A **MICRON** WHITE PAPER

Comparing UFS and NVMe™ Storage Stack and System-Level Performance in Embedded Systems

Bean Huo, Blair Pan, Peter Pan, Zoltan Szubbocsev
Micron Technology

Introduction

Embedded storage systems have experienced a rapid pace of innovation in the last couple of years. While eMMC has been a mainstream storage of choice for many embedded solutions, recently introduced storage technologies — such as UFS and NVMe — are offering great interface speeds that allow systems to make use of the underlying speed of NAND technologies.

In embedded systems, the storage software (SW) stack has a profound impact on user-level speeds. UFS and NVMe differ greatly in their storage SW stacks. The UFS storage stack builds on the SCSI stack, whereas NVMe has a storage stack designed specifically for managed NAND devices, one that has been significantly simplified compared to the SCSI stack to realize high storage speeds at the user level.

In this white paper, we compare and quantify the differences between UFS and NVMe user-level storage performances. We first introduce the methodology used in this study, then we compare the system-level performance between UFS and NVMe using the measured overheads and present the results. All results are based on Micron tracing results used during benchmarks on a HiKey® 960 reference board.

Our conclusion, based on measurements and estimation, show that due to the simpler storage SW stack, NVMe achieves 28% faster system-level speed in sequential writes and 15% faster speeds in sequential reads, and 30% higher IOPS in random writes and 16% higher IOPS in random reads. This inherent advantage of NVMe is expected to yield even greater benefits with faster storage devices.

Methodology

To measure system-level overhead, we used two standard Linux tools: FTRACE and BLKTRACE. FTRACE is an internal tracer tool designed to provide developers the capability to understand what goes on inside the Linux kernel. BLKTRACE is a tool that uses FTRACE to specifically trace the Linux block layer. These tools can be used to measure the time spent in defined areas of the kernel.

Figure 1 illustrates a Linux storage stack, depicted from the application layer down to the storage device. In this image, there are four defined latency areas:

1. VFS-FS latency
2. Block layer submission to storage device submission latency
3. Hardware (HW) transfer latency
4. Request post-processing latency

The VFS-FS latency is the time between when a user submits a file input-output (IO) request to the time the file system (FS) submits a corresponding IO request to the Linux block layer. Block layer submission to storage device submission latency is the period between the time when the block layer receives an IO request to the time a corresponding request is submitted to the storage device. HW transfer latency is defined as the time between the IO submission to the storage device to the interrupt generation signaling IO completion from the storage device. The request post-processing latency is the time between receiving the IO completion interrupt to signaling IO completion to the file system (FS).

Overall system-level overhead is composed of these four latencies. The described tools (FTRACE and BLKTRACE) were used to measure the average time the OS spends in these four sections.

For workload generation, we used the standard LINUX flexible IO (FIO) tester tool and examined the storage stack performance under 4K random read/write, 128KB sequential read/write workloads. We used the 4KB chunk size to model small random accesses and the 128KB chunk size to model larger sequential accesses. We used direct and sync IO configurations and applied both single and eight threaded workloads. By increasing the number of threads, it is expected that the number of concurrently dispatched commands will increase the storage performance and system overhead will decrease, thus increasing overall storage throughput.

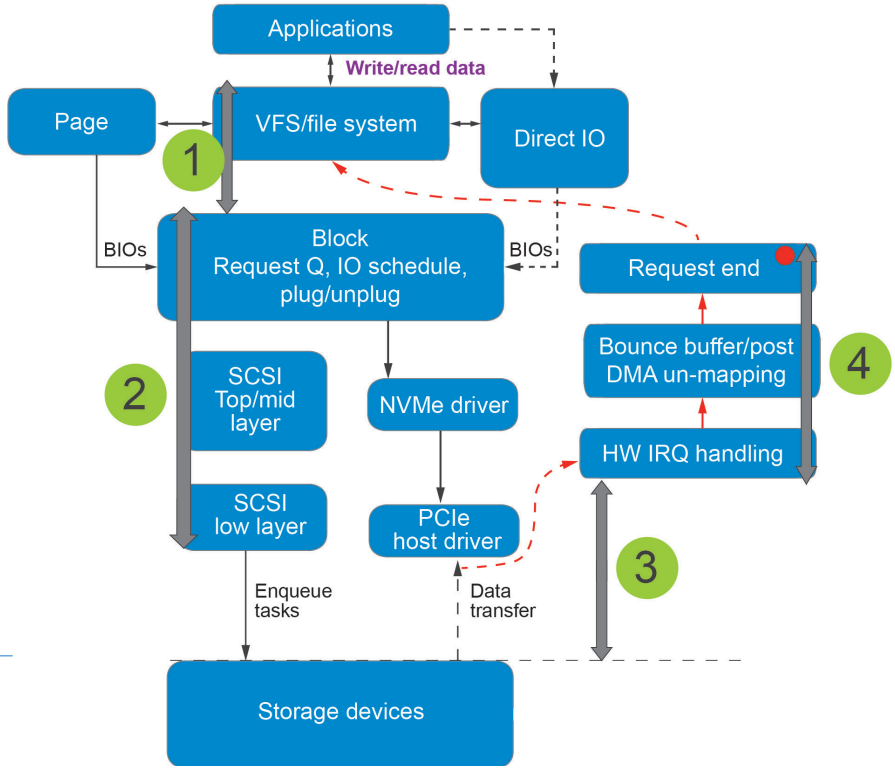

Figure 1: Linux Storage Stack

Figure 1 depicts the Linux storage stack for both NVMe and SCSI paths. The NVMe stack consists of fewer layers; this translates to a significant reduction in code length leading to a significant performance advantage for NVMe. The storage stack impact includes larger IO latencies and associated reduction in effective storage bandwidth.

After measuring the average latencies in the four defined areas, we applied the measured system SW overheads (areas 1, 2 and 4) on assumed HW latencies (area 3) to compare the two storage stacks. We compared them by assuming equivalent HW latencies for both UFS and NVMe (in other words, we assumed latency 3 will be equivalent for UFS and NVMe). This is, of course, is not the case in the real world because UFS and NVMe show different performances, and even between two UFS and NVMe memory devices, performance will vary. This methodology nevertheless can be applied as 1, 2 and 4 are independent from 3. We can substitute a faster or slower device in the stack, but the average latencies in 1, 2 and 4 will not change.

System Description

To measure system overhead for both UFS and NVMe, we used a HiKey® 960 embedded board. This board supports UFS 2.1 and NVMe 1.2 and consists of eight cores (four Cortex®-A73 and four Cortex-A53).

HiKey960 Board Specifications	
CPU	Cortex-A73 (x4) + Cortex-A53 (x4) (big.LITTLE)
OS	Android™, Linux kernel 4.4.80
IO Type	Direct IO
File System	ext4
IO Tool	fio
IO Trace Tool	blktrace/ftrace

Table 1: HiKey 960 Board Specifications

HiKey960 Storage Specifications	UFS	NVMe
Lanes	2	1
Density	128GB	128GB
Phy/Link Interface	M-PHY® Gear3	PCIe® Gen2
Queues	1	8
Queue Depth	UFS supports up to 256, but UFS host capacity supports only 32	1024 per queue
INTs	1	1 (sharing)

Table 2: HiKey 960 Storage Specifications

UFS and NVMe System Overhead

This section presents the measured system overhead results. For both UFS and NVMe, system overhead is significant for 4KB and 128KB chunk sizes; this cannot be ignored when evaluating user-level storage performance. The overall system performance is, of course, a factor of not only the system overhead but also of the underlying storage speed.

For the sake of simplicity, we combined the three system overhead areas (1, 2 and 4) so that the total IO duration is composed of two durations: HW duration and SW duration. We grouped three IO latencies (1, 2 and 4) into a single latency to better isolate total system overhead from HW overhead. Latencies 1, 2 and 4 compose the total SW overhead, and latency 3 consists of the HW duration. The system overhead (latencies 1, 2 and 4) is presented as a percentage of the total IO duration.

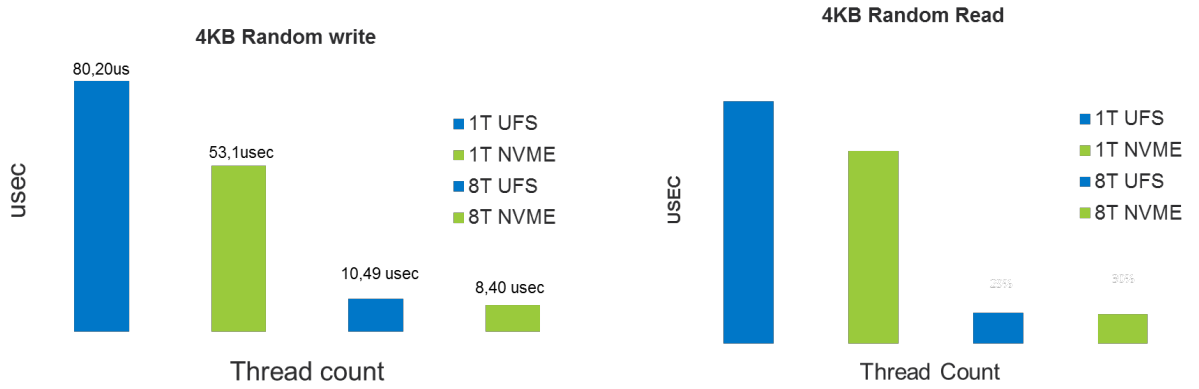

Figure 2. IO Duration Breakdown for 4KB Random Write/Read

Figure 2 represents the system overhead portion of an IO duration for 4K random read and write for UFS and NVMe. Observe the SW overhead for random write makes up most of the IO duration time, ranging as high as 75%. For reads, the performance impact of the SW overhead is smaller; however, it is still significant and ranges between 20-40% depending on interface and number of threads used. Single and 8 threaded workloads were used to measure the benefits of concurrently executed commands, both in the storage device and the storage software. As expected, by increasing the number of threads, the storage stack efficiency increases; however, the storage overhead is still significant with respect to the HW overhead.

Figure 3 represents the system overhead portion of an IO duration for 128KB sequential read and write workloads. The SW overheads are significantly smaller than with 4KB workloads; however, they still make up a significant portion of the IO duration, especially in single threaded workloads. In summary, these two graphs demonstrate that SW overhead cannot be ignored when considering system-level performance numbers. For smaller data access sizes, system overhead makes up a large percentage of the IO duration for both UFS and NVMe.

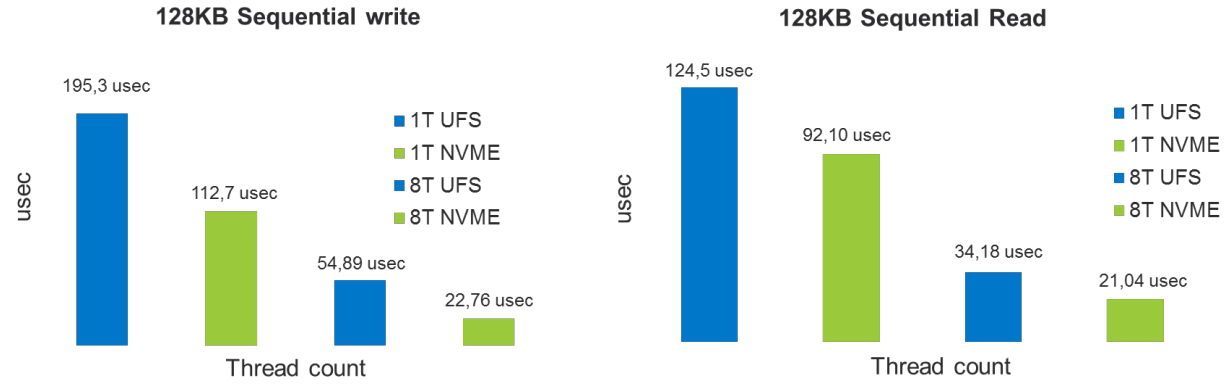

Figure 3. IO Duration Breakdown for 128KB Sequential Read/Write

System Overhead and Performance Comparison

In this section, we compare the system overhead between UFS and NVMe. This is followed by a comparison of system-level performance between UFS and NVMe. As explained in the methodology section, we compare the two by applying the measured system overheads (latencies 1, 2 and 4) to an assumed HW latency, which is assumed to be equivalent for both UFS and NVMe to compare similar devices in terms of performances. This assumption can be made because the system overhead measured in latencies 1, 2 and 4 are independent to the HW latencies, and therefore, can be applied to any HW device. Both UFS and NVMe are assumed to have the same performance characteristics. To compare two equivalent UFS and NVMe devices, we use the measured overheads together with an assumed device speed to calculate expected system performance.

First, we compared the system overhead numbers without considering HW duration. Figures 4 and 5 show the NVMe overhead compared to UFS. In all four cases, NVMe overhead is expressed in terms of UFS overhead as a percentage. In all four cases, NVMe overhead is smaller than that of UFS.

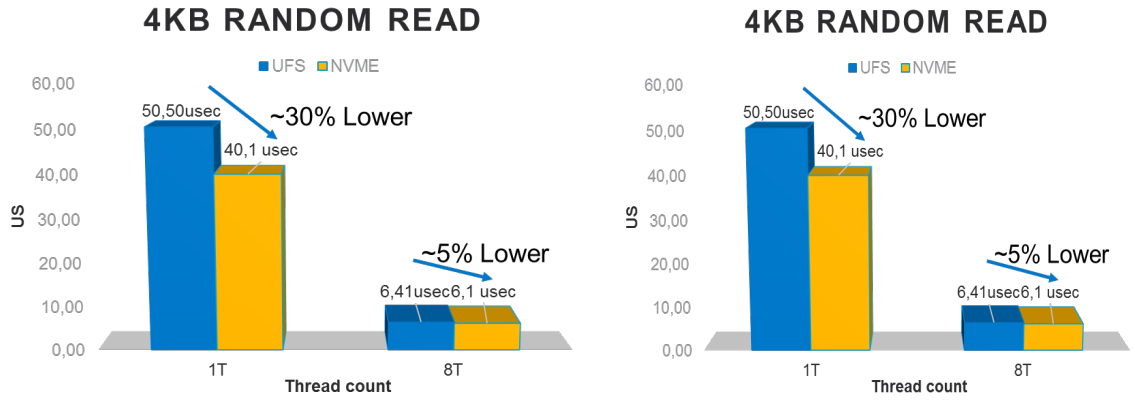

Figure 4: UFS, NVMe Overhead Comparison for 4KB Random Read/Write

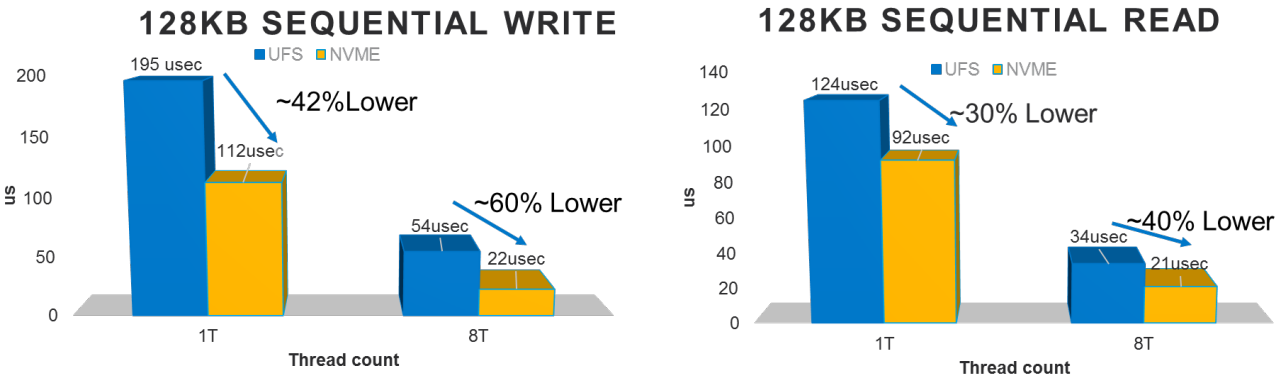

Figure 5: UFS, NVMe Overhead Comparison for 128KB Sequential Read/Write

Next, we compared NVMe and UFS system-level performance considering two identical devices (one UFS and one NVMe). For estimation purposes, we assumed the following latencies: 125 usec for 128KB write, 104 usec for 128KB read, 12.5 usec for 4KB write and 16.6 for 4KB read. These numbers correspond to a performance characteristic of a sample 256GB NVMe device, and the numbers used are actual measured values. We could use other numbers and the methodology used can still be applied. We picked a 256GB device for the sake of this study, but other densities could also be used. With higher capacity devices, it is expected that the system overhead impact will be greater and the advantage of NVMe will be even more pronounced.

Figure 6 represents estimated system-level performance numbers for UFS and NVMe. We applied the measured system overhead numbers to the above specified device characteristics and calculated the expected system performance numbers. We observed an increase of 28% in sequential 128KB write speed and a 15% increase in sequential read speed. In random write, NVMe showed a 30% increase in IOPS over UFS and a 16% increase in random read IOPS.

In summary, in all four examined workloads, NVMe performs significantly better because the storage SW stack is significantly simpler.

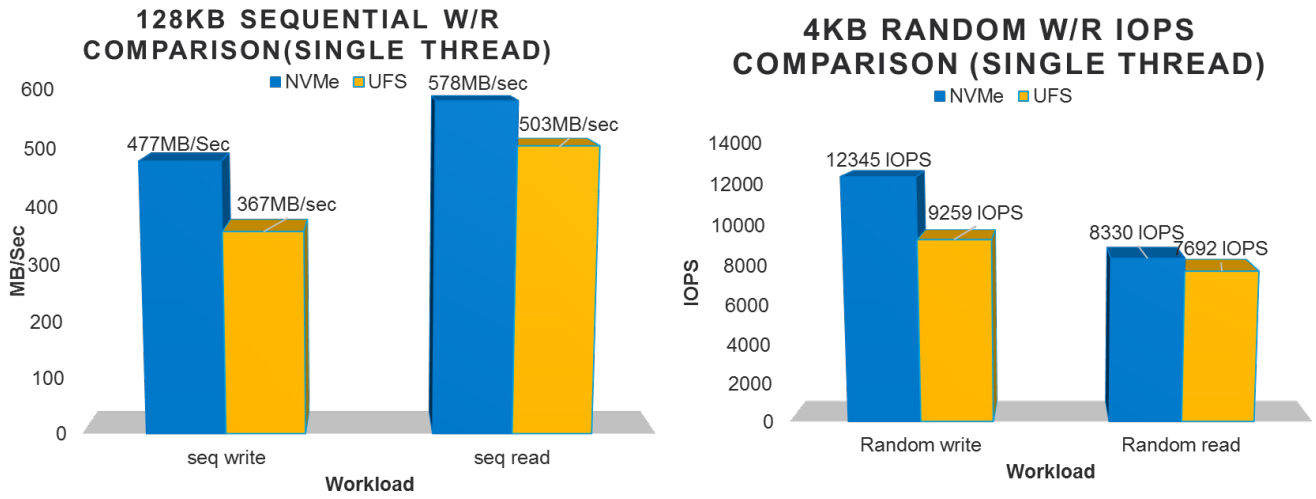

Figure 6. System-Level Performance Estimates

Conclusion

In this study, we explored the impact of system overhead on system-level performance for UFS and NVMe. We found that in the system evaluated, the overhead is significant and it greatly impacts the storage performance users experience. We conclude that faster storage devices will not necessary translate to faster user experiences due to software overheads. We compared the system-level overhead for UFS and NVMe and showed that due to the simpler storage stack of NVMe, significant system-level benefits can be achieved versus UFS, independent of the storage device speeds. These benefits can be attributed to NVMe's more efficient storage SW stack.

micron.com