Energy Implications of IO Interface Design Choices

Sidharth Sundar, William Simpson, Jacob Higdon,
Caeden Whitaker, Bryan Harris, Nihat Altiparmak

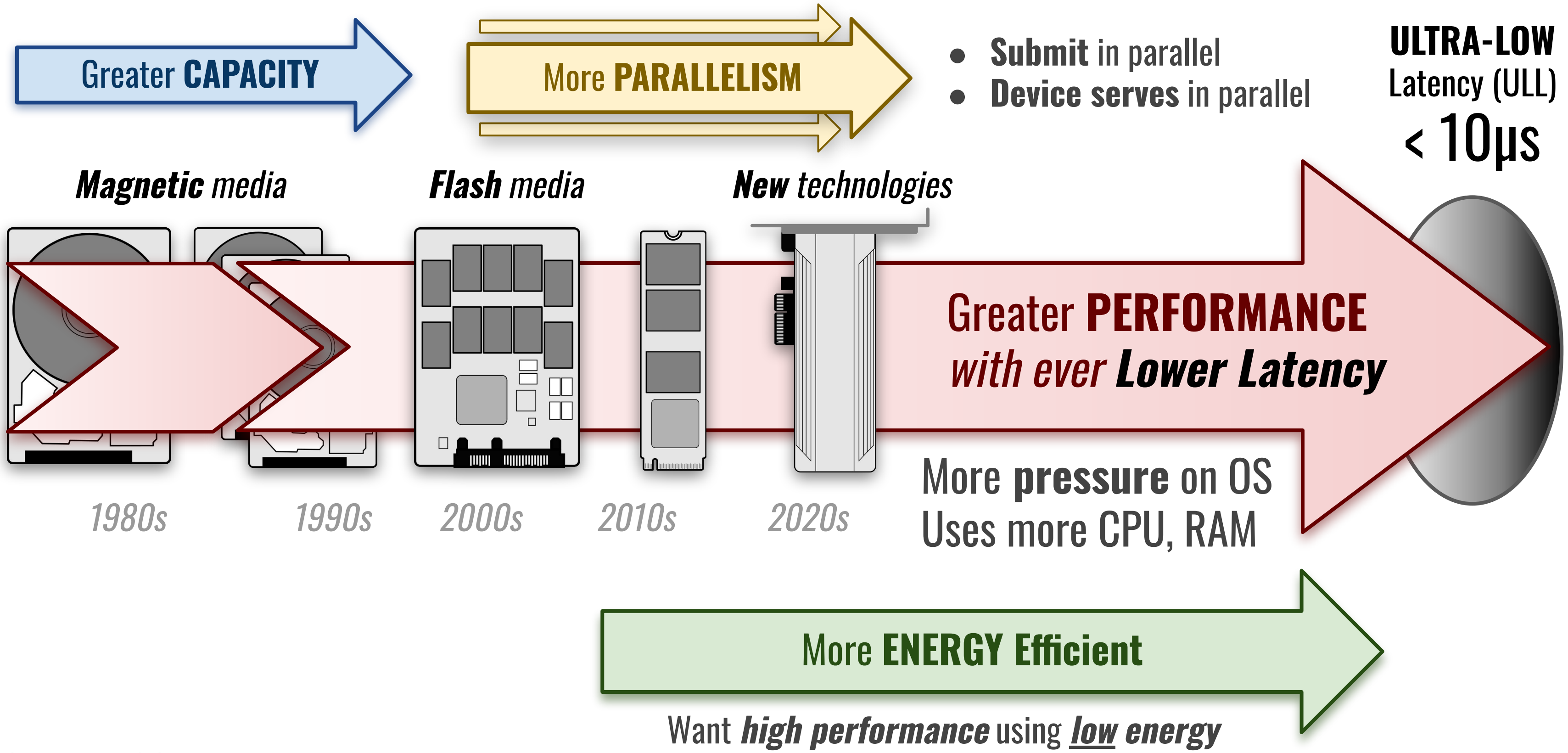
Computer Systems Lab
Computer Science & Engineering Department
UNIVERSITY OF LOUISVILLE®

HotStorage '23 (Boston), July 9, 2023

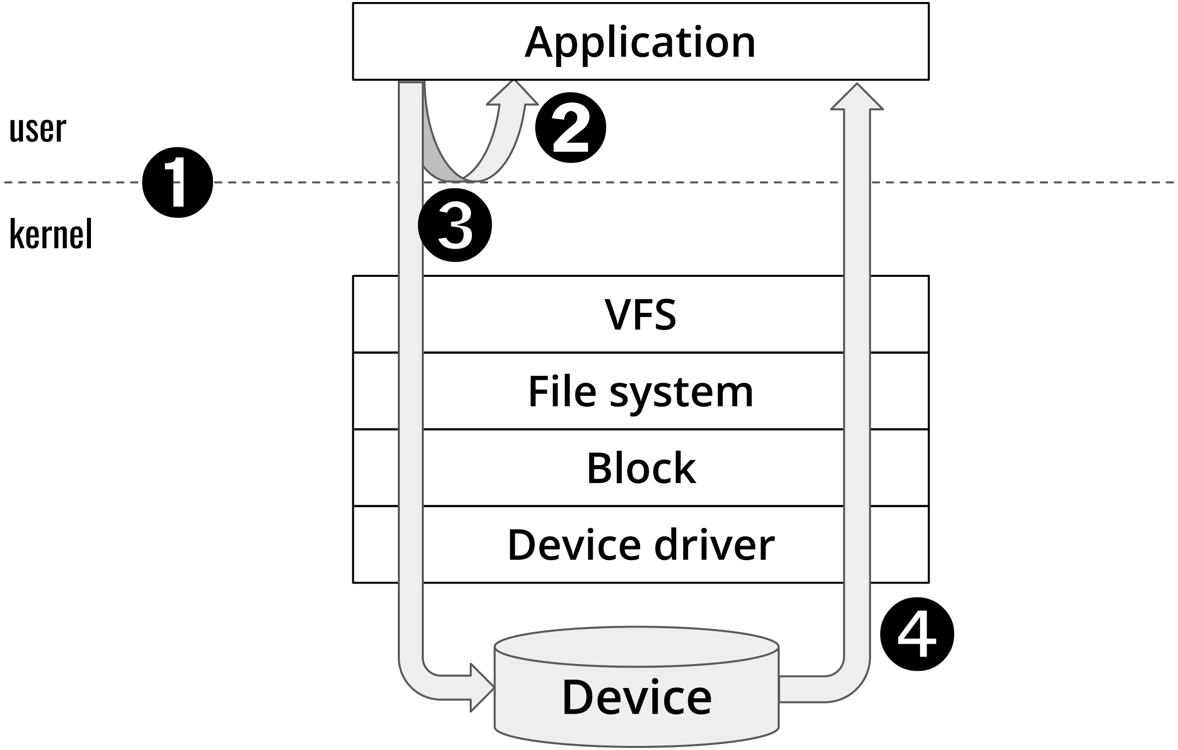
Outline

- Motivation
- IO Interface design choices
- Experimental Results
    - Latency Impact
    - Energy Efficiency
- Conclusion

Trends in data storage



IO Interfaces



IO Interface Design Choices

Existing Linux APIs

posix-sio — Based on traditional POSIX synchronous *read*(2), *pread*(2), etc.

posix-aio — POSIX asynchronous, implemented in library on top of *posix-sio*.

libaio — Linux native asynchronous IO library

io_uring — Recent Linux design for high performance

spdk — Intel framework based on kernel bypass

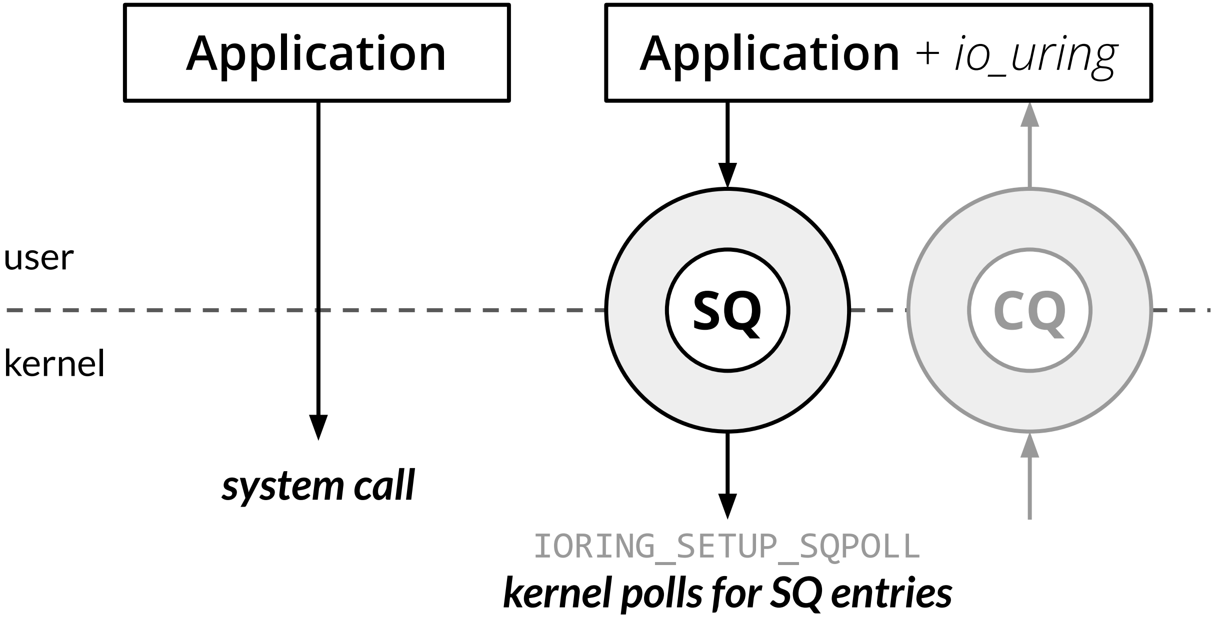
❶ **Execution** in ***kernel*** or ***user*** space

<table>
<tr><th>API</th><th>kernel</th><th>user</th></tr>
<tr><td>posix-sio</td><td rowspan="4">Implemented in kernel</td><td></td></tr>
<tr><td>posix-aio</td><td>Implemented in C lib. on top of posix-sio</td></tr>
<tr><td>libaio</td><td></td></tr>
<tr><td>io_uring</td><td></td></tr>
<tr><td>spdk</td><td></td><td>Uses kernel bypass drivers</td></tr>
</table>

❷ **Synchronous** or **Asynchronous** behavior

API	Synchronous	Asynchronous
posix-sio	synch. only	
posix-aio		asynch. only
libaio		asynch. only
io_uring	both supported	
spdk		asynch. only

❸ **Submission** using ***system call*** or ***polling***



(No need for system calls!)

❸ **Submission** using *system call* or *polling*

API	System call	Submission polling
posix-sio	System call based	
posix-aio		
libaio		
io_uring		*io_uring* feature
spdk	kernel bypass	

❹ **Completion** using *interrupt* or *polling*

API	Interrupt	Poll
posix-sio	Interrupts	Optional polling
posix-aio		
libaio		
io_uring		Optional polling
spdk		Polling required

Interface design choices

API	Execution		Behavior		Submission		Completion	
	kernel	user	sync	async	syscall	spoll	int	cpoll
posix-sio	●	○	●	○	●	○	●	●
posix-aio	●	●	○	●	●	○	●	○
libaio	●	○	○	●	●	○	●	○
io_uring	●	○	●	●	●	●	●	●
spdk	○	●	○	●	○	○	○	●

○ unsupported, ● supported

Experimental Results

Experimental Setup

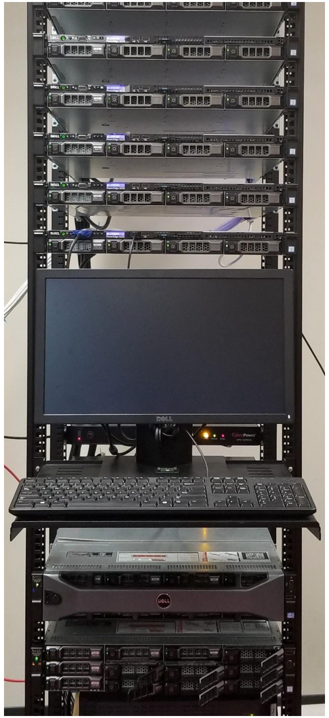
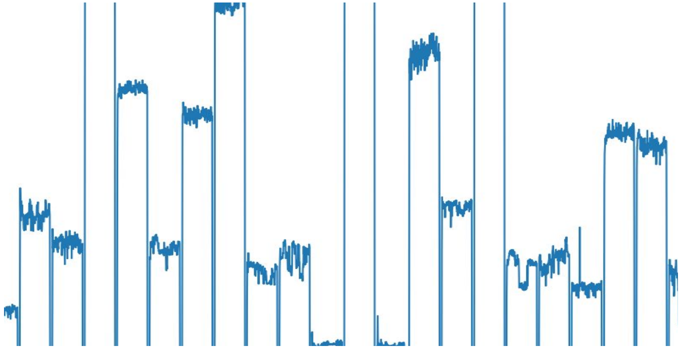


Image from Onset Computer Corporation



Power Measurement
Onset HOBO plug meter logs power, current, etc., every second, for the entire system.

Workloads
fio (“Flexible IO tester”)
   – direct IO
- *xfs* file system
- “none” IO scheduler

Latency Impact

Latency Impact

Interface	Subm.	Comp.	read (μs) 50th	read (μs) 99th	write (μs) 50th	write (μs) 99th	avg (%) slower than *spdk*
spdk	–	cpoll	**6.23**	**7.47**	**8.44**	**10.40**	–
posix-sio	syscall	cpoll	8.69	9.06	**11.11**	**12.24**	27.52%
io_uring	syscall	cpoll	8.65	**9.03**	14.50	15.93	46.18%
io_uring	spoll	cpoll	**7.89**	9.05	12.38	22.33	52.30%
posix-sio	syscall	int	11.68	12.36	14.08	15.64	67.54%
io_uring	spoll	int	8.72	11.15	13.34	25.32	72.69%
libaio	syscall	int	12.35	13.06	16.33	21.87	94.21%
io_uring	syscall	int	12.73	13.45	16.92	23.61	102.97%
posix-aio	syscall	int	19.10	28.89	21.60	34.46	220.15%

Energy Impact

Energy efficiency metric

$$\text{Energy efficiency} = \frac{\text{Performance}}{\text{Power}} = \frac{\text{IOPS}}{\text{Watts (J/s)}} = \text{IO/J}$$

Energy Impact Experiments

- 4KB, 16KB, and 128KB requests
- 100% random read vs 100% random write
- Scaling # of overall requests issued in parallel to device using 2 metrics:

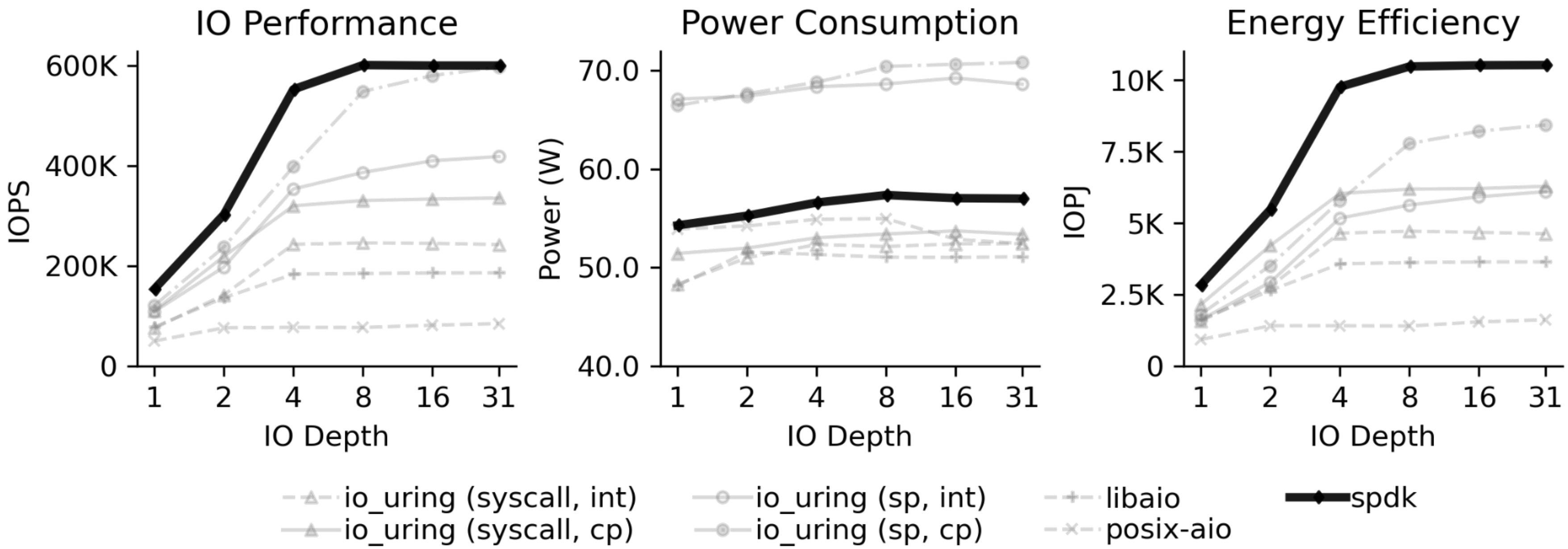
1. Single Thread
   a. Scaling # of outstanding requests (iodepth) on single thread
   b. Single thread issuing IOs
   c. Only considers asynchronous interfaces

2. Multi-threaded
   a. Scaling # of threads issuing IOs (# of jobs)
   b. One request per thread

Energy Impact

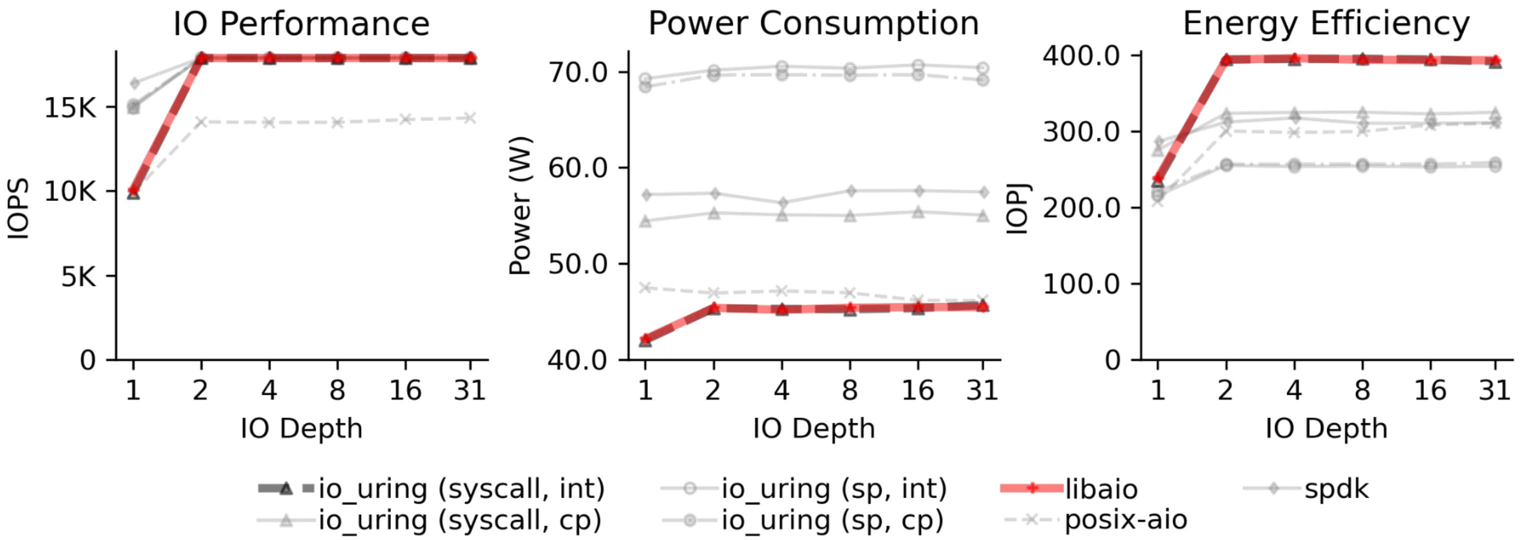
Single Thread

Kernel bypassing is most energy efficient for single thread small requests



4 KB Single Thread Random Reads

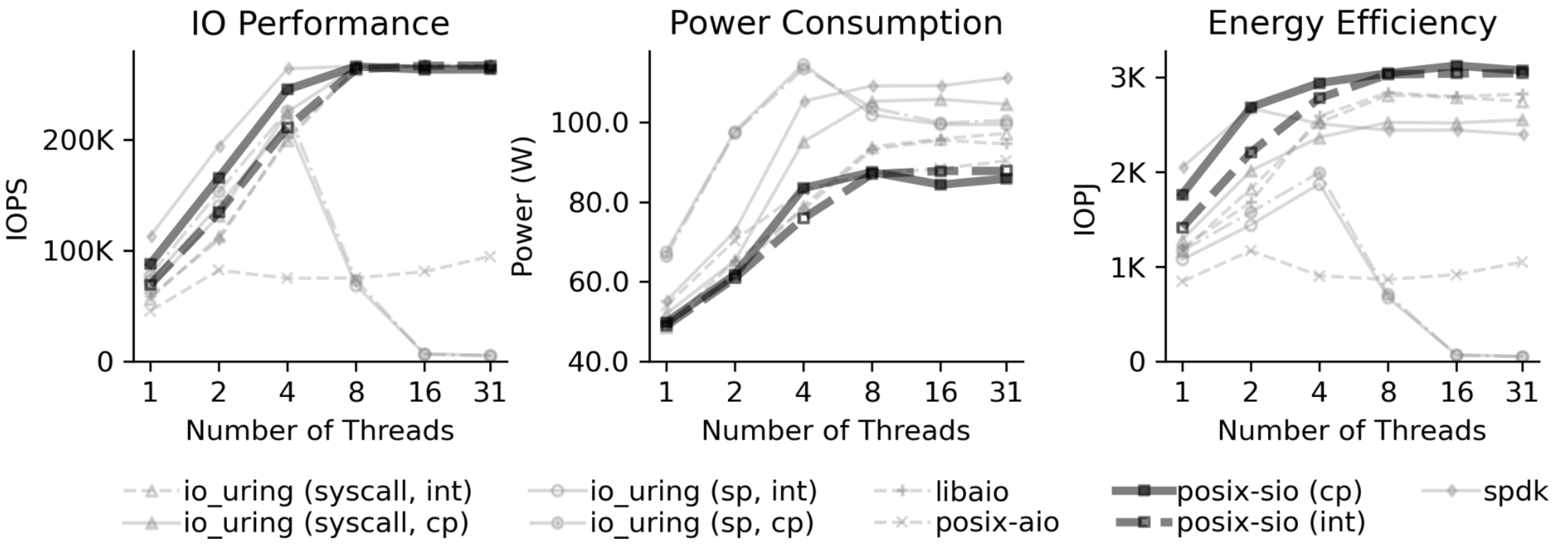
System call and interrupt based kernel space implementations are most efficient for large requests



128 KB Single Thread Random Reads

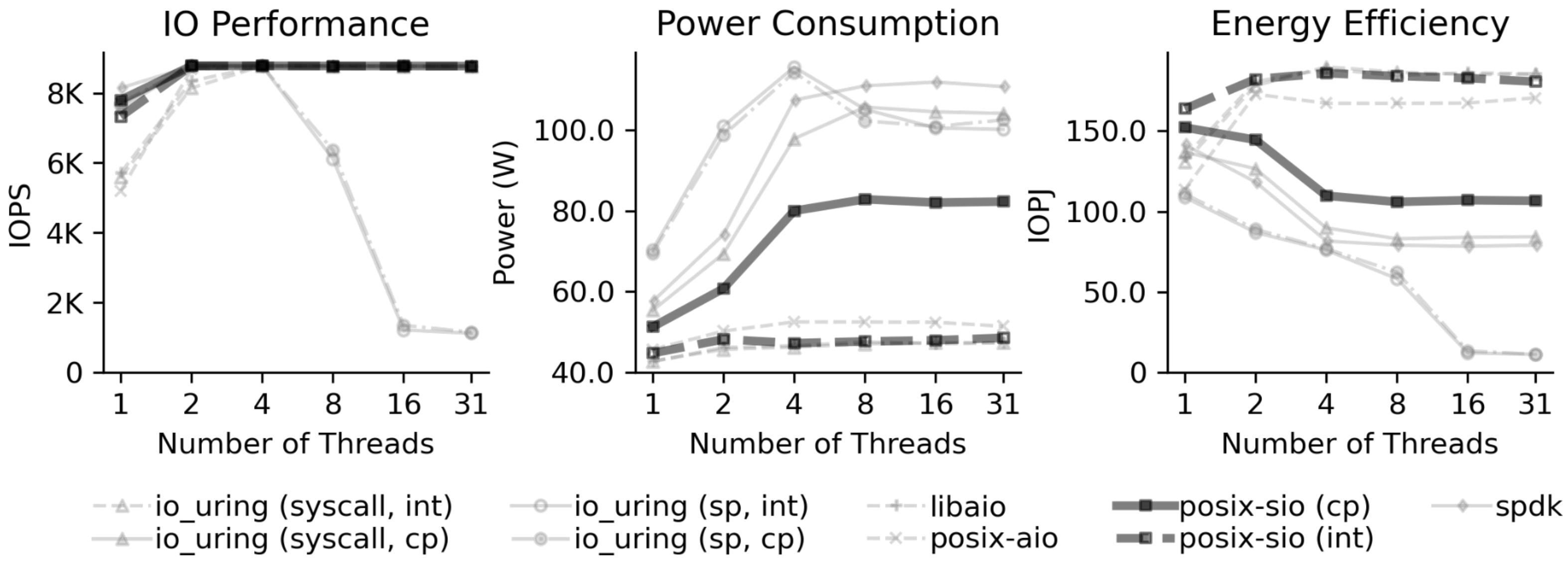
Energy Impact
Multithreaded

For <u>small</u> requests, *posix-sio* with <u>polling</u> based completion is most energy efficient



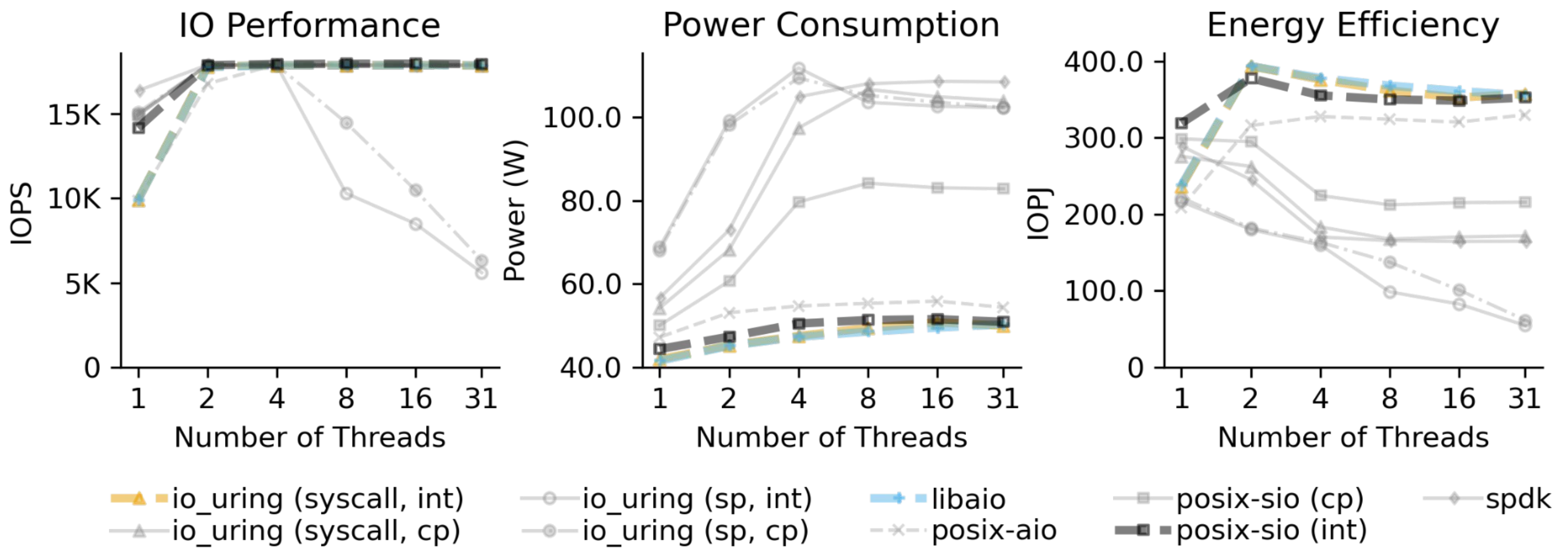
4 KB Multithreaded Random Writes

For <u>large</u> requests, *posix-sio* with <u>interrupt</u> based completion is most energy efficient



128 KB Multithreaded Random Writes

Interrupts are crucial for energy efficiency when the request size gets larger



128 KB Multithreaded Random Reads

Conclusion

❶ Kernel *vs* Userspace

Only bypass kernel if kernel functionalities (such as interrupts) are unnecessary

❷ Synchronous *vs* Asynchronous

Synchronous (posix-sio) tends to be more energy efficient when synchronous is usable

❸ System Call *vs* Submission Polling

Submission polling typically costs too much power to justify

❹ Interrupt *vs* Completion Polling

- **Polling** is more energy efficient for **smaller** IOs
- **Interrupt** is more energy efficient for **larger** IOs

Questions?