SmartSSD를 사용하여 SSD 내에서 두 개의 벡터 데이터 합을 계산하는 작업의 전체적인 흐름의 예시는 아래와 같다.

- 단계1: Vitis 플랫폼을 사용하여 벡터 합 커널 프로그래밍과 컴파일을 통해 커널바이너리 파일 생성함.
- 단계2: 응용 런타임에 커널바이너리 파일을 FPGA 가속기에 적재함.
- 단계3: SmartSSD 내부 SSD의 NAND에 있는 두 개 벡터 데이터를 FPGA 가속기의 DRAM으로 전송함.
- 단계4: 응용의 실행 명령을 통해 FPGA의 벡터 합을 계산하고 결과 값은 FPGA 가속기의 DRAM에 저장함.
- 단계5: 가속기의 DRAM에 저장된 결과 값을 SSD NAND에 저장함.

3 FPGA 성능 최적화

3.1 Local Memory Buffer

FPGA 가속기의 DRAM은 글로벌 메모리 영역과 로컬 메모리 영역으로 나뉜다. 이들은 FPGA 가속기가 동작할 때 구분되어 사용되며 용도는 다음과 같다.

- 글로벌 메모리: 커널 함수 인자로 넘어온 변수 데이터 적재
- 로컬 메모리: 커널 코드 내부에 선언된 변수 데이터 적재

Vitis 플랫폼 문서에 의하면 글로벌 메모리에는 오직 1개의 포트가 있기 때문에, 커널이 작동할 때 접근하는 변수가 2개 이상 글로벌 메모리에 적재되어 있는 경우 성능 저하를 초래할 수 있다. 즉, 커널 함수가 동시에 2개 이상의 넘겨받은 인자(글로벌 메모리 적재)에 대해 연산을 수행해야 한다면 커널 함수 내부에 버퍼(로컬 메모리 적재)를 사용하는 성능 최적화 방법을 권장한다.

알고리즘 1은 Vector Addition을 수행하는 커널 코드 예시이다. 위에 설명했듯이 커널은 함수 인자로 넘겨받은 데이터들에 대해서만 연산을 수행하므로 성능 저하가 발생한다. 알고리즘 2은 로컬 메모리 버퍼를 적용한 예시이다. 커널이 작동할 때 함수 내부에 선언된 버퍼를 사용하여 함수 인자로 넘겨받은 데이터에 대한 동시 접근이 줄어들게 된다. 따라서 메모리 복사(글로벌 → 로컬) 오버헤드를 감수하더라도 성능을 향상시킬 수 있다.

Algorithm 1: Vector Addition (Normal)

```
1  Function Krnl_Vadd(Array1[ ], Array2[ ], Result[ ], N):
       /* N is Length of Vector Array                    */
2      for i ← 1 to N do
3          Result_i ← Array1_i + Array2_i
```

Algorithm 2: Vector Addition (Local Memory Buffer)

```
1  Function Krnl_Vadd(Array1[ ], Array2[ ], Result[ ], N):
       /* N is Length of Vector Array                    */
2      Buf[M]  // Declare Kernel Local Array Variable
3      for i ← 1 to N do
4          for j ← 1 to M do
5              Buf_j ← Array1_{i+j}                  // Read
6          for j ← 1 to M do
7              Buf_j ← Buf_j + Array2_{i+j}          // Addition
8          for j ← 1 to M do
9              Result_{i+j} ← Buf_j                  // Store
10         i ← i + M
```

Algorithm 3: Vector Addition (Loop Pipelining)

```
1  Function Krnl_Vadd(Array1[ ], Array2[ ], Result[ ], N):
       /* N is Length of Vector Array                    */
2      for i ← 1 to N do
3          #pragma LOOP_PIPELINE II = 1
4          Result_i ← Array1_i + Array1_2
```

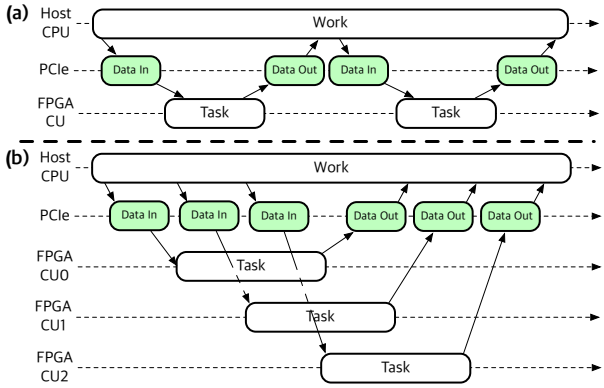

그림 2: Host 작업을 FPGA 가속기로 오프로딩 하여 처리할 때, (a) Single Compute Unit을 사용한 경우와 (b) Multiple Compute Units을 사용한 경우에 대한 Workflow 묘사

3.2 Loop Pipelining

알고리즘 3 Loop Pipeline 최적화 기법을 적용한 커널 코드 예시이다. 커널 코드 Loop 내부에 관련 코드인 **#pragma** LOOP_PIPELINE II = 1을 추가로 작성하면 된다. II는 Initiation Interval의 약자로 Pipeline을 나눌 Cycle단위를 설정한다.

3.3 Multiple Compute Units

FPGA 가속기는 Multiple Compute Units(CU)를 지원한다. 그림 2(a)와(b)는 각각 Single CU와 Multiple CU를 사용하여 Host의 작업을 FPGA 가속기로 오프로딩 했을 때 Workflow를 묘사한다. Single CU를 사용한 경우(그림 2(a)), Task가 수행될 때마다 Host와의 데이터 전송을 해야 할 뿐 아니라 각 Task는 직렬적으로 수행된다. 반면, Multiple CU를 사용한 경우(그림 2(b)), 각 CU는 독립적으로 Task 처리가 가능하므로 여러 작업을 병렬적으로 처리하여 전체적인 성능을 향상 시킬 수 있다.

4 실험 및 평가

우리는 빅데이터 분석 커널 워크로드들을 섹션 3에서 설명한 FPGA 성능 최적화를 SmartSSD (CSD)에 적용했을 때 성능 평가를 비교 분석한다. 표 1은 SmartSSD의 상세 세부 스펙을 설명한다. 사용한 빅데이터 분석 커널 워크로드들은 다음과 같다. (n)은 커널이 작동할 때 주로 접근해야 하는 커널 함수 인자 개수를 의미한다.

- Count(1): 1개 정수형 배열에서 특정 값 개수 확인
- Average(1): 1개 정수형 배열 원소들의 평균 계산
- Vector Addition(3): 2개 정수형 배열 각 원소 합으로 구성된 새로운 배열 1개 생성
- Array Merge(3): 정렬된 2개 정수형 배열에서 중복 원소를 제거하며 합친 새로운 배열 1개 생성

결과 및 분석: 그림 3은 Vector Addition 워크로드에서 로컬 메모리 버퍼를 적용하지 않았을 때 (Normal)를 기준으로 버퍼 크기에 따른 성능 향상 비교를 나타낸 결과이다. 버퍼 사이즈가 1KB일 때 최소 8.8배, 256KB일 때 최대 11.2배 성능이 향상되었다. 버퍼 크기가 증가함에 따라 성능이 증가하는 이유는 워크로드의 최상위 Loop을 수행하는 횟수가 줄어들기 때문이다. 그러나 버퍼 사이즈가 256KB를 초과하면 오히려 성능이 감소하는데, 이는 메모리 복사(글로벌 → 로컬) 오버헤드가 커지기 때문이다.

그림 4(a)는 각 워크로드에 256KB 로컬 메모리 버퍼를 적용했을 때 기존(Normal) 대비 성능 향상을 정규화한 비교 결과이다. Count와 Average

Storage Capacity	3.84TB	
Host Interface	Single Port PCIe Gen3x4	
NAND Flash Memory	Samsung V-NAND	
Programmable Hardware Accelerator	Xilinx Kintex Ultrascale+ KU15P FPGA	
	DRAM	4GB DDR4
	Clock	300MHz

표 1: SmartSSD에 대한 세부 상세 설명

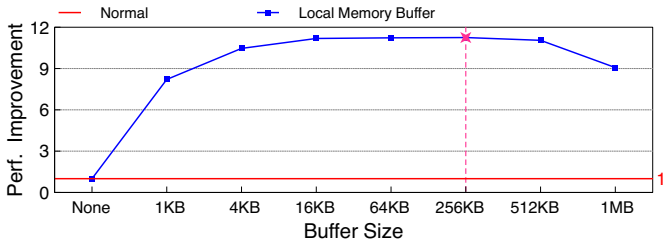

그림 3: Local Memory Buffer 크기에 따른 성능 비교

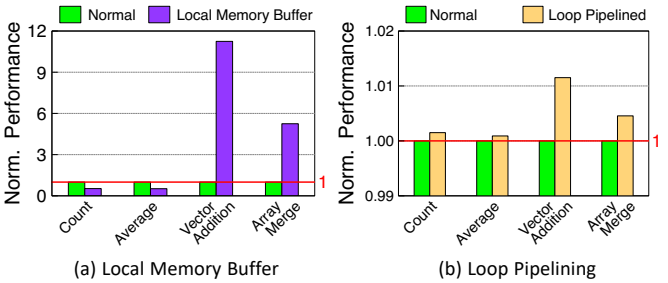

그림 4: 워크로드별 (a) Local Memory Buffer 사용 성능 비교와 (b) Loop Pipelining 사용 성능 비교

의 경우 연산을 수행할 때 커널 함수에서 넘겨받은 인자 중 1개 배열 변수에 접근하므로 글로벌 메모리 접근으로 인한 성능 저하가 발생하지 않는다. 따라서 로컬 메모리 버퍼를 적용하면 오히려 메모리 복사 오버헤드(글로벌 → 로컬)로 인해 성능이 감소한다. 반면 Vector Addition과 Array Merge의 경우 커널 함수에서 넘겨받은 인자 중 3개의 배열 변수에 접근해야 하므로 글로벌 메모리 접근으로 인한 성능 저하가 발생한다. 따라서 로컬 메모리 버퍼를 사용했을 때 각각 11.2배, 5.2배로 높은 큰 성능 향상을 보인다.

그림 4(b)는 Loop Pipelining 기법을 적용한 성능 비교 결과이다. Loop Pipeline의 경우 Vector Addition에서 최대 1%정도로 매우 낮은 성능 향상을 보이고 나머지 워크로드에서도 거의 성능 향상이 없었다.

그림 5는 Multiple CU를 적용한 실험 결과이다. SmartSSD에서 Multiple CU는 최대 15개까지 수행할 수 있음을 실험적으로 확인했다. 그 이유는 SmartSSD 내부 FPGA 메모리는 1개의 Bank를 갖고 있으며, Bank는 최대 15개의 인터페이스를 제공할 수 있기 때문이다. 따라서 CU개수를 최대 15개까지 늘려가면서 실험을 진행하였다.

그림 5(a)는 CU의 개수를 증가시켜가며 Single CU만 사용했을 때를 기준으로 정규화한 성능 향상을 나타낸 결과이다. 연산이 적은 워크로드(Count, Average, Vector Addition)의 경우 CU가 5개 정도에서 각각 3.5배, 3.8배, 2.5배로 최대 성능 향상을 보이고 이후로 불규칙한 변화를 보이는 반면, 상대적으로 연산이 많은 워크로드(Array Merge)는 CU의 개수가 증가할수록 지속적으로 성능이 증가하여 CU가 15개 일때 최대 1.7배 성능이 향상되었다. 즉, 연산이 많은 워크로드일수록 CU의 개수가 늘어남에 따라 Multiple CU 사용 효율이 높아진다.

그림 5(b)는 Multiple CU를 사용할 때 CU 개수에 따른 각 CU의 성능 감소를 나타낸 결과이다. 모든 워크로드에서 CU의 개수가 늘어남에 따라 각 CU의 성능은 떨어진다. 그렇지만 CU의 개수가 N개일 때 각 CU의 성능(pf)이 떨어지더라도 $\frac{1}{N}$보다 높음을 확인했다. $\sum_{i=1}^{N} pf_i \geq 1$을 만족하므로 그림 5(a)와 같이 전체적인 성능은 기존보다 향상될 수 있다. 즉, Multiple CU 사용하면 각 CU의 성능이 감소 하더라도, 전체 성능은 Single CU를 사용했을 때보다 향상시킬 수 있다.

그림 6(a)는 각 워크로드에 본 연구에서 소개한 FPGA 가속화의 최적화 방법을 모두 적용해본 결과이다. 모든 워크로드에 대해서 Loop Pipelining을 적용한 경우는 다른 최적화 방법에 비해 상대적으로 매우 낮은 향상을 보이므로 최종 결과에서는 나타내지 않았다. Count, Average의 경우 로컬 메모리 버퍼는 오히려 성능 저하가 발생하기 때문에 Multiple CU만 적용하였고, Vector Addition과 Array Merge는 로컬 메모리 버퍼와 Multiple

CPU	Intel(R) Core(TM) i7-8700K 3.7GHz (Up to 4.7GHz)
Memory	32GB DDR4
OS	Centos 7.92.2009 (Core)

표 2: Host 서버 세부 상세 설명

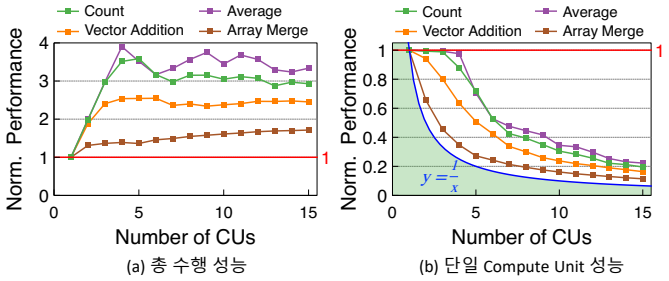

그림 5: CU 개수 증가에 따른 (a) 총 성능 증가와 (b) 각 CU 성능 감소 비교

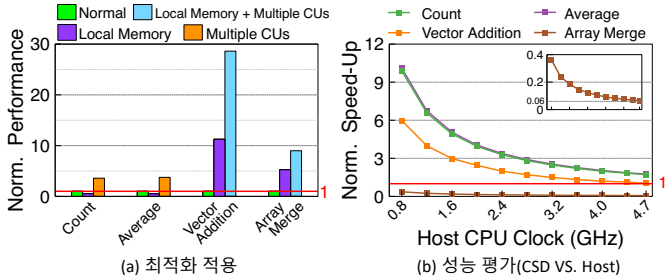

그림 6: 워크로드별 (a) 최적화 적용 성능 비교와 (b) Host CPU 클럭 변화에 따른 SmartSSD와 Host 성능 비교 평가

CU를 모두 적용하였다. 그 결과 Count, Average, Vector Addition, Array Merge 워크로드에서 각각 기존 대비 3.5배, 3.8배, 28.6배, 9.0배 성능이 향상되었다.

그림 6(b)는 본 연구에서 소개한 모든 최적화를 적용한 FPGA 가속기와 Host CPU의 성능 비교이다. FPGA 가속기 성능을 기준으로 Host CPU 성능을 정규화 하였다. 표 2는 Host 서버의 상세 세부 스펙을 설명한다. Host CPU 클럭을 0.8GHz부터 4.7GHz까지 증가시켜가며 각 워크로드를 단일 코어, 단일 쓰레드로 실행했다. 연산이 적은 워크로드(Count, Average, Vector Addition)의 경우 Host CPU 최대 클럭에서도 FPGA 가속기 성능이 좋았으며 클럭이 제한적일 때 Average 워크로드에서 최대 10.1배 좋은 성능을 보인다. 연산이 많은 워크로드(Array Merge)의 경우 Host CPU 클럭이 가장 낮을 때도 FPGA 가속기의 성능이 더 낮았으며 최소 2.7(1/0.36)배, 최대 16.6(1/0.06)배 낮은 성능을 보였다.

5 결론

본 연구에서는 SmartSSD의 내부 FPGA 가속기의 성능 최적화 방법을 소개하고 적용하여 평가 및 비교 분석했다. 이를 위해 로컬 메모리 버퍼, Loop Pipelining, Multiple CU 기법을 고찰하고 다양한 빅데이터 분석 커널들에 대해서 성능 향상을 확인했다. 모든 최적화를 적용하였을 때 Vector Addition 워크로드에서 기존 대비 최대 28.6배 성능이 향상되었고, Host CPU와 비교하여 Host CPU 클럭이 제한적일 때 Average 워크로드에서 최대 10.1배 높은 성능을 보였다.

참고 문헌

[1] W. Cao, Y. Liu, Z. Cheng, N. Zheng, W. Li, W. Wu, L. Ouyang, P. Wang, Y. Wang, R. Kuan, Z. Liu, F. Zhu, and T. Zhang, "POLARDB Meets Computational Storage: Efficiently Support Analytical Workloads in Cloud-Native Relational Database," in *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST)*, p. 29–41, USENIX, 2014.

[2] S. Liang, Y. Wang, Y. Lu, Z. Yang, H. Li, and X. Li, "Cognitive SSD: A Deep Learning Engine for In-Storage Data Retrieval," in *Proceedings of the 2019 USENIX Annual Technical Conference (ATC)*, p. 395–410, USENIX, 2019.

[3] "SmartSSD." https://samsungsemiconductor-us.com/smartssd/, Accessed 2020-10-25.

[4] S. Watanabe, K. Fujimoto, Y. Saeki, Y. Fujikawa, and H. Yoshino, "Column-oriented Database Acceleration using FPGAs ," in *Proceedings of 2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pp. 686–697, 2019.

[5] S. Xu, T. Bourgeat, T. Huang, H. Koim, S. Lee, and Arvind, "AQUOMAN: An Analytic-Query Offloading Machine," in *Proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture(MICRO)*, pp. 386–399, 2020.

[6] "Vitis Document." https://www.xilinx.com/html_docs/xilinx2021_1/vitis_doc/, Accessed 2020-10-25.

[7] "Vitis Accel Examples." https://xilinx.github.io/Vitis_Accel_Examples/, Accessed 2020-10-25.

[8] J. H. Lee, H. Zhang, V. Lagrange, P. Krishnamoorthy, X. Zhao, and Y. S. Ki, "SmartSSD: FPGA Accelerated Near-Storage Data Analytics on SSD," in *IEEE Computer Architecture Letters*, pp. 110–113, 2020.