Evaluating Multiple Endpoints for MPI with libibverbs

Rohit Zambre,[a,b] Abdelhalim Amer,[b] Aparna Chandramowlishwaran,[a] Pavan Balaji[b]
[a]University of California, Irvine, USA; {rzambre, amowli}@uci.edu
[b]Argonne National Laboratory, USA; {rzambre, aamer, balaji}@anl.gov

ABSTRACT

The network adapters of today's high-performance-interconnects expose parallelism at the network interface level by featuring multiple hardware communication contexts. MPI applications transparently utilize this network-level parallelism when multiple ranks run on one node. However, this parallelism remains unexploited when the number of ranks running on a node is smaller than the number of hardware contexts available; no MPI implementation today maps more than one hardware context in a network adapter to a single MPI rank. The overarching goal is to study the following question: when the number of ranks is lesser than the number of hardware contexts available, can MPI benefit from utilizing the idle contexts? In this work, we evaluate message rates observed on the Infiniband network with multiple endpoints.

1 INTRODUCTION

The need for performance and scalability in high performance computing and next-generation data centers is ever-growing. Communication is a critical factor towards delivering higher performance and scalability especially in MPI applications. Modern network hardware features multiple hardware communication contexts to cope with the high communication volume when there are multiple processes running on a node. MPI applications can benefit from these network-level parallelism only when multiple ranks are running on a node because MPI today uses only 1 endpoint per process to issue transfers. In the case where the number of ranks is lower than the number of hardware contexts available on the network hardware, the idle hardware contexts are essentially wasted. If a process gets access to more hardware contexts, it could issue transfers to all those resources to keep the network hardware busy in processing the transfers it issued. However using more resources also means more overhead (such as checks for completion) in managing those resources. The overarching goal of our work is to study how much and what kind of overhead there is in utilizing multiple network resources for communication. Ultimately, we would like to decide if a single MPI rank can benefit from the use of multiple hardware resources.

To start off, we first experiment with libibverbs over the InfiniBand interconnect. This work evaluates the behavior of libibverbs when multiple endpoints exist. Our analysis shows that a straightforward way of utilizing multiple endpoints does not correlate with higher message rates. This work focuses on our analysis of why we see slower throughputs when multiple endpoints are used to issue transfers.

2 EXPERIMENT

To evaluate and conduct our experiments, we write micro-benchmarks; MPI today has no infrastructure in place for opening up multiple endpoints. Our micro-benchmark design consists of a "sender-receiver" pair of processes, each running on a separate node, communicating via `libibverbs`. Our experimental setup includes two nodes on the JLSE cluster [1]. Each node hosts an Intel Xeon CPU with a base frequency of 2.5 GHz; the nodes are connected with InfiniBand fabric (`mlx5`).

The endpoint in InfiniBand terminology is the Queue Pair (QP) [2]. Each QP consists of a send and receive queue along with one completion queue (CQ) that is associated with both the work queues. While we can open up multiple QPs in one libibverbs context (CTX), we create multiple endpoints by creating multiple CTXs to keep all QP resources independent. A send or a receive Work Request (WR) that has been posted to the QP is called a Work Queue Entry (WQE). When a WQE completes, a Completion Queue Entry (CQE) is generated in the CQ.

2.1 Benchmark design

In our micro-benchmark, every transfer (send or receive) will generate a CQE and hence, every WQE needs to be checked for completion. The primary loop of the sender consists of two parts: a post loop which issues sends, and a poll which checks for completion of the issued sends. In each iteration, the sender posts as many WQEs are required to fill up the depth of the CTX's QP. This is decided by how many of the send-WQEs have completed, which is determined in the previous iteration of the loop. The receiver is constantly polling for completions of its receive-WQEs. For every completed receive-WQE, a replacement WQE is posted to the QP, if more messages are expected. When we have multiple CTXs, the primary loop of the sender and the receiver is executed in a round-robin fashion for the QP of each CTX.

Figure 1 shows the message rates with varying number of contexts with this benchmark design. We can observe that a single context performs the best.

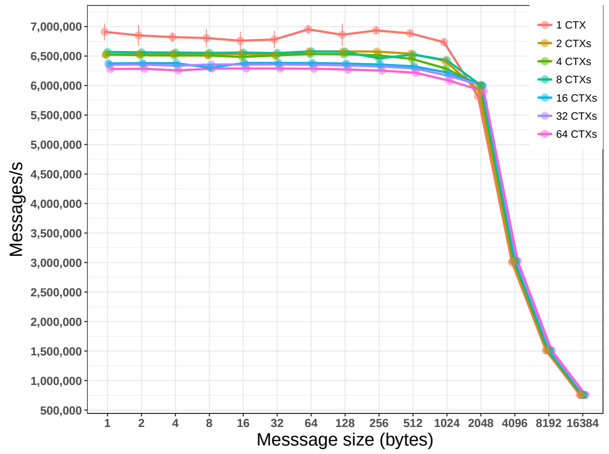

Figure 1: Message Rates with Multiple QPs.

3 ANALYSIS

We measure cycles to explain and analyze the MRs seen with `libibverbs`. To measure cycles, we use Intel's `rdtscp` instruction that reads the processor's time-stamp counter. For all of our analysis below we work with a transfer of 1,000,000 messages; the size of each message is 2 bytes.

3.1 Explaining the MRs observed

The rate of data transfer is contingent the rate of issuing sends. Hence, we first measure the frequency of calls to ibv_post_send. When use 1 CTX, we observe that ibv_post_send is called every 296.18 ± 39.21 cycles in the post loop. However, it takes 3838.88 ± 281.71 cycles after the last call to ibv_post_send in the post loop to start executing the next post loop. Hence, the issue-rate of sends is bottlenecked by the check for their completion. The case is the same when multiple CTXs exist.

So, to explain the message rates observed with different number of CTXs we calculate the issue-rate of what we call a *completed send*. A completed send (only conceptual) entails a post and a successful poll for its completion. We measure the frequency of the post loop and also record the number of successful completions returned in the previous iteration of the sender's primary loop. We calculate the number of cycles for a completed send by dividing the frequency of the post loop by the corresponding number of CQEs returned. The issue-rate of completed sends is simply the CPU frequency divided by the number of cycles taken for a completed send. The calculated issue-rate of completed sends matches the observed message rates and bolsters the fact that the rate of issuing sends is bounded by the check for their completion. Corresponding to the decrease in message rates with the increase in number of CTXs, we observe an increase in the issue-rate of completed sends with an increase in number of CTXs. Figure 2 shows the increase in completed send-cycles for a message size of 2 bytes, and also the corresponding decrease in the calculated issue-rates which match the observed message rates as seen in Figure 1.

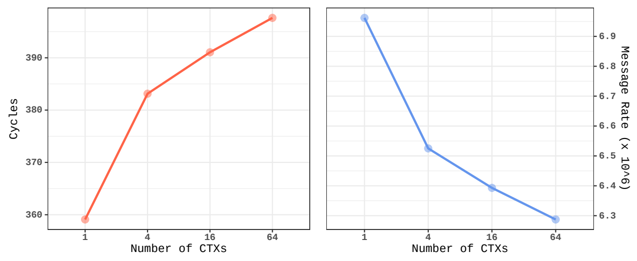

Figure 2: Increase in cycles per completed send (left) and corresponding calculated MRs (right) for a message size of 2 bytes.

3.2 Why the decrease in MRs?

To answer this question, we measure the number of cycles spent in the `ibv_post_send` and `ibv_poll_cq` functions on the sender. With the measurement overhead in the benchmark, the message rates are within 10% of those observed in Figure 1.

Through our measurements we observed that when multiple CTXs exist, different CTXs behave distinctly in both the post and poll operations. CTX_1 and CTX_2 spend a higher time in `ibv_post_send` when 4 CTXs are used. And CTX_2 spends the highest mean number of cycles in `ibv_poll_cq`. This is because the call to `ibv_poll_cq` for CTX_2 returns higher number of CQEs than others; Figure 3 shows that higher number of CQEs returned by `ibv_poll_cq` corresponds to higher number of cycles spent in

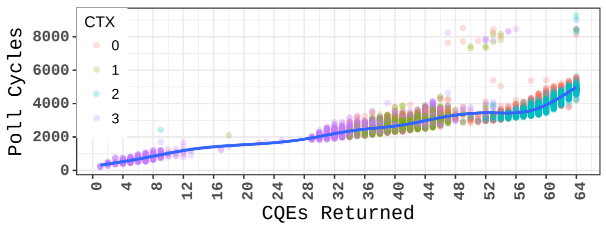

Figure 3: `ibv_poll_cq` cycles VS CQEs returned in its corresponding call

the poll function. Since the number of CQEs returned for CTX_2 is higher, its frequency of calls to the polling function is smaller than that of the other CTXs. Similarly, CTX_1 and CTX_3 spend lower cycles in `ibv_poll_cq` because their number of CQEs returned is relatively lower. But their frequency of calls to `ibv_poll_cq` is higher than other contexts.

Hence, what really matters is the aggregate number of cycles spent in the polling and posting functions. Figure 4 shows the aggregate cycle counts spent in posting and polling against an increasing number of CTXs. The time spent in both posting and polling increases with the increase in number of CTXs, hence the decrease in message rates with increasing the number of CTXs used for transferring data.

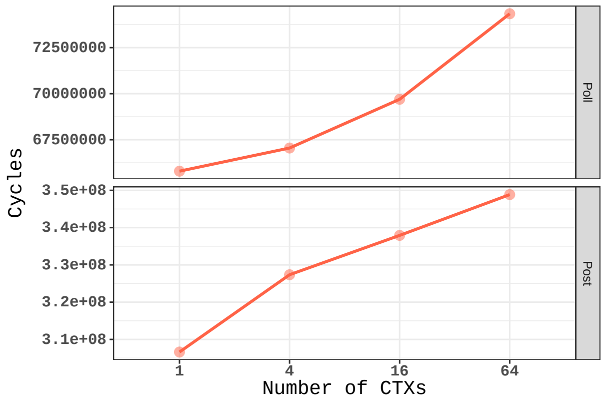

Figure 4: Total number of cycles spent in posting and polling.

4 CONCLUSION

The results of this study show that the check for completion is a bottleneck in issuing sends. Additionally, when multiple endpoints are used with InfiniBand, different QPs behave differently w.r.t. posting and polling operations; the aggregate number of cycles spent in issuing messages is higher with more endpoints. Hence, we can conclude that we are compute-bound for small message sizes. We will extend our analysis to other interconnects and study the differences and similarities with the behavior on InfiniBand on small, medium and large message sizes.

REFERENCES

[1] [n. d.]. JLSE cluster. http://jlse.anl.gov/. ([n. d.]).
[2] 2015. libibverbs API. http://www.rdmamojo.com/2012/05/18/libibverbs/. (Feb. 2015).