ByteExpress: A High-Performance and Traffic-Efficient Inline Transfer of Small Payloads over NVMe

Junhyeok Park
junttang@sogang.ac.kr
Sogang University
Seoul, South Korea

Junghee Lee
j_lee@korea.ac.kr
Korea University
Seoul, South Korea

Youngjae Kim*
youkim@sogang.ac.kr
Sogang University
Seoul, South Korea

ABSTRACT

Recent computational storage devices enable host-side tasks such as SQL filtering and key-value operations to be offloaded to the device. However, these tasks often involve small payloads, typically a few dozen to hundreds of bytes, which are inefficiently handled by the conventional NVMe protocol due to its page-based DMA mechanism. Even tiny payloads incur 4 KB PCIe transfers, leading to severe bandwidth waste and increased latency. Prior approaches either break NVMe compatibility or are only effective for very small payloads on the order of a few dozen bytes. This paper presents ByteExpress, a new mechanism that efficiently transmits small payloads by placing them inline in 64-byte chunks directly into the NVMe submission queue, immediately following the NVMe command. ByteExpress requires only slight modifications to the NVMe driver and controller logic, while preserving full compatibility with existing APIs and SSD architectures. We implemented ByteExpress on the Linux NVMe driver and OpenSSD, demonstrating up to 98% reduction in PCIe traffic and 40% and 39% lower latency compared to PRP and a state-of-the-art approach, respectively, for sub-page payloads.

CCS CONCEPTS

• **Information systems** → **Flash memory**; **Storage architectures**; **Storage management**.

KEYWORDS

Non-Volatile Memory Express Protocol, Solid-State Drive.

ACM Reference Format:
Junhyeok Park, Junghee Lee, and Youngjae Kim. 2025. ByteExpress: A High-Performance and Traffic-Efficient Inline Transfer of Small Payloads over NVMe. In *17th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '25), July 10–11, 2025, Boston, MA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3736548.3737837

*Y. Kim is the corresponding author.

HotStorage '25, July 10–11, 2025, Boston, MA, USA

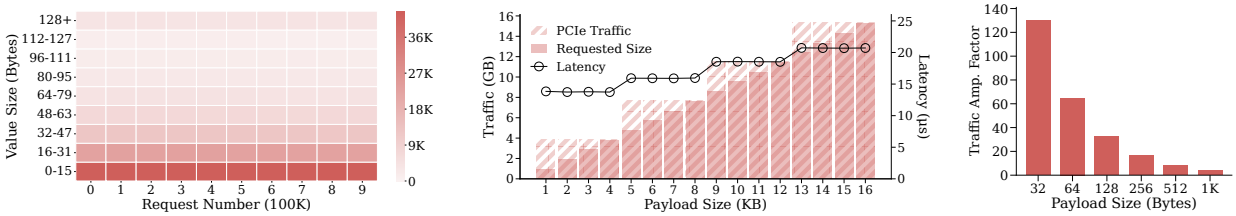
ACM ISBN 979-8-4007-1947-9/2025/07.
https://doi.org/10.1145/3736548.3737837

1 INTRODUCTION

Recent advances in Solid-State Drive (SSD) technology have led to increasingly powerful devices, with modern Non-Volatile Memory Express (NVMe) SSDs now equipped with multi-core Arm processors and substantial internal DRAM. As these capabilities continue to evolve, SSDs are gradually shifting from simple storage solutions to intelligent data-processing devices, opening new opportunities for offloading and executing tasks traditionally handled by the host [43]. These trends have led to the emergence of new classes of computational storage devices, including Computational SSDs (CSDs) that can execute user's analytics tasks such as SQL filters inside the SSD [3, 14, 20], and Key-Value SSDs (KV-SSDs) that natively support key-value operations within the device by bypassing traditional file systems [11, 13, 19, 24, 33].

To interact with these new types of devices, users commonly employ the NVMe passthrough [17] (§2.1), wherein application-level requests, such as SQL predicates for CSDs or key-value pairs for KV-SSDs, are encoded as custom NVMe Commands (CMDs) and sent directly to the device. Interestingly, a closer look at this new storage interface reveals that the actual data payloads (i.e., predicates and key-value pairs) transferred in such requests are often small (§2.2). CSDs process filter operations based on computation task specifications that typically require only a table identifier and predicates, resulting in payloads of just tens to hundreds of bytes [14]. Similarly, the values handled during real-world key-value operations also tend to be a few dozen bytes in size, as evidenced by Meta's internal workload analysis [4].

Ironically, the conventional NVMe protocol is not well-suited for handling such small payloads (§2.3). Specifically, NVMe employs Physical Region Pages (PRPs) for data transfer, which requires data to be transferred in 4 KB memory page units. As a result, even a 32-byte payload incurs 4 KB of PCIe traffic, more than 130× greater than the requested size (see Figure 1(c)). This significant traffic bloating can lead to increased latency and unnecessary power consumption [3],



(a) Value Size Distribution in MixGraph (b) PCIe Traffic and Transfer Latency (c) Traffic Amp. Factor

Figure 1: (a) Value size heatmap from MixGraph All_random with its default settings, (b) PCIe traffic and transfer latency (NAND off) for PRP-based writes across varying sizes, and (c) traffic amplification for sub-1KB payloads.

making it a critical bottleneck for frequent, small direct I/Os. NVMe also supports Scatter-Gather Lists (SGLs), which can enable fine-grained DMA for small payloads. However, SGLs are not supported across all NVMe devices, and even when supported, the Linux kernel is configured by default to use them only for transfers larger than 32 KB [18] (§5). Accordingly, this work focuses on optimizing PRP-based transfers.

Prior works have attempted to address this issue using two primary approaches: (1) bypassing the NVMe stack via PCIe Memory-Mapped I/O (MMIO) to enable byte-level writes directly to the device [1], or (2) embedding the payload inline within one or more custom NVMe CMDs and transmitting them in fragments [35]. However, both solutions suffer from critical limitations. The former requires substantial modifications to SSD architecture and controller firmware, making it difficult to integrate with existing NVMe-based CSDs or KV-SSDs (§3.1). The latter, meanwhile, loses scalability for larger payloads due to repeated CMD generation and processing overhead caused by its mandatory serialization (§3.2).

In this paper, we present ByteExpress, a novel mechanism that efficiently transmits small payloads without disrupting the NVMe-based interface or requiring changes to SSD architecture. ByteExpress leverages a simple yet powerful idea: *it directly places the small payload, in 64-byte chunks, into the NVMe Submission Queue (SQ) after the CMD itself.* This approach allows for fine-grained, performant PCIe transfers.

To realize this idea, ByteExpress addresses two key challenges: *how to identify the payload correctly* and *how to preserve ordering of chunks.* First, it reuses existing NVMe CMD metadata by embedding the payload size into a reserved field, enabling the controller to determine how many 64-byte chunks to fetch. Second, ByteExpress ensures strict intra- and inter-SQ ordering by leveraging the driver's existing SQ locking and enforcing queue-local chunk retrieval on the device, preserving ordering of the CMD and payload chunks.

We implemented ByteExpress on the Linux kernel NVMe driver [18] and the OpenSSD platform [36], and demonstrated that it reduces PCIe traffic by up to 98% and improves latency by more than 40% across key-value, SQL pushdown, and microbenchmark workloads, outperforming both PRP and the state-of-the-art NVMe CMD-based method.

2 BACKGROUND AND MOTIVATION

2.1 NVMe-Based New Storage Interface

Recent computational storage devices, such as Key-Value SSDs (KV-SSDs) [15, 19, 24, 32, 33] and Computational SSDs (CSDs) [3, 14, 20, 45], extend the standard NVMe protocol to deliver key-value pairs or computation tasks directly to the device. In these systems, user-level APIs translate and encapsulate high-level operations, such as key-value operations or SQL predicates pushdown, into custom NVMe Commands (CMDs), along with the relevant data, and submit them to the SSD via the NVMe driver. These requests typically leverage the NVMe passthrough [17], allowing applications to bypass the kernel I/O stack and interact directly with the driver. This enables seamless data exchange with such devices without significantly modifying the I/O stack, as shown in Figure 2.

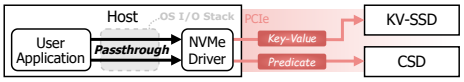
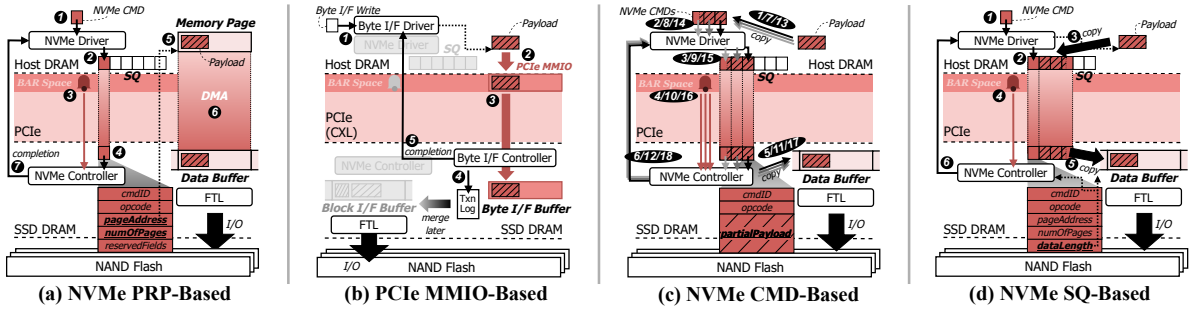


Figure 2: The NVMe passthrough-based interface used in computational storage such as KV-SSDs and CSDs.

2.2 The Advent of Small, Direct I/O

2.2.1 ***Value Transfer in KV-SSDs.*** Recent production-scale analyses reveal that most values written by key-value databases are small. Meta reported that in their deployment of RocksDB [7], a widely used persistent key-value store, the majority of values are only a few tens of bytes in size [4]. Similarly, Twitter observed that their average value size is less than 100 bytes in production environments [5]. Figure 1(a) illustrates this trend using the MixGraph workload [8], a benchmark reflecting Meta's workload characteristics [4].

These small values are persisted individually in KV-SSDs, in accordance with the key-value pair-level transaction model defined in the NVMe key-value extension [31]. As a result, small direct I/O operations are a natural and frequent part of KV-SSD behavior in real-world scenarios. Although batching multiple key-value pairs into a single bulk PUT has been explored in some prior work [21, 33], such approaches may not always be applicable, particularly in use cases where fine-grained persistence is desired for each key-value pair [2, 6, 38, 44].



(a) NVMe PRP-Based (b) PCIe MMIO-Based (c) NVMe CMD-Based (d) NVMe SQ-Based

Figure 3: Illustration of data transfer flows for small, direct I/O writes, including (a) the conventional NVMe PRP mechanism, (b) & (c) representative techniques proposed by prior works, and (d) the proposed ByteExpress.

2.2.2 ***SQL Predicate Pushdown in CSDs.*** In CSDs, SQL filter operations can be early executed within the SSD, which is especially effective for high-selectivity workloads with simple `SELECT-WHERE` queries [14, 20, 29, 42]. A key observation across these systems is that the SSD already stores table schema. As a result, the host only needs to transmit a predicate and a table identifier to initiate filter execution. Even when expressed as a completely unoptimized SQL string without any binary encoding, the total payload size remains small. We analyzed example queries from prior studies [9, 14, 20] as shown in Figure 4.

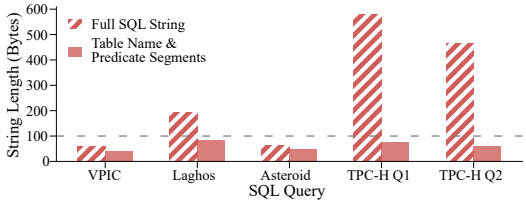


Figure 4: Example queries used in CSD works, showing the lengths of full string and table/predicate segments.

For each workload, we extract the table name and predicate portion from the full SQL string. Scientific workloads such as VPIC [23], Laghos [40], and Asteroid [39] involve simple predicates, resulting in payloads of less than 100 bytes. For TPC-H, we focus on Q1 and Q2 queries used in prior work [9, 14], isolating the filter condition on a single table (e.g., `lineitem` in Q1 and `region` in Q2), which similarly yield payloads under 100 bytes.

2.3 NVMe is Not Small I/O Friendly

Figure 3(a) illustrates the standard data transfer process in NVMe SSDs based on the Physical Region Page (PRP) mechanism. The host prepares the data in memory pages and constructs an NVMe CMD that specifies the address and number of pages to transfer. The CMD is submitted to the NVMe driver (❶). The driver inserts the CMD into the Submission Queue (SQ) (❷) and triggers the doorbell register in the PCIe Base Address Register (BAR) space to notify the device of a new submission (❸). The device, polling the doorbell register, detects the submission and performs a 64-byte Direct Memory Access (DMA) fetch of the CMD (❹). It then reads the indicated host page addresses and counts (❺) and copies the corresponding pages into device DRAM via DMA (❻). Finally, the device signals completion to the host (❼).

We conducted an experiment on the OpenSSD [36], sending 1 million payloads of 1 to 16 KB via NVMe passthrough. As shown in Figure 1(b), the PCIe traffic generated remained aligned to 4 KB boundaries, regardless of the actual payload size. We disabled NAND I/O and measured transfer latency alone, which also exhibited a stepwise increase at 4 KB intervals. Figure 1(c) illustrates this inefficiency for sub-1 KB transfers: for example, a 32-byte request generates over 130 times more traffic than its actual size.

3 SMALL PAYLOAD TRANSFER METHOD

3.1 PCIe MMIO-Based Transfer

PCIe Memory-Mapped I/O (MMIO)-based transfer refers to a mechanism that enables direct data exchange between the host and SSD by utilizing the PCIe BAR space. This region, where doorbell registers and SQ tail pointers typically reside, supports synchronous memory operations, allowing hosts to bypass traditional block-based data paths for small I/O. Representative implementations of this approach include 2B-SSD [1] and ByteFS [26]. These systems expose a Byte Interface API over the BAR space, enabling the host to directly write small payloads with 64-byte cacheline-unit PCIe traffic into a designated device memory buffer (see Figure 3(b)). This design allows for low-latency, fine-grained data transfers [1].

However, applying this approach to existing NVMe-based storage devices introduces several key challenges. First, the device must be significantly modified to include dedicated buffer memory and maintain transactional coordination, typically via a persistent log, between the new Byte Interface for small payloads and the existing block-based I/O path for large payloads to ensure consistency. Second, on the host side, NVMe passthrough-based APIs cannot be reused, requiring the development of a new interface layer.

3.2 NVMe CMD-Based Transfer

NVMe CMD-based transfer refers to the use of NVMe CMDs to deliver data from the host to the SSD. BandSlim [35] demonstrates how NVMe CMDs can be repurposed for fine-grained data movement. It embeds small payload fragments directly into CMD fields and issues a sequence of NVMe CMDs (each carrying a portion of the data) to create fine-grained PCIe traffic patterns (see Figure 3(c)). By utilizing the NVMe passthrough, this method offers the advantage of requiring no modifications to SSD architectures.

However, this reliance on CMD issuance introduces additional complexity and performance overheads. Because payload fragments must be sent through serialized CMDs, a dedicated software layer is required to manage fragment ordering and CMD generation. Moreover, frequent issuance of CMDs significantly increases protocol overhead, particularly as the number of fragments grows. As a result, when the payload size exceeds just a few dozen bytes (e.g., 64 bytes), the cost of repeated CMD submission becomes a major bottleneck, leading to substantial performance drops [35] (§4.2).

3.3 ByteExpress: NVMe SQ-Based Transfer

At the heart of ByteExpress is a concise yet powerful insight: ***NVMe already enables fine-grained data delivery over PCIe***. This capability is embedded in the design of the NVMe SQ, where the host places a CMD in memory and the device performs a 64-byte DMA fetch from the SQ's tail. If we reinterpret the CMD itself as a payload or a portion of payload, this mechanism effectively becomes a built-in fine-grained I/O path. ByteExpress builds on this insight by placing the actual payload into the SQ in 64-byte chunks (see Figure 3(d)), reusing the existing infrastructure for small, direct transfers.

3.3.1 ***Challenge#1: Identifying the Payload.*** ByteExpress leverages the fact that the NVMe driver already has full knowledge of the payload at submission time. Specifically, the payload size is encoded in the data length field of the NVMe I/O CMD, and its address is specified through the PRP entry fields during CMD construction. Right before placing the CMD into the SQ, ByteExpress repurposes a reserved field within the CMD to store the payload length again. This enables the NVMe controller to later retrieve and interpret the exact size of the payload without additional protocol changes. After the submission of the CMD, ByteExpress immediately appends the payload in 64-byte chunks, using the subsequent SQ entries. Each chunk occupies a single SQ entry, which is the same size as the NVMe CMD. Once all data chunks are written, the driver raises the doorbell register.

On the device side, when the NVMe controller fetches a CMD, it first examines the designated field to read the embedded payload length. This indicates that the CMD uses ByteExpress semantics and instructs the controller to expect a sequence of payload chunks following the CMD. The controller then calculates the number of subsequent SQ entries to fetch based on the payload length and issues DMA fetches in 64-byte units, storing the received payload into a designated buffer. The buffer can refer to, for example, a key-value log of KV-SSDs, a workspace for filter processing in CSDs, or even a NAND page buffer entry of normal block SSDs.

3.3.2 ***Challenge#2: Preserving Data Ordering.*** ByteExpress preserves ordering between the NVMe CMD and its associated 64-byte payload chunks through two complementary mechanisms. **First**, on the host side, the NVMe driver guarantees exclusive access to each SQ using spin locks [18]. ByteExpress leverages this by inserting both the CMD and its payload chunks while holding the lock, ensuring that the entries are placed consecutively and in order. **Second**, on the device side, the controller preserves inter-SQ ordering by fetching subsequent entries exclusively from the same SQ once a ByteExpress-applied CMD is detected. In our implementation on OpenSSD [36], the controller polls SQs in round-robin and, upon identifying a ByteExpress CMD, sequentially fetches the following payload chunks from the same queue without switching queues mid-transaction.

While effective in our environment, this strategy assumes that chunk fetching of one payload remains confined to a single SQ, which may not hold across all SSD implementations [16] and may affect load distribution [27]. To relax this constraint, an identifier-based reassembly mechanism can be employed, allowing the controller to accept chunks out-of-order across SQs without enforcing strict ordering. To minimize SRAM usage during tracking of in-flight transactions [22], each chunk can embed metadata including a payload ID, chunk number, and total chunk count, enabling direct placement at the correct DRAM offset. Only lightweight metadata, such as the payload ID and a receive bitmap, is needed in SRAM to monitor reassembly progress. A full exploration of this mechanism is left as future work.

4 EVALUATION

4.1 Experimental Setup

We implemented and evaluated ByteExpress on a Linux host machine connected to the Cosmos+ OpenSSD platform [36]. The host machine is equipped with a 64-core Intel Xeon Gold 6226R CPU and 384 GB of DDR4 memory. The OpenSSD features an Xilinx Zynq-7000 FPGA, 1 TB NAND, and 1 GB DRAM. We set up OpenSSD with the host machine via a PCIe Gen2 8-lane connection using the NVMe protocol.

Implementation: ByteExpress[1] was implemented with minimal changes on both the host and device sides. On the host, we modified the Linux kernel NVMe driver [18]

[1] https://github.com/junttang/byteexpress

(v6.6.31), implementing the logic in under 30 lines of code within the `nvme_queue_rq(...)`. On the controller side, we extended the OpenSSD [36] by adding less than 20 lines to the `get_nvme_cmd(...)`, which fetches CMDs from the SQ.

4.2 Analysis on Various Payload Sizes

We evaluated PCIe traffic and transfer latency (with NAND I/O disabled on the OpenSSD) across various payload sizes using NVMe passthrough, issuing 1 million writes per configuration with NVMe PRP, BandSlim [35], and ByteExpress. PCIe traffic was measured via Intel PCM [12], and the results are shown in Figure 5.

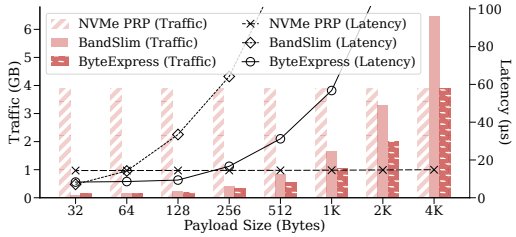


Figure 5: PCIe traffic and average latency for various payload sizes across different transfer methods.

Both ByteExpress and BandSlim significantly reduced traffic for payloads smaller than 4 KB compared to PRP. In particular, ByteExpress reduced traffic by up to 96.3% for the 64-byte case over PRP. Interestingly, in the 64-byte to 4 KB range, ByteExpress outperformed BandSlim by up to 39.8% in traffic reduction. This gain comes from ByteExpress's ability to inject payloads directly into the SQ, avoiding the overhead of issuing separate NVMe CMDs, such as doorbell ringing, tail pointer address updates, and completion signaling. For transfer latency, ByteExpress reduced latency by up to 40.4% over PRP in the 32–128 byte range and outperformed BandSlim beyond 64 bytes, for instance, achieving a 72% reduction at 128 bytes. These results highlight the clear latency advantage of ByteExpress for ~128-byte payloads, typical in real-world key-value store workloads and SQL predicate pushdowns (§2.2).

Table 1: The overheads introduced by ByteExpress.

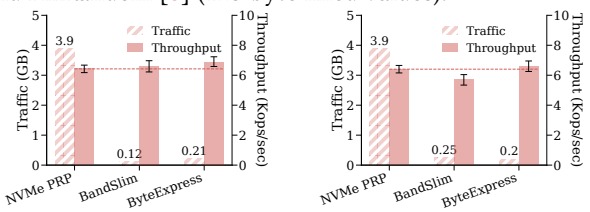
System	Driver SQ Submit	Controller SQ Fetch
NVMe PRP (ALL)	~ 60ns	~ 2400ns
ByteExpress (64B)	~ 100ns	~ 2800ns
ByteExpress (128B)	~ 130ns	~ 3200ns
ByteExpress (256B)	~ 180ns	~ 4000ns

Overhead Analysis: We evaluated ByteExpress's overhead during two stages: host-side insertion of payload chunks into the SQ and device-side fetching from the SQ (see Table 1). On average, inserting one chunk takes ~30ns, and fetching an SQ entry takes ~400ns. This overhead fairly increases with payload size due to more SQ entries being used. As shown in Figure 5, this causes ByteExpress to become slower than the PRP-based transfer starting around the 256-byte. However, this also implies that for smaller payloads, it achieves performance gains despite the added overhead, demonstrating that the cost per SQ entry at both sides is sufficiently low.

ByteExpress's performance drop beyond 256 bytes is a clear limitation compared to PCIe MMIO-based approaches, which sustain low latency even beyond 1 KB [1]. While this constraint stems from ByteExpress's adherence to the NVMe protocol and is therefore fundamental, one obvious optimization remains: ByteExpress can be combined with PRP using a simple threshold-based switching mechanism, just as proposed in BandSlim [35]. When the payload size exceeds a certain threshold (e.g., 256 bytes), PRP can be used; otherwise, ByteExpress is applied. Since ByteExpress requires no modifications to the core NVMe architecture, this hybrid approach can be employed seamlessly.

4.3 Effects on KV-SSD and CSD Workload

KV-SSD: We evaluated ByteExpress on a state-of-the-art OpenSSD-based KV-SSD [25] with NAND I/O enabled, running 1 million PUTs using MixGraph [4] (default settings) and FillRandom [8] (128-byte fixed values).

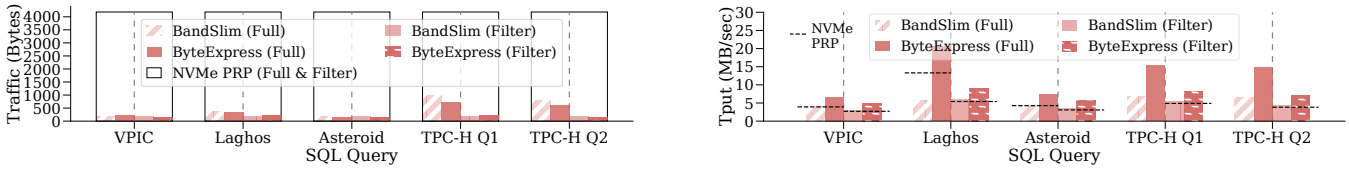


(a) MixGraph **(b) FillRandom (128 bytes)**

Figure 6: PCIe traffic and average write throughput for different transfer methods applied in the KV-SSD.

Figure 6 shows the results. In MixGraph, where over 60% of values are under 32 bytes (see Figure 1(a)), ByteExpress reduced PCIe traffic by up to 95% compared to PRP, though its traffic was 1.75× higher than BandSlim, which transmits sub-32-byte payloads within a single CMD. Despite this, ByteExpress achieved approximately 8% higher throughput than BandSlim (the error bars indicate the 1st–99th percentile range). In FillRandom, ByteExpress not only reduced PCIe traffic further than BandSlim but also achieved about additional 1 Kops/sec in throughput. These results show that ByteExpress robustly covers a wide range of small-payload scenarios, while maintaining both traffic efficiency and high performance even with NAND I/O.

SQL Predicate Pushdown for CSD: We evaluated ByteExpress in SQL predicate pushdown scenarios using example queries from Figure 4. For each query, we extracted both the full SQL string and the substring corresponding to the table identifier and predicate, and treated them as the computation task message to the SSD. As shown in Figure 7(a),



(a) PCIe Traffic **(b) Average Throughput**

Figure 7: (a) PCIe traffic and (b) average throughput for each method when transferring the full SQL strings (left of each vertical dashed line) and only the predicate and table name segments (right) of the queries in Figure 4.

both strings are well under 4 KB, allowing BandSlim and ByteExpress to significantly reduce traffic compared to PRP. For example, in the Asteroid case [39], both methods cut traffic by nearly 98% when sending a single pushdown task.

For transfer performance, ByteExpress achieved higher throughput than PRP for all cases when sending only the predicate and table name (see Figure 7(b)). In contrast, BandSlim's throughput was similar to or slightly below PRP. For workloads like VPIC [23], Laghos [40], and Asteroid [39], where the full SQL string is under 100 bytes, ByteExpress also outperformed both PRP and BandSlim even when sending the entire string. These results show that while some CSD-based systems use compact binary encodings [9, 14] and others [20, 29] adopt slightly larger intermediate representations [37], ByteExpress remains effective across this spectrum from very small to moderately sized payloads.

5 DISCUSSION

Comparison with Scatter-Gather List: In addition to PRPs, NVMe provides an alternative method known as the Scatter-Gather List (SGL) [28, 30], which enables variable-sized DMA transfers and thus can address traffic inefficiencies associated with small payloads. Specifically, for writes, SGL enables fine-grained DMA by allowing a single data block descriptor to directly reference a small, contiguous memory region. For reads, bitbucket descriptors in SGL can act as placeholders for unused segments, enabling completion of small-data read requests without requiring data return.

However, despite their functional overlap, SGL and ByteExpress exhibit clear differences in execution flow. The SGL requires constructing a SGL descriptor within the CMD, submitting the CMD, and having the controller interpret the descriptor before initiating a DMA transaction. ByteExpress also requires submitting the CMD but appends the payload directly into the subsequent SQ entries, enabling the controller to fetch the payload immediately without additional descriptor parsing. This streamlined design of ByteExpress is intended to reduce protocol-level overhead by avoiding descriptor handling and separate DMA setup, which can become increasingly significant as I/O sizes shrink.

Interestingly, the Linux NVMe driver sets a default threshold of 32 KB for enabling SGL [18, 41], which means that PRP is used for transfers smaller than this threshold unless reconfigured by the user. This indicates that while SGL is fully capable of supporting small transfers, PRP remains commonly used for small payloads in practice. Given this, and the fact that PRP is mandatory in NVMe over PCIe [30] and thus supported across all NVMe SSDs, we chose to focus our optimization efforts on PRP-based transfers in this study. A broader comparative analysis encompassing PRP, SGL and mechanisms such as ByteExpress would help complete the performance landscape for small I/O transfers.

Page Granularity and PCIe Generation Variants: Our current evaluation is limited to a 4 KB page granularity due to platform constraints [36]. While this granularity is standard in many NVMe-based systems, some storage configurations support finer-grained transfer units (e.g., 512 bytes) [10], which may affect the performance characteristics of ByteExpress. Likewise, higher-bandwidth PCIe generations (e.g., PCIe 4.0/5.0) could influence the relative impact of data movement optimizations [34]. Although not covered in our current evaluation, these variations represent important axes that could affect the applicability and benefits of ByteExpress under broader system conditions.

6 CONCLUSION

ByteExpress offers a lightweight, practical solution to the long-standing inefficiency of small, direct data transfer over NVMe. By repurposing the NVMe submission queue to carry 64-byte payload chunks, ByteExpress enables efficient, fine-grained small-data transmission without modifying SSD internal architecture or NVMe passthrough-based APIs. Evaluation results show that ByteExpress reduces PCIe traffic by up to 98% and significantly improves performance for small I/O patterns common in KV-SSDs and CSDs.

ACKNOWLEDGMENTS

We thank the reviewers and our shepherd, Javier González, for their constructive comments that have significantly improved the paper. This work was funded in part by the National Research Foundation of Korea (NRF) grant funded by the Korean Government (MSIT) (RS-2024-00453929), in part by the Institute of Information & Communications Technology Planning Evaluation (IITP) grant funded by the Korea Government (MSIT) under Grant RS-2021-II210528.

REFERENCES

[1] Duck-Ho Bae, Insoon Jo, Youra Adel Choi, Joo-Young Hwang, Sangyeun Cho, Dong-Gi Lee, and Jaeheon Jeong. 2018. 2B-SSD: the Case for Dual, Byte- and Block-Addressable Solid-State Drives. In *Proceedings of the 45th Annual International Symposium on Computer Architecture* (Los Angeles, California) *(ISCA '18)*. IEEE Press, Piscataway, NJ, USA, 425–438. https://doi.org/10.1109/ISCA.2018.00043

[2] Lawrence Benson, Hendrik Makait, and Tilmann Rabl. 2021. Viper: an efficient hybrid PMem-DRAM key-value store. *Proc. VLDB Endow.* 14, 9 (May 2021), 1544–1556. https://doi.org/10.14778/3461535.3461543

[3] Wei Cao, Yang Liu, Zhushi Cheng, Ning Zheng, Wei Li, Wenjie Wu, Linqiang Ouyang, Peng Wang, Yijing Wang, Ray Kuan, Zhenjun Liu, Feng Zhu, and Tong Zhang. 2020. 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* (Santa Clara, CA, USA) *(FAST'20)*. USENIX Association, USA, 29–42.

[4] Zhichao Cao, Siying Dong, Sagar Vemuri, and David H. C. Du. 2020. Characterizing, modeling, and benchmarking RocksDB key-value workloads at facebook. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies* (Santa Clara, CA, USA) *(FAST'20)*. USENIX Association, USA, 209–224.

[5] Walter Daelemans, Mike Kestemont, Enrique Manjavacas, Martin Potthast, Francisco Rangel, Paolo Rosso, Günther Specht, Efstathios Stamatatos, Benno Stein, Michael Tschuggnall, Matti Wiegmann, and Eva Zangerle. 2019. Overview of PAN 2019: Bots and Gender Profiling, Celebrity Profiling, Cross-Domain Authorship Attribution and Style Change Detection. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction. Proceedings of the 10th International Conference of the CLEF Association (CLEF 2019)* (Lugano, Switzerland). Springer-Verlag, Berlin, Heidelberg, 402–416.

[6] etcd Authors. 2023. etcd Raft Log Durability and Performance. https://etcd.io/docs/v3.5/faq/#why-is-etcd-so-slow Accessed: 2025-05-25.

[7] Facebook. 2012. *RocksDB*. Facebook. https://rocksdb.org Last accessed: 2024-09-01.

[8] Facebook. 2017. *DB Bench*. Facebook. https://github.com/facebook/rocksdb/wiki/Benchmarking-tools Last accessed: 2024-10-01.

[9] Boncheol Gu, Andre S. Yoon, Duck-Ho Bae, Insoon Jo, Jinyoung Lee, Jonghyun Yoon, Jeong-Uk Kang, Moonsang Kwon, Chanho Yoon, Sangyeun Cho, Jaeheon Jeong, and Duckhyun Chang. 2016. Biscuit: a framework for near-data processing of big data workloads. In *Proceedings of the 43rd International Symposium on Computer Architecture* (Seoul, Republic of Korea) *(ISCA '16)*. IEEE Press, Piscataway, NJ, USA, 153–165. https://doi.org/10.1109/ISCA.2016.23

[10] Gabriel Haas and Viktor Leis. 2023. What Modern NVMe Storage Can Do, and How to Exploit it: High-Performance I/O for High-Performance Storage Engines. *Proc. VLDB Endow.* 16, 9 (May 2023), 2090–2102. https://doi.org/10.14778/3598581.3598584

[11] Junsu Im, Jinwook Bae, Chanwoo Chung, Arvind Arvind, and Sungjin Lee. 2020. PinK: high-speed in-storage key-value store with bounded tails. In *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC'20)*. USENIX Association, USA, Article 12, 15 pages.

[12] Intel. 2022. Intel® Performance Counter Monitor - A Better Way to Measure CPU Utilization. https://www.intel.com/content/www/us/en/developer/articles/tool/performance-counter-monitor.html. Last Accessed: 2024-10-01.

[13] Yanqin Jin, Hung-Wei Tseng, Yannis Papakonstantinou, and Steven Swanson. 2017. KAML: A Flexible, High-Performance Key-Value SSD. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Piscataway, NJ, USA, 373–384. https://doi.org/10.1109/HPCA.2017.15

[14] Insoon Jo, Duck-Ho Bae, Andre S. Yoon, Jeong-Uk Kang, Sangyeun Cho, Daniel D. G. Lee, and Jaeheon Jeong. 2016. YourSQL: a high-performance database system leveraging in-storage computing. *Proc. VLDB Endow.* 9, 12 (Aug. 2016), 924–935. https://doi.org/10.14778/2994509.2994512

[15] Min-Gyo Jung, Chang-Gyu Lee, Donggyu Park, Sungyong Park, Jungki Noh, Woosuk Chung, Kyoung Park, and Youngjae Kim. 2021. GPUKV: an integrated framework with KVSSD and GPU through P2P communication support. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing* (Virtual Event, Republic of Korea) *(SAC '21)*. Association for Computing Machinery, New York, NY, USA, 1156–1164. https://doi.org/10.1145/3412841.3441990

[16] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. 2014. The multi-streamed solid-state drive. In *Proceedings of the 6th USENIX Conference on Hot Topics in Storage and File Systems* (Philadelphia, PA) *(HotStorage'14)*. USENIX Association, USA, 13.

[17] Linux Kernel. 2025. libnvme ioctl.h Source Code. https://github.com/linux-nvme/libnvme/blob/master/src/nvme/ioctl.h Accessed: 2025-04-01.

[18] Linux Kernel. 2025. NVMe PCI Driver Source Code. https://github.com/torvalds/linux/blob/master/drivers/nvme/host/pci.c Accessed: 2025-04-01.

[19] Yang Seok Ki. 2017. Key-Value SSD Explained: Concept, Device, System and Standard. https://www.snia.org/sites/default/files/SDC/2017/presentations/Object_ObjectDriveStorage/Ki_Yang_Seok_Key_Value_SSD_Explained_Concept_Device_System_and_Standard.pdf. Presented at SNIA Storage Developer Conference (SDC), 2017.

[20] Jongryool Kim. 2023. Accelerating Data Analytics Using Object-based Computational Storage System in HPC. In *Supercomputing Conference (SC23), Exhibitor Forum*. SK hynix Inc., SC Conference Organizing Committee, Denver, CO, USA, N/A. Presented at SC23 Exhibitor Forum, Booth #2101.

[21] Sang-Hoon Kim, Jinhong Kim, Kisik Jeong, and Jin-Soo Kim. 2019. Transaction support using compound commands in key-value SSDs. In *Proceedings of the 11th USENIX Conference on Hot Topics in Storage and File Systems* (Renton, WA, USA) *(HotStorage'19)*. USENIX Association, USA, 1.

[22] Taejin Kim, Duwon Hong, Sangwook Shane Hahn, Myoungjun Chun, Sungjin Lee, Jooyoung Hwang, Jongyoul Lee, and Jihong Kim. 2019. Fully automatic stream management for multi-streamed SSDs using program contexts. In *Proceedings of the 17th USENIX Conference on File and Storage Technologies* (Boston, MA, USA) *(FAST'19)*. USENIX Association, USA, 295–308.

[23] Los Alamos National Laboratory. 2025. Vector Particle-In-Cell (VPIC) Project. https://github.com/lanl/vpic Accessed: 2025-04-01.

[24] Chang-Gyu Lee, Hyeongu Kang, Donggyu Park, Sungyong Park, Youngjae Kim, Jungki Noh, Woosuk Chung, and Kyoung Park. 2019. iLSM-SSD: An Intelligent LSM-Tree Based Key-Value SSD for Data Analytics. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, Piscataway, NJ, USA, 384–395. https://doi.org/10.1109/MASCOTS.2019.00048

[25] Seungjin Lee, Chang-Gyu Lee, Donghyun Min, Inhyuk Park, Woosuk Chung, Anand Sivasubramaniam, and Youngjae Kim. 2023. Iterator Interface Extended LSM-tree-based KVSSD for Range Queries. In *Proceedings of the 16th ACM International Conference on Systems and Storage* (Haifa, Israel) *(SYSTOR '23)*. Association for Computing Machinery, New York, NY, USA, 60–70. https://doi.org/10.1145/3579370.3594775

[26] Shaobo Li, Yirui (Eric) Zhou, Hao Ren, and Jian Huang. 2025. ByteFS: System Support for (CXL-based) Memory-Semantic Solid-State Drives. In *Proceedings of the 30th ACM International Conference on Architectural*

Support for Programming Languages and Operating Systems, Volume 1 (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 116–132. https://doi.org/10.1145/3669940.3707250

[27] Xiaojian Liao, Youyou Lu, Zhe Yang, and Jiwu Shu. 2023. Efficient Crash Consistency for NVMe over PCIe and RDMA. *ACM Trans. Storage* 19, 1, Article 7 (Jan. 2023), 35 pages. https://doi.org/10.1145/3568428

[28] Robert E. Marks. 2013. NVM Express: A Standard for Non-Volatile Memory. https://files.futurememorystorage.com/proceedings/2013/20130812_PreConfD_Marks.pdf. Presented at Flash Memory Summit Pre-Conference Seminar, August 12, 2013.

[29] Aldrin Montana, Yuanqing Xue, Jeff LeFevre, Carlos Maltzahn, Josh Stuart, Philip Kufeldt, and Peter Alvaro. 2023. A Moveable Beast: Partitioning Data and Compute for Computational Storage. arXiv:cs.DC/2212.11459 https://arxiv.org/abs/2212.11459

[30] NVM Express Inc. 2011. NVM Express Specification. https://nvmexpress.org/developers/nvme-specification. Last Accessed: 2024-09-12.

[31] NVM Express Inc. 2021. NVM Express Key Value Command Set Specification. https://nvmexpress.org/developers/nvme-specification/. Last Accessed: 2024-09-12.

[32] Chanyoung Park, Jungho Lee, Chun-Yi Liu, Kyungtae Kang, Mahmut Taylan Kandemir, and Wonil Choi. 2025. AnyKey: A Key-Value SSD for All Workload Types. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 47–63. https://doi.org/10.1145/3669940.3707279

[33] Inhyuk Park, Qing Zheng, Dominic Manno, Soonyeal Yang, Jason Lee, David Bonnie, Bradley Settlemyer, Youngjae Kim, Woosuk Chung, and Gary Grider. 2023. KV-CSD: A Hardware-Accelerated Key-Value Store for Data-Intensive Applications. In *2023 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, Piscataway, NJ, USA, 132–144. https://doi.org/10.1109/CLUSTER52292.2023.00019

[34] Junhyeok Park, Chang-Gyu Lee, Soon Hwang, Seung-Jun Cha, Woosuk Chung, and Youngjae Kim. 2025. Maximizing Interconnect Bandwidth and Efficiency in NVMe-Based Key-Value SSDs with Fine-Grained Value Transfer. *IEEE Micro* (2025). https://doi.org/10.1109/MM.2025.3572475

[35] Junhyeok Park, Chang-Gyu Lee, Soon Hwang, Soonyeal Yang, Jungki Noh, Woosuk Chung, Junghee Lee, and Youngjae Kim. 2024. BandSlim: A Novel Bandwidth and Space-Efficient KV-SSD with an Escape-from-Block Approach. In *Proceedings of the 53rd International Conference on Parallel Processing* (Gotland, Sweden) *(ICPP '24)*. Association for Computing Machinery, New York, NY, USA, 1187–1196. https://doi.org/10.1145/3673038.3673064

[36] OpenSSD Project. 2017. Cosmos+ OpenSSD Platform. http://www.openssd-project.org/platforms/cosmospl. Last accessed: 2024-09-22.

[37] Substrait Project. 2024. Substrait. https://substrait.io/.

[38] Redis. 2024. Redis Persistence - Appendfsync Always. https://redis.io/docs/latest/operate/oss_and_stack/management/persistence/ Accessed: 2025-05-25.

[39] LANL Asteroid Impact Team. 2017. Deep Water Impact Dataset. https://github.com/lanl-asteroid-impact/deep-water-impact-dataset-1 Accessed: 2025-04-01.

[40] LANL-OCS Team. 2025. Laghos Sample Dataset. https://github.com/lanl-ocs/laghos-sample-dataset Accessed: 2025-04-01.

[41] Mellanox Technologies. 2017. nvme: add Scatter-Gather List (SGL) support in NVMe driver. https://lore.kernel.org/all/04aaed5c-1a8a-f601-6c9c-88bf1cf66e8a@mellanox.com/T/. Email on Linux Kernel Mailing List.

[42] Devesh Tiwari, Simona Boboila, Sudharshan S. Vazhkudai, Youngjae Kim, Xiaosong Ma, Peter J. Desnoyers, and Yan Solihin. 2013. Active flash: towards energy-efficient, in-situ data analytics on extreme-scale machines. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies* (San Jose, CA) *(FAST'13)*. USENIX Association, USA, 119–132.

[43] Xiangqun Zhang, Janki Bhimani, Shuyi Pei, Eunji Lee, Sungjin Lee, Yoon Jae Seong, Eui Jin Kim, Changho Choi, Eyee Hyun Nam, Jongmoo Choi, and Bryan S. Kim. 2025. Storage Abstractions for SSDs: The Past, Present, and Future. *ACM Trans. Storage* 21, 1, Article 2 (Jan. 2025), 44 pages. https://doi.org/10.1145/3708992

[44] Yiwen Zhang, Jian Zhou, Xinhao Min, Song Ge, Jiguang Wan, Ting Yao, and Daohui Wang. 2023. PetaKV: Building Efficient Key-Value Store for File System Metadata on Persistent Memory . *IEEE Transactions on Parallel & Distributed Systems* 34, 03 (March 2023), 843–855. https://doi.org/10.1109/TPDS.2022.3232382

[45] Qing Zheng, Jason Lee, Dominic A. Manno, and Gary Grider. 2023. Toward Standardized, Open Object-Based Computational Storage For Large-Scale Scientific Data Analytics. In *Proceedings of the 8th International Parallel Data Systems Workshop (PDSW'23)*. IEEE, Denver, CO, USA, N/A. https://pdsw.org/pdsw23/index.shtml