In-Depth Evaluation of a Lower-Level Direct-Verbs API on InfiniBand-based Clusters: Early Experiences

Benjamin Michalowicz, Kaushik Kandadi Suresh, Bharath Ramesh,
Aamir Shafi, Hari Subramoni, Mustafa Abduljabbar, Dhabaleswar Panda
Department of Computer Science and Engineering
The Ohio State University Columbus, USA
{michalowicz.2, suresh.1, ramesh.113, shafi.16, subramoni.1, abduljabbar.1, panda.2}@osu.edu

***Abstract*—Many High-Performance Computing (HPC) clusters around the world use some variation of InfiniBand interconnects, all of which are powered by the “Verbs” API. Verbs supply a quick, efficient, and developer-friendly method of passing data buffers between nodes through their interconnect(s). In more recent years, the MLX5-DV (Direct Verbs) API has made itself known as a method of providing mechanisms to access and expose low-level structures and buffers to a developer. In principle, MLX5-DV is meant to give improved performance over Verbs thanks to the removal of intermediate layers of software abstraction. In this paper, we examine the inner workings of what this means for potential performance improvement and how MLX5-DV compares to its higher-level counterpart. In addition, we will offer insights on how application developers and MPI programmers can use this to their advantage based on initial experiences with benchmark and application-level results.**

***Index Terms*—HPC, Interconnects, InfiniBand, RDMA-Core, HCA, Verbs, MLX5DV, MLX5**

I. Introduction

The InfiniBand Verbs (“Verbs”, “IB-Verbs”) API [5] has been an integral component in HPC clusters for over twenty years with its portability and ease of use. It is a well-known library of functions and data structures used as the backbone of several libraries in the HPC community. Primarily, it is used in the realm of inter-node data transfers through send and receive-based operations and supports a wide range of low-level data structures to manage how these messages are processed at the network level.

A. IB-Verbs Are Fast; Can We Go Faster?

Accelerating traditional HPC workloads is a non-trivial task. HPC researchers spend countless hours profiling applications and libraries to understand bottlenecks and design better algorithms. Researchers and end users alike want high-level, portable, and fast solutions. IB-Verbs offers a high-performance, low-latency answer for these applications and libraries. However, IB-Verbs is a relatively high-level API for those that work on MPI libraries and poses some limitations despite all of its capabilities. For example, beneath a call to `ibv_post_send`, multiple steps are taken to write data into the memory location from which it will be sent, such as placing data in the proper spot in memory, examining the type of “send” to be performed, and alerting the hardware to finally transfer said data across the network. Optimizations can be made here, though not with the IB-Verbs API.

*This research is supported in part by NSF grants #1818253, #1854828, #1931537, #2007991, #2018627, #2112606, and XRAC grant #NCR-130002.

B. MLX5-DV — “Direct” Verbs

The MLX5-DV API was designed in mind to do three main things: 1) remove the layers of abstraction presented in the IB-Verbs library around data structures and send/receive routines; 2) expose special memory regions and mechanisms to a developer for potential optimization; and 3) offer a method of bridging the IB-Verbs API through InfiniBand’s“Extended API” [15].

C. Additional Motivation: Rapid Advances in IB Interconnects

As newer network interface cards (NIC) enter the HPC market, software and firmware must be designed and developed to fully harness the resources available on them. For example, NVIDIA’s ConnectX-7 NICs and Quantum-2 switches [11], [13] are capable of **400Gb/s per port**. Algorithms will be made to harness this kind of power in MPI libraries running over IB-Verbs and/or MLX5-DV, and an insight as to how the API can be used both on current and next-generation hardware will further accelerate the research and design that is done on HPC clusters.

Furthermore, the MLX5-DV API has gone through several modifications over the years (see Section II). Some papers have given small evaluations with MLX5-DV such as [18], though no other work has conducted an in-depth investigation of the MLX5DV-API on benchmarks *and* applications.

D. Contribution and Paper Breakdown

This paper provides the following contributions:

1) An evaluation and analysis of the MLX5-DV API in the context of benchmarks and applications against IB-Verbs. This includes run-time performance, CPU-level behavior, and memory/heap footprint analyses.

2) High-and-low-level trade-offs from both APIs, including runtime, flexibility, and the scope of usability for each API.
3) Insights on how to approach development with the MLX5-DV API in standalone applications and MPI libraries.
4) Correlation of performance to CPU-level counters measured via the PAPI library — primarily **in the number of stalled read/write cycles, instructions-per-cycle values, and branch mispredictions**.

To the best of our knowledge, this is the first paper that makes this comparison between both APIs.

The rest of this paper is organized as follows: Section II discusses some more in-depth background and what each API provides as well as work related to this study. Section III explains the general structure of what an MLX5-DV-based send might look like and how it can be designed. Section IV Details our experiments and subsequent analysis at the benchmark and application level. Section V discusses related work in both design and evaluation along this direction. Section VI will summarize our findings and provide insights.

II. Background

A. Background

Here, we will provide background on the Verbs and MLX5-DV APIs and the tradeoffs that come with each of them. In addition, we will briefly explain how MPI libraries can use this API.

1) The IB-Verbs API: As mentioned in Section I, the IB-Verbs API has a long history dating back to the early 2000s. In [1], it is noted that "libibverbs" has been the defacto standard for Verbs-based transfers since the early 2000s and has been a component of the Linux kernel since kernel version 2.6.11 (2005). In addition to InfiniBand-based clusters, numerous protocols can run over Verbs such as TCP/IP and other socket-based interfaces. It provides high-level and easy-to-use functions for an array of operations. This includes managing Send/Receive queues, querying devices to allocate necessary resources, and even providing a level of security for the prevention of data corruption/overwriting.

IB-Verbs' structures have layers to them. They refer to other structures/structure pointers, which refer to other structures and pointers, and so on. This can potentially result in passing around relatively large/heavy structures, which can result in potentially massive memory requirements as an application's scale increases. For example, the `ibv_qp` structure for queue-pairs (QP) holds a reference to a `ibv_context` structure, which contains references to even more data structures. The IB-level QP also has references to other data structures for completion queues, shared-receive queues, and so on.

2) The MLX5-DV API: The MLX5-DV API has shown up under various names over the years. Initially, it was presented as the "mlx4" (and eventually "mlx5") interface in older OFED releases, though operations under this name and other "experimental" Verbs functionalities — where each function has the prefix `ibv_exp_XYZ` [14] — have since been deprecated [16].

Aside from having its own set of data structures and mechanisms, it is compatible with IB-Verbs, requiring the higher-level API to create some set of its structures. There are two more important aspects to MLX5-DV's makeup. The first is that it exposes a host of hardware/software features to perform send/receive functions. These include using programmable I/O through Mellanox/NVIDIA's "BlueFlame" mechanism [12], [24] and exposing the region directly onto which data/WQE's are placed before a transfer — the "doorbell" region. The second is that its data structures are substantially more lightweight. For example, the `mlx5dv_qp` QP structure might look like it contains more fields than its IB-Verbs counterpart, but there are fewer pointers to other data structures [4]. This creates a comparably lighter-weight structure to be passed around from one node to the next.

Structures at this level are also available for specific control/data segments needed to post a message, further breaking down the components needed in send and receive code paths, which may further open up the possibility of performance improvement. All user-level code is available in a publicly available "RDMA-core" repository [4] with documentation from [15].

3) Use in MPI libraries: The Message Passing Interface (MPI) [6] has been around since 1994. Many MPI libraries use IB-Verbs and MLX5-DV mainly in one of three ways: 1) by directly calling IB-Verbs/MLX5-DV API functions/in wrappers; 2) through the OpenFabrics Interface's "Verbs" provider; or 3) through the Unified Communications-X (UCX) library (See Section V).

III. Inside Look: Analysis Of The MLX5-DV Send/Recv Paths

A. The MLX5-DV Send Path

In this section, we will break down how a "send" operation can be performed using MLX5-DV.

On the IB-Verbs side, the main steps of performing a send include setting up a work request (`struct ibv_wr`), appropriately setting up a QP, and calling `ibv_post_send` while letting the underlying levels of the library handle the remaining steps.

The first step in an MLX5-DV-based send is to determine what kind of "send" we wish to perform — a "standard" send, an atomic operation, or an RDMA write. We will examine the first option.

For a standard send, we deal with the QP as well as intermediate data segments and the need to determine the proper sizes for each portion of the send-based data structures involved. Firstly, we must create and manage the WQE that will eventually be posted. We represent the first data segment of it as a triple — a byte count, a local key (`lkey`), and an address to represent the buffer of data being sent. The lkey will later be used to match a send operation with a receive operation. Next, a data structure for the control segment is initialized to contain the correct send opcode, a QP number, WQE size, immediates for offsets, etc. The lkey is also given to the actual buffer being sent as metadata.

At the MLX5-DV level, the doorbell record is an address space to which the counter for a given WQE is written – similar to how a WQE is posted from the IB-Verbs level in `ibv_post_send`. Once this record is posted, the destination buffer — a `void *` pointer — is assigned a reference to an address used inside the work queue. From here, we transfer the data from the source buffer to the destination buffer in one of two ways: 1) we copy the message through the BlueFlame/programmable I/O, or 2) use `memcpy`. Lastly, pointers for the work queue are updated so that the next WQE in line can be placed in this memory address and posted to the HCA, and the BlueFlame register is given an XOR operation using its size/length. This "rings" the doorbell and sends a signal to the hardware to send our payload.

Assembly-level "barriers" are inserted into the code for correctness and to prevent possible latency degradations. For example, UCX has such barriers surrounding writes to the doorbell region and copying the source buffer to its destination.

B. The MLX5-DV Receive Path

The MLX5-DV-based "post-receive" involves fetching WQEs from an SRQ in an infinite loop. The loop exits when setting a corresponding data segment on the receive side is no longer possible. This happens 1) if The SRQ entry has been posted but not released, or 2) if setting the next data segment results in an error. The latter involves iterating over a bitmap and setting data segments as needed. After this loop, the doorbell record is then written to update the SRQ's resource counts. Compared to the send operation, assembly-level "barriers" are not needed on the receive side.

IV. Experiments and Evaluation

Here, we discuss our experimental setup, tests, and results. Our primary cluster of choice features NVIDIA ConnectX-6 HDR100 100Gb/s InfiniBand/VPI adapters and Dual-Socket Intel Xeon E5-2697A V4 CPUs @ 2.60 GHz (32 cores/node). The next several subsections, in order, show results on the uni-directional bandwidth, bi-directional bandwidth, and latency OSU Microbenchmarks (OMB) v5.9 [10], followed by OMB collectives at various scales and processes per node (PPN). In addition, we show various metrics obtained through PAPI version 6.0.0 [22] on five different metrics: Total Cycles, Total Instructions, Stalled Cycles from ALL resource requests, L3 Cache Misses, and Total Branch Mispredictions.

Our software stack is MVAPICH2-3.0a as released by The Ohio State University [9] running over UCX 1.13.1. UCX provides communication and transport-layer protocols for InfiniBand-based clusters (see more in Section V).

Each graph contains three lines: The blue line represents IB-Verbs-based runs, the orange line represents MLX5-DV, and the gray line represents the percent improvement MLX5-DV obtains with respect to IB-Verbs. The percent improvement runs along the right-hand vertical axis.

For PAPI results, we note the following:

1) For total cycles, lower is better.
2) For stalled cycles, lower is better.
3) For L3/LLC cache misses, lower is better.

All results with OMB were obtained over six back-to-back runs, with the inner loop of each microbenchmark being profiled through PAPI. This allows our performance metrics obtained from OMB to correlate directly with the information obtained by PAPI. The performance counters are totaled over the course of each message size — the standard/default message sizes set by each OMB. Each collective was run from two to sixteen nodes, increasing from 1 PPN to 32 PPN by powers of 2. Higher bandwidth (MB/s) results are better, and lower latency (microseconds) are better.

In an attempt to examine further tradeoffs between both APIs, we utilize Massif [8], a memory checker that analyzes memory footprint throughout a program's runtime within Valgrind. Massif outputs human-readable files that can be further processed to include graphs beyond human-readable content, and we analyze files from each run of the collective OMB to further examine how each API behaves.

A. Runtime Results

In this section, we will display and explain our analysis of the pure runtime numbers for various point-to-point and collective OMB run over IB-Verbs and MLX5-DV.

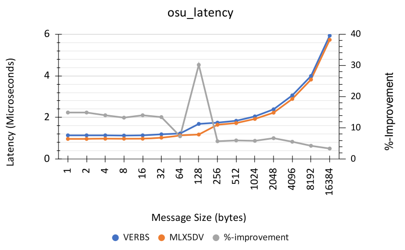
1) Point-to-Point: Through OMB, we deduce that the MLX5-DV API is beneficial for smaller, and sometimes medium message sizes. Point-to-point latency, as shown in Figure 1a, starts consistently performing less than five percent better than IB-Verbs after 4KB, and the difference becomes negligible beyond 16KB. These messages are all sent over an eager protocol, which becomes a bottleneck as message sizes increase. Both APIs need handling of messages in the context of an MPI library such as MVAPICH2 to account for larger messages, hence the increasingly small performance improvements.

2) Collectives: This section showcases and analyzes performance results on collective operations at various scales (2 to 16 Nodes, up to 32 PPN).

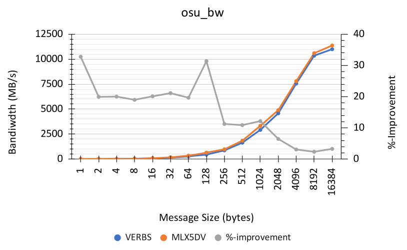
Our first two figures detail the performance of osu_allgather at various node and PPN counts. In Figure 2, we see up to a 30% benefit within osu_allgather across various scales Between each set of results, smaller performance benefits/degradations may vary, with larger degradations largely unchanging.

In Figure 3, we see some factors that can influence performance either positively or negatively in the context of osu_allgather. The first subgraph demonstrates the potential performance degradations seen if full subscription is used with UCX, as the library allocates a helper thread for asynchronous progress. As we will see later, some collectives may perform better, and some worse, when attempting to utilize all cores on a node. System variation can sometimes be a culprit and lead to results shown where an already-low latency's fluctuations can mean massive performance degradations.

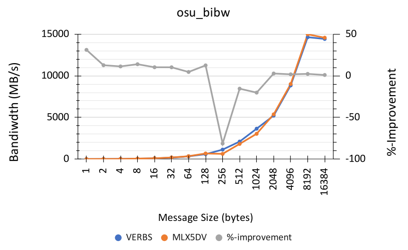
Pure inter-node runs (1-PPN) with any number of nodes will generally perform as well or better for most collectives.



(a) osu_latency on 2 Nodes, 1 PPN

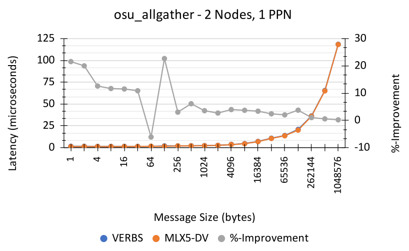


(b) osu_bw on 2 Nodes, 1 PPN

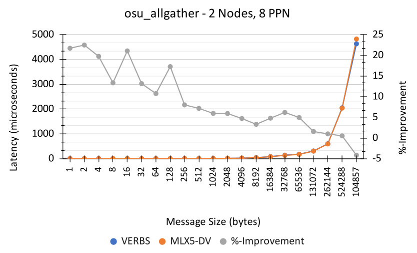


(c) osu_bibw on 2 Nodes, 1 PPN

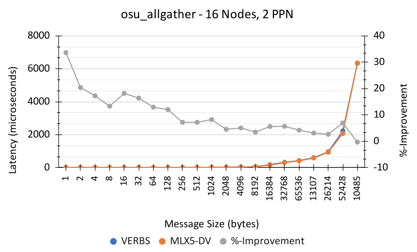
Fig. 1: OMB Pt2Pt up to 16KB with percent difference in performance



(a) osu_allgather at 2 Nodes, 1 PPN

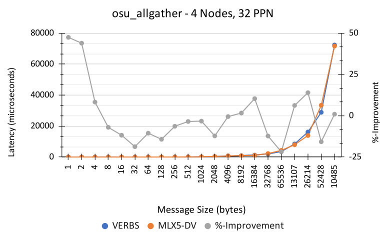


(b) osu_allgather at 2 Nodes, 8 PPN

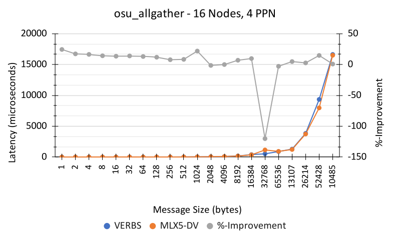


(c) osu_allgather at 16 Nodes, 2 PPN

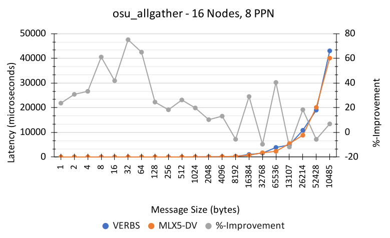
Fig. 2: osu_allgather: A sampling of MLX5-DV's Performance Benefits against IB-Verbs



(a) osu_allgather at 4 Nodes, 32 PPN



(b) osu_allgather at 16 Nodes, 4 PPN



(c) osu_allgather at 16 Nodes, 8 PPN

Fig. 3: osu_allgather: A sampling of MLX5-DV's Performance Fluctuations

Figure 4 details similar trends to previous figures with a slowly decreasing performance improvement as message size increases.

Certain collectives are a bit less predictable with the MLX5-DV API, such as osu_scatter and osu_alltoall. The performance of osu_scatter changes vastly between 8 nodes/32 PPN and 16 Nodes/32 PPN, with the latter showing MLX5-DV obtaining half the runtime as its IB-Verbs counterpart. Figure 5 shows 1-megabyte messages gaining more than a 50% performance against IB-Verbs. osu_alltoall's performance improvements vary heavily with respect to IB-Verbs.

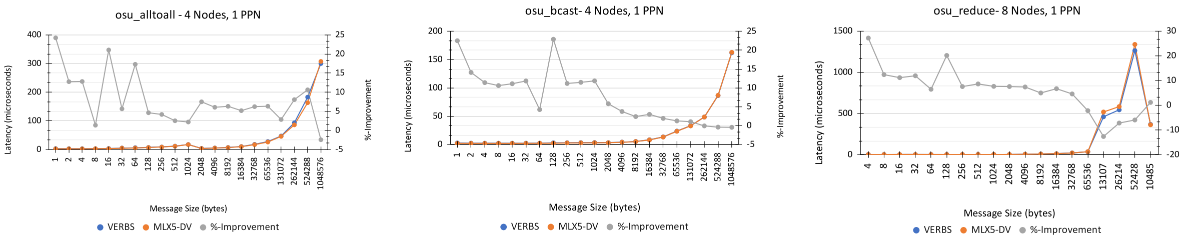
B. *PAPI Counter Analysis*

As previously mentioned, we use a subset of PAPI counters to profile the inner loop of various OMB — that is, the loop performing both warmup and timed iterations of each point-to-point and collective operation.

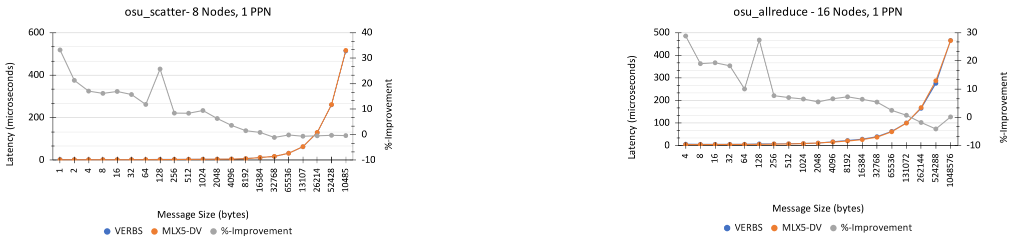
1) PAPI counters on Point-to-Point OMB: In general, the PAPI counters taken correlate with our runtime results. As per Figure 6, we can see that MLX5-DV's direct access to lower-level memory improves various performance metrics, such as a decrease in stalled cycles as well as Instructions and Clock Cycles. In addition, MLX5-DV is shown to have a smaller number of total stalled cycles in bi-directional bandwidth — one whole order of magnitude — which explains the larger performance benefits at the point-to-point level.

2) PAPI counters on Collectives: Here, we will try to correlate our runtime results from Section IV-A with the PAPI counter results shown here.

Figure 7 showcases the PAPI counters obtained from the runs in Figure 2. Almost immediately, we see how decreased numbers in metrics like stalled cycles and identical clock-cycle counts aid in the small message performance benefits for the 16-node/2-PPN and 2-Node/1-PPN cases. While helpful

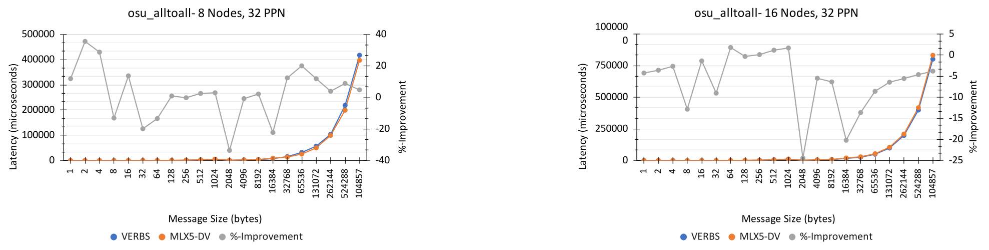


(a) osu_alltoall at 4 Nodes, 1 PPN (b) osu_bcast at 4 Nodes, 1 PPN (c) osu_reduce at 8 Nodes, 1 PPN

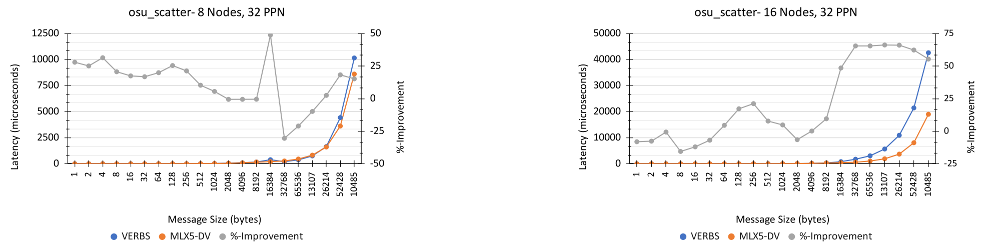


(d) osu_scatter at 8 Nodes, 1 PPN (e) osu_allreduce at 16 Nodes, 1 PPN

Fig. 4: IB-Verbs v. MLX5-DV: 1-PPN Collectives-level Performance



(a) osu_alltoall at 8 Nodes, 32 PPN (b) osu_alltoall at 16 Nodes, 32 PPN



(c) osu_scatter at 8 Nodes, 32 PPN (d) osu_scatter at 16 Nodes, 32 PPN

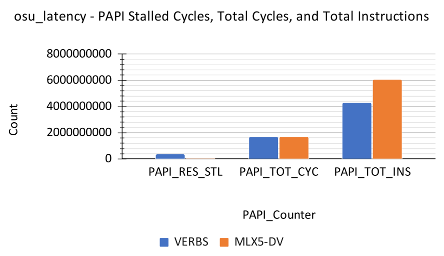
Fig. 5: IB-Verbs vs. MLX5DV: Performance changes in osu_scatter and osu_alltoall at full subscription (32 PPN)

in analysis, branch mispredictions and total instruction count appear to have a smaller impact on the performance benefits thus far than the rest of the metrics.

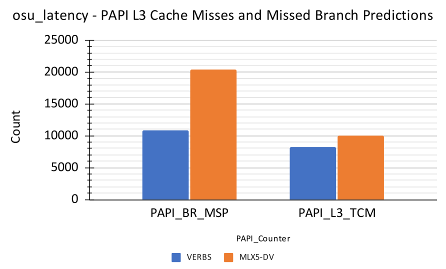
osu_scatter's PAPI counters over MLX5-DV have a stronger correlation with improved runtime; the substantially smaller number of stalled cycles, L3 cache misses, and increased IPC values all contribute, especially at the larger messages in both the 8 and 16-node cases. However, extensive analysis is needed for osu_alltoall given a) its communication-intensive nature, and b) the sporadic change in performance across both scales and message sizes.

C. Memory footprint

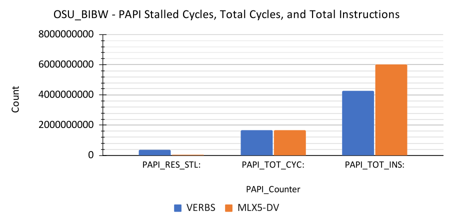
We profiled four different collective OMB — osu_alltoall, osu_gather, osu_scatter, and osu_allreduce — at 8 nodes and various PPN with the Massif tool that comes along with Valgrind [8]. Figure 10 shows how MLX5-DV's heap footprint remains constant against the increasing amount of memory requested by IB-Verbs. This initial reduction in memory footprint comes from the shallower MLX5-DV structures mentioned in Section II-A2.



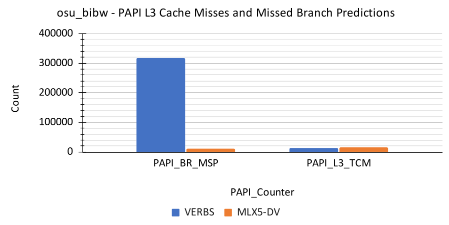
(a) osu_latency performance counters on total stalled cycles, total cycles, and total instructions



(b) osu_latency performance counters on branch mispredictions and L3 cache misses



(c) osu_bibw performance counters on total stalled cycles, total cycles, and total instructions



(d) osu_bibw performance counters on branch mispredictions and L3 cache misses

Fig. 6: PAPI counters on both MLX5-DV and IB-Verbs for osu_latency and osu_bibw. For this and subsequent graphs, the majority of the performance benefits are influenced by L3 Cache Misses, number of cycles, and Total number of Stalled Cycles

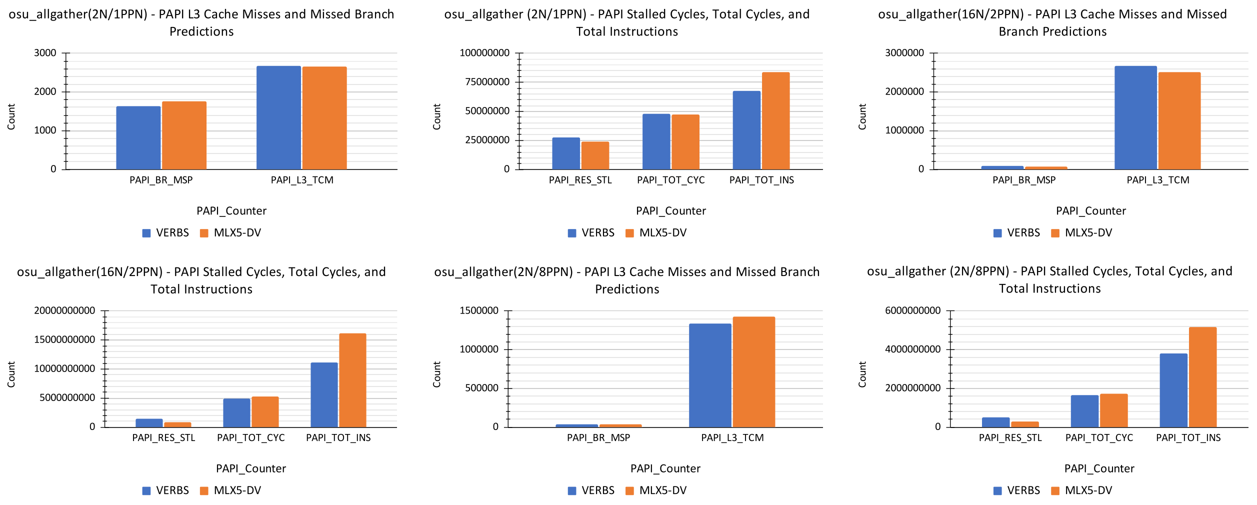


Fig. 7: PAPI counters on both MLX5-DV and IB-Verbs, focused on osu_allgather results from Figure 2

Figure 11 shows a deviation in the increased memory footprint exhibited by IB-Verbs-backed variant of MVAPICH2-3.0a over UCX. Once a microbenchmark is run at more than 512 processes, we see a sudden drop in IB-Verbs' memory consumption. At runtime, unless a user selects specific protocols (UD, RC, etc.), UCX will default to the best protocol for initialization at runtime (such as UD), hence the drop in IB-Verbs memory consumption beyond 512 processes.

This difference in heap usage, however, will not be apparent at scale with applications, as medium and large problem sizes will dwarf the heap usage of an MPI library.

D. Collectives at Scale

We briefly examine larger-scale runs on the Frontera [21] supercomputer. We showcase 128-node collectives at 16-PPN (2048 processes). Figure 12 shows how how performance can change at a large scale for collectives like osu_allreduce, osu_bcast, osu_gather, and osu_scatter. Whereas we saw smaller message benefits at smaller scales, larger messages *generally* exhibit larger benefits here.

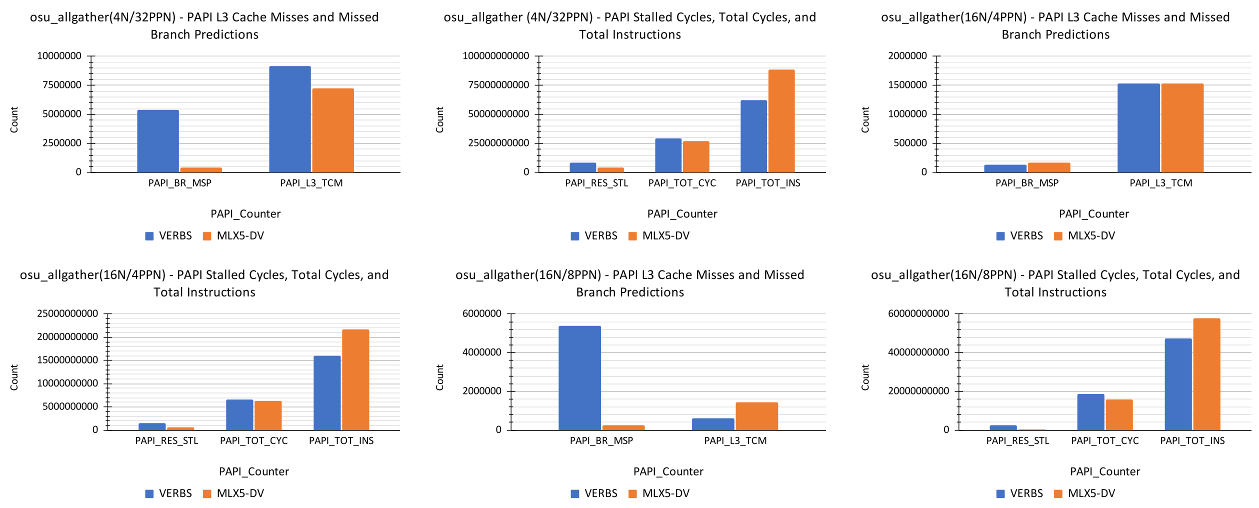


Fig. 8: PAPI counters on both MLX5-DV and IB-Verbs, focused on osu_allgather results from Figure 3

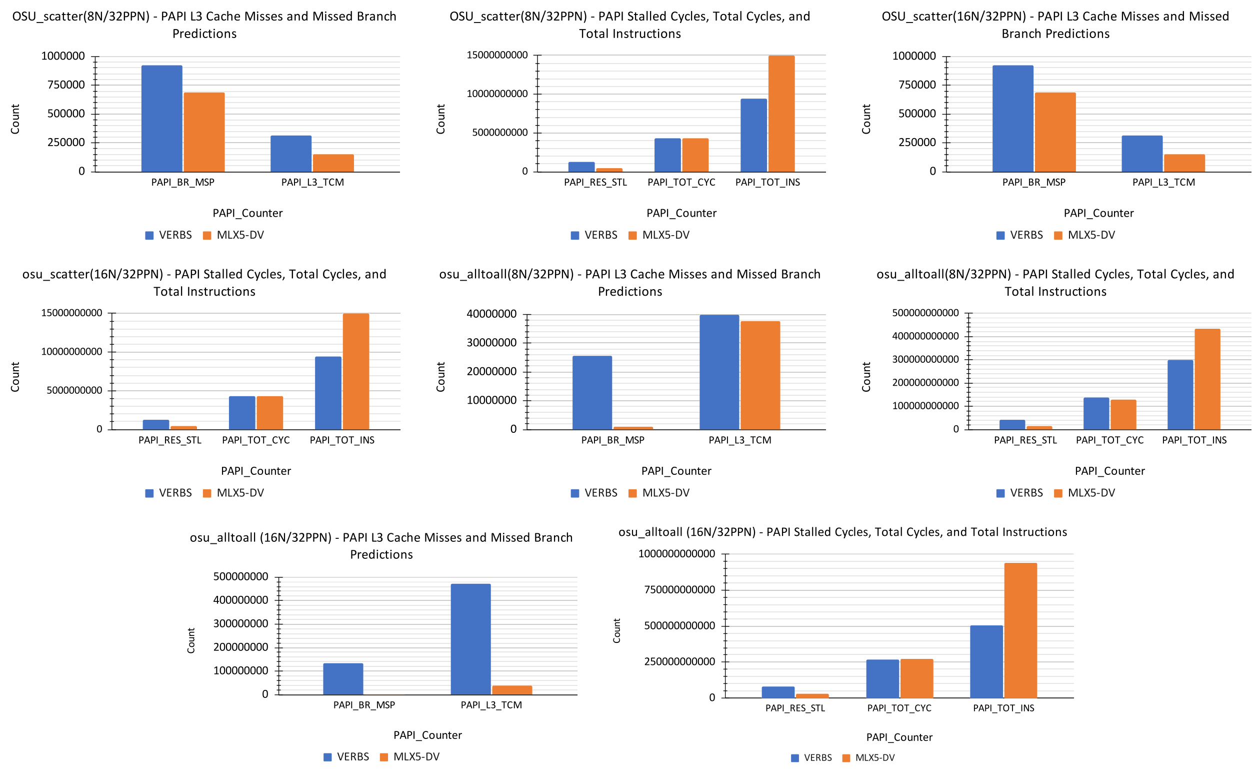
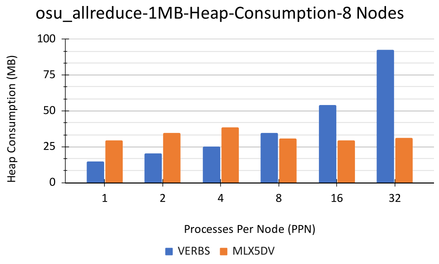


Fig. 9: PAPI counters on both MLX5-DV and IB-Verbs, focused on osu_scatter and osu_alltoall results from Figure 5.

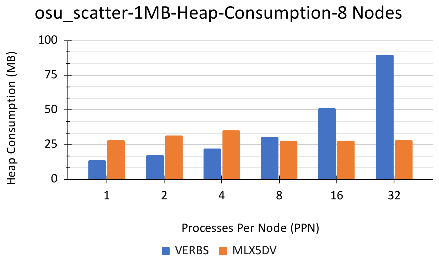
We previously saw benefits at small messages for osu_allreduce shrank until the 512KB message size. Overall, MLX5-DV gives up to 20% performance benefits at larger scales.

E. Application-Level Results

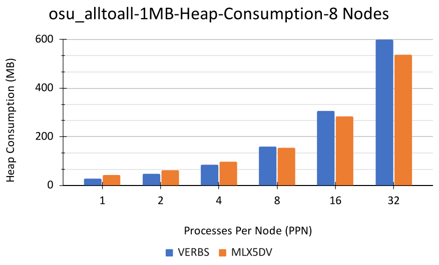
Here, we present the results of two applications running MVAPICH2-3.0a over IB-Verbs and MLX5-DV at a medium scale. Contrary to OMB, 32 PPN cannot be done on account of UCX's asynchronous thread. One application is MILC [7] — a lattice-based physics application. The other is LAMMPS [23] — a molecular-dynamics simulation code.



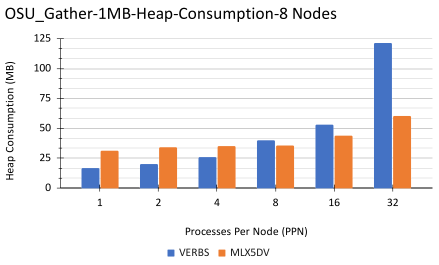
(a) Heap usage in osu_allreduce



(b) Heap usage in osu_scatter

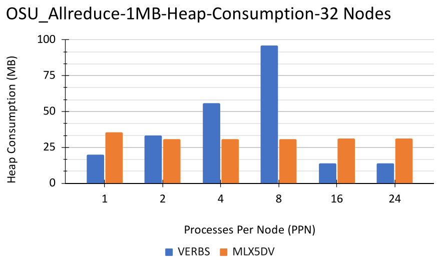


(c) Heap usage in osu_alltoall

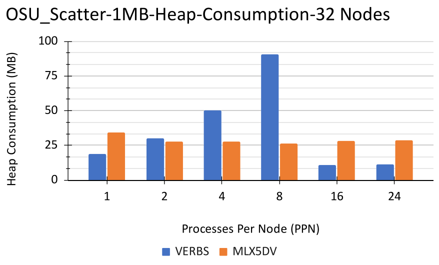


(d) Heap usage in osu_gather

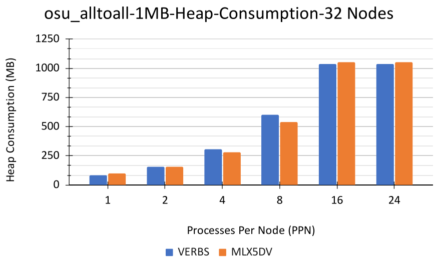
Fig. 10: Heap Profiling with Massif on Different Communication Patterns/OMB (8 Nodes)



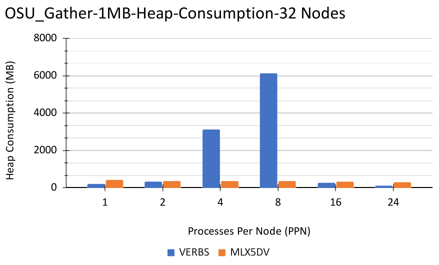
(a) Heap usage in osu_allreduce



(b) Heap usage in osu_scatter



(c) Heap usage in osu_alltoall

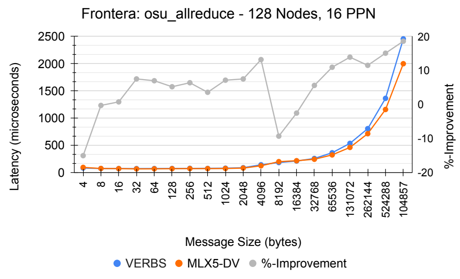


(d) Heap usage in osu_gather

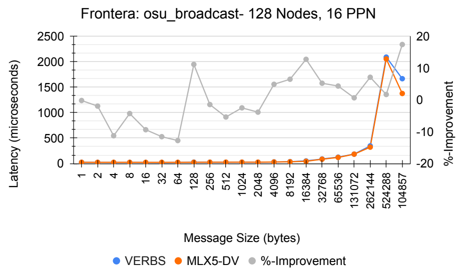
Fig. 11: Heap Profiling with Massif on Different Communication Patterns/OMB (32 Nodes)

In MILC, we run the pure gauge "su3_rmd" test with a mesh of dimensions 170 x 180 x 160 x 7. The numbers for our MILC experiments (averaged over six back-to-back executions) are shown in Figure 13. Here we see that in this problem set and at these scales, MLX5-DV generally obtains either small, consistent, positive performance improvement, or virtually no improvement (less than 1%) in either the positive or negative direction. Smaller problem sizes offer similar/consistent results. TAU [20] profiles with MILC indicate mild fluctuations in runtime between both MLX5-DV and IB-Verbs during MPI communication, leading us to believe that the message sizes used within MILC relate to the areas of little performance difference for given collectives.

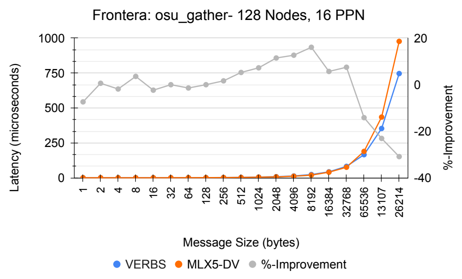
Our LAMMPS experiments use an enlarged version of the



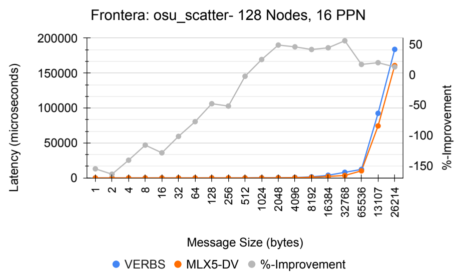
(a) 128N-16PPN osu_allreduce



(b) 128N-16PPN osu_bcast

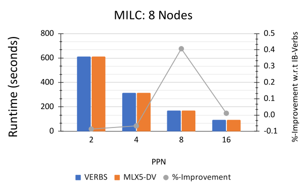


(c) 128N-16PPN osu_gather

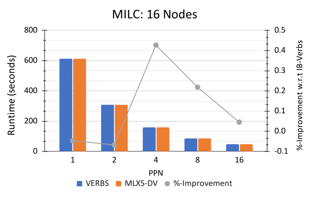


(d) 128N-16PPN osu_scatter

Fig. 12: Large-Scale Collectives on the Frontera Supercomputer



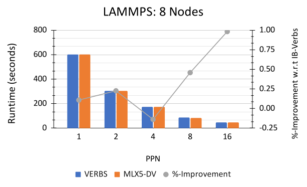
(a) MILC on 8 nodes



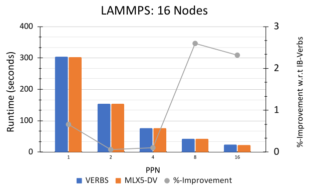
(b) MILC on 16 nodes

Fig. 13: MILC 8-and-16-Node, Multi-PPN Experiments

3D Lennard-Jones melt experiment with numbers obtained identically to that of the MILC experiments shown in Figure 14. Here, we see that while MLX5-DV offers a slightly larger improvement compared to that of MILC's executions in these experiments, neither API vastly outperforms the other. Further experimentation on different problem sets and scales will be needed to further evaluate performance impact.



(a) LAMMPS on 8 nodes



(b) LAMMPS on 16 nodes

Fig. 14: LAMMPS 8-and-16-Node, Multi-PPN Experiments

V. Related Work

UCX [19] is a standalone library that can be incorporated into other HPC libraries and act as the communication layer for MPI and PGAS environments. UCX also provides the ability to turn on/off support for MLX5-DV-based functionality at the point-to-point and collective levels[1]. [25] performed an analysis on high-performance communication using PCIe hardware counters to measure low-level transfers and analyze bottlenecks within HPC communication. While a very insightful work, not all clusters have such hardware to perform the same level of analysis. [3] uses a performance monitoring library (found at [2]) to analyze PCIe metrics. This library requires root access to use and, to the best of our knowledge, cannot be run inside an MPI application. While we attempted to use this library (with root access enabled), we found it was suitable for our analysis. [17] also performs an initial evaluation of UCX over InfiniBand, focusing on its performance under MPICH-3.3, though with no focus on the MLX5 (later MLX5-DV) support. [18] dedicates a section to a comparison of MLX5-DV and IB-Verbs in the context of RMA operations.

VI. Insights and Conclusion

A. *Insights: Trade-offs in APIs*

MLX5-DV already shows significant benefits with the potential to out-perform its IB-Verbs counterpart. At the same time, it also has its own set of drawbacks. For example, clusters running on older OFED releases may not be able to benefit from the usage of this API. Furthermore, utilization of this API requires more effort on the developer than if they

[1]This is a configure-time option, not a run-time option

were using IB-Verbs. As mentioned in Section III, there is no exact counterpart for `ibv_post_send`. As it currently stands, MLX5-DV appears to be more susceptible to system jitters that lead to potentially wild performance benefits and degradations. As we have seen, MLX5-DV's memory consumption does not change when transitioning between the UD and RC communication protocols.

Based on our results, we propose the following suggestions: 1) Interchange MLX5-DV and IB-Verbs at different message sizes. Smaller message sizes appear more advantageous for MLX5-DV – such as those below 16KB — in the majority of cases; 2) Utilize different APIs at different scales — while more analysis is needed, MLX5-DV generally shows benefits at larger scales. Fine-tuning for each collective is needed to obtain more improvements beyond what is shown in Figure 12, and further profiling is needed to explore the large-message degradations occurring within the large-scale gather. MPI libraries traditionally run either over a layer such as UCX or the Open Fabrics Interface (OFI) or by calling IB-Verbs functions directly. Being a low-level extension of IB-Verbs, MLX5-DV could be made more user-friendly by providing an option for standardized send and receive-based abstractions so that they can be more readily integrated should an MPI library attempt to utilize it — similar to `ibv_post_send/recv`.

B. Conclusion and Future Work

In this paper, we performed an in-depth analysis and evaluation of two approaches to communication in InfiniBand-based clusters. We have presented results at various scales including runtime analysis, collection of performance counters at the CPU level, and analyses on the memory footprint of both the MLX5-DV API and the IB-Verbs API in the case of the OSU Microbenchmark suite. We have also provided insights and results on two well-known applications at medium-large scales. We have provided insights on the performance tradeoffs between both APIs and how to integrate them into both standalone applications and, e.g., MPI libraries. Future directions along this line include further profiling and analysis, further exploration of features, and attempts to stabilize the performance shown in this paper. It would be prudent to analyze instructions executed in the "Send" and "Receive" sections of each code path, such as through Intel's Software Development Emulator, compared to the results shown through PAPI in Section IV-B. Investigating other communication protocols within the IB stack in the manner of this paper will be interesting, such as with the DC protocol. In addition, we hope to further profile MLX5-DV to stabilize the performances shown in this paper.

REFERENCES

[1] Dotan Barak. Verbs programming tutorial, 2014. https://www.cs.mtsu.edu/ waderholdt/6430/papers/ibverbs.pdf.

[2] Intel. Intel/pcm: Processor counter monitor, Jan 2017. https://github.com/intel/pcm.

[3] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Design guidelines for high performance RDMA systems. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 437–450, Denver, CO, June 2016. USENIX Association.

[4] Kernel.org team. Rdma-core userspace libraries and daemons repository. https://github.com/linux-rdma/rdma-core.

[5] Patrick MacArthur, Qian Liu, Robert D. Russell, Fabrice Mizero, Malathi Veeraraghavan, and John M. Dennis. An integrated tutorial on infiniband, verbs, and mpi. *IEEE Communications Surveys & Tutorials*, 19(4):2894–2926, 2017.

[6] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 4.0*, June 2021.

[7] MIMD Lattice Computation Collaboration. MIMD Lattice Computation (MILC). https://web.physics.utah.edu/ detar/milc/milc_qcd.html.

[8] Nicholas Nethercote, Robert Walsh, and Jeremy Fitzhardinge. "building workload characterization tools with valgrind". In *2006 IEEE International Symposium on Workload Characterization*, pages 2–2, 2006.

[9] Network-Based Computing Laboratory. Mvapich. https://www.mvapich.cse.ohio-state.edu/.

[10] Network-Based Computing Laboratory. Osu microbenchmarks. https://mvapich.cse.ohio-state.edu/benchmarks/.

[11] NVIDIA. NVIDIA ConnectX-7 NDR 400 InfiniBand Adapter Card.

[12] NVIDIA. Nvidia mlnx_ofed documentation rev 5.6-2.0.9.0 user manual. https://docs.nvidia.com/networking/display/MLNXOFEDv562090/Introduction.

[13] NVIDIA. Nvidia quantum-2 infiniband platform.

[14] NVIDIA. Mlnx_ofed v4.5-1.0.1.0 documentation, 2018. https://docs.nvidia.com/networking/display/MLNXOFEDv451010.

[15] NVIDIA. Migration to rdma-core, 2020. https://docs.nvidia.com/networking/display/rdmacore50/Migration+to+RDMA-Core#MigrationtoRDMACore-Overview.

[16] NVIDIA. Mlnx_ofed documentation rev 5.0-1.0.0.0, 2021. https://docs.nvidia.com/networking/display/OFEDv501000/MLNX_OFED+Documentation+Rev+5.0-1.0.0.0.

[17] Nikela Papadopoulou, Lena Oden, and Pavan Balaji. A performance study of ucx over infiniband. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 345–354, 2017.

[18] Pavel Shamis, M. Graham Lopez, and Gilad Shainer. Enabling one-sided communication semantics on arm. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 805–813, 2017.

[19] Pavel Shamis, Manjunath Gorentla Venkata, M Graham Lopez, Matthew B Baker, Oscar Hernandez, Yossi Itigin, Mike Dubman, Gilad Shainer, Richard L Graham, Liran Liss, et al. Ucx: an open source framework for hpc network apis and beyond. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 40–43. IEEE, 2015.

[20] Sameer S. Shende and Allen D. Malony. The tau parallel performance system. *Int. J. High Perform. Comput. Appl.*, 20(2):287–311, may 2006.

[21] Dan Stanzione, John West, R. Todd Evans, Tommy Minyard, Omar Ghattas, and Dhabaleswar K. Panda. Frontera: The evolution of leadership computing at the national science foundation. In *Practice and Experience in Advanced Research Computing*, PEARC '20, page 106–111, New York, NY, USA, 2020. Association for Computing Machinery.

[22] Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. Collecting performance data with papi-c. In Matthias S. Müller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel, editors, *Tools for High Performance Computing 2009*, pages 157–173, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[23] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton. LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Comp. Phys. Comm.*, 271:108171, 2022.

[24] Rohit Zambre, Aparna Chandramowlishwaran, and Pavan Balaji. Scalable communication endpoints for mpi+threads applications. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 803–812, Dec 2018.

[25] Rohit Zambre, Megan Grodowitz, Aparna Chandramowlishwaran, and Pavel Shamis. Breaking band: A breakdown of high-performance communication. In *Proceedings of the 48th International Conference on Parallel Processing*, ICPP 2019, New York, NY, USA, 2019. Association for Computing Machinery.