Master's Thesis of Yunho Jin

Cost-effective Extreme-scale DNN Inference on a Flash-based Memory System

플래쉬 기반 효율적 초대규모 인공신경망 모델 추론 시스템

August 2021

Graduate School of Engineering
Seoul National University
Computer Science and Engineering

Yunho Jin

Cost-effective Extreme-scale DNN Inference on a Flash-based Memory System

Advisor Jae W. Lee

Submitting a master's thesis of Engineering

July 2021

Graduate School of Engineering
Seoul National University
Computer Science and Engineering

Yunho Jin

Confirming the master's thesis written by Yunho Jin

July 2021

Chair	Kunsoo Park
Vice Chair	Jae W. Lee
Examiner	Byung-ro Moon

Abstract

The size of deep neural network (DNN) models has been exploding rapidly, demanding a colossal amount of memory capacity. For example, Google has recently scaled its Switch Transformer to have a parameter size of up to 6.4 TB. However, today's HBM DRAM-based memory system for GPUs and DNN accelerators is suboptimal for these extreme-scale DNNs as it fails to provide enough capacity while its massive bandwidth is poorly utilized. Thus, we propose *Leviathan*, a DNN inference accelerator, which integrates a cost-effective *flash-based* memory system, instead. We carefully architect the flash-based memory system to provide enough memory bandwidth while preventing performance drop caused by read disturbance errors. Our evaluation of Leviathan demonstrates a 2.39× throughput gain compared to the iso-FLOPS DNN accelerator with conventional SSDs and up to 19.47× higher cost-efficiency than the HBM-based DNN accelerator.

Keywords: Neural Network, Flash Memory, Read Disturbance
Student Number: 2019-22004

Table of Contents

Chapter 1. Introduction

Deep neural networks (DNNs) have become more than just cutting-edge technology. They have now permeated diverse domains such as natural language processing (NLP) ([16], [36], [38], [44]), computer vision ([24], [52], [61]), and recommendation systems ([40]). One of the most recent trends in DNNs is the increase in their model sizes. In particular, Transformer-based models ([57]) have been explosively expanding their sizes. For example, BERT ([16]) has hundreds of millions of parameters, translating to 1.36 GB memory space requirements. Similarly, GPT-3 ([7]) requires 350 GB memory space, and the most recent Switch Transformer ([17]) requires 6.4 TB memory space. This scaling of model size poses a significant challenge in the existing DNN inference systems having a relatively small memory capacity of tens of GBs.

One straightforward way to secure more capacity is to increase the number of sockets (i.e., scaling out). For example, to house a GPT-3 model requiring 350 GB memory capacity, one needs at least five NVIDIA's A100 GPUs ([1]), each having 80 GB memory to secure 400 GB memory space. However, such a use of multiple devices incurs a high hardware cost. This is because HBM-based memory in GPUs is costly for their high memory bandwidth; however, this high memory bandwidth is underutilized for those extreme-scale DNNs as their bandwidth requirements are often much smaller due to a higher degree of data reuse ([32]).

NAND-based flash memory is an attractive alternative to the HBM-based memory system for such large-scale DNN models as it provides a large memory capacity at a low cost ([4]). However, several challenges need to be addressed. First, a flash device exhibits far lower bandwidth than HBM DRAM and what is necessary for DNN inference. For example, it takes 50 seconds to read all weights in GPT-3 from a commercial SSD providing 7 GB/s ([49]), which is too

slow. Second, as DNN inference necessitates repeated reads to the devices, a chronic problem in NAND flash known as *read-disturbance (RD)* ([6], [9], [21], [37], [53]) is exacerbated and causes severe deterioration in their performance. RD-induced errors are typically addressed with read reclaim (RR) ([9], [23], [29]). RR in the prior work is designed for a generic case of non-uniformly distributed reads across blocks. However, in a DNN inference system, memory reads are distributed uniformly across blocks to reach the threshold of RD-induced errors and call RRs simultaneously. These bursty RRs cause a significant drop in flash throughput to increase performance variability.

To address these challenges, we present *Leviathan*, a NAND flash-based DNN inference accelerator specialized for extreme-scale DNNs. To boost the bandwidth of the storage system, we conduct vertical optimization from the host to the storage system by leveraging the domain-specific characteristics of DNN inference. Specifically, Leviathan exploits the read-intensive nature and static data-access pattern to facilitate flash optimizations such as cache-read and multi-plane operations for user I/O. Leviathan also resolves performance degradation caused by RD. Identifying that bursty RR operations are the cause, Leviathan effectively distributes them over time in the course of multiple DNN inference requests. Thus, it smooths out the peak bandwidth demanded from RR operations and minimizes interference with user I/Os for computation.

Chapter 2. Background and Motivation

2.1 Extreme-scale DNN Inference

To inference a DNN model is to exploit a trained model to carry out certain tasks it was trained for. It does so by multiplying learned weights and input data which is being propagated through the model in the form of activations. One interesting aspect of this process is that the dataflow strictly adheres to a predefined path. Therefore, it is easy to sequentialize data accesses. Also, the task features a WORM (write-once-read-many) access pattern. It reads many times from weight data that is written once. This is especially helpful in flash memories where a typical read is over 10× faster than write. Leviathan exploits these properties to boost its flash bandwidth.

2.2 Reading from NAND-based memory

GPT-3 model introduced in 2020 contains 175 billion parameters, whereas BERT model introduced in 2018 contains merely 110 million parameters. Just within two years gap, the size of these Transformer-based models has scaled by more than 1000×. To account for the enormous memory capacity demand from the extreme-scale models, recent literature has been shifting the direction to heterogeneous memory ([5, 32, 48, 58]). Specifically, solutions using NAND-based SSDs ([5, 32]) have been announced. With gigantic memory capacity, these solutions adequately addressed the memory capacity wall problem induced by the extreme-scale models. However, the NAND-based storage also induces a few problems of its own when being exploited to accommodate DNN models for inference. In this section, we first provide some necessary background on reading from NAND flash memory. Next, we discuss read disturb, a common error which is more susceptible in read-intensive workloads such as DNN inference.

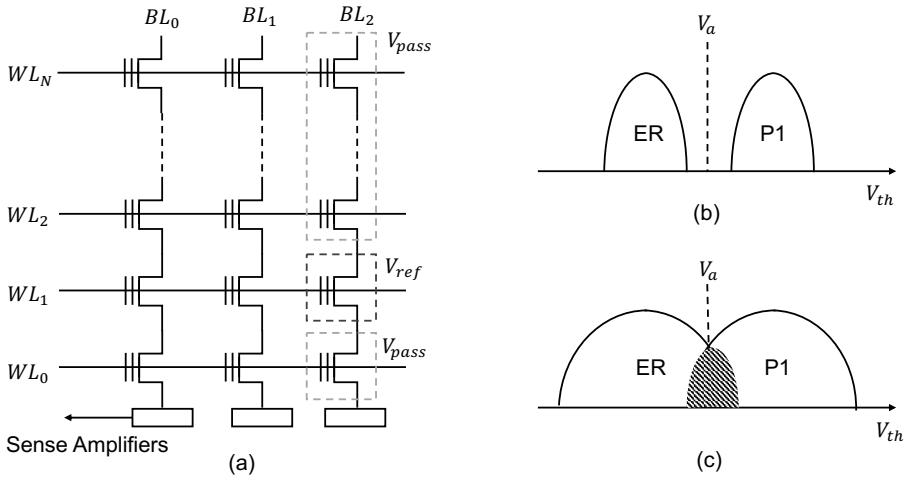

Figure 2.1: Reading from NAND array and read disturbance during the process

Read path. A typical NAND array in flash memory is laid out as a grid, as in Figure 2.1(a) referred to as a *block*. To read from this block, a read reference voltage V_{ref} is applied to the target wordline (WL) that contains the *page* to be read. The cells that are programmed with threshold voltages (V_{th}) lower than V_{ref} will be turned off while others will be turned on. Sense amplifiers at the bottom of the array will sense whether or not the cell placed on their bitline (BL) is turned on and send the message to the memory controller.

This is the case of single-level cell (SLC) where each cell can represent only one bit, namely two states. This is a simplification compared to its counterparts such as multi-level (MLC) or triple-level (TLC) cells. These cells must apply multiple different voltages to read a page in a block. Taking MLC which can represent two bits, or four states, as an example, there are 3 V_{th}'s that separate the four states in equal portions. First V_{ref} that is equal to the middle V_{th} is applied. The least significant bit (LSB) pages are determined in this step. Afterward, other two V_{th}'s are applied sequentially to determine the most significant bit (MSB) pages. With V_{ref} applied three times, reading from MLCs induces at least 3× latency and 7× for TLCs.

As mentioned earlier, the cell to be read from is connected to the sense amplifier via a BL. This BL contains cells in other WLs. Thus, to read from

a cell, all other cells in the same BL must be turned on to allow current flow into the sense amplifier. To turn the other cells, pass-through voltage (V_{pass}) is applied to all other cells as in Figure 2.1. V_{pass} is set as the maximum possible V_{th} to ensure that all cells receiving V_{pass} are turned on.

Read Disturbance (RD). Read disturb is an inherent problem in SSDs ([6], [9], [21], [37], [53]). Reading a page from a block necessitates applying V_{pass} to all the other WLs. Applying such a high voltage to a cell will degrade its dioxide which in turn shifts the threshold voltage due to charge loss. Figure 2.1(b) shows the original distribution of cells right after being programmed. Cells whose threshold voltage is lower than V_a are sensed as in erased (ER) state with a bit value of 1 and the others are sensed as P1 state with bit value of 0. With the shift of V_{th} of cells due to repeated appliance of V_{pass}, the distribution will be expanded as in Figure 2.1(c). When the reference voltage of V_a is applied to the cells, cells in the dashed region will be read incorrectly. Namely, cells originally in P1 state will be read as in ER state, and cells originally in ER state will be read as in P1 state. Thus *read disturb errors* have been generated.

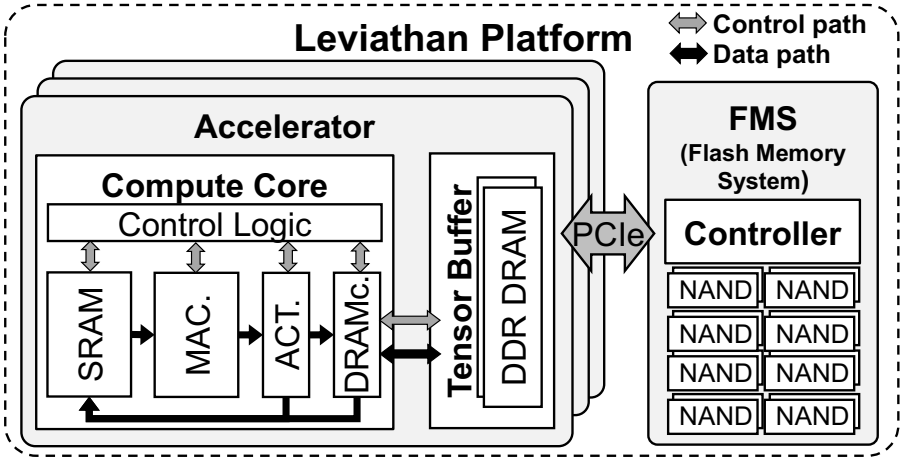

Figure 3.1: Overview of Leviathan

Chapter 3. Leviathan Overview

Overview. Leviathan is a specialized DNN inference platform designed to accommodate extreme-scale models. It employs a low-cost storage medium that satisfies the bandwidth demand of the workloads. As depicted in Figure 3.1, Leviathan has multiple accelerators with DDR DRAM in place of HBM. Each accelerator is connected to Leviathan Flash System (LFS) via 64 lanes of Gen4 PCIe (42).

DNN inference in Leviathan. Initially, a host analyzes the requested DNN model (2, 43) to confirm that Leviathan's LFS can provide the bandwidth demanded by the model inference. If so, it writes the trained model weights to Leviathan's LFS. Afterward, the host generates computation and DMA command sequences for each accelerator and LFS to execute the model. It then passes the commands to Leviathan's LFS, and the sequences guide operations in the system. This sequence is stored in a separate stream (32) of LFS from storing weight data.

The commands are first distributed to the accelerators they are assigned to. Each accelerator is assigned a part of the model so that multiple accelerators

can process a single model in a model-parallel manner. For example, when three accelerators are attached to a single LFS, a fully connected layer in the form of $3M \times N$ matrix is partitioned into three $M \times N$ matrices and distributed to each accelerator. The result is accumulated in the last accelerator, which is distributed to the Tensor Buffers of the others for the computation of the next layer.

The inside of an accelerator is depicted in Figure 3.1. Control Logic first sequences the computation and DMA commands it was assigned. Computation commands are issued to components in Compute Core after their dependencies are met. Leviathan resembles the architecture of popular DNN accelerators ([10], [11], [12], [28]) designed solely for multiply-accumulate (MAC) operations. This MAC array is a massive grid of processing elements, each carrying out one MAC operation every cycle. At every cycle, a part of activation is pumped into the MAC array from SRAM buffer and multiplied to the stationed weights. The generated result is then passed onto activation unit (ACT) if the layer in computation contains an additional activation layer ([3], [20]) to apply the corresponding activation function. The final product is saved in SRAM to maintain a copy of the result and distributed to other accelerators through Tensor Buffers.

DMA commands are issued to LFS controller, which corresponds to the conventional SSD controller, and DRAM controller (DRAMc) to load weights from LFS to Tensor Buffers, then to SRAM in each accelerator respectively. The weights are finally stationed in the MAC arrays. Tensor Buffer is a DRAM buffer that acts as a staging area for weight data to smooth the traffic between accelerator and LFS. LFS is a storage system designed to accommodate extreme-scale models with its large capacity while at the same time providing high enough bandwidth to support DNN inference. Both read and write datapath of LFS is automated by hardware logic based on a DNN-specific lightweight FTL ([32]). LFS is further delineated in Section 4.

Chapter 4. Leviathan Flash System

The main challenges in architecting NAND flash-based memory system (LFS) for extreme-scale DNN inference are insufficient read bandwidth and acceleration of read disturb (RD) errors due to the WORM access pattern. This section describes how Leviathan FMS provides hundreds of GB/s read bandwidth without deterioration from RD-induced errors caused by the WORM pattern.

4.1 Boosting Read Bandwidth

NAND product with over 2.0GT/s (i.e., 2.0Gbps ×8 DDR bus) I/O bandwidth ([14], [25]) and controller with automated read path ([13]) have been introduced to support the high bandwidth in various storage applications. But this still falls short of providing hundreds of GB/s required by DNN inference workloads. Therefore, Leviathan augments the high bandwidth NAND device and read path hardware automation with additional optimizations. Specifically, Leviathan adopts widely known yet un- or partially used cache-read and multi-plane operations to elicit the full potential of NANDs.

Cache-read operation. Cache read pipelines data read through channel and sensing from NAND page to data register ([41]). Figure 4.1(a) and (b) show page read without and with cache-read operation, respectively. Through the cache-read operation, the channel transfer time is hidden by the page sensing time. Most modern NAND chips employ multiple registers and additional commands to enable cache read ([14], [30], [41]).

In spite of the availability of the hardware logic, most commodity SSDs cannot use cache-read. This is because the SSD firmware or controller must delay the current read I/O and append the next one unless multiple read I/Os

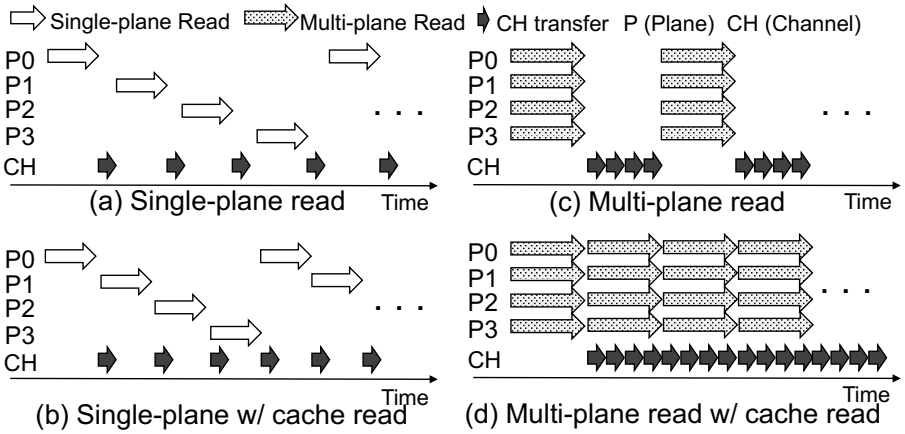

Figure 4.1: A NAND chip read pipeline diagram according to read methods

are pending on the same plane ([27]). This demand for an extraneous scheduler increases the read I/O latency ([59]). Thus, conventional SSDs do not employ cache-read since they emphasize lowering latency for more common I/Os (e.g., small-sized random read with low queue depth (QD)) than large-sized high-QD sequential reads.

In contrast, Leviathan only issues sequential read requests with high QD and in large chunks to its LFS (e.g., 2MB command × 12 QD). It always requests more than two pending read I/Os to a particular plane at any given time. Therefore, LFS does not incur additional read latency while issuing cache-read commands. This operation, coupled with multi-plane read operation based on plane/channel/way (PCW) striping ([54]), dramatically increases the effective bandwidth.

Multi-plane operation. Multi-plane operation is a widely known technique that partitions a NAND die into multiple planes to extract more parallelism. The pages are sensed from multiple pages in parallel as depicted in Figure 4.1(c). This technique is especially attractive because it significantly increases the write bandwidth in NANDs where a program operation is over tens times slower than a read operation. Also, it enhances the read/erase/program efficiency of garbage collection (GC) operations ([50]).

Unfortunately, this useful technique is restricted to reads in GC (i.e., non-user read). There are three reasons for this. (1) The multi-plane read (and

program) operation is only allowed for pages that have the same page offset in NAND blocks of the adjacent plane ([26]). This restriction makes it highly unlikely that the pages, which can be read by multi-plane operation at a specific time, are the data a user wants to read. That is, unless the user's read request refers to the pages in the order in which they were written (e.g., sequential read after strict append-only sequential write). Additionally, (2) a multi-plane read operation is slower than a single-plane read operation (i.e., bandwidth oriented using large chunk) and (3) most commercial SSDs adopt channel/way/plane (CWP) striping to maximize channel utilization and minimize latency during user data reads ([19]).

Figure 4.2 shows the difference between typical CWP striping and Leviathan's plane/channel/way (PCW) scheme that allows multi-plane read operation for user data. In an SSD with M channels, N ways, and K planes, using CWP scheme results in a distance of MN between two adjacent planes even in sequential writes as in Figure 4.2(a). This adds back pressure to the controller or firmware because the initial read request to a chip has to wait for $MN \times (K-1)-1$ subsequent read requests to request all pages in a particular stripe. In contrast, as illustrated in Figure 4.2(b), sequential writes to a PCW-striped SSD will have planes with contiguous pages. Therefore, reading from the PCW-striped SSD will allow multi-plane read also for user data if the data is written in sequential manner. As shown in Figure 4.1(d), the collaboration of cache-read and multi-plane operation will boost the effective bandwidth of the Leviathan.

4.2 Preventing RD-induced Performance Degradation

RD errors in DNN inference. Although DNN does provide opportunities for specializing LFS to increase throughput, it is a double-edged sword. The intensive reads and their WORM pattern accelerates the RD-induced reliability problem in the flash-based memory system.

The WORM nature of DNN inference quickly fills the RD threshold. All NAND blocks in the flash memory experience almost the same degree of read stress as all weights are read once for a single inference. Therefore, every NAND block containing model weights exceeds the RD threshold *at the same time*

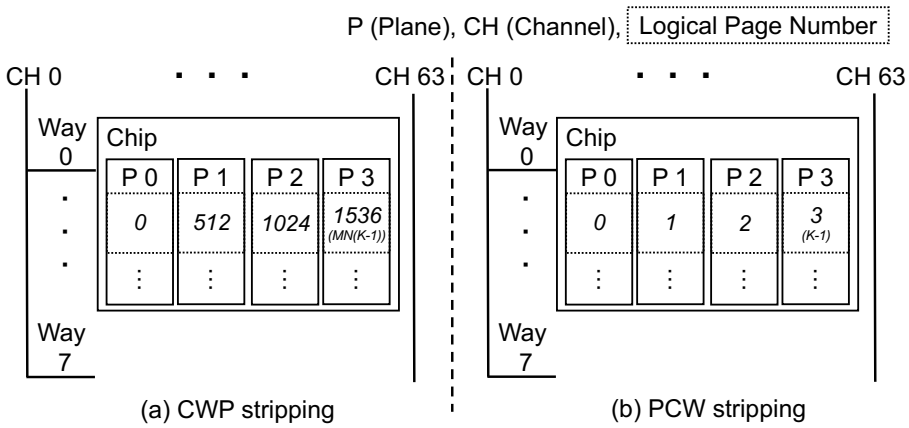

Figure 4.2: Logical page mapping with two striping schemes ((a) CWP, (B) PCW) after sequential write on an SSD with 64 channels (M=64), 8 ways (N=8), and 4 planes (K=4).

and necessitates read reclaim (RR) operation simultaneously. Such bursty RR operations firing over a small time window severely impede the user because RR for blocks that exceeded RD threshold must be given higher priority than user read requests. In this case, RR operations significantly degrade the device's performance, thereby worsening end-to-end quality-of-service (QoS). Note that this degradation continues until every block is reclaimed. Thus, we propose a new RR smoothing technique that effectively eliminates the performance degradation induced by the bursty RR operations.

RR timing. The main difference between the conventional and Leviathan's RR procedures is that the former migrates *after* and the latter migrates *before* the read count reaches RD threshold ([37]). The static, deterministic pattern of data accesses in DNN inference enables accurate prediction of read counts (i.e., RD stress) for all blocks. Exploiting this characteristic, Leviathan issues RR operations alongside user read requests before read count in any block exceeds the RD threshold. To put it another way, LFS gives the same priority to RR requests and inference reads, unlike the RR priority scheduling in conventional SSDs. The distribution is carefully tuned to prevent any inference performance degradation and reprogram every NAND blocks only once before reaching the RD threshold.

The RR timing is calculated by dividing the RD threshold associated with a P/E cycle by pages per block. For example, 768 pages per block and 6M read counts from 75K P/E cycles require full read reclaim of all blocks for every 7813 inference requests. Note that NAND block allocation policy for RR is based on an existing proposal ([32]).

LFS resource reservation for RR. Though RR requests in the above example is not frequent, the corresponding bandwidth must be reserved by the SSD for execution of DNN inference without interruption via effective RD resolution. Unlike the DNN inference workload that only issues read requests sequentially, a RR operation, which is a type of GC, *reads* source pages, *erases* a destination block, and *programs* into the block. This resource consumption calls for the need to pay careful attention to factors such as NAND topology, channel bandwidth, FTL scheme, and degree of hardware automation. Specifically, user read I/O to a NAND chip is blocked for a few milliseconds during the block erase operation for RR operation; thus a read prefetch buffer of appropriate size is required for a host system ([5]) or an SSD. Otherwise, erase suspension technique ([31], [60]) can be adopted to meet the QoS requirement for DNN inference. Leviathan reserves about 1% of channel bandwidth for the RR operation as the use of low latency SLC NAND ([15]), automated read/write datapath ([13], [32]), and bandwidth boost techniques (Section 4.1) place the bottleneck of LFS on channel. Also, each accelerator reserves 1 GB space in its Tensor Buffers to prevent QoS degradation caused by block erase operations in RR.

Chapter 5. Evaluation

We use both MAESTRO ([33]) and MQSim ([55]) for cycle-level modeling DNN accelerator and SSD, respectively, and PyTorch ([43]) DNN models. The evaluated system has 12 compute cores to process GPT-3 model in 1.3s, which is 10× of the known time constraint for a BERT model ([46]). This constraint is conservative considering that GPT-3 model is about 500× larger than the BERT model.

We evaluate twelve workloads with Leviathan. The models represent two types of widely adopted transformer-based models: 1) BERT/GPT-like and 2) T5-like, as listed in Table 5.2. They are grouped in this way such that the models in the group have identical architecture. More encoders/decoders are stacked (**D**epth) and the dimensions in FC layers are expanded (**W**idth) for diverse comparison. They are referred as $W \times D$ in the rest of this paper.

5.1 Performance of DNN Inference

Baseline system and Leviathan. To perform inference of extreme-scale models such as GPT-3, both the compute capability and storage capacity should match their demands. Thus, we configure Leviathan to accommodate these demands. The baseline accelerator has the same compute capability as Leviathan but with conventional SSDs. We also compare Leviathan with accelerators equipped with additional HBM DRAM to run those models. The accelerator and storage configurations are tabulated in Table 5.1 and Table 5.3, respectively.

Overall inference throughput. Figure 5.1 illustrates the throughput of the three systems. Leviathan achieves the same throughput as the HBM-based accelerator in all cases, which is 2.39× of the baseline throughput on average. In 1×1 case of the BERT/GPT-like model, the presented Leviathan system uti-

Table 5.1: Platform configurations for the cost evaluation of Leviathan.

NPU Parameters		
Number of cores	12 cores (52.5 TFLOPs per core)	
Number of PEs	393,216	
Peak throughput	630 TFLOPs	
Host I/F conf.	PCIe Gen4 × 256 lane (42)	
Memory Parameters		
	Resembled TPU (28)	**Leviathan**
Buffer conf.	128GB HBM	16GB DDR4 DRAM + 2TB NAND flash
Peak bandwidth	2400GB/s	290GB/s

Table 5.2: DNN models evaluated with Leviathan. We use a sequence length of 2048 (tokens) for each model.

Model Size	**BERT/GPT-like**			**T5-like**		
	Act. (GB)	Weight (GB)	PFLOP	Act. (GB)	Weight (GB)	PFLOP
1×1	44	350	2.15	40	305	0.62
1×2	88	698	4.42	80	609	1.25
1×4	175	1393	8.56	160	1218	2.49
2×1	88	1395	8.56	80	1218	2.49
2×2	175	2786	17.12	160	2436	4.99
2×4	349	5569	34.21	319	4871	9.97

lized 282 GB/s aggregate memory system bandwidth. About 1% extra bandwidth of the 282 GB/s is reserved for RR. Since Leviathan's LFS can provide 290 GB/s bandwidth, the system maintains high throughput without being limited by the LFS bandwidth.

Cost efficiency. The difference in memory cost between Leviathan and the conventional HBM-based accelerator is shown in Figure 5.2. Here, we assume \$20/GB, \$4/GB, and \$0.81/GB for HBM DRAM, DDR4 DRAM, and NAND flash, respectively (32). Note that Leviathan assumes 64Gb SLC NAND based

Table 5.3: LFS and conventional storage configuration.

Storage Parameters		
	Baseline SSD	**Leviathan LFS**
NAND Configurations	512B, 128 channels, 1 chips/channel, 1 die/chip	
Channel Speed Rate	2400MT/s (MT/s: Mega Transfers per Second (14, 18))	
NAND Structure	32Gb SLC / die: 8 planes / die, 171 blocks / plane, 768 pages / block, 4KB page	
NAND Latency	Read: 3μs, Program: 100μs, Block erase: 5ms	
Buffer Configurations	DRAM 512GB: FTL metadata SRAM 8MB: I/O buffer, GC Buffer	SRAM 16MB: 6MB for FTL metadata, 10MB for I/O buffer
Stripe strategy (54)	Channel/Way/Plane	Plane/Channel/Way
FTL Schemes	Page mapping, Preemtible GC (34)	Block mapping Cache-read, Multi-plane
OP ratio	7%	N/A
Firmware Latency	Write: 1.45μs / a page (4KB)	N/A
Contoller Latency	Read: 1.93μs / an NVMe Cmd	Read: 1.93μs / an NVMe Cmd, Write: 1.18μs / an NVMe Cmd

on the same semiconductor process as the 128Gb SLC NAND ($0.67/GB) (32). Therefore, we have added an appropriate NAND peripheral cost (30) to the 64Gb SLC NAND. The cost gap between the two systems widens with increasing model size. The maximum cost difference between the two systems is $109K for BERT/GPT-like models and $94K for T5-like models.

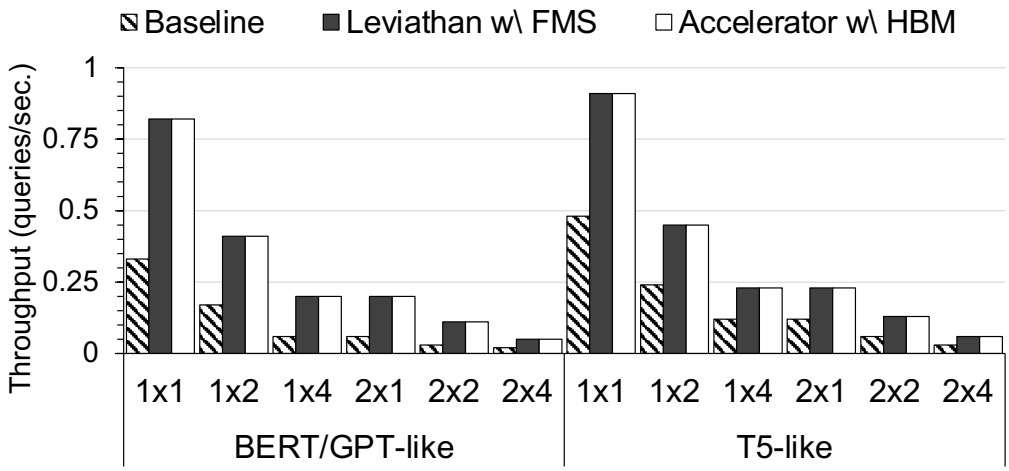

Figure 5.1: Inference throughput comparison.

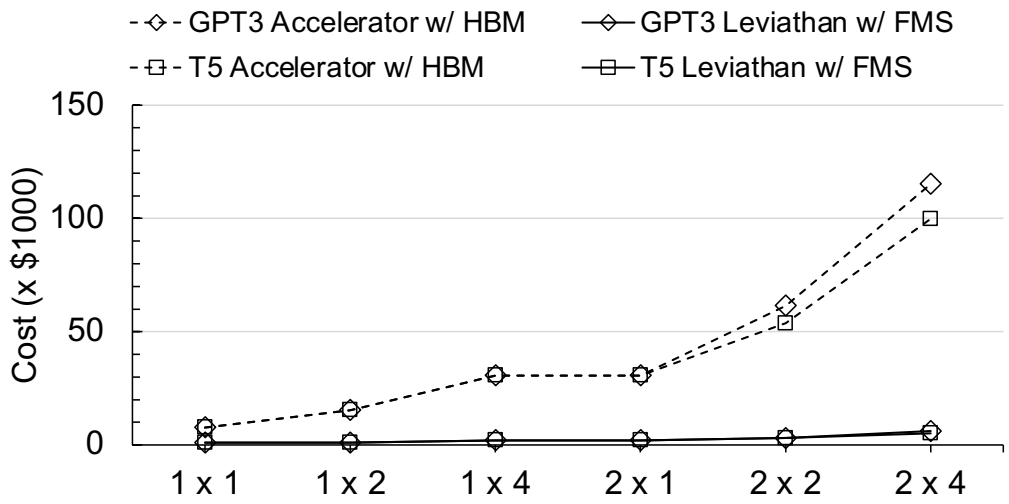

Figure 5.2: Memory system cost comparison.

5.2 Bandwidth variability during RR

Figure 5.3 depicts the bandwidth of four SSD configurations. We align the start time of RR operations at n for all four configurations. Originally, the start times are different due to the different speeds in reaching RD threshold as they provide disparate read speeds for inference. In normal situations, applying cache-read only maintains 60% of LFS bandwidth. Integration with multi-plane operation, the bandwidth reaches that of LFS. However, during

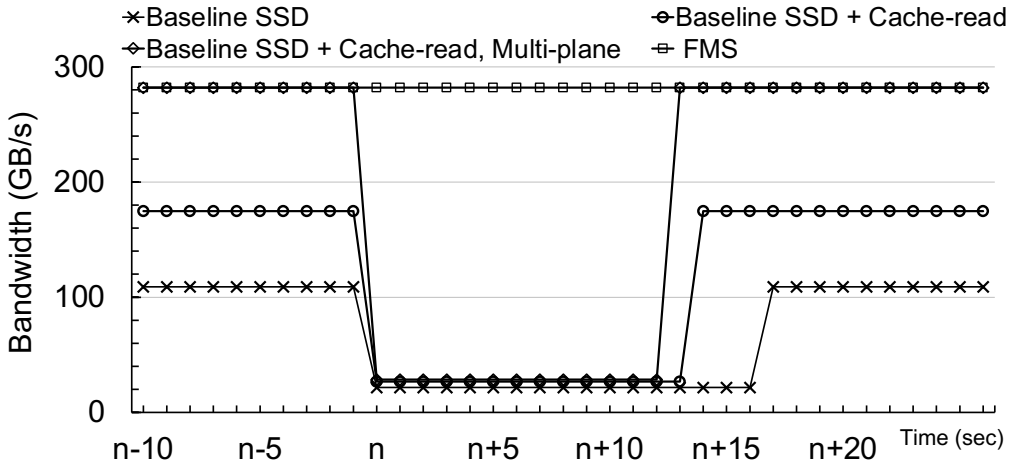

Figure 5.3: Bandwidth variations during read reclaim (RR). RR starting time of each storage configuration (n) is (1) Baseline: 4,180, (2) Cache-read: 2,608, (3) Cache-read, Multi-plane: 1,615, (4) LFS: 1,615 (sec)

RR, all systems but LFS show severe bandwidth deterioration (maximum of 88%) from bursty RRs after n seconds for a maximum of 16 seconds. This is the time needed to reprogram all weight tensors. All three systems maintain comparable bandwidth during RR since multi-plane operation is already being carried out for RR thus cache-read is the sole factor for increasing the bandwidth during this period. In contrast, LFS shows more robust performance, free from RD-induced bandwidth drop 之 by distributing RR operations over time.

Chapter 6. Related Work

Extreme-scale DNN model. With the advent of extreme-scale models, much prior literatures have focused on processing these models, mostly from industry. Megatron-LM ([51]), from NVIDIA, proposes an enlarged version of GPT2 ([44]), the largest model at the time of its writing. Due to the memory capacity wall, they were forced to adopt their massive fleet of GPUs in a model parallel manner. Microsoft followed up with Turing-NLG ([56]), and proposed ZeRO ([45]) and subsequently, ZeRO-offload ([47]). These techniques devised by Microsoft enabled the data parallel training of the enormous model. Each device held a contiguous portion of the model and distributed it to other devices when the portion was next in line to be processed. Within a year, OpenAI introduced GPT3 ([7]), a model with phenomenal size, which was soon topped by Google's GShard ([35]). All of the prior work was focused on training in model- arallel manner but Microsoft. They required the use of expensive HBM to store the model, leading to a cost-efficacious system. Behemoth ([32]) was the first to mitigate this with NAND-based flash. However, as it focused on training of DNN models, it does not mention the unique problem generated in DNN inference system with NAND-based memory.

Read Disturbance. The problem of RD was given much attention relative to its counterpart, write-induced disturbance. However, some prior works take interesting approaches to tackle this issue. Cai's team ([8]) introduces read-retry, where the device reads the read-disturbed cells with the shifted reference voltages to adapt to the changed threshold voltages. Park ([41]) et al. advances this technique using pipeline and by adaptive reduction of chip-level read latency. On the other hand, read-refresh technique is presented in WARM ([39]), which our work is based upon. Ha ([22]) et al. proposes to isolate the read-hot data from the cold data, such that the blocks with cold data are not affected when

reading from hot data. Liu (37) et al. focuses on reducing the number of RD-induced rewrites by examining the application's expected read throughput. With the exception of (37), all prior literature does not leverage the application's characteristics to specialize the NAND flash. In contrast to all prior works, Leviathan fully takes advantage of the deterministic nature of DNN and use it to completely hide RD induced latency and throughput issues.

Chapter 7. Conclusion

This paper presents Leviathan, a DNN inference platform designed to fully accommodate extreme-scale models using cost-efficient NAND flash memory. Leveraging the static, deterministic access pattern of DNN inference, it successfully increases the flash memory bandwidth and averts detrimental cases arising from simultaneous, bursty read reclaims. The bandwidth is boosted using cache-read and multi-plane operations, while RD related issue is taken care of by distributing RR between user read requests. With Leviathan, DNN inference can be run on a much cheaper memory system than the conventional HBM-based memory system.

Bibliography

[1] a100. NVIDIA A100 GPU. https://www.nvidia.com/en-us/data-center/a100/.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, 2015.

[3] Abien Fred Agarap. Deep Learning using Rectified Linear Units (ReLU). *arXiv e-prints*, art. arXiv:1803.08375, March 2018.

[4] J. Bae, H. Jang, J. Gong, W. Jin, S. Kim, J. Jang, T. J. Ham, J. Jeong, and J. W. Lee. Ssdstreamer: Specializing i/o stack for large-scale machine learning. *IEEE Micro*, 39(5):73–81, 2019. doi: 10.1109/MM.2019.2930497.

[5] Jonghyun Bae, Jongsung Lee, Yunho Jin, Sam Son, Shine Kim, Hakbeom Jang, Tae Jun Ham, and Jae W. Lee. Flashneuron: Ssd-enabled large-batch training of very deep neural networks. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 387–401. USENIX Association, February 2021. ISBN 978-1-939133-20-5. URL https://www.usenix.org/conference/fast21/presentation/bae.

[6] A. Brand, K. Wu, S. Pan, and D. Chin. Novel read disturb failure mechanism induced by flash cycling. In *31st Annual Proceedings Reliability Physics 1993*, pages 127–132, 1993. doi: 10.1109/RELPHY.1993.283291.

[7] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen

Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *arXiv e-prints*, art. arXiv:2005.14165, May 2020.

[8] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai. Threshold voltage distribution in mlc nand flash memory: Characterization, analysis, and modeling. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1285–1290, 2013. doi: 10.7873/DATE.2013.266.

[9] Y. Cai, Y. Luo, S. Ghose, and O. Mutlu. Read disturb errors in mlc nand flash memory: Characterization, mitigation, and recovery. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 438–449, 2015. doi: 10.1109/DSN.2015.49.

[10] Yiran Chen, Yuan Xie, Linghao Song, Fan Chen, and Tianqi Tang. A survey of accelerator architectures for deep neural networks. *Engineering*, 6(3):264–274, 2020. ISSN 2095-8099. doi: https://doi.org/10.1016/j.eng.2020.01.007. URL https://www.sciencedirect.com/science/article/pii/S2095809919306356.

[11] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the 43rd International Symposium on Computer Architecture*, page 367–379. IEEE Press, 2016.

[12] Yunji Chen, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. DianNao family: Energy-efficient hardware accelerators for machine learning. *Commun. ACM*, 59(11):105–112, October 2016. ISSN 0001-0782. doi: 10.1145/2996864. URL https://doi.org/10.1145/2996864.

[13] Wooseong Cheong, Chanho Yoon, Seonghoon Woo, Kyuwook Han, Daehyun Kim, Chulseung Lee, Youra Choi, Shine Kim, Dongku Kang, Geunyeong Yu, Jaehong Kim, Jaechun Park, Ki-Whan Song, Ki-Tae Park,

Sangyeun Cho, Hwaseok Oh, Daniel DG Lee, Jin-Hyeok Choi, and Jaeheon Jeong. A flash memory controller for 15μs ultra-low-latency SSD using high-speed 3D NAND flash with 3μs read time. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 338–340. IEEE, 2018. doi: 10.1109/ISSCC.2018.8310322.

[14] J. Cho, D. C. Kang, J. Park, S. W. Nam, J. H. Song, B. K. Jung, J. Lyu, H. Lee, W. T. Kim, H. Jeon, S. Kim, I. M. Kim, J. I. Son, K. Kang, S. W. Shim, J. Park, E. Lee, K. M. Kang, S. W. Park, J. Lee, S. H. Moon, P. Kwak, B. Jeong, C. A. Lee, K. Kim, J. Ko, T. H. Kwon, J. Lee, Y. Lee, C. Kim, M. W. Lee, J. y. Yun, H. Lee, Y. Choi, S. Hong, J. Park, Y. Shin, H. Kim, H. Kim, C. Yoon, D. S. Byeon, S. Lee, J. Y. Lee, and J. Song. 30.3 a 512gb 3b/cell 7th -generation 3d-nand flash memory with 184mb/s write throughput and 2.0gb/s interface. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 426–428, 2021. doi: 10.1109/ISSCC42613.2021.9366054.

[15] Client 0ZSSD. Samsung Z-SSD SZ985. https://www.samsung.com/semiconductor/global.semi.static/Brochure_Samsung_S-ZZD_SZ985_1804.pdf.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv e-prints*, art. arXiv:1810.04805, October 2018.

[17] William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *arXiv e-prints*, art. arXiv:2101.03961, January 2021.

[18] Alan Freedman. MT/sec. *The Computer Desktop Encyclopedia*. https://www.computerlanguage.com/results.php?definition=MT/sec.

[19] Congming Gao, Liang Shi, Chun Jason Xue, Cheng Ji, Jun Yang, and Youtao Zhang. Parallel all the time: Plane level parallelism exploration for high performance ssds. In *2019 35th Symposium on Mass Storage*

Systems and Technologies (MSST), pages 172–184, 2019. doi: 10.1109/MSST.2019.000-5.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[21] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf. Characterizing flash memory: Anomalies, observations, and applications. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 24–33, 2009. doi: 10.1145/1669112.1669118.

[22] Keonsoo Ha, Jaeyong Jeong, and Jihong Kim. A read-disturb management technique for high-density nand flash memory. In *Proceedings of the 4th Asia-Pacific Workshop on Systems*, APSys '13, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450323161. doi: 10.1145/2500727.2500743. URL https://doi.org/10.1145/2500727.2500743.

[23] Keonsoo Ha, Jaeyong Jeong, and Jihong Kim. An integrated approach for managing read disturbs in high-density nand flash memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35 (7):1079–1091, 2016. doi: 10.1109/TCAD.2015.2504868.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv e-prints*, art. arXiv:1512.03385, December 2015.

[25] Tsutomu Higuchi, Takuyo Kodama, Koji Kato, Ryo Fukuda, Naoya Tokiwa, Mitsuhiro Abe, Teruo Takagiwa, Yuki Shimizu, Junji Musha, Katsuaki Sakurai, Jumpei Sato, Tetsuaki Utsumi, Kazuhide Yoneya, Yasuhiro Suematsu, Toshifumi Hashimoto, Takeshi Hioka, Kosuke Yanagidaira, Masatsugu Kojima, Junya Matsuno, Kei Shiraishi, Kensuke Yamamoto, Shintaro Hayashi, Tomoharu Hashiguchi, Kazuko Inuzuka, Akio Sugahara, Mitsuaki Honma, Keiji Tsunoda, Kazumasa Yamamoto, Takahiro Sugimoto, Tomofumi Fujimura, Mizuki Kaneko, Hiroki Date, Osamu Kobayashi, Takatoshi Minamoto, Ryoichi Tachibana, Itaru Yamaguchi,

Juan Lee, Venky Ramachandra, Srinivas Rajendra, Tianyu Tang, Siddhesh Darne, Jiwang Lee, Jason Li, Toru Miwa, Ryuji Yamashita, Hiroshi Sugawara, Naoki Ookuma, Masahiro Kano, Hiroyuki Mizukoshi, Yuki Kuniyoshi, Mitsuyuki Watanabe, Kei Akiyama, Hirotoshi Mori, Akira Arimizu, Yoshito Katano, Masakazu Ehama, Hiroshi Maejima, Koji Hosono, and Masahiro Yoshihara. 30.4 a 1tb 3b/cell 3d-flash memory in a 170+ word-line-layer technology. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 428–430, 2021. doi: 10.1109/ISSCC42613.2021.9366003.

[26] Yang Hu, Hong Jiang, Dan Feng, Lei Tian, Hao Luo, and Shuping Zhang. Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity. In *Proceedings of the International Conference on Supercomputing*, ICS '11, page 96–107, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450301022. doi: 10.1145/1995896.1995912. URL https://doi.org/10.1145/1995896.1995912.

[27] Sitaram Iyer and Peter Druschel. Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous i/o. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, page 117–130, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 1581133898. doi: 10.1145/502034.502046. URL https://doi.org/10.1145/502034.502046.

[28] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire

Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News*, 45(2): 1–12, June 2017. ISSN 0163-5964. doi: 10.1145/3140659.3080246. URL https://doi.org/10.1145/3140659.3080246.

[29] Bryan S. Kim, Jongmoo Choi, and Sang Lyul Min. Design tradeoffs for SSD reliability. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pages 281–294, Boston, MA, February 2019. USENIX Association. ISBN 978-1-939133-09-0.

[30] Chulbum Kim, Ji-Ho Cho, Woopyo Jeong, Il-han Park, Hyun-Wook Park, Doo-Hyun Kim, Daewoon Kang, Sunghoon Lee, Ji-Sang Lee, Wontae Kim, Jiyoon Park, Yang-lo Ahn, Jiyoung Lee, Jong-hoon Lee, Seungbum Kim, Hyun-Jun Yoon, Jaedoeg Yu, Nayoung Choi, Yelim Kwon, Nahyun Kim, Hwajun Jang, Jonghoon Park, Seunghwan Song, Yongha Park, Jinbae Bang, Sangki Hong, Byunghoon Jeong, Hyun-Jin Kim, Chunan Lee, Young-Sun Min, Inryul Lee, In-Mo Kim, Sung-Hoon Kim, Dongkyu Yoon, Ki-Sung Kim, Youngdon Choi, Moosung Kim, Hyunggon Kim, Pansuk Kwak, Jeong-Don Ihm, Dae-Seok Byeon, Jin-yub Lee, Ki-Tae Park, and Kye-hyun Kyung. 11.4 a 512gb 3b/cell 64-stacked wl 3d v-nand flash memory. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 202–203, 2017. doi: 10.1109/ISSCC.2017.7870331.

[31] Shine Kim, Jonghyun Bae, Hakbeom Jang, Wenjing Jin, Jeonghun Gong, Seungyeon Lee, Tae Jun Ham, and Jae W. Lee. Practical erase suspension for modern low-latency ssds. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 813–820, Renton, WA, July 2019. USENIX Association. ISBN 978-1-939133-03-8.

[32] Shine Kim, Yunho Jin, Gina Sohn, Jonghyun Bae, Tae Jun Ham, and Jae W. Lee. Behemoth: A flash-centric training accelerator for extreme-scale dnns. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 371–385. USENIX Association, February 2021. ISBN 978-1-939133-20-5. URL https://www.usenix.org/conference/fast21/presentation/kim.

[33] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. Understanding reuse, performance, and hardware cost of DNN dataflow: A data-centric approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, page 754–768. ACM, 2019. ISBN 9781450369381. doi: 10.1145/3352460.3358252. URL https://doi.org/10.1145/3352460.3358252.

[34] Junghee Lee, Youngjae Kim, Galen M. Shipman, Sarp Oral, and Jongman Kim. Preemptible I/O scheduling of garbage collection for solid state drives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(2):IEEE, 247–260, 2013. ISSN 0278-0070. doi: 10.1109/TCAD.2012.2227479.

[35] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding. *arXiv e-prints*, art. arXiv:2006.16668, June 2020.

[36] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv e-prints*, art. arXiv:1910.13461, October 2019.

[37] Chun-Yi Liu, Yunju Lee, Myoungsoo Jung, Mahmut Taylan Kandemir, and Wonil Choi. Prolonging 3d nand ssd lifetime via read latency relaxation. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASP-

LOS 2021, page 730–742, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383172. doi: 10.1145/3445814.3446733. URL https://doi.org/10.1145/3445814.3446733.

[38] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv e-prints*, art. arXiv:1907.11692, July 2019.

[39] Y. Luo, Y. Cai, S. Ghose, J. Choi, and O. Mutlu. Warm: Improving nand flash memory lifetime with write-hotness aware retention management. In *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–14, 2015. doi: 10.1109/MSST.2015.7208284.

[40] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. Deep Learning Recommendation Model for Personalization and Recommendation Systems. *arXiv e-prints*, art. arXiv:1906.00091, May 2019.

[41] Jisung Park, Myungsuk Kim, Myoungjun Chun, Lois Orosa, Jihong Kim, and Onur Mutlu. Reducing solid-state drive read latency by optimizing read-retry. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS 2021, page 702–716, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383172. doi: 10.1145/3445814.3446719. URL https://doi.org/10.1145/3445814.3446719.

[42] pcie. PCI Express 4. https://pcisig.com.

[43] pytorch. PyTorch. https://pytorch.org.

[44] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and

Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[45] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. *arXiv e-prints*, art. arXiv:1910.02054, October 2019.

[46] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 446–459, 2020. doi: 10.1109/ISCA45697.2020.00045.

[47] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. ZeRO-Offload: Democratizing Billion-Scale Model Training. *arXiv e-prints*, art. arXiv:2101.06840, January 2021.

[48] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W. Keckler. vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 18:1–18:13. IEEE, 2016.

[49] Samsung. Samsung SSD 980 PRO. https://www.samsung.com/semiconductor/minisite/ssd/product/consumer/980pro/, 2020.

[50] Narges Shahidi, Mahmut T. Kandemir, Mohammad Arjomand, Chita R. Das, Myoungsoo Jung, and Anand Sivasubramaniam. Exploring the potentials of parallel garbage collection in ssds for enterprise storage systems. In *SC '16: Proceedings of the International Conference for High Performance*

Computing, Networking, Storage and Analysis, pages 561–572, 2016. doi: 10.1109/SC.2016.47.

[51] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv e-prints*, art. arXiv:1909.08053, September 2019.

[52] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv e-prints*, art. arXiv:1409.1556, September 2014.

[53] K. Takeuchi, S. Satoh, T. Tanaka, K. Imamiya, and K. Sakui. A negative vth cell architecture for highly scalable, excellently noise immune and highly reliable nand flash memories. In *1998 Symposium on VLSI Circuits. Digest of Technical Papers (Cat. No.98CH36215)*, pages 234–235, 1998. doi: 10.1109/VLSIC.1998.688097.

[54] Arash Tavakkol, Pooyan Mehrvarzy, Mohammad Arjomand, and Hamid Sarbazi-Azad. Performance evaluation of dynamic page allocation strategies in ssds. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 1(2), June 2016. ISSN 2376-3639. doi: 10.1145/2829974. URL https://doi.org/10.1145/2829974.

[55] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. MQSim: A framework for enabling realistic studies of modern multi-queue SSD devices. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies*, pages 49–66. USENIX Association, 2018. ISBN 978-1-931971-42-3. URL https://www.usenix.org/conference/fast18/presentation/tavakkol.

[56] turing-NLG. turing-NLG Microsoft. https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/.

[57] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention

is all you need. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS, 2017.

[58] Linnan Wang, Jinmian Ye, Yiyang Zhao, Wei Wu, Ang Li, Shuaiwen Leon Song, Zenglin Xu, and Tim Kraska. SuperNeurons: Dynamic GPU memory management for training deep neural networks. In *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 41–53. ACM, 2018. ISBN 978-1-4503-4982-6. doi: 10.1145/3178487.3178491.

[59] Jiwon Woo, Minwoo Ahn, Gyusun Lee, and Jinkyu Jeong. D2fq: Device-direct fair queueing for nvme ssds. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 403–415. USENIX Association, February 2021. ISBN 978-1-939133-20-5.

[60] Guanying Wu and Xubin He. Reducing SSD read latency via NAND flash program and erase suspension. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST'12, pages 117–123. USENIX Association, 2012.

[61] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated Residual Transformations for Deep Neural Networks. *arXiv e-prints*, art. arXiv:1611.05431, November 2016.

국문초록

인공신경망 모델들의 크기가 급속하게 증가함에 따라 이에 상응하는 거대한 용량이 요구돼어진다. 예시로, 구글에서 최근에 발표한 Switch Transformer는 6.4 TB의 크기를 가지고 있다. 하지만, 근래의 GPU 혹은 DNN 가속기에 탑재되어 있는 HBM DRAM 기반 메모리 시스템은 이러한 초대규모 DNN 모델들을 저장하기에는 용량이 작을 뿐더러 높은 대역폭이 충분히 활용되지 않아 부적합하다. 이 사실에 기반하여, 본 논문에서는 효율적인 플래쉬 기반 메모리 시스템을 탑재한 DNN 추론 가속기인 Leviathan 을 제안한다. DNN 추론의 특성을 활용하여 기존의 플래쉬 기반 시스템의 대역폭을 충분히 증가시키며 읽기 방해에 의해 발생하는 성능 저하를 예방한다. Leviathan 의 성능 분석을 통해 동일 연산 능력에 기존 SSD를 장착한 DNN 가속기에 비해 2.39× 처리율을 달성하고 HBM 기반 DNN 가속기에 비해 19.47× 가격 효율성을 가지는 점을 확인한다.

주요어: 인공신경망, 플래쉬 메모리, 읽기 방해
학번: 2019-22004