Manycore-Based Scalable SSD Architecture Towards One and More Million IOPS

Jie Zhang[1], Miryeong Kwon[1], Michael Swift[2], Myoungsoo Jung[1]
Computer Architecture and Memory Systems Laboratory,
Korea Advanced Institute of Science and Technology (KAIST)[1], University of Wisconsin at Madison[2]
http://camelab.org

I. Introduction

NVMe is designed to unshackle flash from a traditional storage bus by allowing hosts to employ many threads to achieve higher bandwidth [1]. While NVMe enables users to fully exploit all levels of parallelism offered by modern SSDs [2], current firmware designs are not scalable and have difficulty in handling a large number of I/O requests in parallel due to its limited computation power and many hardware contentions.

We propose `DeepFlash`, a novel manycore-based storage platform that can process more than a million I/O requests in a second (1MIOPS) while hiding long latencies imposed by its internal flash media. Inspired by a parallel data analysis system, we design the firmware based on many-to-many threading model that can be scaled horizontally. The proposed `DeepFlash` can extract the maximum performance of the underlying flash memory complex by concurrently executing multiple firmware components across many cores within the device. To show its extreme parallel scalability, we implement `DeepFlash` on a many-core prototype processor that employs dozens of lightweight cores, analyze new challenges from parallel I/O processing and address the challenges by applying concurrency-aware optimizations. Our comprehensive evaluation reveals that `DeepFlash` can serve around 4.5 GB/s, while minimizing the CPU demand on microbenchmarks and real server workloads.

II. High Performance NVMe SSDs

Baseline. Figure 1 shows an overview of a high-performance SSD architecture that Marvell recently published [3]. The host connects to the underlying SSD through four Gen 3.0 PCIe lanes (4 GB/s) and a PCIe controller. The SSD architecture employs three embedded processors, each employing two cores [4], which are connected to an internal DRAM controller via a processor interconnect. The SSD employs several special-purpose processing elements, including a low-density parity-check (LDPC) sequencer, data transfer (DMA) engine, and scratch-pad memory for metadata management. All these multi-core processors, controllers, and components are connected to a flash complex that connects to eight channels, each connecting to eight packages, via flash physical layer (PHY).

Future architecture. The performance offered by these devices is by far below 1MIOPS. For higher bandwidth, a future device can extend storage and processor complexes with more flash packages and cores, respectively, which are highlighted by red in the figure. The bandwidth of each flash package is in practice tens of MB/s, and thus, it requires employing more flashes/channels, thereby increasing I/O parallelism. This flash-side extension raises several architectural issues. First, the firmware will make frequent SSD-internal memory accesses that stress the processor complex. Even though the PCIe core, channel and other memory control logic may be implemented, metadata information increases for the extension, and its access frequency gets higher to achieve 1MIOPS. In addition, DRAM accesses for I/O buffering can be a critical bottleneck to hide flash's long latency. Simply making cores faster may not be sufficient because the processors will suffer from frequent stalls due to less locality and contention at memory. This, in turn, makes each core bandwidth lower, which should be addressed with higher parallelism on computation parts.

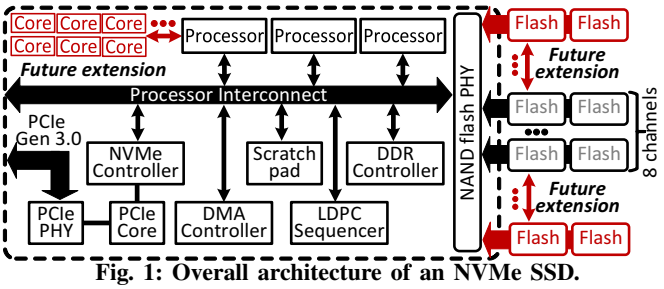
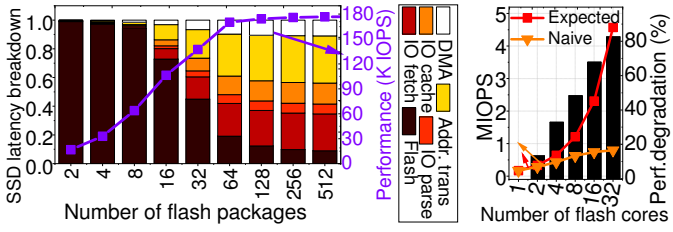


Fig. 1: Overall architecture of an NVMe SSD.



(a) Flash scaling. (b) Core scaling.

Fig. 2: Perf. with varying flash packages and cores.

III. Challenges to Exceeding 1MIOPS

Flash scaling. The bandwidth of a low-level flash package is several orders of magnitude lower than the PCIe bandwidth. Thus, SSD vendors integrate many flash packages over multiple channels, which can in parallel serve I/O requests managed by NVMe. Figure 2a shows the relationship of bandwidth and execution latency breakdown with various number of flash packages. For the breakdown analysis, we decompose total latency into i) NVMe management (`I/O parse` and `I/O fetch`), ii) `I/O cache`, iii) `address translation` (including flash scheduling), vi) NVMe data transfers (`DMA`) and v) flash operations (`Flash`). The SSD performance saturates at 170K IOPS with 64 flash packages, connected over 16 channels. As the number of flash packages increases (more than 32), the layered firmware operations on a core become the performance bottleneck. There are two reasons. First, NVMe queues can supply many I/O requests to take advantages of the SSD internal parallelism, but a single-core SSD controller is insufficient to fetch all the requests. Second, it is faster to parallelize I/O accesses across many flash chips than performing address translation only on one core. These new challenges make it difficult to fully leverage the internal parallelism with the conventional layered firmware model.

Core scaling. To take flash firmware off the critical path in scalable I/O processing, one can increase computing power with the execution of many firmware instances. This approach can allocate a core per NVMe SQ/CQ and initiate one layered firmware instance in each core. However, we observe that this naive approach cannot successfully address the burdens brought by flash firmware. To be precise, we evaluate IOPS with varying number of cores, ranging from 1 to 32. Figure 2b compares the performance of aforementioned naive many-core approach (e.g., `Naive`) with the system that expects perfect parallel scalability (e.g., `Expected`). `Expected`'s performance is calculated by multiplying the number of cores with IOPS of `Naive` built on a single core SSD. `Naive` can only achieve 813K IOPS even with 32 cores, which exhibits 82.6% lower performance, compared to `Expected`. This is because contention and consistency management

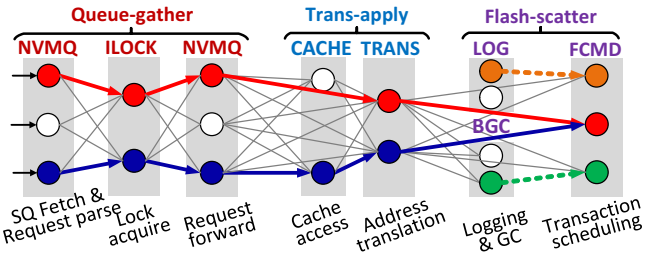


Fig. 3: Many-to-many threading firmware model.

for the memory spaces of internal DRAM introduces significant synchronization overheads. In addition, the FTL must serialize the I/O requests to avoid hazards while processing many queues in parallel. Since all these issues are not considered by the layered firmware model, it should be re-designed by considering core scaling.

The goal of our new firmware is to fully parallelize multiple NVMe processing datapaths in a highly scalable manner while minimizing the usage of SSD internal resources. `DeepFlash` requires only 12 in-order cores to achieve 1M or more IOPS.

IV. Scalable Flash Firmware

We propose `DeepFlash`, a manycore-based NVMe SSD platform that can process more than one million I/O requests within a second (1MIOPS) while minimizing the requirements of internal resources. To this end, we design a new flash firmware model, which can extract the maximum performance of hundreds of flash packages by concurrently executing firmware components atop a manycore processor. The layered flash firmware in many SSD technologies handles the internal datapath from PCIe to physical flash interfaces as a single heavy task [5], [6]. In contrast, `DeepFlash` employs a many-to-many threading model, which multiplexes any number of threads onto any number of cores in firmware.

Many-to-many threading firmware. We identify scalability and parallelism opportunities for high-performance flash firmware. Our many-to-many threading model allows future manycore-based SSDs to dynamically shift their computing power based on different workload demands without any hardware modification. As shown in Figure 3, `DeepFlash` splits all functions from the existing layered firmware architecture into three stages (i.e., Queue-gather, Trans-apply and Flash-scatter), each with one or more thread groups. Different groups can communicate with each other over an on-chip interconnection network within the target SSD (cf. Figure 3).

Parallel NVMe queue management. While employing many NVMe queues allows the SSD to handle many I/O requests through PCIe communication, it is hard to coordinate simultaneous queue accesses from many cores. `DeepFlash` dynamically allocates the cores to process NVMe queues rather than statically assigning one core per queue. Thus, a single queue is serviced by multiple cores, and a single core can service multiple queues, which can deliver full bandwidth for both balanced and unbalanced NVMe I/O workloads. We show that this parallel NVMe queue processing exceeds the performance of the static core-per-queue allocation by 6x, on average, when only a few queues are in use. `DeepFlash` also balances core utilization over computing resources.

Efficient I/O processing. We increase the parallel scalability of many-to-many threading model by employing non-blocking communication mechanisms. We also apply simple but effective lock and address randomization methods, which can distribute incoming I/O requests across multiple address translators and flash packages. The proposed method minimizes the number of hardware core to achieve 1MIOPS. Putting all it together, `DeepFlash` improves bandwidth by 3.4× while significantly reducing CPU requirements, compared

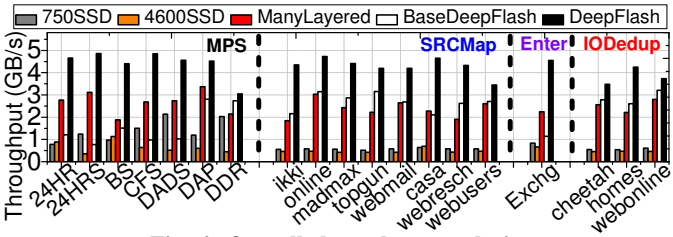


Fig. 4: Overall throughput analysis.

to conventional firmware. Our `DeepFlash` requires only a dozen of lightweight in-order cores to deliver 1MIOPS.

V. Evaluation and Conclusion

Experiment. `BaseDeepFlash` is the emulated SSD platform that applies many-to-many threading firmware. `DeepFlash` is the emulated SSD platform including all the proposed designs of this paper. We evaluate the performance of a real Intel customer-grade SSD (`750SSD`) [7] and high-performance NVMe SSD (`4600SSD`) [8] for a better comparison. We also emulate another SSD platform (`ManyLayered`), which is an approach to scale up the layered firmware on many cores.

Performance. Figure 4 illustrates the throughput of server workloads. As shown in the figure, `BaseDeepFlash` exhibits 1.6, 2.7, 1.1, and 2.9 GB/s, on average, for MPS, SRCMap, Enterprise and IODedup workload sets, respectively, and `DeepFlash` improves those of `BaseDeepFlash`, by 260%, 64%, 299% and 35%, respectively. `BaseDeepFlash` exhibits a performance degradation, compared to `ManyLayered` with MPS. This is because MPS generates multiple lock contentions, due to more small-size random accesses than other workloads. Interestingly, while `DeepFlash` outperforms other SSD platforms in most workloads, its performance is not as good under *DDR* workloads (slightly better than `BaseDeepFlash`). This is because core utilization is lower than 56% due to the address patterns of *DDR*. However, since all NVMQ threads parse and fetch incoming requests in parallel, even for such workloads, `DeepFlash` provides 3 GB/s, which is 42% better than `ManyLayered`.

In conclusion, we designed scalable flash firmware inspired by parallel data analysis systems, which can extract the maximum performance of the underlying flash memory complex by concurrently executing multiple firmware components within a single device.

VI. Original Publication

J. Zhang, M. Kwon, M. Swift and M. Jung. 2020. Scalable Parallel Flash Firmware for Many-core Architectures. USENIX FAST. *https://www.usenix.org/system/files/fast20-zhang_jie.pdf*

VII. Acknowledgement

This research is mainly supported by NRF 2021R1A2C4001773, MemRay grant (G01190170), Samsung grant (G01190271/G01200447), and KAIST start-up package (G01190015).

References

[1] W. Choi *et al.*, "An in-depth study of next generation interface for emerging non-volatile memories," in *NVMSA*, IEEE, 2016.
[2] A. Tavakkol *et al.*, "MQSim: A framework for enabling realistic studies of modern multi-queue SSD devices," in *FAST*, 2018.
[3] marvell, "Marvell 88ss1093 flash memory controller," *https://www.marvell.com/storage/assets/Marvell-88SS1093-0307-2017.pdf*, 2017.
[4] X. Iturbe *et al.*, "A triple core lock-step (TCLS) ARM® Cortex®-R5 processor for safety-critical and ultra-reliable applications," in *DSN*, IEEE, 2016.
[5] Z. Weiss *et al.*, "ANViL: Advanced virtualization for modern non-volatile memory devices.," in *FAST*, 2015.
[6] Y. Zhou *et al.*, "An efficient page-level FTL to optimize address translation in flash memory," in *Eurosys*, ACM, 2015.
[7] Intel, "Intel SSD 750 series," *http://tiny.cc/qyzdcz*, 2015.
[8] Intel, "Intel SSD DC P4600 Series," *http://tiny.cc/dzzdcz*, 2018.