~~Approaching~~ Surpassing 10M I/Ops on a Single CPU Core

Ben Walker

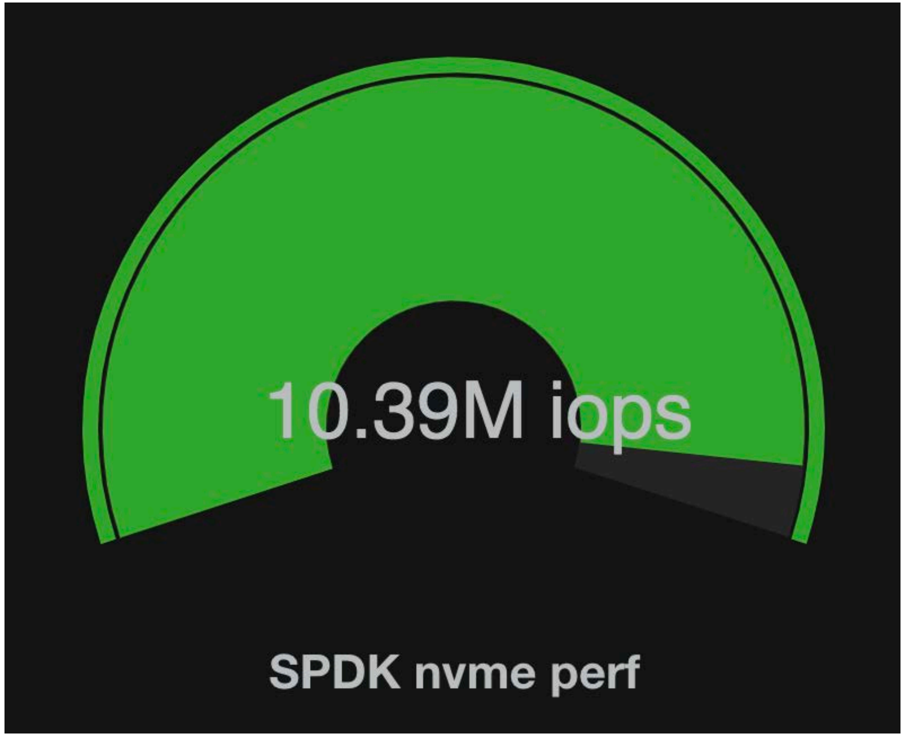
Technical Lead

Intel

Storage Performance Development Kit

- User-space block storage stack with similar features to an OS
  - Includes an NVMe driver
- Open Source, BSD 3-Clause License (https://spdk.io)
- Very active community
  - 1200 commits from 56 committers in last 3 months

BIG NUMBERS



4KiB random reads at queue depth 128 to each device

System Configuration

Configuration	
CPU	Intel® Xeon® Platinum 8280L CPU @ 2.70GHz
Memory	12x 16GB 26667
Storage	21x Intel® SSD DC P4600 1.6TB

Outline

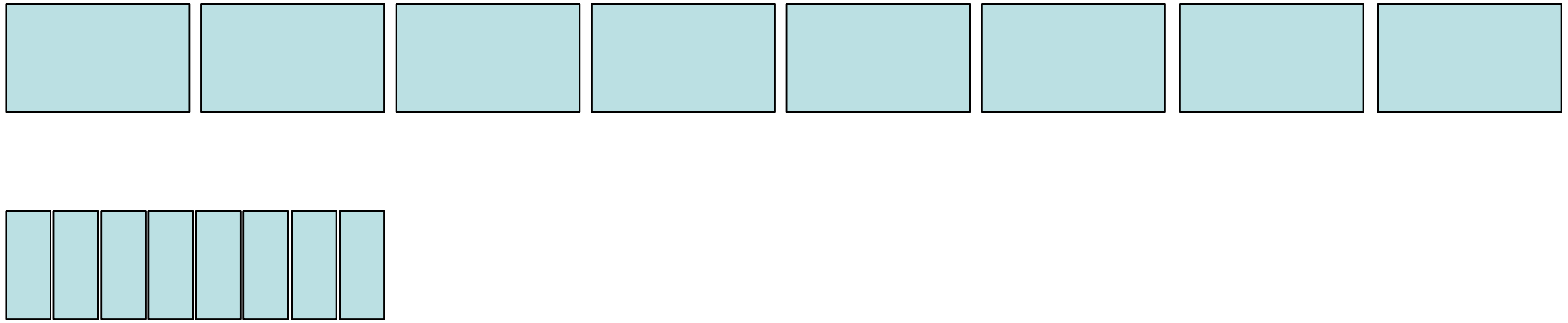
Talk based on this blog post:
https://spdk.io/news/2019/05/06/nvme/

Covering only 3 techniques from that post today for time reasons.

Not covering any active areas of research, but there are several!

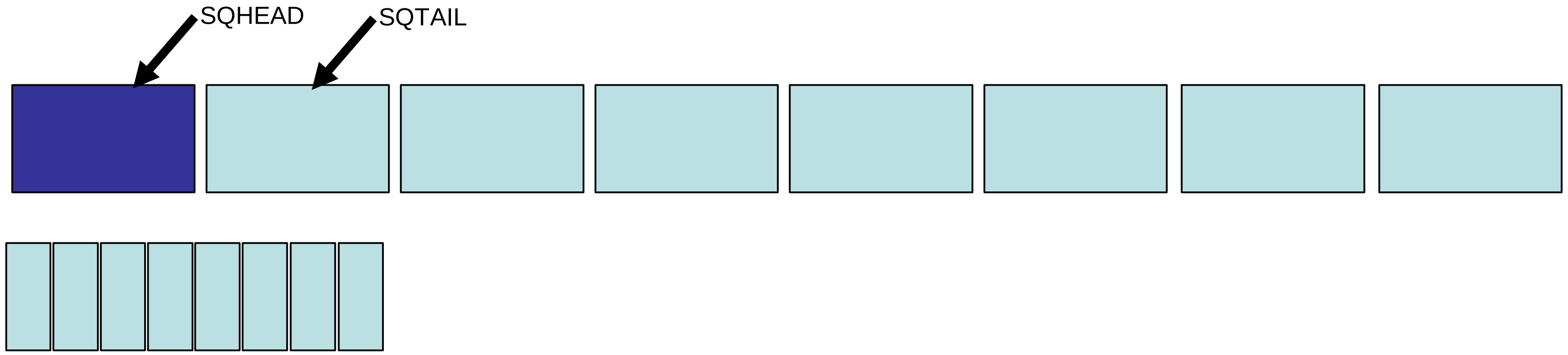
Mechanics of Submitting an I/O

NVMe queues consist of two arrays in host memory (submission queue and completion queue) plus two doorbells (SQTAIL, CQHEAD) in the BAR



Mechanics of Submitting an I/O

To submit: Copy command into next slot. Write SQTAIL.



To complete: Compare phase bit. If flipped, contains valid completion. When done, write CQHEAD.
Completions hold updates to SQHEAD.

SPDK NVMe Design Basics

- Assign NVMe queue pairs to threads.
  - No locks!
- Disable interrupts
  - Completions are handled when the application is ready to handle them. No context switch.
- Code is compiled with –O2, LTO enabled, PGO disabled.

General Rules of Thumb

Don’t let the CPU stall!

- No cross-thread coordination (locks, etc.)
- Poll instead of interrupt
- Minimize MMIO
- Get the right things into the CPU cache at the right time
  - Pack structures. Separate hot data from cold.

Tricks For Minimizing MMIO

```
while (true) {
    ... work ...
        spdk_nvme_ns_cmd_read(..., cb_fn);
        spdk_nvme_ns_cmd_read(..., cb_fn2);
    ... work ...

    spdk_nvme_qpair_process_completions(...); /* calls cb_fns if the read is done */
}
```

Submit several commands. Check for completions. Repeat.

Naïve implementation: For each command, 1 MMIO on submit, 1 MMIO on complete

Tricks For Minimizing MMIO

Trick 1 (well known):

When checking for completions by reading the phase bit, don’t write the completion queue doorbell until all currently outstanding completions have been discovered.

Everyone knows this trick.

Tricks For Minimizing MMIO

Trick 2 (getting smarter):

When a completion is posted, don’t write the completion queue doorbell unless the device actually needs more slots free.

Devices often have large queues – say 1024. We can write the completion queue doorbell every ~512 commands.

I’ve only seen SPDK do this. (Doesn’t help this benchmark because trick 1 is already finding 30 to 50 completions at a time)

Tricks For Minimizing MMIO

Trick 3 (really smart):

When a command is submitted, copy the command into the SQE slot but don’t ring the submission queue doorbell. Instead, ring the doorbell only when the user polls (which is happening very frequently).

This is a tricky way to batch command submissions transparently to the user in a polling system.

2.89M I/Ops with this disabled. 10.39M with this enabled.

Completing I/O

```
for each cqe {
    if (cqe->phase != phase_flipped) {
        break;
    }

    struct tracker *tr = tracker_array[cqe->cid];

    struct request *req = tr->req;

    req->cb_fn(req->cb_arg);
}
```

- Trackers are 1:1 with slots in the NVMe queue pair.
- Requests are N:1 with trackers.
- Trackers are looked up by CID obtained from CQE

Eliminate Data Dependent Loads

```
struct nvme_request {
          spdk_nvme_cmd_cb cb_fn; /* Callback function */
          void *cb_arg;
};

struct nvme_tracker {
          struct nvme_request *req;

          spdk_nvme_cmd_cb cb_fn; /* Copied callback function */
          void *cb_arg;

          /* Other stuff */
};
```

On submission, copy cb_fn and cb_arg into tracker

Eliminating Data Dependent Loads

```
for each cqe {
    if (cqe->phase != phase_flipped) {
        break;
    }

    struct tracker *tr = tracker_array[cqe->cid];

    tr->cb_fn(tr->cb_arg);
}
```

500K I/O per second improvement

Clever Pre-fetching

```
for each cqe {
    if (cqe->phase != phase_flipped) {
        break;
    }

    next_cqe = cqe + 1;
    if (next_cqe->phase == phase_flipped) {
        __builtin_prefetch(tracker_array[next_cqe->cid]);
    }

    __builtin_prefetch(next_cqe + 1);

    struct tracker *tr = tracker_array[cqe->cid];

    struct request *req = tr->req;

    req->cb_fn(req->cb_arg);
}
```

Questions?

- https://spdk.io
- https://spdk.io/news/2019/05/06/nvme/
- https://spdk.io/doc/nvme_spec.html
- https://spdk.io/doc/ssd_internals.html