UCDAVIS

Design Guidelines for High Performance RDMA Systems

Anuj Kalia, Michael Kaminsky, David G. Andersen

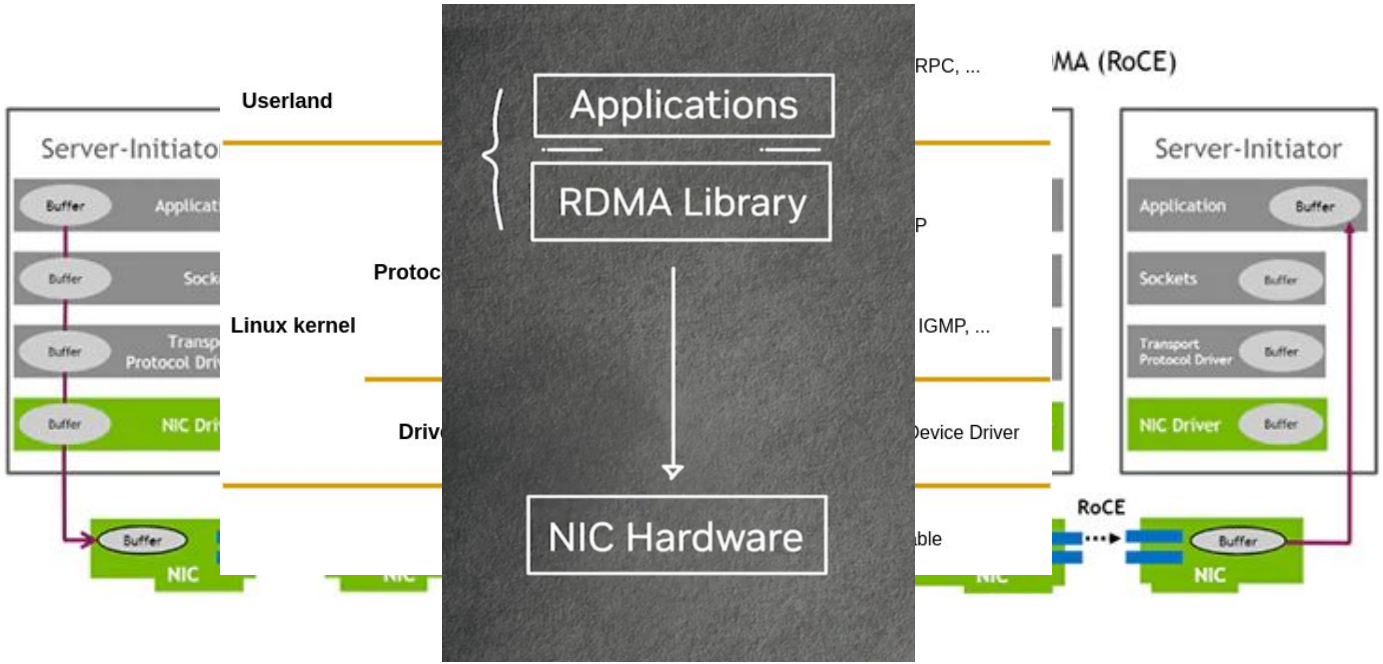
2016 USENIX Annual Technical Conference (USENIX ATC '16).

Presenter

Harish Gokul Krishnakumar

INTRODUCTION

- RDMA Stands for **R**emote **D**irect **M**emory **A**ccess.
- Network Feature that allows direct access to the memory of a remote computer.
- Why RDMA?
- Zero Copy - Application Layer to **N**etwork **I**nterface **C**ard - OS Bypass



BACKGROUND

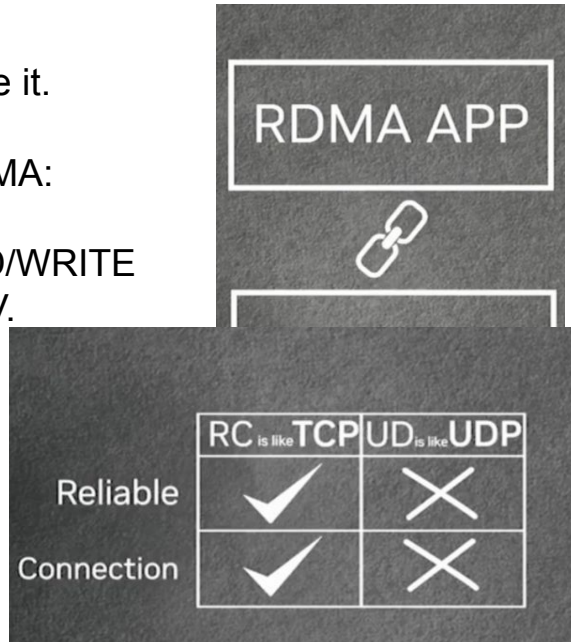
BACKGROUND - RDMA

- RDMA - Read or Write data directly from remote device's memory.
- Server making a request - **Requester**.
- Server receiving the request - **Receiver**.
- It uses the Verbs API - Uses Verbs like READ, WRITE, SEND, RECV etc to interact with remote servers.
- The application has to link with the Verbs API to use it.
- There are two ways for a servers to talk using RDMA:

  - **One Sided:** Do not Involve receiver CPU. READ/WRITE
  -**Two Sided:** Involve receiver CPU. SEND / RECV.

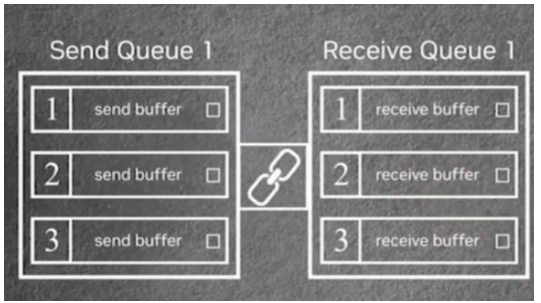
- There are two types of Reliability Guarantees:

  - Connection Oriented: RC / UC ~ TCP
  - Connectionless: UD ~ UDP

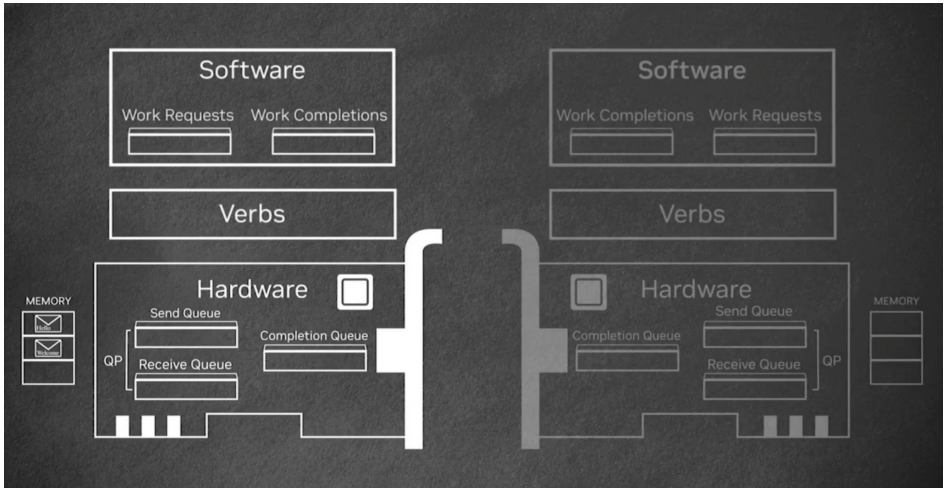


BACKGROUND - RDMA MECHANICS

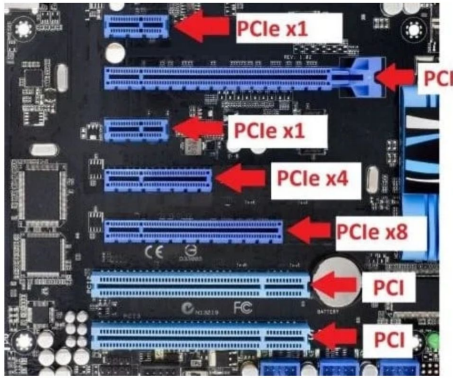
- RDMA - Works in using Queues (FIFO) - Order Preserved
- **W**ork **Q**ueue consist of **W**ork **Q**ueues **E**lements.
- RDMA works in **Q**ueue **P**airs



- **Send** Queue enqueue SEND / WRITE Requests.
- **Receive** Queue enqueue RECV / READ Requests.
- **Completion** Queue enqueues Completed Send / Recv entries with status.



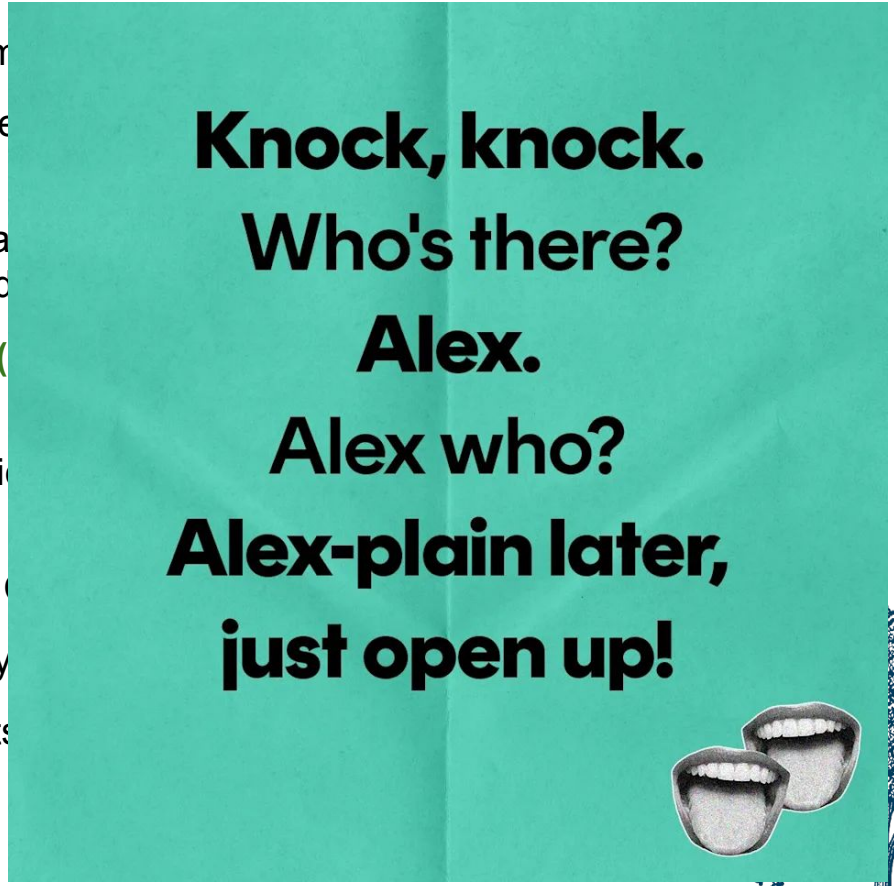
BACKGROUND - Peripheral Component Interconnect - Express



- PCIe: Data transfer between the CPU (and host memory) and the NIC.
- Two Models of Data Transfer:
  - Push Based (CPU MMIO)
  - Pull Based (DMA Read)
- **CPU MMIO stands for CPU Memory Mapped I/O:** Here the CPU pushes channels to talk to the NIC
- **DMA Reads:** Here the NIC is notified of a message to be read, the NIC then issues a DMA read call to get the data.
- -> WQ is established in host memory.
- -> WQ talks to the NIC and our application talks to the WQ.
- -> So data transfer from WQ to NIC is a potential bottleneck.
- Why ? There could be multiple WQ's , there could WQ's pinned to different CPU cores.
- If my CPU spends most of the time in sending data to the WQ then that is an area of Improvement.
- Goal is always -> Use CPU for app logic , use it as less as possible for other stuff

RDMA Guidelines - Reduce CPU Initiated MMIO

- Use CPU as less as possible for WQ to NIC comm
- Instead of pushing data continuously through PCIe batch them?
- So lets accumulate ‘N’ requests, and once these a to the NIC - “Hey N Requests are ready to be reac
- NIC then issues “N” DMA reads to read the data. (**data spans cache lines - lookout)**
- We now issue N + 1 calls , N DMA reads + 1 notifi joke).
- So now problem solved right, CPU is not used so
- DMA reads are better than MMIO generally so yay
- What can we optimize next? Can I reduce N to lets



RDMA Guidelines - Reduce NIC Initiated DMA

- DMA adds one extra read but avoid many MMIO -> Acceptable tradeoff
- Reduce DMA reads by **inlining data** in a single cache line (64 bytes).
- Can we do **HEADER only** RDMA calls like HEADER only RECV and pack the data payload in it.
- Can we do away with a CQ event and not waste a DMA read there (common problem with RECV)
- In datacenters message payloads are often small, this is a good assumption and we can apply these 2 optimisations.

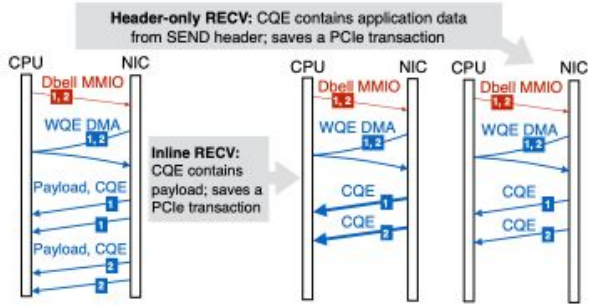


Figure 6: Optimizations for RECVs with small SENDs.

RDMA Guidelines - Leverage NIC Parallelism

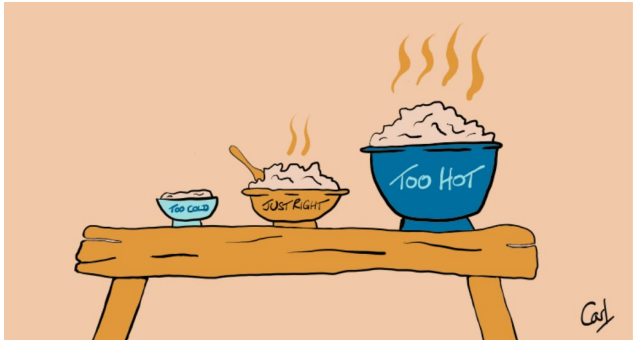
- So NIC's showcase parallelism by having multiple PU's (Parallel Processing Units).
- Let's say we have 4 PU cores, if we have 4 CPU cores then each CPU core can have 1 PU core.
- Each PU Core can have multiple ports.
- Now let's say my WQ is on CPU Core 1 using PU Core 1.
- CPU Core processing speed is fast, PU Core processing speed is slow. CPU is bottlenecked by PU speed.
- **Solution:** Let each CPU core use multiple QPs, so it can distribute work across multiple PUs, leveraging NIC parallelism and keeping all PUs busy efficiently.
- So what about using WQ's across CPU cores, is shared memory worth the performance boost?

RDMA Guidelines - Avoid NIC Contention

- Simple Answer is No.
- It is idea to have WQ's belonging to a CPU core talk to memory allocated for it.
- Resource Contention at the NIC's is not great, they use hardware locks and are not efficient.
- NIC SRAM is small, so number of available locks is small too.
- Let NIC's do the talking and application level code do concurrency control.
- Atomic Operations on NIC is not efficient as a result of this.
- Solutions which involve CAS and atomic swaps on NIC suffer from internal hardware lock mechanisms.
- Question - How do caching work on NIC's?

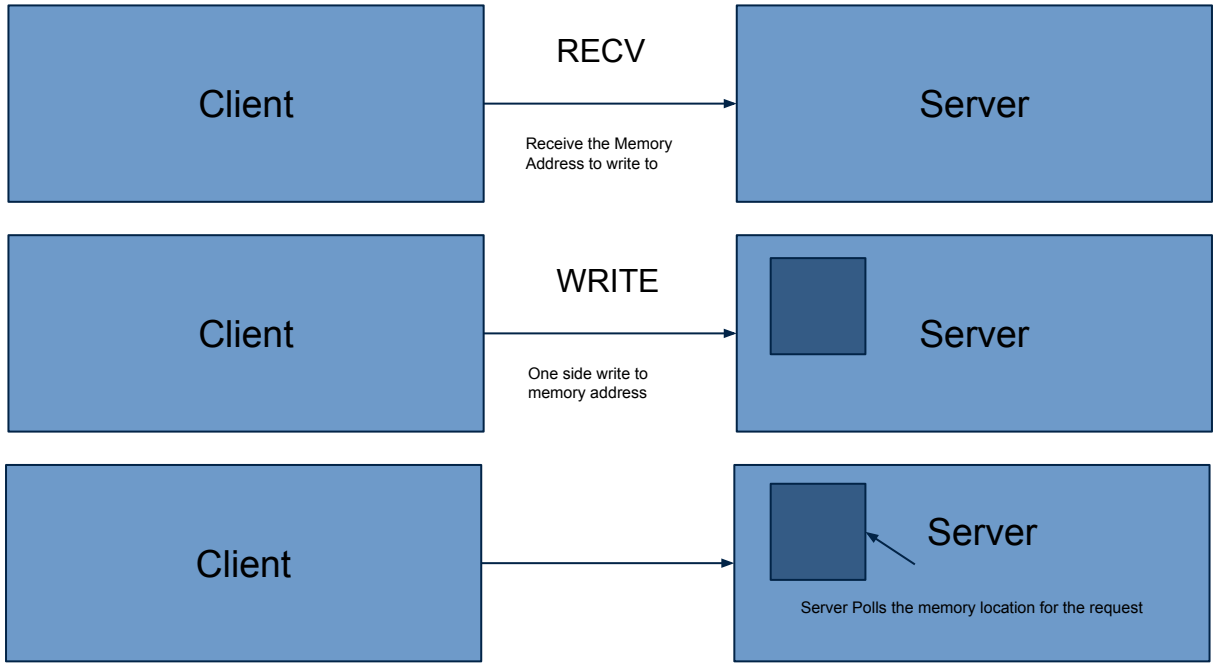
RDMA Guidelines - NIC Cache Lines

- NIC's can cache WQ's or WQ addresses for better performance.
- Since NIC's talk to WQ's directly using DMA reads.
- Now if we have many WQ's from (Multi Queue Optimization) then we have more Cache misses.
- More Cache Misses correspond to not great performance.
- But more MQ's mean better CPU utilisation.
- Solution: Classic Tradeoff -> Stay in Goldilocks zone.
- Right amount of WQ's per core for CPU Utilisation while ensuring good NIC cache lines.
- The authors detect this very brilliantly using PCIe Counters. (Yes just like perf counter, there are PCIe counters).

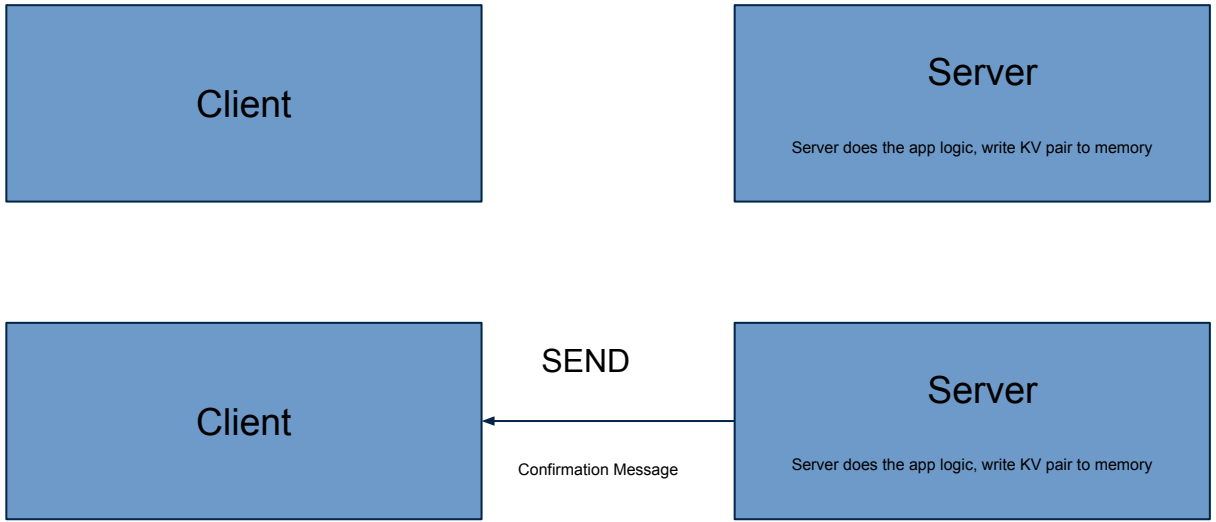


Implementing Guidelines - HERD KV Store

- PS: I Love Key Value Stores.
- HERD is a RPC protocol where a client can write a Key Value Pair to a server.
- Well you could use gRPC, TCP, Arrow RPC …. but we will use you guessed it - RDMA RPC.
- HERD RPC Overview:



Implementing Guidelines - HERD KV Store



HERD Optimizations

- **Batc**
  sing
- **Mult**
  Proc
- **Opp**
  wait
- **DMA**
  CPU

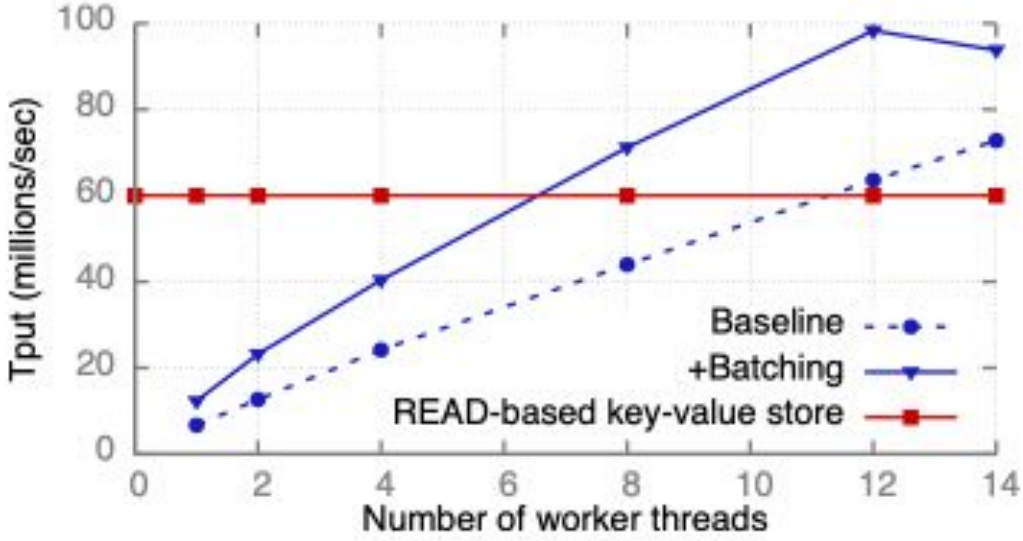


Figure 9: Improvement in HERD's throughput with 5% PUTs

Implementing Guidelines - Network Sequencer

- Problem Statement - Generate Unique Monotonically increasing 8 bytes numbers.

er has to respond with RDMA

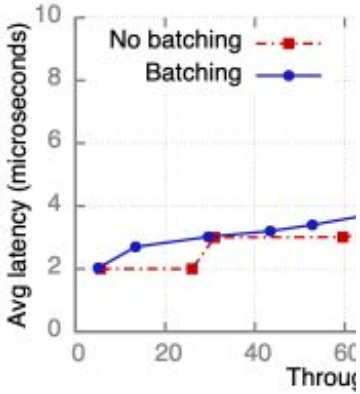


Figure 8: Impact of respons

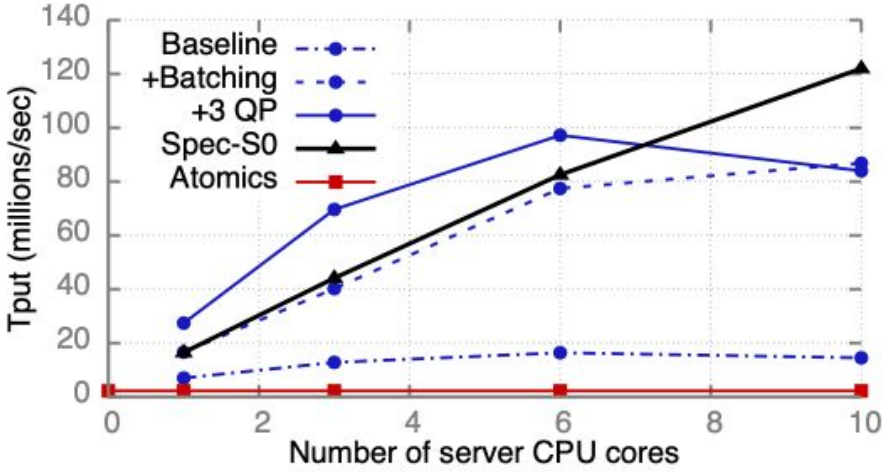


Figure 7: Impact of optimizations on HERD RPC-based sequencer (blue lines with circular dots), and throughput of `Spec-S0` and the atomics-based sequencer

l 2 DMA
t for a

ts by +N

ache line

er,
esponse.

Salient Low Level Observations

- **PCIe Bandwidth is the Ultimate Bottleneck** - Once you optimize everything else, PCIe becomes the hard limit.
- **Batching Only Works Well for Datagram (UD) -** Larger batch size, more machines the better. One to One might not benefit as much.
- **Header-Only SEND/RECV = Game Changer -** Throughput jumps from 82 Mops → 122 Mops (49% gain)
- **Multi-Queue Magic** - Enough QPs to utilize all PUs, not so many that you thrash caches
- **Atomics Are Terrible for Contention -** 2.24 Mops vs 122 Mops → 54x slower than optimized approach.
- **The Bottom Line:** Everything is a tradeoff**,** more queues means cache misses, less queues means less CPU utilisation, m**easure -> implement -> use data to drive decision.**

THANK YOU

UC DAVIS