DPU-Bench: A Micro-Benchmark Suite to Measure Offload Efficiency Of SmartNICs

Benjamin Michalowicz
michalowicz.2@osu.edu
The Ohio State University
Columbus, Ohio, USA

Kaushik Kandadi Suresh
kandadisuresh.1@osu.edu
The Ohio State University
Columbus, Ohio, USA

Hari Subramoni
subramoni.1@osu.edu
The Ohio State University
Columbus, Ohio, USA

Dhabaleswar K. Panda
panda@cse.ohio-state.edu
The Ohio State University
Columbus, Ohio, USA

Steve Poole
swpoole@lanl.gov
Los Alamos National Laboratory
Los Alamos, NM, USA

ABSTRACT

Smart Network Interface Cards (SmartNIC) have experienced massive growth in popularity over the last few years such as the NVIDIA BlueField-2 Data Processing Unit (DPU). Being equipped with their own set of cores and memory allows them to perform actions beyond a regular NIC, and HPC researchers are designing new ways to use them. For example, offloading communication to one enables the CPU "host" to perform more computationally heavy tasks. However, one question remains: How much of that work can be distributed among processes placed on the SmartNIC before facing performance degradation? We present DPU-Bench: A low-level micro-benchmark suite using IB-Verbs primitives to enable HPC users to examine the number of processes to be placed on one or more SmartNICs in order to efficiently offload a given communication pattern. We examine direct algorithms in this paper at a medium scale with different work assignment mechanisms and give insights into the trends found with varying numbers of worker processes and message sizes.

KEYWORDS

SmartNICS, High-Performance Computing Interconnects, Network-Based Computing, Benchmarks, Micro-Benchmarks, InfiniBand

ACM Reference Format:
Benjamin Michalowicz, Kaushik Kandadi Suresh, Hari Subramoni, Dhabaleswar K. Panda, and Steve Poole. 2023. DPU-Bench: A Micro-Benchmark Suite to Measure Offload Efficiency Of SmartNICs. In *Practice and Experience in Advanced Research Computing (PEARC '23), July 23–27, 2023, Portland, OR, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3569951.3593595

1 INTRODUCTION

High-Performance Computing has evolved rapidly in recent years. In particular, accelerators have become the dominating force among the majority of the Top500 [16]. MPI libraries and benchmarks have been designed to test and become aware of these accelerators as they've appeared and evolved over the years. A substantially more recent development is that of the SmartNIC. In particular, NVIDIA's BlueField Data-Processing Units (DPU) have rapidly developed to become a part of HPC clusters and have popularized the abilities of SmartNICs to have a plethora of resources beyond what's provided by both CPUs and a regular NIC. However, designing applications and/or libraries to efficiently utilize the added resources is a non-trivial task, and naively placing processes onto the DPU for applications and/or benchmarks does not necessarily mean enhanced improvement. To combat this, we propose DPU-Bench: an IB-Verbs-level[9] micro-benchmark suite to calculate the offload efficiency of staging utilized in collective operations.

1.1 Motivation

1.1.1 ***The need to design a new benchmark suite:*** Benchmark suites such as the OSU Microbenchmarks and Intel Microbenchmarks are primarily used to test functionality and baseline performance of MPI operations [6, 12] (For more on these, see Section 5). They answer the question of whether an HPC cluster is running as expected and how new MPI-level designs compare to the current state of the art. These benchmarks cannot measure the latency incurred when offloading to devices like DPUs, nor what the offload potential is — including time taken to exchange information between a host and a worker process (processes spawned on the DPUs that assist in communication). This benchmark suite measures the offload efficiency of collective communication patterns one or more DPUs so as to determine whether the same pattern will benefit from such offloading in an application. This brings us to our first major challenge: How can we design a benchmark that will allow us to measure the amount of overlap that can be achieved by offloading a custom communication pattern to the DPUs?

Listing 1: Template of a Scatter-Dest Alltoall using Nonblocking MPI_Isend/Irecv calls

```
for (i = 0 ; i<comm_size ; i++){
    s = comm_size - i;
    for(j = 0; j < s; j++){
        dst = (cur_rank + i + j)
            % comm_size;
        // Offset is calculated
        MPI_Irecv(recvbuf+offset, count, type,
            dst, tag, comm_ptr, &req_ptr[j]);
```


PEARC '23, July 23–27, 2023, Portland, OR, USA

ACM ISBN 978-1-4503-9985-2/23/07…$15.00
https://doi.org/10.1145/3569951.3593595

```
    }
    for(j = 0; j < s; j++){
        dst = (cur_rank - i - j + comm_size)
            % comm_size;
        // Offset is calculated elsewhere
        MPI_Isend(sendbuf+offset, count, type,
            dst, tag, comm_ptr, &req_ptr[j+s]);
    }
    dummy_compute();
    MPI_Waitall(2*s, req_ptr, status_arr);
}
```

1.1.2 ***Developing IB-Level benchmarks:*** Our second challenge is: How can we design benchmarks to reflect the true amount of communication and overlap measured when offloading communication to DPUs? Listing 1 shows a common MPI-level algorithm for a non-personalized all-to-all communication pattern. Naively offloading communication to the DPUs such as the sends and receives in this code snippet is costly: while the number of function calls may not be an issue, message progression on both the send and receive paths is within any MPI library — MPI libraries have what is known as a "progress engine" that ensures whether non/blocking send and receive operations have fully completed. This issue is further exacerbated by operations taking place between CPU and DPU processes. We decided to forgo MPI in favor of IB-Verbs [2, 9]. IB-Verbs allows us to perform operations and stage data to DPUs more easily for a number of reasons. The first is that RDMA semantics do not face the issue of message progression, guaranteeing that data sent will reach its target immediately. The second is that RDMA-level semantics also allow for the recipient of an RDMA read or write to not be involved with data exchange; this is crucial for us, as we want to ensure that there is no communication performed by the CPU-based processes in our benchmark once data is staged to the DPUs. Thirdly, the only bottlenecks involved at this level are the limitations of the hardware and the need to poll a completion queue.

1.1.3 ***Benefits of this Benchmark Suite:*** **To the best of our knowledge, this is the first benchmark suite of its kind in existence.** Previous works [3, 13] have presented staging-based solutions within an MPI library on offloading communication to DPUs. However, not many MPI libraries are DPU-aware as of the writing of this paper. Because of this, and since no other HPC benchmark suite takes the notion of offloading communication to DPUs, we propose these benchmarks to determine if an application's communication can be further improved via offloading to them.

1.2 Contributions

This paper makes the following contributions:

(1) Design of a new micro-benchmark suite that lies closer to hardware to analyze the offload potential of various collectives.
(2) Explores the impact of staging-based offload of communication to DPUs on three direct-algorithm-based implementations for one-to-all, all-to-one, and non-personalized-all-to-all communication patterns.
(3) An analysis on said designs and presenting results on both the offload potential/efficiency and the possible runtime impact that offloading communication provides.
(4) Empirical analysis of how to obtain the maximum possible benefits from offloading communication over a variable number of worker processes.

1.3 Paper Breakdown

The rest of the paper is broken down as follows: Section 2 explains the background needed to understand our motivation and design. Section 3 explains the design and implementation of these benchmarks. Section 4 breaks down our experiments and analyzes our results. Section 5 details work related to our paper, primarily on that of the benchmarking side. Lastly, Section 6 recaps the paper and presents ideas for future work in this direction.

2 BACKGROUND

In this section, we break down the building blocks leading us to design this benchmark suite.

2.1 The Message Passing Interface (MPI)

The MPI standard is the de-facto method of communication for HPC clusters [10]. While this work does not explicitly focus on MPI, we do use it for the placement of processes to perform IB-Verbs-level operations on both the Host and the DPU. With DPUs becoming another competitor in the accelerator race, MPI libraries may need to become aware of their capabilities and generalize their approaches to account for them. Libraries such as MPICH and MVAPICH [1, 11], among others, are already able to enable a naive approach to this through their multi-program, multi-data (MPMD) executions[1] — that is, at runtime, a user can specify how many processes are used to execute one or more applications in parallel. This not only allows multiple programs to potentially communicate with each other, but it also enables the use of multiple architectures, such as x86 CPUs and the ARM cores on the DPUs used in our experiments. Previous work has been done for specific collectives ([3, 13]) at the MPI level, also utilizing staging among other techniques.

2.2 The NVIDIA Bluefield-2 DPU

The NVIDIA BlueField-2 DPU [4] is a System-on-Chip (SoC) that combines traditional network capabilities with a multi-core ARM CPU (the ARM Cortex A-72), on-chip memory, and acceleration engines for specific operations in other realms such as Artificial Intelligence (AI) and security — the latter components are not within the scope of this paper, so we will not be discussing them. Aside from advanced networking capabilities, the aforementioned works, among others, have presented designs in offloading communication, computation, or even different phases of Deep Learning (DL) onto these DPUs [7].

[1] https://www.intel.com/content/www/us/en/develop/documentation/mpi-developer-guide-linux/top/running-applications/mpmd-launch-mode.html has a good explanation of MPMD executions.

2.3 The IB-Verbs Library

InfiniBand interconnects have powered HPC clusters for over 20 years. The IB-Verbs library, and its standardized version "libibverbs", have been the de facto standard for any Verbs-based transfers and has been a component of the Linux kernel since 2005 [2, 9]. 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. The NVIDIA BlueField-2 DPUs run over IB-Verbs, enabling us to design these benchmarks and perform RDMA transfers between them and their respective hosts.

3 DESIGN

In this section, we will break down the design of our benchmarks.

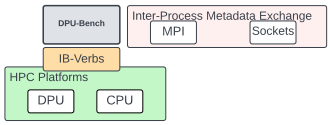


Figure 1: Where DPU-Bench Exists in the Lower Levels of the HPC Ecosystem

3.1 General Approach

Figure 1 shows where DPU-Bench stands in the HPC Ecosystem. While we use MPI to distribute processes, communication at the IB-Verbs level can also be facilitated by sockets. Performing RDMA-based operations at the IB-Verbs levels requires steps both in initializing the runtime environment and when preparing RDMA reads and writes. Listing 2 gives a high-level overview of how each benchmark is designed.

Runtime initialization for each process is similar to the case in which a non-RDMA operation is performed — an IB-Verbs level protection domain is initialized to allow for the creation and utilization of an IB-Verbs level memory region (MR). The major change present is the addition and distribution of each MR's remote key ("rkey") and remote addresses. These can be facilitated by calls to `MPI_Allgather` so that every process ultimately has every other process' rkey and remote buffer address handle.

Listing 2: General Approach to Each Benchmark

```
/* Setup - assume options are
passed in through CLI */
MPI_Init(...);
/*Record-keeping struct*/
global_struct g;
/* Every process makes its own memory region
* which makes the needed rkeys
*/
setup_ib_counters(&g, msg_size,
    num_workers, num_host_procs);
/* Exchange of rkeys between processes */
MPI_Allgather(...);
/* Exchange of RDMA Buffer addresses */
MPI_Allgather(...);
create_sends_and_recvs_pure_host();
run_pure_host(); //gives reference time
create_sends_and_recv_worker_procs();
```

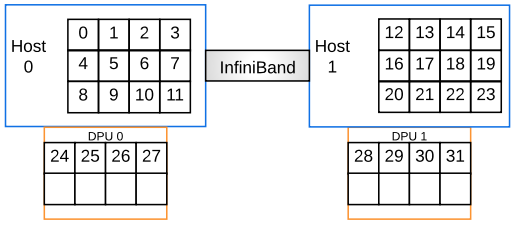


Figure 2: Depiction of a 2-node, 12 PPN, 4 WPN case and how processes are placed. The DPUs act as "separate" hosts alongside the actual CPU-based hosts to allow for offload of operations like communication. The numbers inside each box refer to the MPI rank number.

```
if (proc_is_on_dpu()){
/* IB-level RDMA-operations */
    run_benchmark();
}else{
    perform_compute_on_host(ref_time);
}
MPI_Barrier(MPI_COMM_WORLD);
obtain_max_of_lat_and_comp();
compute_overlap();
cleanup();
MPI_Finalize();
```

During a program's runtime, an RDMA read/write requires three modifications compared to a "standard" `IBV_WR_SEND`: 1) Change of the IB-Verbs level opcode used to one of `IBV_WR_RDMA_READ` or `IBV_WR_RDMA_WRITE`; 2) Utilize both the buffer of data on the host's side *and* the buffer address handle of the corresponding process; 3) the use of the rkey for validation from the process being sent to/received from. In addition, both processes can simply use `ibv_post_send` for their operations instead of needing to also match a call to `ibv_post_recv`.

Every benchmark has the same structure: 1) Perform a "pure-host" — no worker processes involved — version of the benchmark designed to obtain a reference latency. 2) Use that latency to perform a "dummy" computation on the host processes while 3) we offload the communication to the worker processes placed on the DPUs and have them perform all RDMA-based operations.

At runtime, a user of these benchmarks can enable the use of one or more worker processes. Given the need to run on different architectures, these experiments must be run in the MPMD fashion mentioned in Section 2. This is done either explicitly on the command line or by providing a configuration file. For simplicity in setting up MPI runtime commands, hostfiles, configuration files, and process placement, all of the worker processes are associated with the higher-numbered MPI ranks spawned (see Figure 2). We also make the assumption that the end user will request additional processes beyond those needed for the "pure-host" variant of the program. For example, if a pure-host run of an application requires eight processes and the user wishes to utilize four extra workers, then the beginning of their MPI runtime command will start out as `mpirun -np 12` instead of `mpirun -np 8`.

When obtaining the final offload efficiency metric, we take the maximum of the DPU-based communication time and the hosts'

compute time (see Section 4 for extra details) by placing a barrier at the end of the main loop in the benchmark. To analyze how the distribution of work impacts offloading performance, we design two variants of work assignment to the workers: one in which work is distributed cyclically among worker processes, and one in which work is distributed in blocks. Section 4 shows how this ultimately impacts work based on collective design and the number of workers for a given message size. In addition, we have also designed some rudimentary data validation to ensure that each benchmark transferred the staged data as expected.

3.2 Distribution of Workers

Work is distributed statically during runtime. Users must make sure that their hostfiles and configuration files reflect our MPI process placement as shown by Figure 2.

Performing a cyclic distribution of work uses a counter such that workers get assigned to their respective host processes in a round-robin fashion. Performing a block-based distribution of work reverses the order of logic in the cyclic distribution case: the counter for which worker gets assigned work at that instance is placed outside of the loop iterating over the host-based processes, and that an interval is decided as $\frac{num\_host\_procs}{num\_workers}$. We note that a naive block-based work distribution may not use all workers present and/or create a workload imbalance if the number of host processes is not evenly divisible by the number of workers, so we add an extra check to account for the remaining host processes as necessary; this resets the worker-based counter back to the rank value of the first worker process and we iterate over the remaining host processes not yet accounted for.

To ensure that we have a reasonable number of host processes still doing computation and have data for communication, we restrict the number of workers to be less than that of the number of host processes — or at most half of the total number of MPI ranks requested at runtime. This creates a degree of robustness to prevent unrealistic tests from being run such as every process being a worker process with no dummy compute being done.

Given the example in Figure 2, one would run these benchmarks with a command similar to the following: `mpirun -np 32 -hostfile /path/to/hostfile -configfile path/to/configfile /path/to/benchmark`, where the configuration file will contain extra parameters such as number of iterations, number of workers, message size, number of warmup iterations, etc.

3.3 Staged Broadcast

Our staged one-to-all benchmark employs a direct algorithm. When staging begins, every worker will perform an RDMA read from the root process — assumed to be rank 0 — before sending them to one or more host processes. During this, the host processes perform the computation, and as mentioned above, all ranks get synchronized via a barrier at the end of each iteration. See Figure 3 for a high-level view of how the cyclic distribution of work is employed. In a block-based distribution of work, the orange arrows pointing to, for example, Host-1 and Host-2, would spawn out of Worker-1, with the next two spawning out of Worker-2, and so on.

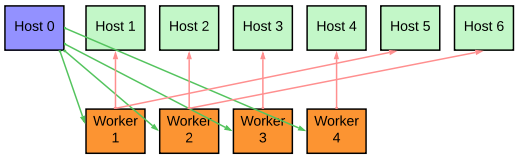


Figure 3: Cyclic Bcast Design. Green arrows represent RDMA reads (performed first), with orange arrows representing RDMA writes (performed second).

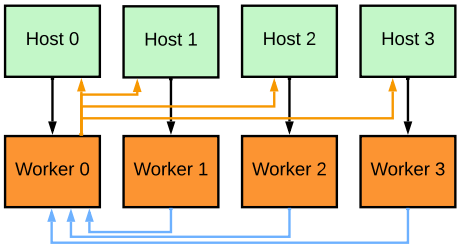


Figure 4: Single-Leader Cyclic Allgather Design. This is a three-step approach with a singular root among the workers (Black lines perform RDMA reads first, followed by blue lines for RDMA writes, followed by orange lines for Worker-to-Host RDMA writes).

3.4 Staged Gather

Our staged all-to-one benchmark also employs a direct algorithm. Essentially, it performs the same operations that the one-to-all benchmark performs, but in reverse: the worker processes will read from each of the hosts — in a round-robin or block-based fashion — before writing to appropriate offsets in the root's memory.

3.5 Staged Allgather/"Non-Personalized Alltoall"

3.5.1 ***Version 1: Single-Leader Approach:*** Performing an Allgather — or a non-personalized all-to-all communication — with staging is trickier than a one-to-all or all-to-one communication pattern. Each process has data at the base offset of its individual buffer, with the gathered result being distributed across all processes. In the case of multiple workers, we buffer all data to a single "leader" of the worker processes, which in turn performs a broadcast to the remaining host processes.

Figure 4 gives a graphic representation of the cyclic work assignment approach. The black line from one host to one worker is "Step one": every worker initiates an RDMA read. The blue lines are step two — a single "leader" is selected to gather the data from each worker — with the orange line acting as step "three" of returning the data to the host processes involved. In particular, the block-based work distribution requires a wrap-around as performing RDMA writes among the other worker processes required more work than just writing back to a single host process. Section 4 will show how

this ultimately does not scale well, nor does this kind of pattern exhibit good offload efficiency.

3.5.2 ***Version 2: An "All-In" Approach:*** In a multi-leader approach, every worker process that gets added to the DPU side becomes involved in transmitting their messages. Instead of transmitting their data to that of the single "leader" as mentioned in Version 1, each worker is aware of what host processes have given it work and which ones have not such that when it comes time to distribute the data back to the hosts after performing communication, no worker will overwrite another's already-written data. This mainly affects the implications of using multiple workers. Utilizing a single worker remains unchanged here. To better visualize this, imagine Figure 4, except with the removal of blue lines and orange lines extending from each of the workers to each of the hosts.

3.6 Offload Efficiency Calculation

The central part of this benchmark is how efficiently a communication pattern can be offloaded to the DPU (as opposed to traditionally calculating overlap). "Offload Efficiency" is calculated as $\frac{reference\_time}{max(pure\_comm, compute\_time)} * 100$, where "Pure_Comm" refers to the communication time of the communication excuted over the DPUs and "compute_time" refers to the time taken to perform a dummy compute as shown by Listing 2.

4 EXPERIMENTS AND EVALUATION

In this section, we will describe our experimental setup, the experiments run, and insights gained from our results.

4.1 Experimental Setup

Our experimental testbed consists of 32 Intel Dual-Socket E5-2697A V4 CPUs @ 2.60 GHz (16 cores/socket), and each node is equipped with NVIDIA ConnectX-6 HDR100 100Gb/s InfiniBand/VPI adapters and one NVIDIA BlueField-2 SoC, HDR100 100Gb/s InfiniBand/VPI adapter.

4.2 Experiments

We demonstrate our benchmarks at 8 nodes and 8 processes per node (PPN) on the host, with an increasing number of workers placed onto the DPUs. At a smaller total number of workers, we assign one per worker per node (WPN) on each DPU until one worker exists on every DPU requested, though as we increase the number of workers beyond the number of host nodes we work with, we also see the effects of 2, 4, 6, and 8 WPN.

We test medium-to-large message sizes: 256KB to 4MB. Smaller messages such as those below 16KB transfer rather quickly and thus do not incur massive overhead, leading to "low" offload efficiency. The numbers we show of each message size within each benchmark are the average of four back-to-back executions, each of which has an internal loop running for one thousand iterations. In addition, we show the effects of work being assigned to workers in both cyclic and block-based fashions. During allocations on said HPC cluster, we assume that the number of host nodes present is equivalent to the number of DPUs available. In our experiments, we show results for one-to-all (broadcast), all-to-one (gather), and non-personalized all-to-all (allgather) communication patterns.

4.3 Experimental Results: 1 WPN

Figure 5a shows how a smaller number of workers can achieve reasonable offload efficiency up until a "sweet spot" is approached. For all of our results, We empirically evaluated that this is when the number of worker processes equals the number of host **nodes** present in a given allocation — or otherwise 1 worker per node (WPN). With one worker being offloaded all the communication, we see that it clearly is not close to the "sweet spot" to maximize overlap with respect to all message sizes. Similarly, Figure 5b shows a larger difference when using two or more workers compared to one worker in the all-to-one (gather) operation. Part of this lies in increased amounts of RDMA reads and writes issued by each worker to their respective host process/processes. Figures 7a and 7b show similar trends — asymptotically approaching a maximum offload efficiency as we approach the 1 WPN threshold.

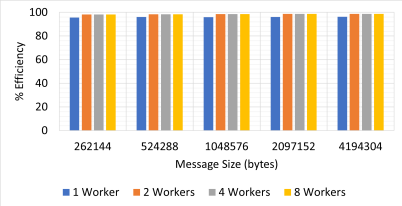
We recognize that a single-root-worker approach to implementing allgather is a naive and inefficient approach, hence the substantially smaller overlap efficiencies shown in Figures 5c and 7c, as well as later on in Figures 8c and 9c. Utilizing OMB's and IMB's calculations for overlap does not necessarily help here, as they will always result in 100% overlap between communication and computation. Figure 6 showcases a more intelligent and efficient design to showcase 1) how DPUs can improve offload efficiency in a more complex scheme and 2) the effects of offloading different communication patterns for the same collective operation to them. As we approach placing one worker on every DPU (1 WPN), we obtain a sweet spot for maximizing offload efficiency.

4.4 Experimental Results: 2, 4, 6, and 8 WPN

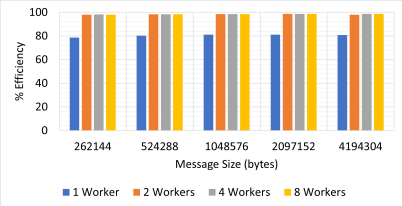
Increasing the total number of workers and workers per node shows different trends for each work assignment. Both Figures 8 and 9 show that having both 2 and 4 workers per node have only mildly decreasing offload efficiency results compared to the 8-worker, 1-WPN case — all revolving around 98.6 and 98.7%. However, the work distribution — block versus cyclic — appears to have a substantial impact at 6 WPN.

At 6-WPN, the block-work-assignment variant of broadcast/one-to-all takes a hit at smaller message sizes on account of the extra RDMA writes from more workers. This and the increased degradation at 8 WPN come up given that the BlueField-2 DPUs have one memory controller and a substantially smaller number of cores compared to host CPUs these days. The same analysis can be applied to the 8-WPN case for the block-based work assignment for the all-to-one/gather benchmark.

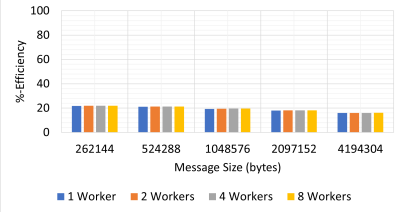
Block-based work assignment appears to give substantial degradations as the number of workers increases to 8 WPN, despite higher/comparable overlaps being achieved with larger message sizes to that of the cyclic work distribution. Figure 9a shows this in particular as 6 WPN only reaches maximal overlap at 4MB message sizes. Part of this is thanks to the architectural limitations of the BlueField-2 DPU — a single memory controller for its 8 cores eventually has to deal with the contention of resource requests as each core requires memory slots to create and issue RDMA reads and/or writes. Similarly, 6 WPN provides a mix of uneven work distribution compared to 8 WPN, and the degradations that come with it also show in the block-based work assignment. This degradation at



(a) Cyclic Bcast Work Assignment, 1 WPN, up to 8 workers total



(b) Cyclic Gather Work Assignment, 1 WPN, up to 8 workers total



(c) Cyclic AllGather (Single Leader) Work Assignment, 1 WPN, up to 8 workers total

Figure 5: Offload efficiency of 1-WPN Cyclic assignment of work for Gather, Broadcast, and Single-Leader Allgather (8 nodes, 8 PPN)

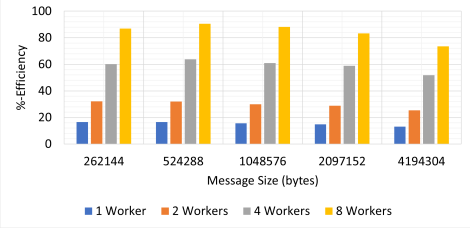
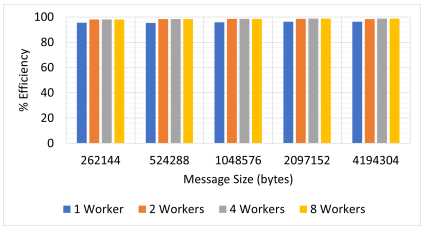


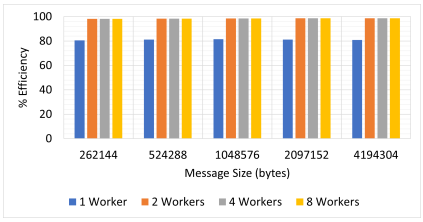
Figure 6: Results of a more intelligent offloading of the Direct, Cyclic-Work-Assignment Allgather up to 1 WPN (8 nodes, 8 PPN)

6 WPN does NOT occur within the cyclic case, as shown by each of the subfigures in Figure 8.

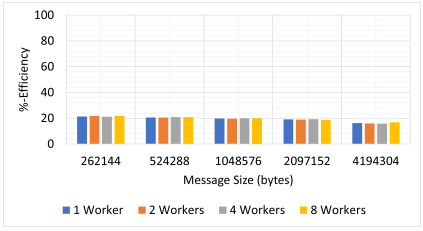
Figure 10 examines what happens when DPUs ultimately receive less and less work to do after hitting the empirically observed sweet spot. While each DPU is ultimately performing less operations, the number of processes communicating through the HCA increases. The spike at 8 WPN in which the 8 DPUs used in our experiments are fully subscribed is uncharacteristic compared to previous elements. One running hypothesis is that a work threshold has been



(a) Block Bcast Work Assignment, 1 WPN, up to 8 workers total



(b) Block Gather Work Assignment, 1 WPN, up to 8 workers total



(c) Block AllGather (Single Leader) Work Assignment, 1 WPN, up to 8 workers total

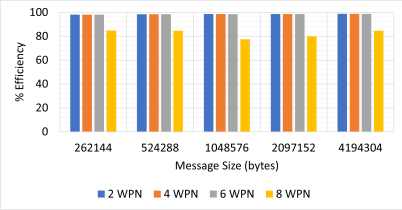
Figure 7: Offload efficiency of 1-WPN Block assignment of work for Gather, Broadcast, and Single-Leader Allgather (8 nodes, 8 PPN)

reached in which the distribution allows for greater offload efficiency/improved runtime on the DPU side, though further profiling is needed on this side.

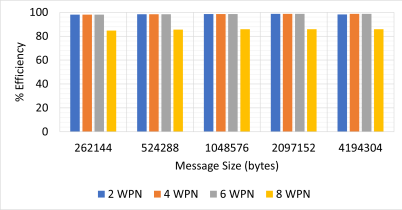
Figure 11 shows general runtime trends of cyclic work assignment on the three benchmarks designed thus far for DPU-Bench. These compare the pure-host reference time with that of the pure communication time that ultimately becomes the dominating factor in determining both traditional overlap measurements and measurements of our offload efficiency. As mentioned above, the overhead of all of the workers sending to a "root" worker in our allgather implementation leads to substantial overhead in the pure communication times shown in Figure 11c and the lowered offload efficiency previously shown.

4.5 Profiling Performance On the DPU

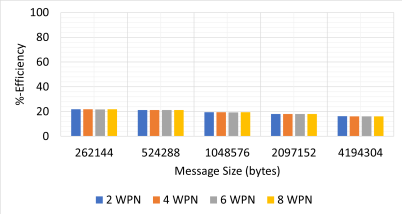
Profiling the NVIDIA BlueField-2 is non-trivial. Compiling and building PAPI [15] on the BlueField-2 is possible, though there are no DPU-based performance counters available. Other profiling tools may be able to analyze DPUs at a sample-based level such as TAU [14], though this still requires building two separate versions



(a) Cyclic Bcast Work Assignment, 2/4/6/8 WPN

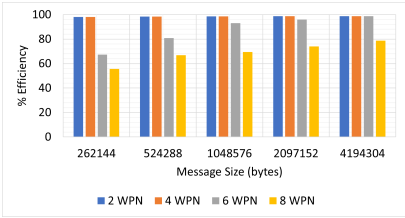


(b) Cyclic Gather Work Assignment, 2/4/6/8 WPN

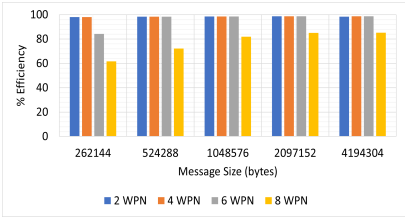


(c) Cyclic AllGather (Single Leader) Work Assignment, 2/4/6/8 WPN

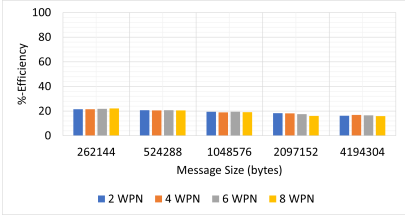
Figure 8: Offload efficiency of 2/4/6/8-WPN Cyclic assignment of work for Gather, Broadcast, and Single-Leader Allgather (8 Nodes, 8 PPN)



(a) Block Bcast Work Assignment, 2/4/6/8 WPN



(b) Block Gather Work Assignment, 2/4/6/8 WPN



(c) Block Allgather (Single Leader) Work Assignment, 2/4/6/8 WPN

Figure 9: Offload efficiency of 2/4/6/8-WPN Block assignment of work for Gather, Broadcast, and Single-Leader Allgather (8 Nodes, 8 PPN)

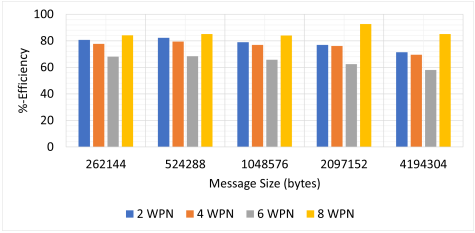


Figure 10: Results of a more intelligent offloading of the Direct, Cyclic-Work-Assignment Allgather from 2 to 8 WPN (8 nodes, 8 PPN)

of the software to run on both CPU and DPU and ensuring that it can work in conjunction with calls to, e.g., an MPI library.

5 RELATED WORK

In this section, we detail related works to both designing HPC microbenchmarks as well as microbenchmarks for DPUs.

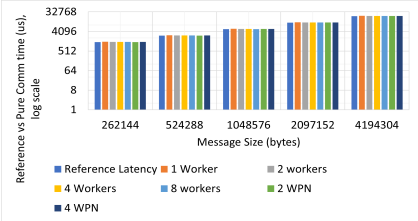
5.1 Current State of the Art Micro-benchmark suites

Our group has developed the OSU Microbenchmark suite (OMB) [12], which stands as one of the standards for MPI-based HPC microbenchmarks. Along with the MVAPICH library, it has been downloaded over 1.6 million times as of February 2023 [11] and contains applications to evaluate both MPI and PGAS libraries for CPUs and GPUs.

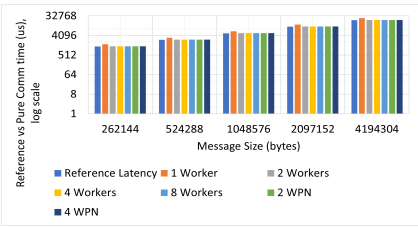
The Intel MPI benchmarks (IMB) [6] are analogous to that of OMB, providing an equally vast array of benchmarks over which users can test and benchmark MPI libraries across blocking, non-blocking, point-to-point, and one-sided communication, similar to that of OMB.

The OpenHPCA benchmark suite [5] employs, along with OMB and the Sandia Microbenchmarks, a time-driven model to find the maximum amount of work that can be injected while executing a given nonblocking collective (NBC) in a given problem size and scale — for reference, most other benchmarks utilize what is known as a "Data-Driven Model."

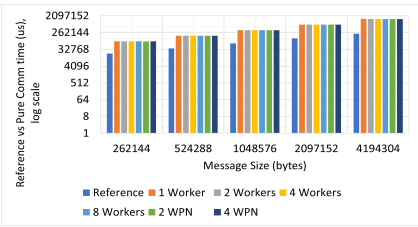
In all of the above cases, hostfiles and MPMD configuration files can be used to arbitrarily place processes in virtually any given configuration. However, this will not necessarily always give an optimal overlap measurement for offloading communication to SmartNICs, as eventually every process in e.g., an OMB nonblocking



(a) Cyclic Bcast Runtimes up to 4 WPN



(b) Cyclic Gather Runtimes up to 4 WPN



(c) Cyclic Allgather (Single Leader) Runtimes up to 4 WPN

Figure 11: Runtime comparison of Cyclic-Based Work Assignment on One-to-All (Broadcast), All-to-One (Cyclic), and Non-Personalized All-to-All (Single-Leader Allgather) at 8 Nodes, 8 PPN, up to 4 WPN (32 total workers)

collective eventually gets called to do a dummy compute, which would defeat the purpose of offloading collective communication and having only host processes perform their computation.

Most of the developments of our benchmarks — adjusting problem sizes, number of iterations for which the benchmark runs, warm-up iterations, and basic data validation/verification — came from ideas presented in these benchmarks to ensure that they can be equally as useful to HPC researchers and application developers.

6 CONCLUSION AND FUTURE WORK

In this paper, we proposed DPU-Bench: A micro-benchmark suite designed to actively measure the offload efficiency of offloading communication from various collectives onto a SmartNIC such as the BlueField-2 DPU. We present designs regarding one-to-all, all-to-one, and non-personalized all-to-all transfers and measure them at a medium/large scale, using up to 64 workers on eight NVIDIA BlueField-2 DPUs. We discuss the possibilities of how work is distributed among worker processes — block-based versus round-robin — and how each approach can benefit or potentially degrade performance at different numbers of workers and message sizes. We have shown that it is possible to relatively quickly, and empirically, find a number of workers based on the number of nodes and host processes involved that can give maximum offload efficiency.

We plan to perform the following tasks on these benchmarks: 1) Investigate the performance of other non-trivial flat and hierarchical algorithms (recursive doubling, k-nomial, etc.), 2) ultimately integrate into the OMB repertoire, and 3) develop them to send information relevant to routines like staging, pure communication time, etc., to monitoring tools such as OSU INAM[8].

ACKNOWLEDGMENTS

This research is done as a part of contract #19537 from Los Alamos National Laboratory/US Department of Defense to The Ohio State University and is partially supported by National Science Foundation grants #2007991 and #2018627. We would also like to thank the HPC-AI Advisory Council for allowing us access to their resources so that we could perform our experiments.

REFERENCES

[1] Argonne National Laboratory 2022. *MPICH.* https://www.mpich.org/.
[2] Dotan Barak. 2014. Verbs Programming Tutorial. https://www.cs.mtsu.edu/~waderholdt/6430/papers/ibverbs.pdf
[3] Mohammadreza Bayatpour, Nick Sarkauskas, Hari Subramoni, Jahanzeb Maqbool Hashmi, and Dhabaleswar K. Panda. 2021. BluesMPI: Efficient MPI Non-blocking Alltoall Offloading Designs on Modern BlueField Smart NICs. In *High Performance Computing*, Bradford L. Chamberlain, Ana-Lucia Varbanescu, Hatem Ltaief, and Piotr Luszczek (Eds.). Springer International Publishing, Cham, 18–37.
[4] Idan Burstein. 2021. Nvidia Data Center Processing Unit (DPU) Architecture. In *2021 IEEE Hot Chips 33 Symposium (HCS).* 1–20. https://doi.org/10.1109/HCS52781.2021.9567066
[5] High Performance Compute Availability Group 2022. *OpenHPCA Benchmark Suite.* https://github.com/openucx/openhpca.
[6] Intel 2022. *Intel MPI Benchmarks.* https://www.intel.com/content/www/us/en/developer/articles/technical/intel-mpi-benchmarks.html.
[7] Arpan Jain, Nawras Alnaasan, Aamir Shafi, Hari Subramoni, and Dhabaleswar K Panda. 2021. Accelerating CPU-based Distributed DNN Training on Modern HPC Clusters using BlueField-2 DPUs. In *2021 IEEE Symposium on High-Performance Interconnects (HOTI).* 17–24. https://doi.org/10.1109/HOTI52880.2021.00017
[8] Pouya Kousha, Kamal Raj Sankarapandian Dayala Ganesh Ram, Mansa Kedia, Hari Subramoni, Arpan Jain, Aamir Shafi, Dhabaleswar Panda, Trey Dockendorf, Heechang Na, and Karen Tomko. 2021. INAM: Cross-Stack Profiling and Analysis of Communication in MPI-Based Applications. In *Practice and Experience in Advanced Research Computing* (Boston, MA, USA) *(PEARC '21).* Association for Computing Machinery, New York, NY, USA, Article 14, 11 pages. https://doi.org/10.1145/3437359.3465582
[9] Patrick MacArthur, Qian Liu, Robert D. Russell, Fabrice Mizero, Malathi Veeraraghavan, and John M. Dennis. 2017. An Integrated Tutorial on InfiniBand, Verbs, and MPI. *IEEE Communications Surveys & Tutorials* 19, 4 (2017), 2894–2926. https://doi.org/10.1109/COMST.2017.2746083
[10] Message Passing Interface Forum. 2021. *MPI: A Message-Passing Interface Standard Version 4.0.* https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf
[11] Network-Based Computing Laboratory 2022. *MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE.* http://mvapich.cse.ohio-state.edu/.
[12] Network-Based Computing Laboratory 2022. *OSU Microbenchmarks.* http://mvapich.cse.ohio-state.edu/benchmarks/.
[13] Nick Sarkauskas, Mohammadreza Bayatpour, Tu Tran, Bharath Ramesh, Hari Subramoni, and Dhabaleswar K. Panda. 2021. Large-Message Nonblocking MPI_Iallgather and MPI Ibcast Offload via BlueField-2 DPU. In *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC).* 388–393. https://doi.org/10.1109/HiPC53243.2021.00054
[14] Sameer S. Shende and Allen D. Malony. 2006. The Tau Parallel Performance System. *Int. J. High Perform. Comput. Appl.* 20, 2 (may 2006), 287–311. https://doi.org/10.1177/1094342006064482
[15] Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. 2010. Collecting Performance Data with PAPI-C. In *Tools for High Performance Computing 2009*, Matthias S. Müller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 157–173. https://doi.org/10.1007/978-3-642-11261-4_11
[16] Top500 2022. *Top500.* https://www.top500.org/lists/top500/2022/11/.