Linux Storage System Bottleneck Exploration

Bean Huo / Zoltan Szubbocsev

Beanhuo@micron.com / *zszubbocsev@micron.com*

Outline

- Introduction
- Introduction to Linux storage stack analysis methodology
- eMMC stack analysis
- UFS and NVMe stack analysis comparison
- Summary

Introduction

About us

- Work for Micron
- Part of the embedded business unit
- Focusing on embedded storage SW
- Based in Munich part of the embedded system architecture and engineering group
- Areas we work on, embedded file systems, eMMC, UFS, NVMe for embedded

Goals of our project

- Quantify storage system overhead in embedded systems for eMMC, UFS and NVMe
  - Understand how much of the physical speed is realized at user level
  - Get an idea on how much Linux storage stack impacts the overall user space performance
- Quantify NVMe storage stack improvements over UFS storage stack
  - Does NVMe provide a better user level speed in embedded systems?
  - If we put two equivalent UFS and NVMe devices in a system which would be better and how much?
  - How much improvements does it provide over a UFS stack?

Introduction to eMMC, UFS and NVMe

- All three are solid state drive technologies, below in chronological order
  - Consisting of NAND chips plus controller and firmware
- eMMC (embedded multi-media card)
  - HS400 and Maximum speed can reach to 400 MB/s
- UFS (universal flash storage)
  - UFS Gear3 728 MB/s per lane
- NVMe (non-volatile memory express)
  - Gen3 1000MB/s per lane

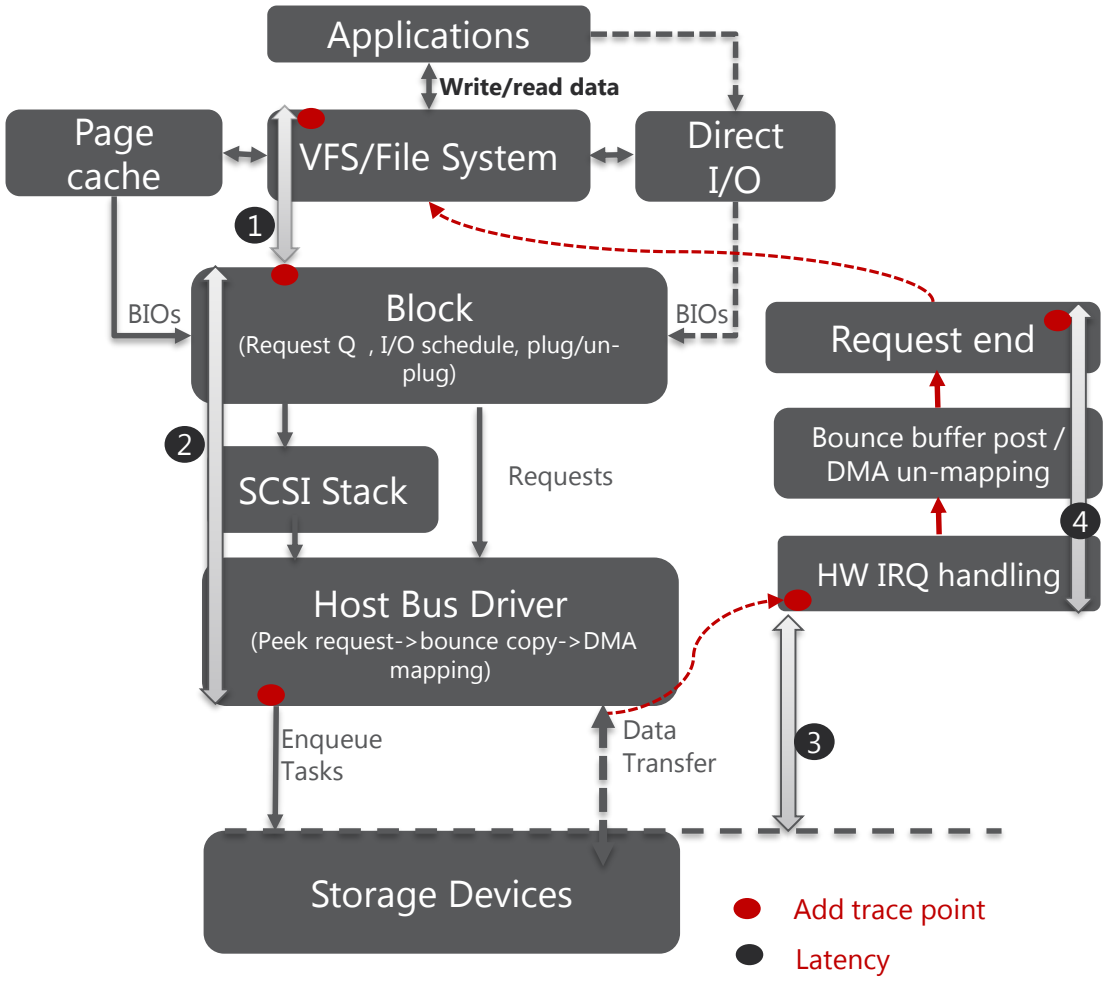
Introduction to Linux storage stack analysis methodology

Micron

Methodology

- Workload generation
  - Fio tool
  - Single and multi-threaded workloads (1, 8 thread)
    - 4KB random read, write
    - 128 KB sequential read, write
    - Direct IO and sync IO
- The utilities of tracing
  - Ftrace
    - Tracer: function_graph
    - trace_printk() add tracing point
  - Blktrace
    - Cannot trace VFS-FS layer
    - blk_add_trace_msg()

Methodology



- Latencies broken down to 4 sections
  - VFS-FS latency: user space submission to block layer receiving a BIO
  - Block layer submission to storage device submission
  - HW transfer, storage submission to completion interrupt
  - Request post, completion interrupt to block layer completion

eMMC stack analysis

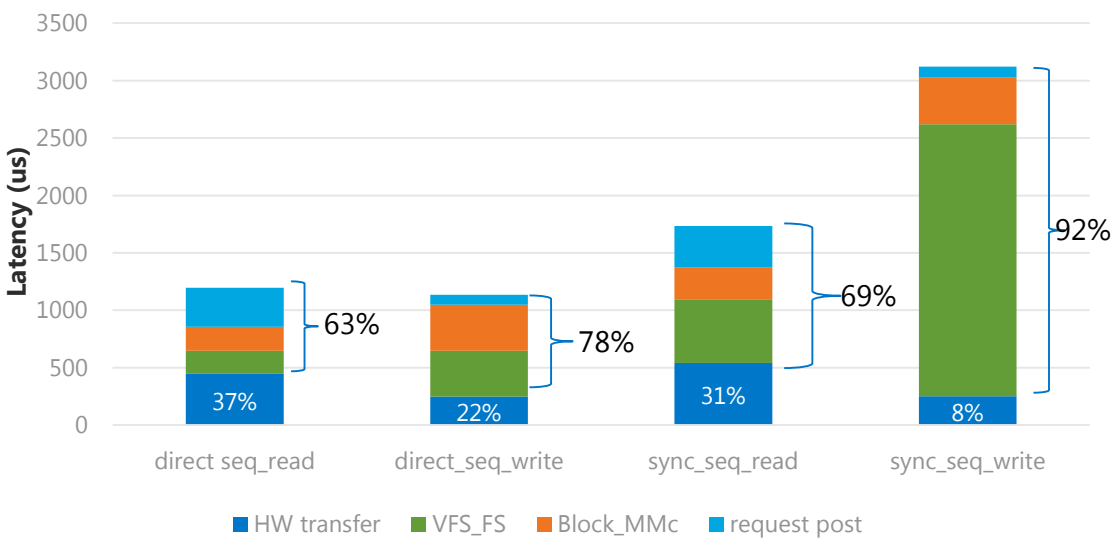
Linux® Storage System Analysis for e.MMC With Command Queuing
freely download from ----------https://www.micron.com/resource-details/1ccd41ac-8196-4987-8d46-83b2067d1ba5

Speciation Of eMMC Target Platforms

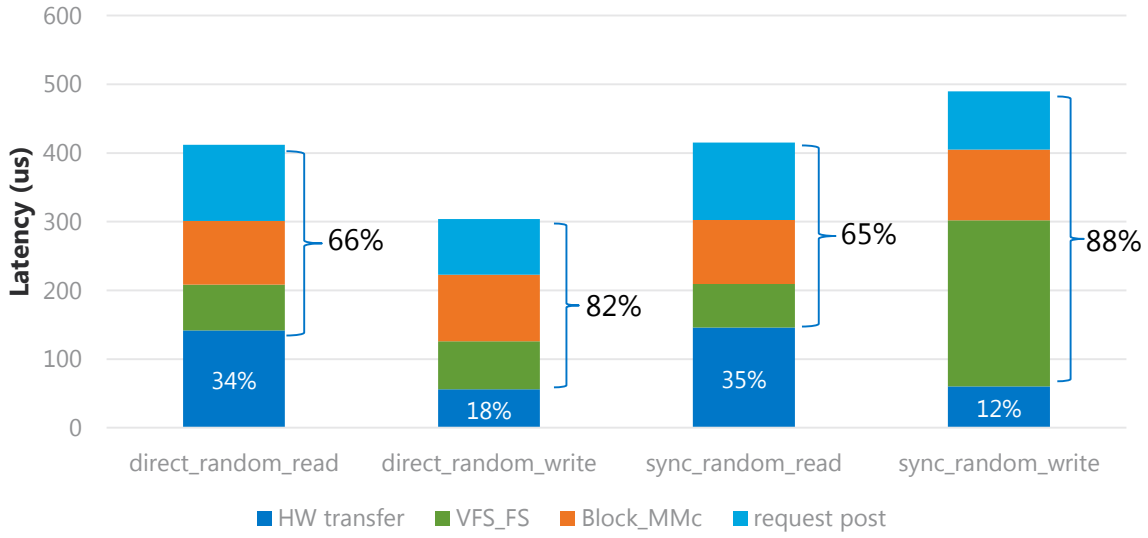
	Xilinx Zynq Zed board	Jenson TX1 NVIDIA Board
CPU type	ARM Cortex A9	ARM Cortex-A57 MPcore
Core number	2	4
L1 cache	32KB L1 I-cache and 32KB L1 D-caches with parity per core	48KB L1 I-cache per core; 32KB L1 D-cache per core
L2 cache	512KB (Unified Cache)	2MB (Unified Cache)
CPU frequency	667MHz	1,73GHz
DDR type	512MB DDR3 (32bit)	4GB LPDDR4 (64 bit)
DDR frequency	533MHz	1600MHz
CQE	Hardware CQ engine	software simulation CQ
eMMC	32GB@HS400	8GB@HS400
Block FS	Ext4	Ext4

eMMC system overhead on the Xilinx Zed

64KB data I/O request system overhead on the Zed

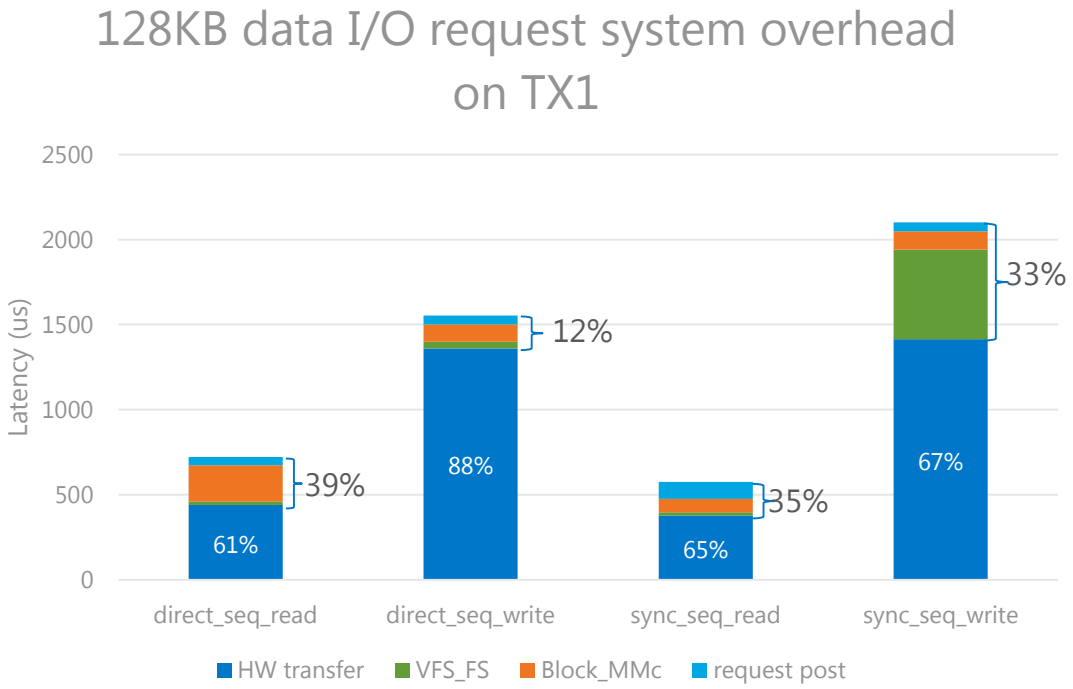


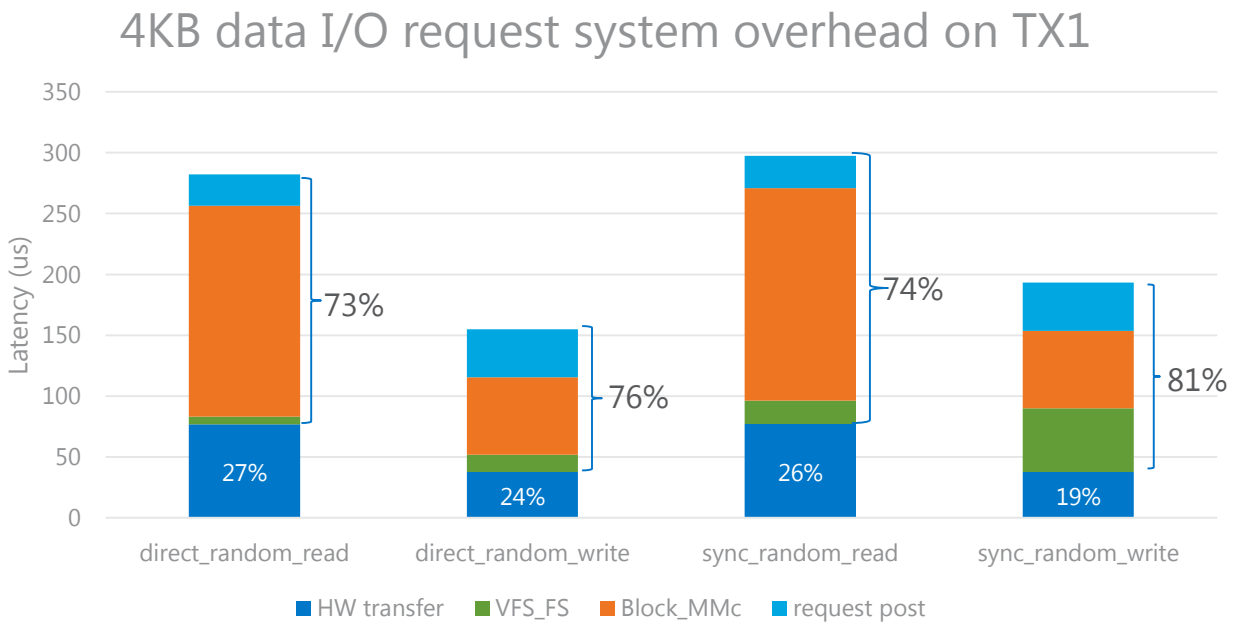
4KB data I/O request system overhead on the Zed



- On the Zed board, the eMMC performance is dominated by software overhead, rather than device.

eMMC system overhead on the TX1





- The performance with 128KB chunk size is impacted by 12-39% system overhead.
- With 4KB chunk size, we observe 73-81% system overhead.

Summary

- Direct I/O has better performance than Sync I/O due to lack of memory copy operation and page cache usage.
- System overhead is observed to be a significant contributor to I/O duration and even in higher end system with small chunk accesses it is significant.
- Even for high speed hardware platform, the conventional Linux software stack can not fully exploit the speed provided by high speed eMMC devices.

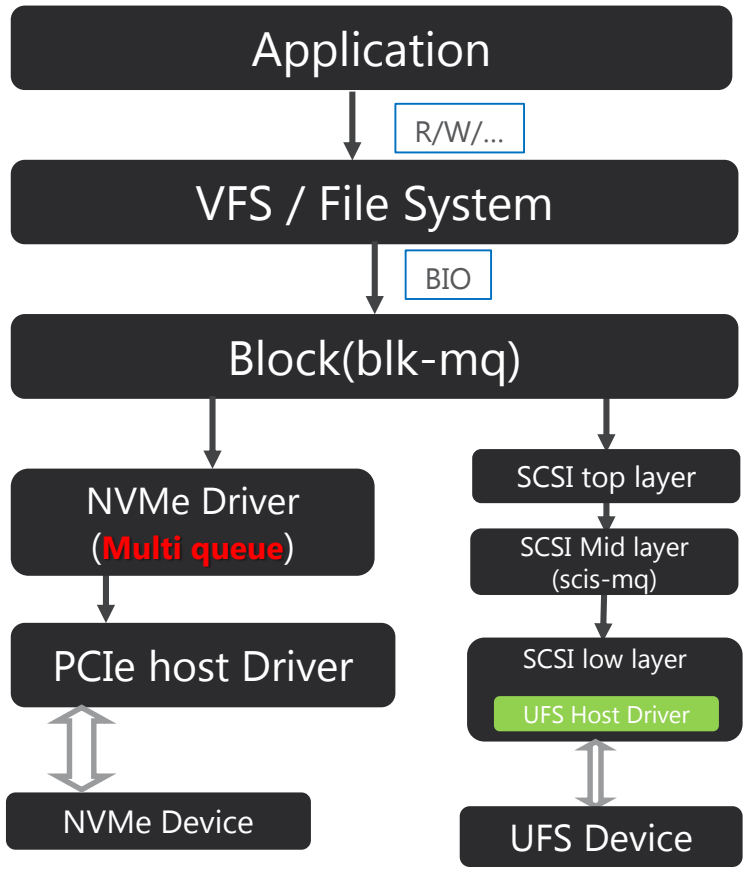
UFS and NVMe stack analysis comparison

Speciation Of UFS/NVMe Target Platform

	Hikey960
CPU #	A73 x 4 + A53 x 4(BigLittle)
OS	Android, Linux kernel 4.4.80
IO Type	Direct IO
File System	ext4
IO Tool	fio
IO Trace Tool	Blktrace / ftrace

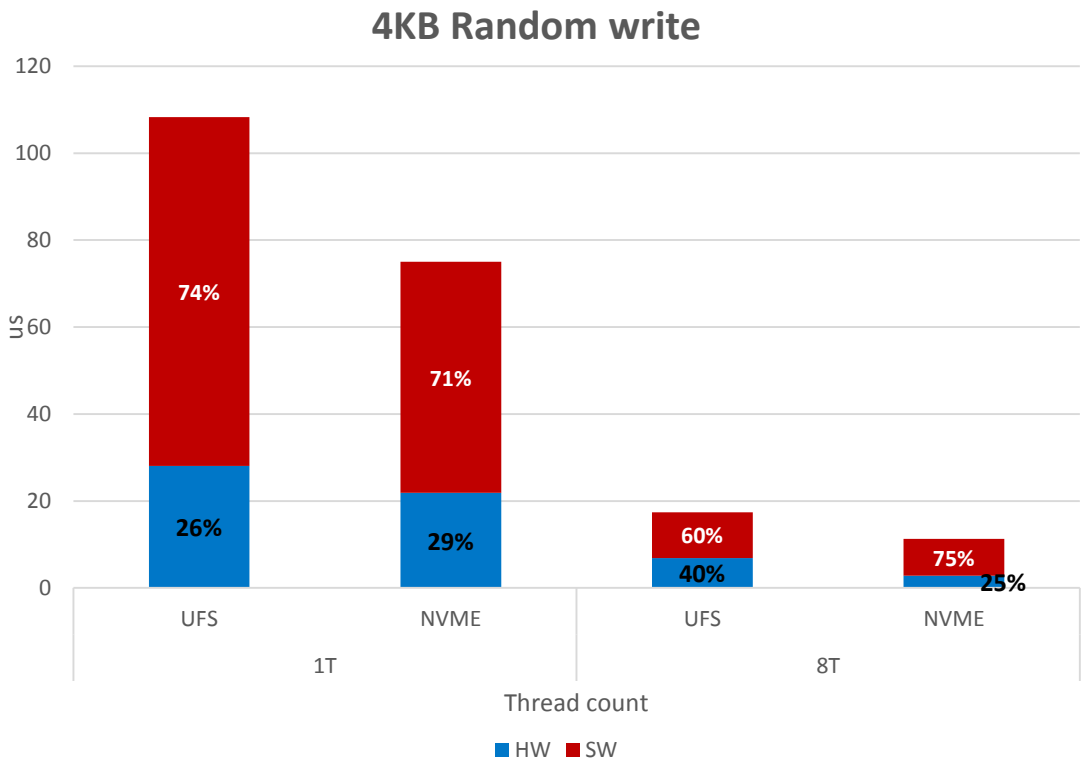
	UFS	NVMe
Lanes	2	1
Density	128GB	128GB
Phy / link Interface	M-phy Gear 3	PCIe Gen2
Queues	1	8
Queue depth	UFS supports to 256, but UFS host capacity only can support 32.	1024 per queue
INTs	1	1(Sharing)

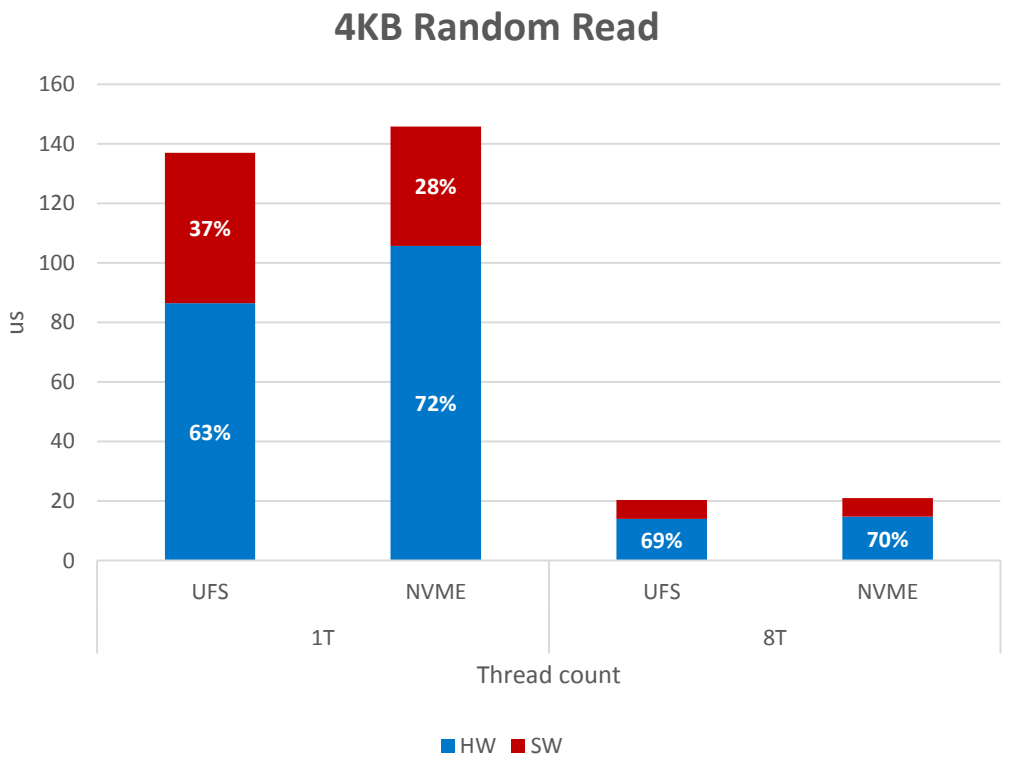
UFS/NVMe SW Stack



- NVMe has more compact/simpler SW stack and provides shorter code paths and lower overhead.
- NVMe stack is newer optimized for managed NAND devices
- Nvme can better support parallelism due to advanced multi-queue capability

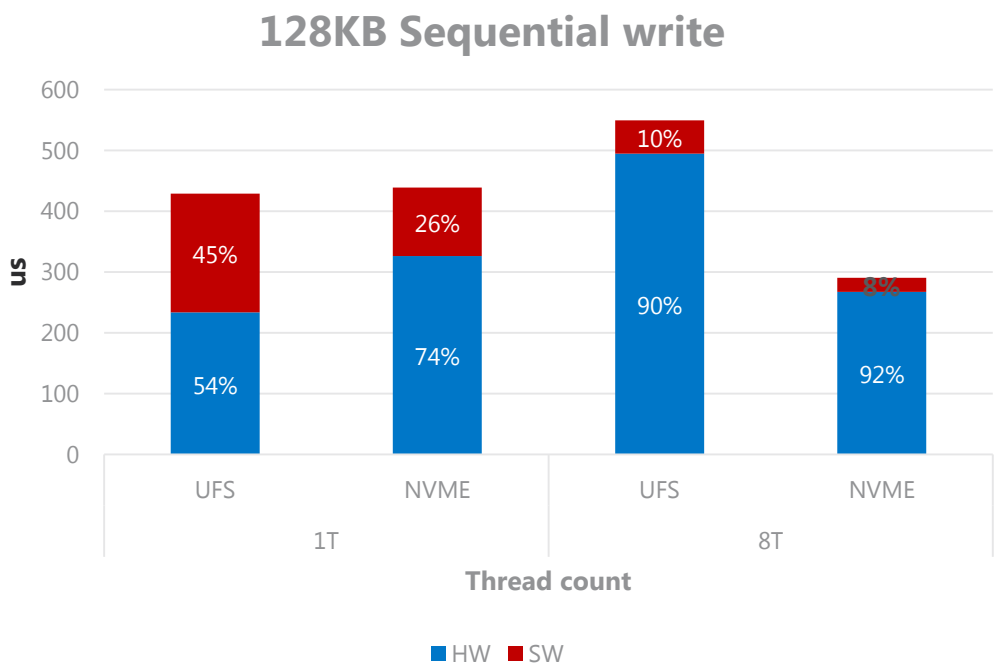
4KB Random HW/SW latency comparison(per thread)

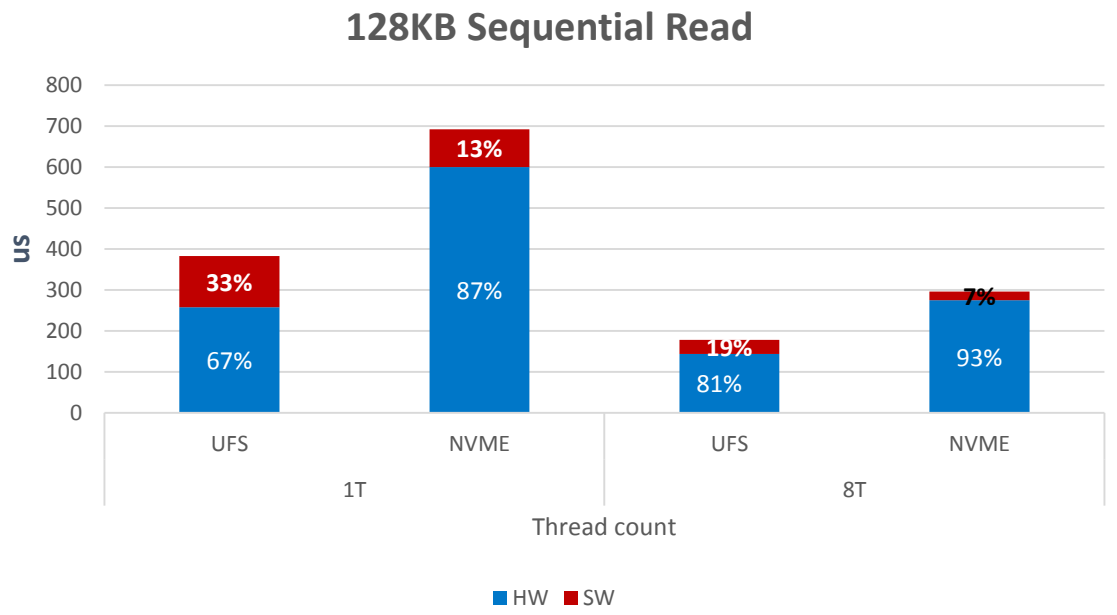




- System overhead is significant in both UFS and NVMe ranging from 60-75% of total IO time for random write and 10-37% from random read

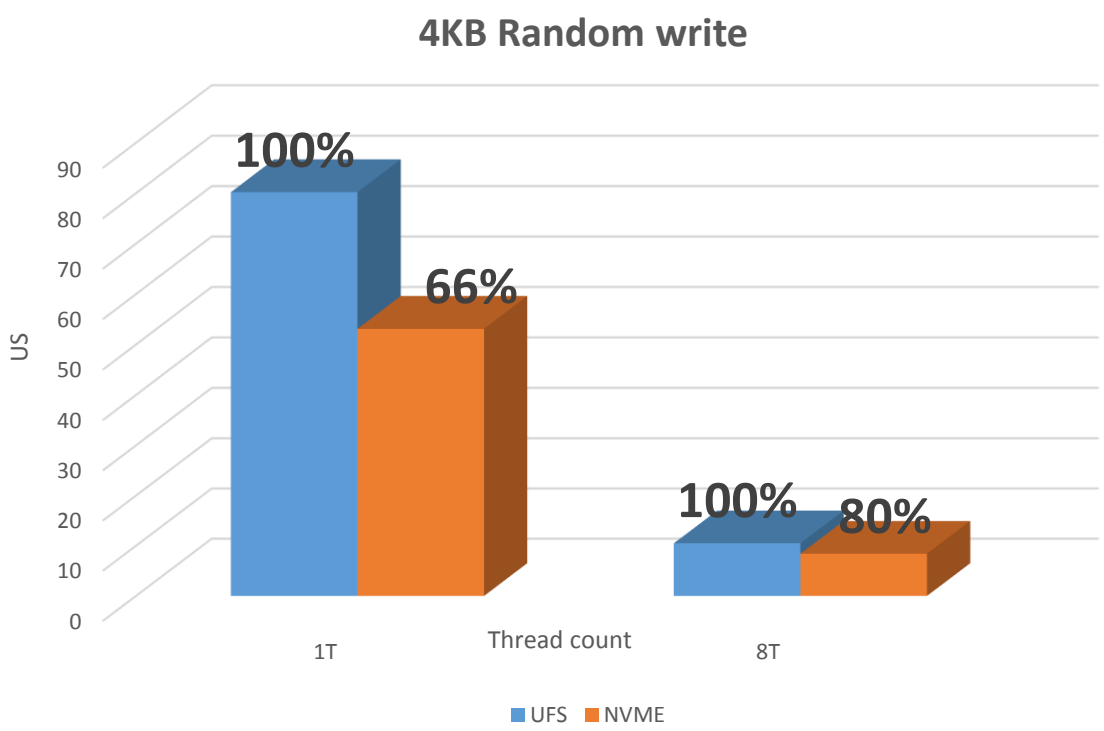
128KB sequential HW/SW latency comparision (per thread)

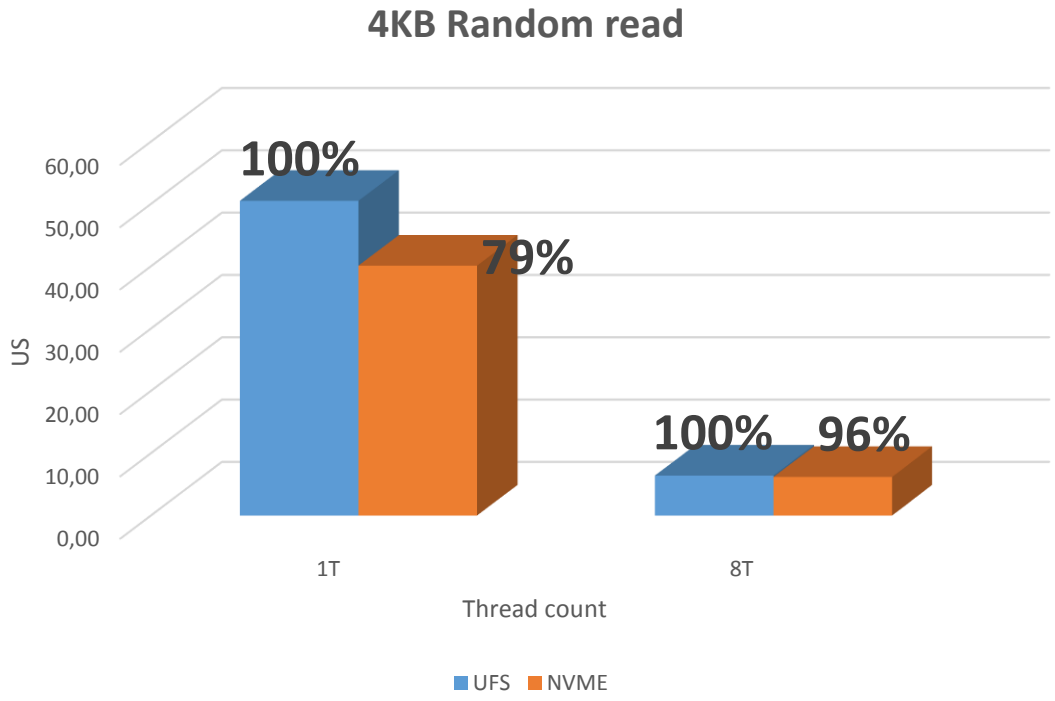




- System overhead on sequential large chunk writes in ranging between 26-45% with single thread and between 8-10% with multiple threads
- Sequential reads experience overhead between 13-33% with single threads and 7-19% with multiple threads

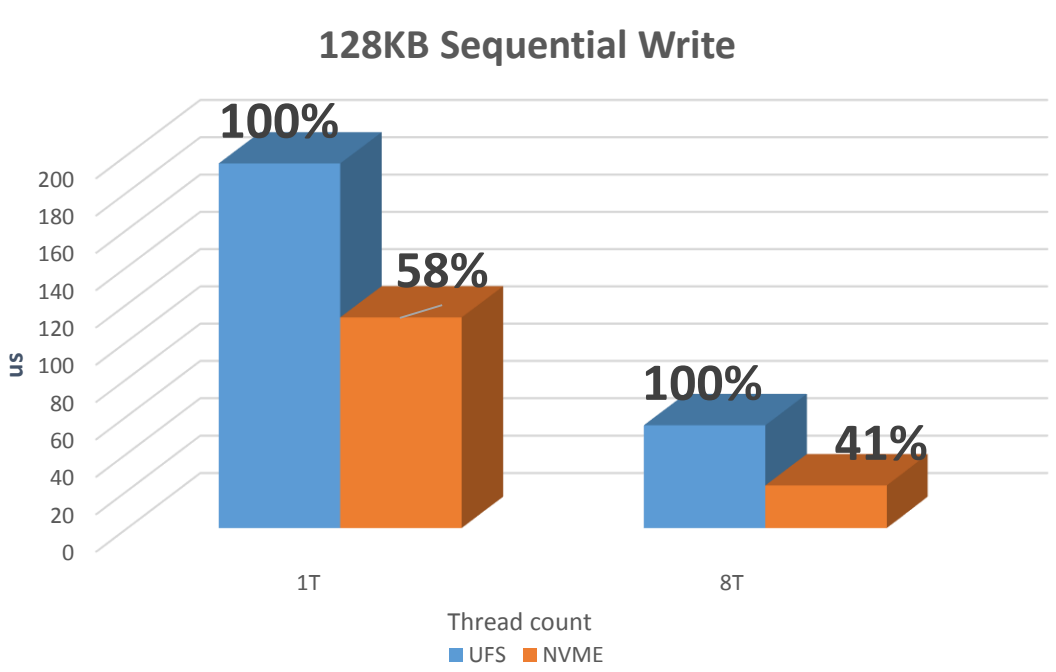
System overhead comparison with 4KB

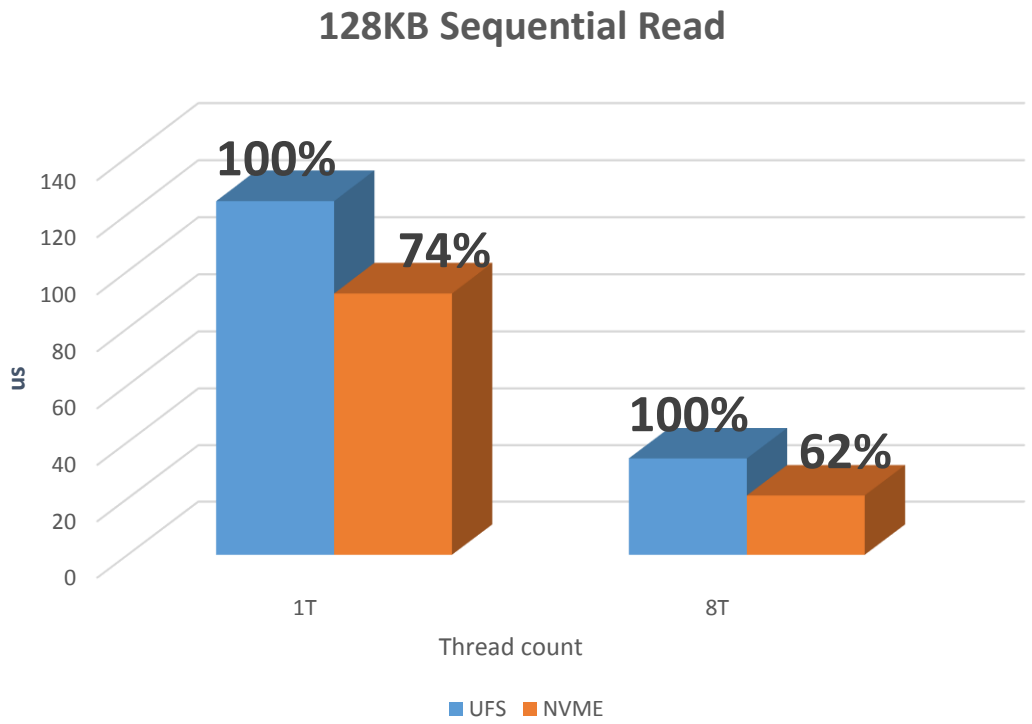




- NVMe shows a significantly lower system overhead vs UFS in small chunk accesses
- The difference decreases with increase in number of threads

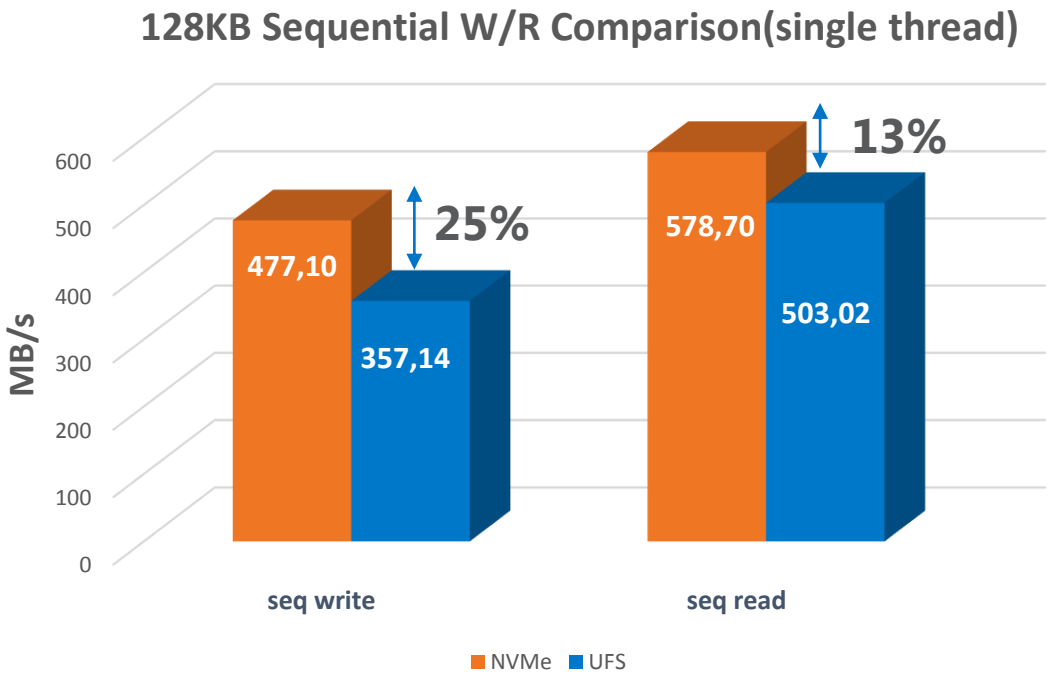
System overhead comparison with 128KB

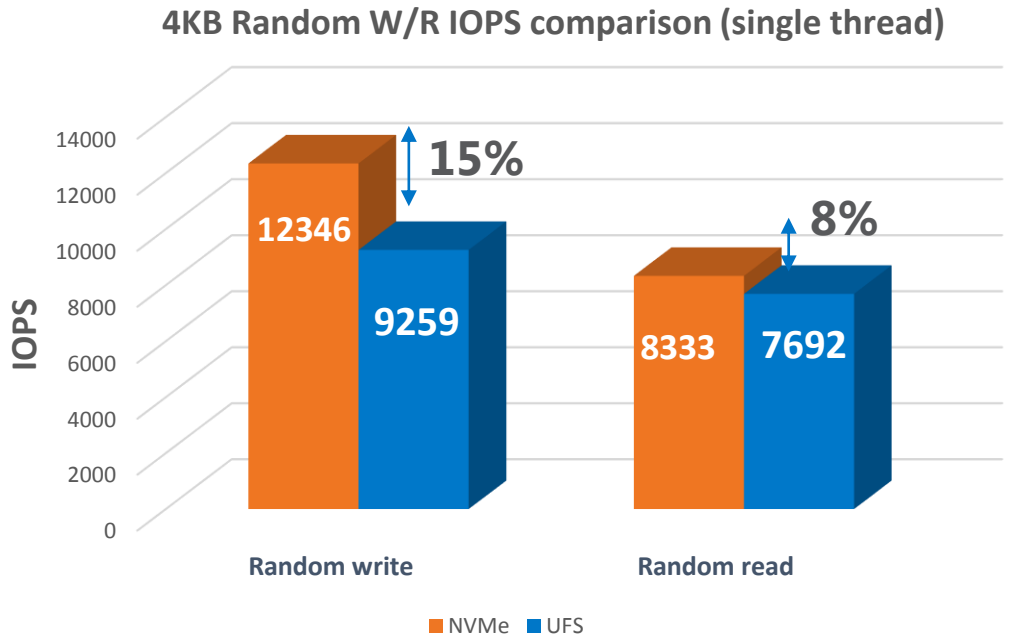




– NVMe shows a significantly lower system overhead vs UFS in small chunk accesses

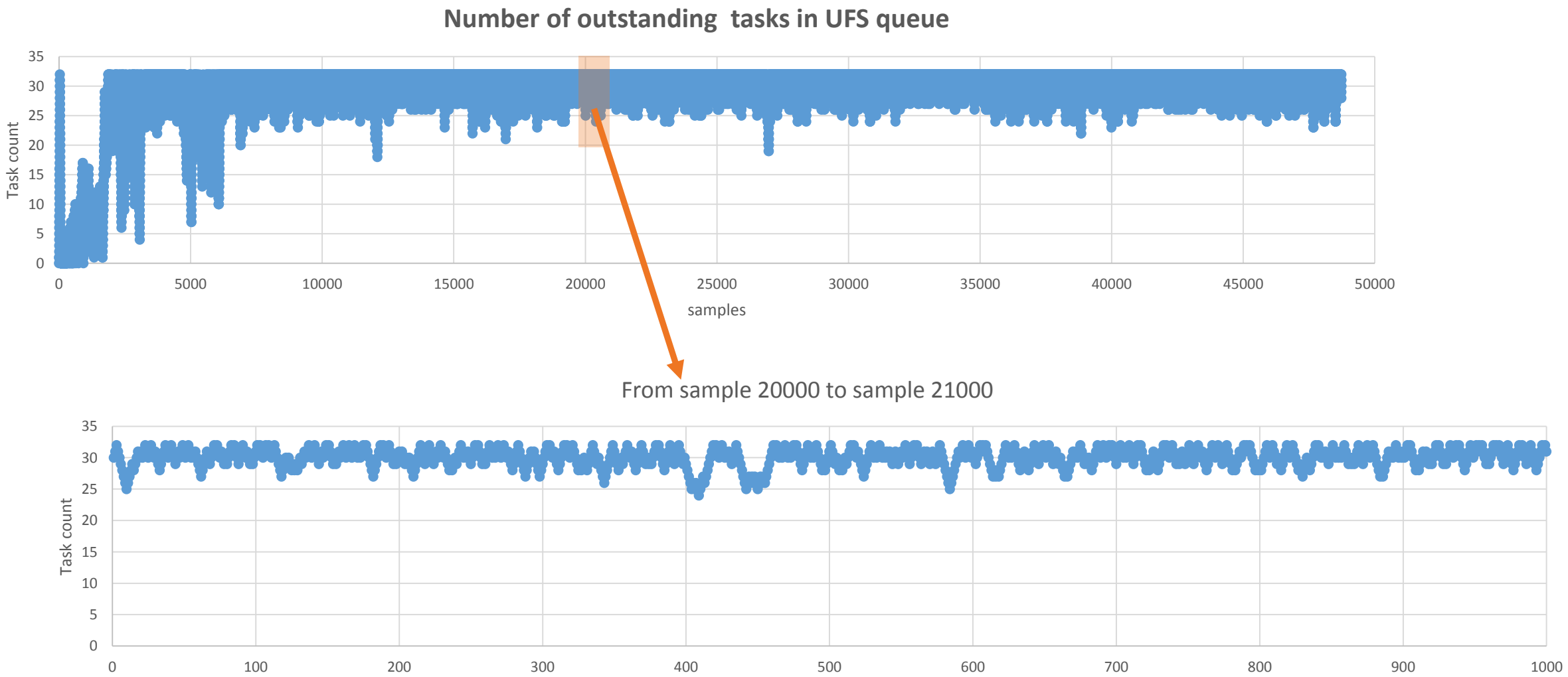
Estimated system level performance comparison with typical device access times



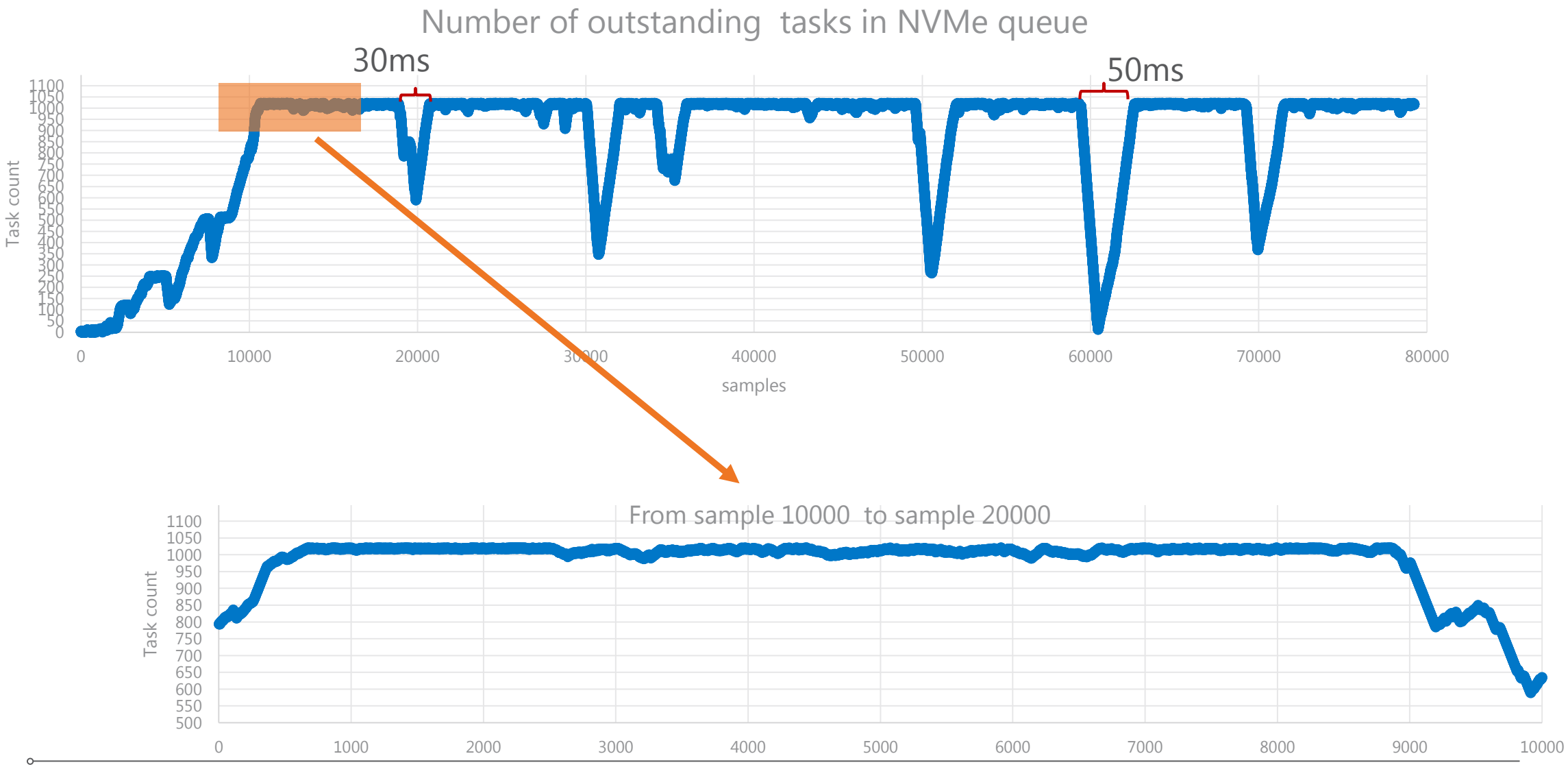


- Assuming identical NVMe and UFS devices, graph represents performance differences
- Sequential write access time:150 usec, Sequential read access time:124 usec
- Random write access time: 28 usec, Random read access time: 80 usec
- NVMe shows 25% benefits in sequential write and 13% benefit in sequential read. 15% improvement in random write and 8% improvement in random read

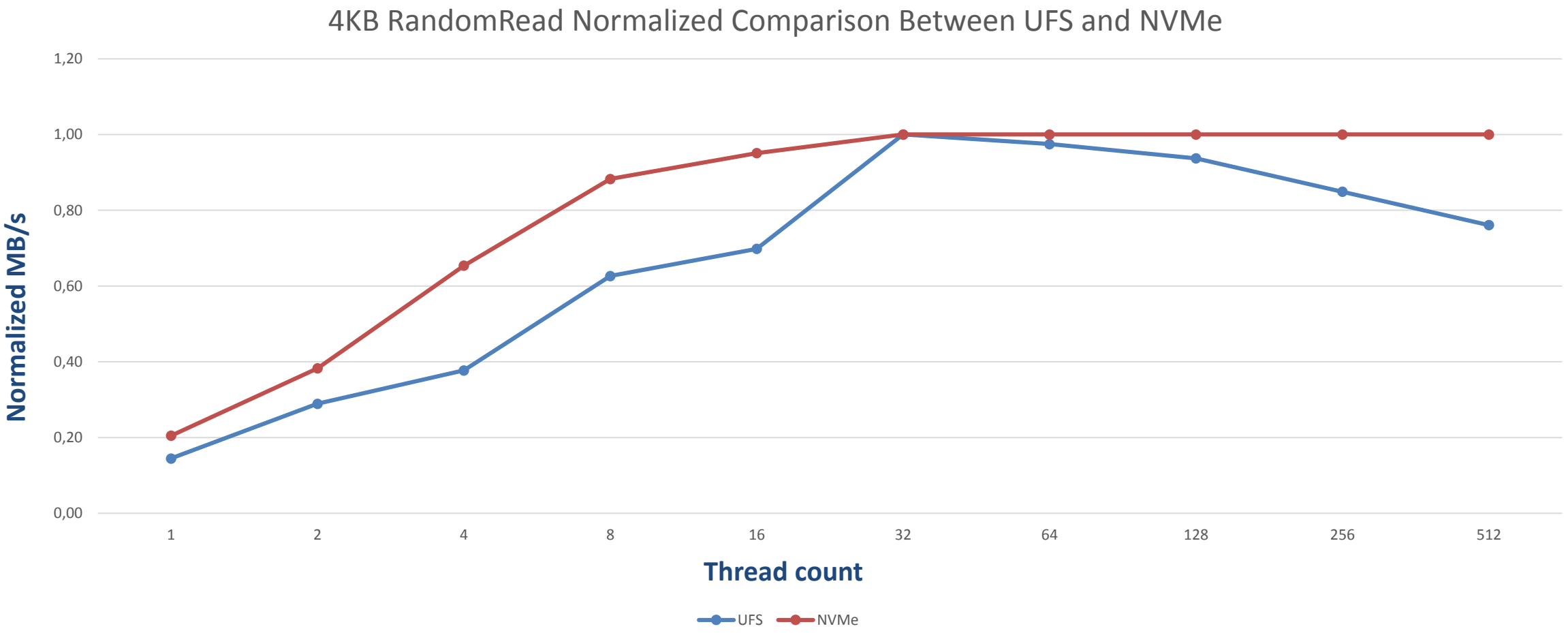
UFS queue limitation



NVME queue limitation(single queue)



UFS/NVMe performance comparison in high parallel level



Summary

- We observe storage system overhead that eats into underlying storage device bandwidth. Overhead is more significant with small chunk IO accesses
- This overhead is expected to be more significant with faster storage devices
- NVMe shows improved performance due to leaner storage system up to 25% in certain cases
- NVMe provides a richer queuing infrastructure, this has an observable benefit in high thread count

Q&A


Micron®