# Optimizing Application Performance With BlueField: Accelerating Large-Message Blocking and Nonblocking Collective Operations

Richard Graham, George Bosilca, Yong Qin,
Bradley Settlemyer, Gilad Shainer, Craig Stunkel,
Geoffroy Vallee, Brody Williams, and
Gerardo Cisneros-Stoianowski
*NVIDIA*
Santa Clara, CA, USA
{richgraham,gbosilca,yqin,bsettlemyer,shainer,
cstunkel,geoffroy,brodyw,gcisneross}@nvidia.com

Sebastian Ohlmann and Markus Rampp

*Max Planck Computing and Data Facility*
Garching, Germany

{sebastian.ohlmann,markus.rampp}@mpcdf.mpg.de

***Abstract*—With the end of Dennard scaling, specializing and distributing compute engines throughout the system is a promising technique to improve applications performance. For example, NVIDIA's BlueField Data Processing Unit (DPU) integrates programmable processing elements within the network and offers specialized network processing capabilities. These capabilities enable communication via offloads onto DPUs and present new application opportunities for offloading nonblocking or complex communication patterns such as collective communication operations. This paper discusses the lessons learned enabling DPU-based acceleration for collective communication algorithms by describing the impact of such offloaded collective operations on two applications: Octopus and P3DFFT++. We present new algorithms for the nonblocking MPI_Ialltoallv and blocking MPI_Allgatherv collective operations that leverage DPU offloading, which are used by the above applications, and evaluate them. Our experiments show a performance improvement in the range of 14% to 49% for P3DFFT++ and 17% for Octopus, even though the performance of those collectives in well-balanced OSU latency benchmarks shows comparable performance to well-optimized host-based implementations of these collectives. This demonstrates that taking into account load imbalance in communication algorithms can help improve application performance where such imbalance is common and large in magnitude.**

## I. Introduction

In parallel computing, one highly-desired objective is to fully overlap computation and communication, thus hiding the cost of expensive network updates. While progress has been made over the years, the goal remains elusive, with a lack of sufficient hardware and software capabilities for fully asynchronous support, short of dedicating a nontrivial amount of computational resources. This paper utilizes a programmable network device that may be used for effective offloading a new set of communication library capabilities. In particular, this paper describes the offload of two MPI collective functions commonly used by HPC simulation codes and Machine Learning applications. New algorithms were devised to reduce the effects of application load imbalance and to deal with application scenarios that block widely-used algorithms from progressing under such circumstances, thereby improving overall application performance.

The end of Dennard scaling [1] and the slowing of improvements in processor speed have led to an increase in compute specialization to improve performance. General Purpose Graphical Processing Units (GPGPUs), Tensor Processing Units (TPUs), and Field-Programmable Gate Arrays (FPGAs) are well-known examples of such specialization.

In recent years, a class of specialization has emerged that couples network interface controllers (NICs) and computational capabilities in devices generally known as SmartNICs [2], [3], [4]. SmartNICs are enhanced network interfaces that provide a processing pipeline for network packets arriving at the interface. The processing capabilities may exist solely within the NIC's packet processing pipeline, alongside the packet processing pipeline, or both. A SmartNIC may support a set of predefined, specialized hardware-based processing programs, it may support more general programmable processing capabilities, or may provide a mix of each. SmartNICs with more general capabilities can be used for offloading network management, complex network messaging, and even computational work that is typically done on a host. NVIDIA's BlueField Data Processing Units (DPUs) [5] are an example of this trend of increasing capabilities in SmartNICs.

Effective utilization of DPUs within emerging architectures is predicated on the existence of a simple, performant interface. To do this, we use the offloading software Multi-tenant Intelligent Modular Offload Service Architecture (MIMOSA) for the research carried out in this work. In addition to providing run-time support for middleware, this modular software provides implementations of new topology and memory access primitives required to offload MPI collective communication work using advanced DPU capabilities.

Previous collective communication offload research has concentrated principally on non-blocking collectives [6], [7] wherein available computation work can proceed on the host in parallel with the communication work that was offloaded to other engines. Once the offload engines are given the information to initiate the collective work, the original host engines are not involved except to check the completion of

the communication work.

However, in this paper, we show that blocking collectives, in addition to nonblocking collectives, can also benefit significantly from offloading processing steps within a collective. Blocking collective algorithms are in much wider use in the HPC application community and as such are an important class of algorithms to optimize. Achieving performance improvements when blocking collectives are in use requires thinking broadly about how offload engines and subsystems can cooperate with host engines and with each other to achieve low latency communication and/or high bandwidths. Although algorithm latency considerations are important when nonblocking collectives are utilized, the primary focus for optimization is to maximize communication-computation overlap in order to enhance user-level algorithm throughput. As a result, we developed a larger set of offload design principles to guide our decisions.

Some of the key features of the BlueField class of devices include an NVIDIA Connect-X network processing core (the same one used by the host), user-programmable ARM CPUs, and access to the network or the host through a PCIe bus. The ARM cores are not server-grade and their number is typically much smaller than then number of host-side computational engines, and their memory bandwidth is also smaller than that of the server compute complex. The BlueField compute cores may be used independently of the state of the host-side compute engines. As such, this class of devices is very well suited for lightweight asynchronous computational tasks. In the context of collective communication algorithms, this class of devices is particularly well suited for implementing large-data loosely coupled operations, such as blocking and nonblocking versions of alltoall, alltoallv, allgather and allgatherv, which are used by many HPC and emerging AI applications.

This paper defines and justifies these principles that underlie state-of-the-art DPU-based communication frameworks, and describes the design and performance of DPU-accelerated MPI_Allgatherv and MPI_Ialltoallv collective communication algorithms implemented using MIMOSA. These collectives are used by the applications studied here. We make the following contributions:

1) A description and justification of the tenets used to build a communication framework accelerated with DPUs.
2) Descriptions of offloaded collective communication algorithms that are able to simultaneously exploit the strengths of DPUs and host processors.
3) Performance analysis of these offloaded collective communication algorithms on standard benchmarks and scientific applications.

The remainder of this paper is organized as follows: Section II describes the relevant background on DPUs and describes the specialized capabilities of the NVIDIA BlueField DPU. Section III presents the offloaded algorithms and provides a brief description of the MIMOSA communication library, service processes (SPs), and changes made to the UCX library in order to support implementation of offloaded algorithms. Section IV presents a detailed performance analysis of offloaded collectives and applications, Section V overviews related work. Section VI discusses our overall findings and provides an expanded set of algorithm offloading and library design principles based on this work.

## II. Background

### A. From SmartNICs to DPUs

Early SmartNICs [8], [9], [10] provided a limited set of in-pipeline processing capabilities for delivering packets to user space [11], traffic shaping [12], or tailored offloads for TCP packet processing [13]. Emerging SmartNICs provide much richer hardware capabilities such as regular expression matching [14], compression [15], encryption [16], and SDN controllers [17]. Current commercial SmartNIC products exist at multiple points along this design space and range from generally programmable in-pipeline processing capabilities [18], [19], SmartNICs that provide in-pipeline processing with general programs and predefined hardware functions [20], and the most capable SmartNICs providing a complete feature set that enables packet processing both alongside and within the packet pipeline using both predefined programs and programmable processors [5], [21]. Although there is no clear delineation where SmartNICs end and DPUs begin, the industry trend appears to be for SmartNICs that are generally programmable and that provide high levels of in-pipeline processing performance to take on the DPU moniker.

### B. BlueField Data Processing Unit

The NVIDIA BlueField family of DPUs [5] are systems-on-a-chip (SOCs). They include an InfiniBand Connect-X Host Channel Adaptor (HCA) core, programmable ARM cores, which run the Linux operating system, memory, domain-specific accelerators, DMA engines for transferring data between the host and the DPU, and a PCIe switch that allows DPU memory to be accessed from the network or the attached host. The DPUs are connected to the host via a PCIe bus. The host-side computational engines and the DPU ARM cores both access the network through the Connect-X core. Data transfers between the host and DPU memory may be realized using either InfiniBand capabilities or DPU DMA engines.

The DOCA software stack provides system-level application support for BlueField DPUs [22] including standard programming environment tools that can be used to compile the user-level binaries that will run on the DPU itself. Since the DPU functions independently of the host, such binaries can be used to run code asynchronously with respect to host-side applications. As such, the DPU provides additional resources to host-resident applications and can be used to enable the realization of a truly asynchronous communication progress engine.

The BlueField-3 platform is the device utilized in this paper. The networking element is the ConnectX-7 400 Gb/s core, which supports 32 lanes of PCIe Gen 5.0 between the host and the DPU. It is configurable for either dual ports of NDR / NDR200 or a single port of NDR and a PCIe switch bifurcation with 8 downstream ports. The device features up to

16 Armv8 A78 cores (64-bit) with an 8 MB L2 cache per two cores and a 16 MB L3 cache. The chipset supports two DDR5 5600 MT/s DRAM5 controllers and up to 64 GB of DRAM5 memory with 8 MB of L2 cache and 16 MB of L3 cache. The maximum ARM CPU memory bandwidth is 80 GB/s, depending on how the device is configured.

### C. Alias Memory Keys

The InfiniBand specification defines an object called a memory key (MKey) that is used to describe an InfiniBand accessible region of memory and access rights to this memory. For example, the MKey includes a reference to a virtual-to-physical translation table. These MKeys are associated with a specific instance of a virtual Host Channel Adapter (HCA). By default, access to such memory regions is allowed only through objects, such as queue pairs, that belong to the same instance of the virtual HCA.

Recently, a secure protocol has been added to allow objects from one instance of a virtual HCA to access and use MKeys in another virtual HCA when both instances are mapped back to the same physical HCA. The result is an alias-Mkey with InfiniBand access to the original memory region. As host-side memory and DPU memory access the network via the same HCA, an alias-Mkey memory key can be created that provides the DPU InfiniBand access to host-resident memory. This allows the DPU to initiate network activity, such as posting a send request, on behalf of the host and is a key capability for supporting efficient offload of network traffic management from the host to the DPU. Alias-Mkeys are critical to the DPU offload work conducted in this study because they avoid the explicit transfer of data from the host to DPU, or DPU to host, as would typically be required for an RDMA operation.

## III. BlueField Accelerated Collective Support

The BlueField accelerated collective algorithms described in this paper generally include algorithms whose implementation is split on a per MPI process basis between an MPI process resident on the host and an associated process on the DPU. Such implementations leverage the newly developed MIMOSA infrastructure and the UCX point-to-point communication library, and are implemented via extensions to the Unified Collective Communication Library (UCC) [23]. Both blocking and nonblocking algorithms are supported. The overall goal of the accelerated collective support is to improve application throughput. This results in different design considerations based upon the accelerated collective algorithm in question. More specifically, for blocking algorithms the target is to offer the best operation latency possible, while for nonblocking algorithms the target is to maximize communication-computation overlap opportunities.

With the above in mind, the primary principles used to guide the design of new BlueField accelerated collective algorithms in this work include:

- The DPU is truly asynchronous with respect to the host. As a result, the average latency of a response within a DPU-based algorithm is much lower than a host-based algorithm wherein compute resources are shared with the application.
- DPU compute resources are not as powerful as the server side resources. This limits the amount of computation that can be done without harming overall performance.
- The number of DPU-side general-purpose computational elements is a small fraction of the host-side resources. Therefore, a single DPU-side compute resource is generally required to service several server-side entities, which serializes processing on the DPU.
- Algorithmic coordination between the host and DPU resources adds control path latency.
- The BlueField DPU support for alias-Mkeys allows for memory that is server-side resident to be accessed without involving the host-side computational capabilities.
- The BlueField memory bandwidth is additive with respect to server-side memory bandwidth and may be used to supplement the available server-side bandwidth for operations such as data packing.

This section briefly describes the concept of Service Processes (SPs), the MIMOSA library, and modifications made to the UCX and UCC libraries in order to support alias-Mkeys and the BlueField offloaded MPI_Ialltoallv and MPI_Allgatherv algorithms, respectively.

### A. Service Processes (SP)

Service processes are daemon processes running on the DPU that execute the BlueField portion of the offloaded collective algorithms. Each SP participates in establishing connections with host-resident MPI processes taking part in the job as well as with other SPs as dictated by implementation requirements. During communication establishment, service processes exchange endpoint information as well as topology metadata for host application processes. This topology metadata includes MPI communicator and rank information along with runtime residency details such as socket and core binding specifics for the host-resident MPI processes. This data is used to establish the needed communication channels between SPs and local and remote host MPI processes. After completion of the bootstrapping process, the daemons move to the progression cycle and wait for control messages that direct the algorithmic flow for any ongoing collective operations. Figure 1 shows how differing numbers of host processes and service processes can be associated with each other and the communication pathways that control messages use.

### B. MIMOSA

This subsection provides brief descriptions of key MIMOSA components used for enabling BlueField accelerated collectives. In order to facilitate the implementation of DPU offloaded algorithms, MIMOSA distinguishes between *control* and *data* paths. The control path is active-message based and enables the exchange of messages and data that is inherently required for the implementation of offloaded algorithms, i.e. coordination messages exchanged between the host-resident MPI ranks and the SPs as well as between SPs. The exchange

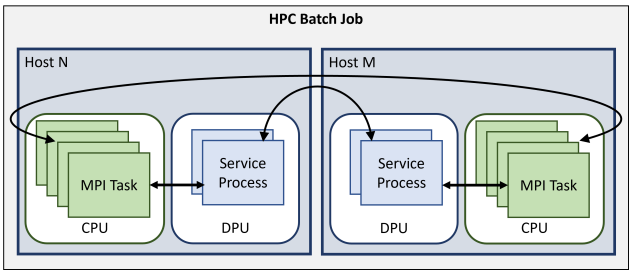



Figure 1: Process layout for a parallel MPI job with MIMOSA service processes. Arrows between processes show the communication paths available for algorithm control-plane messages. Data plane messages are sent using RDMA transfers as described in the text.

of application data within the offloaded algorithms presented in this paper, typically large in size, is handled by the data path. For performance reasons, these data path exchanges most often rely on RDMA operations built upon alias-Mkeys and, therefore, do not utilize MIMOSA directly. Control path messages, on the other hand, are usually accomplished using MIMOSA events that are designed to be easy to use and optimized for such exchanges. Information necessary for performing alias-Mkey enabled data movement as described above, for example, is exchanged using MIMOSA events.

MIMOSA provides the following capabilities for DPU-enabled systems:

- *Offload engines* that implement offload services and create and manage the progress of active offloaded operations.
- *Offload service bootstrapping* to establish and tear down a foothold from applications to the offload services.
- *Group and topology management* in support of distributed objects like MPI communicators, which enables load balancing between entities involved in the offloading of a collective operation by exposing the various mappings between MPI communicators, ranks, and SPs.
- *Endpoint cache* supporting high-performance communication.
- *Execution context* used to manage a given offloaded operation such as an MPI collective, on behalf of a single user-level communication context like an MPI process.
- *Notifications and events* that encapsulate information, resources, and tasks associated with progressing a communication operation as well as completion notifications. For example, when an MPI_Allgatherv operation is invoked by an application process, the communication library uses a notification to inform the DPU-side proxy that the operation has started and passes the relevant metadata to the corresponding DPU context.

### C. UCX Modifications

In this work, UCX [24] is used as the low-level communication library to create and post work requests to the network. Host-only UCX users create one memory key for each region of memory used for network communication, but DPU-initiated network operations involving host-resident memory regions requires setting up an alias-Mkey. The workflow to create these keys involves the following steps:

- Create an Mkey (target Mkey) for the relevant memory region on one virtual HCA (target virtual HCA).
- Modify target Mkey to allow it to be accessible from another virtual HCA (initiator virtual HCA).
- Transfer target Mkey from target virtual HCA to initiator virtual HCA.
- Create the alias Mkey from target Mkey on the initiator virtual HCA.

An optimization that eliminates the need to create the alias-Mkey for each region of memory being registered is to create an empty predefined User-Mode Memory Registration (UMR) MKey (MKeyA) and create its corresponding alias-Mkey on the DPU (MKeyA-DPU), as described above, and maintain this pairing for the duration of the given MKey. When a new memory region needs to be used, a regular host-side Mkey (MkeyB) is created. This memory key is associated with MKeyA by posting an InfiniBand send request filling MKeyA. At this stage MKeyA and MKeyA-DPU are both ready for use in initiating network activity for host-resident memory regions described by MKeyB. This requires only local host-side setup once after MKeyA-DPU is initially set up.

### D. BlueField Optimized Collectives

Modifying an MPI collective to leverage DPU offloading motivates a careful *algorithm distribution* between host and DPU. Algorithm offloading can be done in many ways, from offloading the entire algorithm, wherein the data is copied in and out of the DPU and the algorithm is executed almost as-is on the DPU, to a redesign of the algorithm for a finer-grain offload where the algorithm specifies pieces running on the host and DPU in order to optimize for the respective technologies. These options impact the balance between leveraged compute capabilities, coordination, and data transferred between the hosts and DPU. The algorithm distribution should therefore be carefully considered since these choices have a clear impact on performance. Moreover, asynchronous tasks should not be run on resources shared with other entities, such as on the host-side where the application is run, in order to ensure optimized performance. With these considerations in mind, we design two BlueField-accelerated collective algorithms.

The algorithms we accelerate in this work are designed to optimize operations that involve large data transfers where the goal is to orchestrate the global data transfers in a manner that reduces the likelihood of encountering network congestion. A basic building block for such algorithms is the use of some sort of ring algorithm with the specifics dependent on the collective operation being implemented. For our accelerated algorithms, a common feature across implementations is running the per-process ring-management on the DPU in order to avoid the situation where the orchestration described above may compete with the application computation for resources. This optimization allows the collective operations to progress

independently of the state of the running application. Further, it also enables application computation to continue without the need to stall in order to progress ongoing collective operations.

*1)* ***MPI_Ialltoallv BlueField Algorithm****:* Within an MPI_Ialltoallv collective operation, each MPI process sends unique data to every other MPI process in the communicator wherein each data block size is unique. Large-data optimized algorithms for this collective, both blocking and nonblocking, typically use a ring-like algorithm to schedule data transfers as they are readied as described above.

Our offloaded algorithm builds upon this approach by using a service process (SP) to progress the ring-like algorithm without host-side assistance. At a high level, the algorithm is performed as follows:

- A host process notifies its associated SP on the DPU that the collective has started. It passes metadata to the SP that includes items such as a unique group (MPI communicator) context, group rank, collective ID, virtual addresses and memory keys for the source and destination buffers, data type, and the counts arrays.
- The SP progresses a ring-like algorithm to distribute data to each of the other MPI ranks in the communicator. MIMOSA notifications are used to notify the destination MPI process's corresponding SP that data is ready to be read. Each notification includes the addressing information needed by the SP to read user data from the source process.
- When the notification arrives at the remote SP, the SP enqueues the metadata for in-turn processing. If the associated receive-side MPI process has not entered the iAlltoallv and given its metadata to this SP, the SP will wait to handle this request after it receives that metadata.
- The receive-side SP performs a UCX RDMA read operation, coupled with the alias-Mkey capabilities of the NVIDIA DPU, to transfer the data from the source MPI process directly to the receive-side MPI process. As part of this procedure, the SP ensures that the number of active read requests does not exceed a predefined upper limit. It also checks for completion of the transfer to complete this step of the algorithm. Note: while RDMA Writes from multiple sources can potentially flood a destination and cause tree saturation, RDMA Reads can more effectively limit network congestion.
- After all local data transfers are completed, the SPs notify the host-side communication library of local collective completion.

*2)* ***MPI_Allgatherv BlueField Algorithm****:* An MPI_Allgatherv collective operation enables each MPI process to send its data to every other MPI process in the communicator. The data size each MPI process sends is unique to the given MPI process. Similar to MPI_Ialltoallv, large-data optimized algorithms use a ring-like algorithm to schedule data transfers as they are readied. Optimized MPI_Allgatherv implementations often use some form of a hierarchical ring algorithm [25] and a more recent implementation [26] segments messages to improve network utilization and load balancing.

Our offloaded algorithm is a hierarchical algorithm wherein a given MPI rank's data is sent to a host only once. A service process (SP) is used to progress communication and to manage the segmented ring-like algorithm. This allows the the algorithm to take advantage of a DPU's asynchronous capability to pick an available local MPI process to receive the remote data and load balance data delivery across ready local MPI processes. At a high level, the algorithm is as follows:

- A host process notifies its associated SP on the DPU that the collective has started. It passes metadata to the SP that includes items such as a unique group (MPI communicator) context, group rank, collective ID, virtual addresses and memory keys for the source and destination buffers, data type, and the counts arrays.
- The SP progresses a ring-like algorithm to distribute each MPI rank's data to every other host running an MPI rank participating in the collective operation. MIMOSA notifications are used to notify one of the SPs running on each remote host that data is ready to be read. Each notification includes the addressing information needed by that SP to read user data from the source process.
- When the notification arrives at the remote SP, the process of determining which of that DPU's SPs will initiate the data transfer from the source to one of that SP's associated MPI processes is started. If the receiving SP recognizes the collective operation, it will manage the data transfer. If not, it will pass the request in a ring to the remaining local SPs until one is found that is ready to handle the request. If no such SP is found, the request is placed on a shared list that all local SPs will check for work such that the request will eventually be satisfied.
- The receive-side SP performs a UCX RDMA read operation, coupled with the alias-Mkey capabilities of the NVIDIA DPU, to transfer the data from the source MPI process directly to the receive-side MPI process. As part of this procedure, the SP ensures that the number of active read requests does not exceed a predefined upper limit. It checks for completion of the transfer to complete this step of the algorithm.
- After this host-memory-to-host-memory RDMA data transfer completes, data transfer to the remaining local MPI processes commences. These MPI processes are notified that the requisite data is available by passing metadata that includes the MPI rank of the local process that has already received the data as well as the virtual address and data type of the available data. These host processes then initiate the data transfer within the host, thereby avoiding an unnecessary waste of network bandwidth.
- After all local data transfers are completed, the SPs notify the host-side communication library of local completion.

## IV. Evaluation

We evaluate the effectiveness of our BlueField-accelerated collective algorithms using a variety of Open MPI-based collective implementations. As described in this paper, our implementation is based on Open MPI [27] using a modified UCX transport implementation and a modified UCC collective algorithm component. For comparisons to the existing state of the art, we compare to the standard Open MPI implementation and to NVIDIA's closed source HPC-X MPI implementation [28]. Latency tests are used to measure the average time it takes a given collective to complete. The Overlap benchmark measures the relative amount of computational work that can be done while a single nonblocking collective is active. To evaluate collective communication performance in isolation, we use the OSU Micro-Benchmark suite [29], release 5.8. We do not modify the default number of warm-up trials (10 for large messages) or measured trials (100 for large messages) in any test. On OSU runs, error bars show the reported minimum and maximum latencies across the 100 trials. In addition, we also evaluate performance using the P3DFFT++ distributed Fast Fourier Transform (FFT) and Octopus scientific applications. Note that because the OSU collective benchmarks are well load-balanced, their measured performance tends to be better than for the same collectives executed within full applications that exhibit various degrees of load imbalance.

### A. System Configuration

The numerical experiments presented in this paper have been performed on a 32-node Intel Broadwell system with NVIDIA BlueField-3 DPUs. The system configuration is described in Table I. Note that BlueField-3 DPUs run at NDR200 speed, which has a theoretical throughput of 200 Gbps. However, since the host architecture only supports PCIe 3.0, the NDR200 BlueField-3 is limited to 128 Gbps when used as a host channel adapter (HCA) from the host. When used as an HCA from the DPU, it is capable of delivering native 200 Gbps.

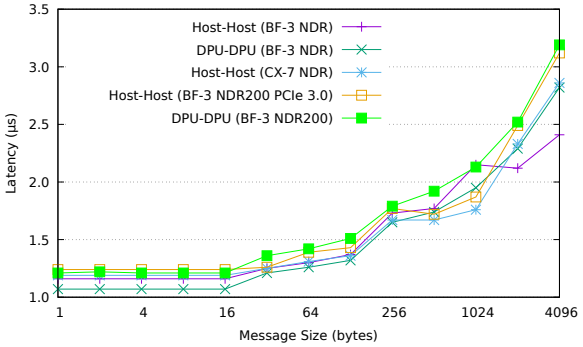
Table I: 32-node system configuration. Each host is connected via PCI 3.0 to one BlueField-3 DPU.

| | Host | BlueField-3 DPU |
|---|---|---|
| OS | Rocky Linux 9.2 | Rocky Linux 9.2 |
| Kernel | 5.14.0-284.25.1.el9_2 | 5.15.0-1015-bluefield |
| CPU | Intel(R) E5-2697A | Cortex-A78AE |
| Frequency | 2.60 GHz | 2.10 GHz |
| Sockets | 2 | 1 |
| Cores / Socket | 16 | 16 |
| RAM | 256 GB | 32 GB |
| Switch | NVIDIA Quantum-2 QM9700 | |
| HCA | NVIDIA ConnectX-7 (with NDR200 links) | |
| HCA FW | 32.38.1002 | |
| Storage | NFS & Lustre | |

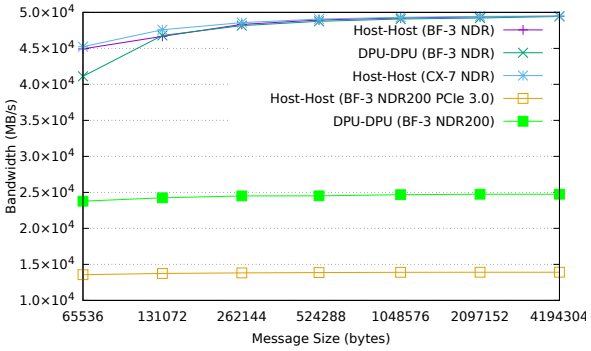
### B. Point-to-point Performance

In order to understand the behavior of any collective operation, we first need to understand the BlueField-3 performance metrics for relevant point-to-point communications. These metrics, collected using the OSU latency and bandwidth benchmarks, have been measured host to host and DPU to DPU on a native NDR system, as well as the NDR200 system. The native NDR data was collected from a pair of Intel Sapphire Rapids nodes while the NDR200 data was collected from the aforementioned test system that is utilized for the rest of this paper.

Figure 2 shows the OSU latency and bandwidth comparisons for different configurations. We can see that the BlueField-3 zero-byte latency is very close to that of ConnectX-7 at 1.16 µs and 1.19 µs, respectively. The latency for host-host and DPU-DPU are also identical for small messages, which indicates that processing work requests on the ARM side is as efficient as on the host side. For large messages, performance on the native NDR is better than NDR200 due to increased bandwidth, where 49.47 GB/s and 24.75 GB/s performance measurements with 4 MB message sizes are observed, respectively. The PCIe 3.0 on the test system limits the host to host bandwidth to 128 Gbps so it is less than DPU to DPU even though they are both NDR200, at 13.92 GB/s and 24.75 GB/s, respectively.




(a) OSU Latency

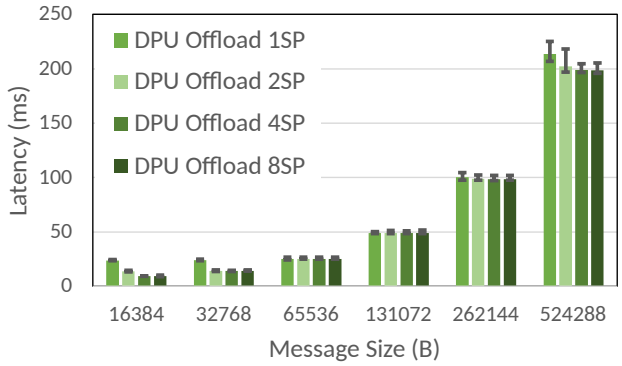



(b) OSU Bandwidth

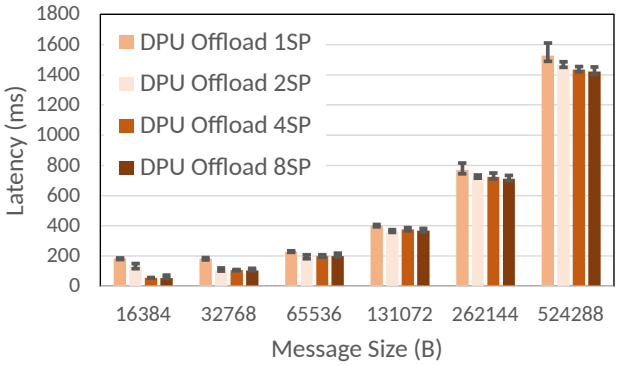
Figure 2: Point-to-point latency and bandwidth between BlueField-3 and ConnectX-7 with different configurations.

### C. Service Process Count Selection

MIMOSA provides the ability to configure a varying number of service processes to run on the DPU. For our DPU-accelerated algorithms, each service process acts as an asynchronous agent for a statically configured set of MPI




(a) MPI_Allgatherv




(b) MPI_Ialltoallv

Figure 3: MPI_Allgatherv and MPI_Ialltoallv — 1024 Process, 32 processes per node latency measurements, as a function of SP count and message size.

processes. As such, a given SP serializes the handling of work for the MPI processes it handles. It is desirable to minimize the number of SPs used in order to make the DPU cores available for other work and, therefore, we study the impact of the number of SPs per DPU on operation latency.

Figure 3 (a) and (b) show the performance of DPU-accelerated MPI_Allgatherv and MPI_Ialltoallv, respectively, as a function of the number of SPs. The OSU MPI_Allgatherv and MPI_Ialltoallv benchmarks are used to collect data using one, two, four, and eight SPs servicing 32 MPI processes per node. The data show that for smaller sized messages up to 64 KB one SP cannot quite keep up with the amount of work required to service 32 MPI processes. For larger messages, four SPs are needed, which is equivalent to one SP per eight MPI processes. Increasing the count of SP to eight improves performance slightly further but does so with diminishing returns. Given these results, we used four service processes for our remaining experiments.

### *D. DPU-Accelerated MPI_Allgatherv*

Latency measurements for MPI_Allgatherv are shown in Figure 4. This data is collected using the BlueField offloaded algorithm, Open MPI, and HPC-X. Two MPI process layouts are used: 1) MPI ranks are distributed block-wise across hosts within the communicator, i.e. “contiguously” distributed, and 2) MPI ranks are distributed in a round-robin manner across the communicator hosts, or “strided”. As the data show, the offloaded algorithm’s performance is often better and at least comparable to the latency of the best host-only algorithm implemented in HPC-X, being 31% better at 64 KB message sizes and 1% worse at 512 KB message sizes.

The offloaded algorithm’s performance is much better than HPC-X’s when the “strided” process distribution is used. In the strided configuration, DPU offload performance is 6.9 times better than Open MPI’s at 64 KB message sizes and 6.7 times better at 512 KB message sizes. The extra overheads that come from initiating the offload and in handling late arriving processes are small relative to the overall cost of data transfer,

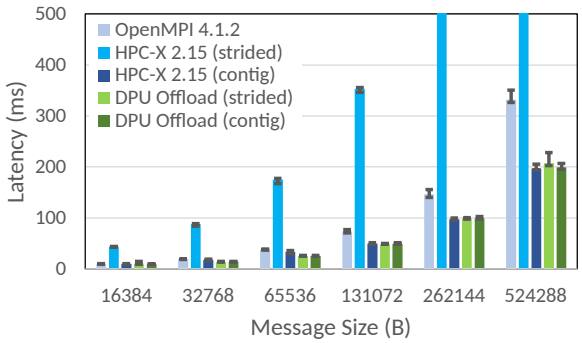



Figure 4: 1024 Process, 32 processes per node MPI_Allgatherv latency measurements as a function of message size. For the HPC-X and Offloaded algorithms we present data with MPI ranks mapped contiguously within a node (contig) and ranks mapped round-robin across nodes (strided). The additional topology information within the DPU offloaded algorithm ensures that performance is stable across arbitrary rank mappings while existing host-based algorithms can be extremely sensitive to rank mapping algorithms.

so even when all MPI processes are actively participating in the collective operation all of the time, the overall overhead is low. Surprisingly, we find that late arrivals are common even under the ideal conditions of an OSU benchmark. Across approximately 13,000 benchmarking trials, slightly more than 1% of the MPI ranks find that a message intended for the given rank has arrived at its DPU before the host process begins its collective processing. However, for 96% of those late arrivals, another process on the same host has arrived at the collective, and the SP is able to forward the data to an available and waiting host process.

A common implementation for large-data MPI_Allgatherv is that of a pipelined-ring algorithm based on MPI rank in the communicator. With a block-wise, contiguous distribution of ranks across host, a given message is sent over the interconnect

only once per host, with distribution within the host done by a host-optimized mechanism, such as a single copy through the kernel. Host memory bandwidth is typically quite a bit higher than network bandwidth, therefore being a better choice for data transfer. Network transfers also consume host-network bandwidth at both the sender and the receiver. With the "strided" distribution of ranks a large fraction the pipelined-ring algorithm will be sent over the interconnect. The offloaded algorithm described in this paper minimized the number of messages sent over the interconnect, independent of MPI process layout.

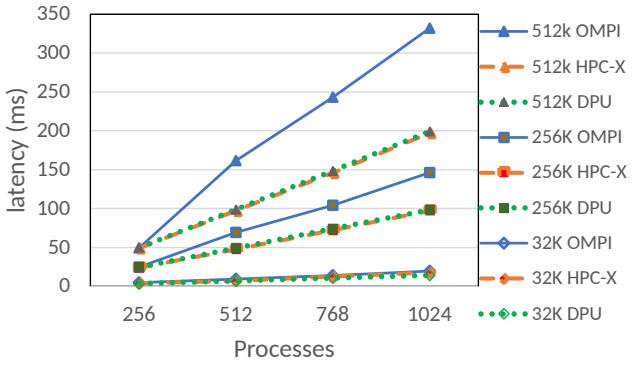



Figure 5: OSU MPI_Allgatherv latency for fixed message sizes (32, 128 and 512 KB) as a function of MPI process count (with 32 processes per node).

Figure 5 shows the latency of MPI_Allgatherv for message sizes of 32, 256, and 512 KB as a function of MPI process count in the range of 256 to 1024 processes. This data is collected using the BlueField offloaded algorithm, Open MPI, and HPC-X. With the BlueField offloaded algorithm, the latency scales linearly with the MPI process count, as expected, given that the message sizes are in the range where network bandwidth dominates the cost of data transfer. Here, the performance of the BlueField offloaded algorithm is very similar to that obtained using HPC-X and much better than that obtained using Open MPI.

### *E. DPU-Accelerated MPI_Ialltoallv*

As mentioned in section II, although absolute performance is also important, nonblocking collective algorithms are primarily optimized for communication-computation overlap. The OSU benchmark calculated concurrency of the MPI_Ialltoallv operation is shown in Figure 6 wherein the BlueField offloaded algorithm, Open MPI, and HPC-X are compared using a block-wise process layout. It is interesting to note that the concurrency produced by the DPU offloaded algorithm remains above 99% for the full range of message sizes displayed, which is well above that of the Open MPI and HPC-X implementations. At the same time, from a performance standpoint, the BlueField algorithm's performance with 1024 MPI processes is 6.3 times better at 32 KB message sizes, 2.07 times at 256 KB, and 2.02 times better at 512 KB message sizes than HPC-X, while Open MPI's performance is consistently slower than HPC-X, as shown in Figure 7. As expected, for a fixed message size, the latency of the collectives scales linearly with the number of MPI processes in the communicator.

We find that late arrivals are relatively common during the OSU latency benchmark runs, with approximately 0.1% of the MPI ranks finding a message has arrived at the DPU for that collective prior to the rank starting collective processing. Although this percentage seems low compared to what was observed for MPI_Allgatherv, the MPI_Ialltoallv algorithm requires per rank communication where only the first few communications to a rank are likely to arrive late given the nature of OSU benchmark runs. Even with the much higher volume of total communication, more than 290,000 late arrivals occurred during the approximately 6000 trials conducted as part of the data collection for Figure 6.

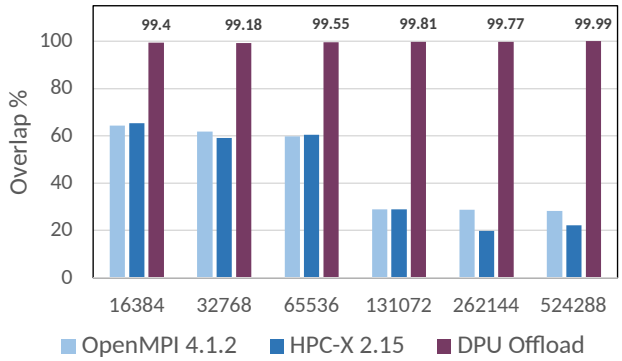



Figure 6: MPI_Ialltoallv algorithm concurrency using several algorithm implementations and 32 processes per node and the DPU offloaded version uses four service processes.

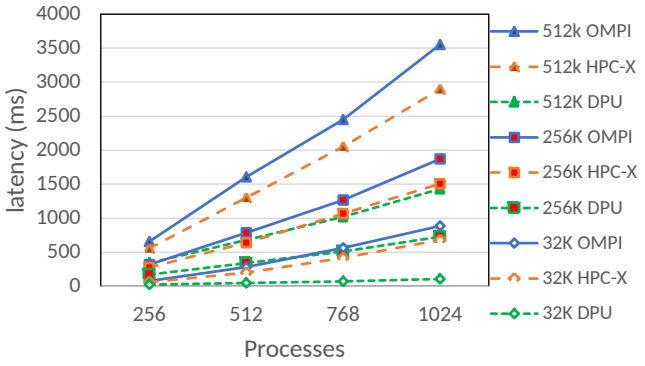



Figure 7: OSU MPI_Ialltoallv latency for fixed message size (32, 128 and 512 KB) as a function of MPI process count (with 32 processes per node).

### *F. P3DFFT++ Application Performance*

P3DFFT++ is a library of spectral transforms that includes a 3D Fast Fourier Transform (FFT) implementation [30], [31],

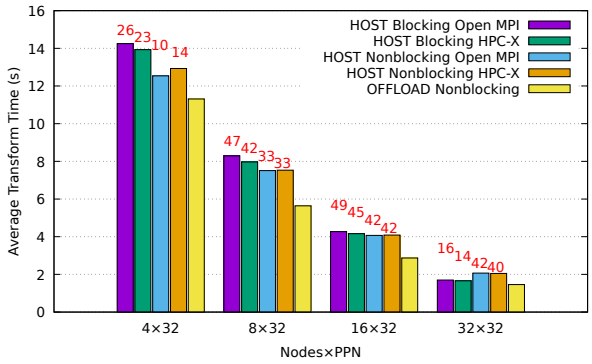



Figure 8: P3DFFT++ 4-variable test3D_c2c_cpp benchmark-reported average transform time for 10 repetitions of 1024×1024×1024 grids on a 2D processor decomposition scheme of Nodes×PPN with traditional (blocking and non-blocking) and offloaded (nonblocking) Alltoallv implementation. The numbers above each bar represent the percentage improvement of the offloaded nonblocking implementation relative to the given implementation.

[32]. By using two-dimensional processor decomposition, it is capable of scaling to a large number of compute cores/tasks. More generally, 3D FFTs represent a ubiquitous algorithm essential to many application areas in HPC programs. Therefore, any improvement in scalable performance of 3D FFTs and related spectral algorithms, particularly on top of existing solutions, would be extremely beneficial, since it would translate into higher throughput and better resource utilization for a large number of important user applications. When run at large scale (such as thousands of cores), the main performance challenge for spectral algorithms is efficient implementation of an all-to-all data-exchange, which is typically performed using an MPI_Alltoallv routine. In P3DFFT++ this routine is used in two orthogonal communicators, once per 3D FFT call. Overall, the performance of MPI_Alltoallv implementations is often the limiting factor in the overall performance of many applications relying on 3D FFTs. Moreover, this problem will likely worsen as applications move into the world of exascale computing, as bisection bandwidth is not projected to keep up with compute performance [33], [34].

To address this issue, we have implemented a pipelined multi-variable version of P3DFFT++ that takes advantage of overlapped communication and computation. This version (briefly described here and to be revealed in more detail in future publications) can be found in the newly released branch *nb-mv* (NonBlocking MultiVariable) in the P3DFFT++ repository [32]. Whereas the default version (*master* branch) takes a 3D FFT of one variable at a time and relies on blocking MPI_Alltoallv calls, the *nb-mv* branch offers a batch 3D FFT routine with several independent variables such as velocity components, pressure, passive scalars, etc. It pipelines the execution of these variables in a staggered fashion: while one variable goes through a local 1D FFT computation on a host CPU, another variable is undergoing an inter-processor exchange. To make this possible, the nonblocking MPI_Ialltoallv routine is used in lieu of MPI_Alltoallv. Taken together, these modifications provide a framework for overlapping communication with computation, provided that such overlap is supported by the underlying system's hardware and software capabilities.

Table II: P3DFFT++ process average MPI_Ialltoallv and MPI_Waitany times for 4-variable test3D_ c2c_cpp benchmark with 1024×1024×1024 grids 10 repetitions running on 16 and 32 nodes, with host-based and DPU-offloaded implementations.

| Function | 16N HOST | 16N OFF. | 32N HOST | 32N OFF. |
|---|---|---|---|---|
| Communication | 17.58 | 7.54 | 10.15 | 5.46 |
| MPI_Waitany | 10.20 | 4.88 | 6.32 | 3.22 |
| MPI_Ialltoallv | 2.98 | 1.38 | 1.64 | 0.77 |
| other | 4.40 | 1.28 | 2.19 | 1.47 |

In this paper we have used the P3DFFT++ *nb-mv* version to evaluate the performance of our offloaded MPI_Ialltoallv algorithm in real-world application scenarios. Performance gathered using our offloaded algorithm and *nb-mv* P3DFFT++ is compared against runs performed using the same P3DFFT++ code using a host-only version of MPI_Ialltoallv and runs using host-only MPI_Alltoallv. The messages sizes used in the MPI_Ialltoallv range from up to 16 MB for the 4×32 use case and up to 256 KB for the 32×32 use case. Figure 8 shows that the overall performance improvement between our DPU-offloaded implementation and the host-only MPI_Ialltoallv implementation ranges from 14% on four nodes to 49% on node counts in the range of four to 32. Table II shows the average process cost breakdown for the two main MPI functions in the 3D FFT algorithm for the 16 and 32-node runs. It shows that the time spent in the communication library waiting for the MPI_Ialltoallv operations to complete is reduced by close to 50%, which demonstrates the effectiveness of overlapping communication and computation.

### G. Octopus Application Performance

The Octopus [35], [36] simulation code applies density-functional theory (DFT) in its time-dependent form to model non-equilibrium phenomena in molecular complexes, low-dimensional materials, and extended systems by accounting for electronic, ionic, and photon quantum mechanical effects using *ab initio* methods. Octopus release 11.4 [37] was used in this study. As a benchmark, we run a time-dependent simulation of an adenine molecule with 38 states and ca. 70 million grid points in order to assess the performance of various MPI_Allgatherv algorithms. For each measurement, Octopus restarts from a pre-computed ground state and computes 100 time steps of the evolution. Each measurement was performed three times with the average time reported. Across all three runs the minimum and maximum iterations per hour are always within 0.5% of the reported average.

Figure 9 shows the application performance for two configurations: 32 nodes with eight processes per node (PPN) and four threads per process (TPP), and 32 nodes with four

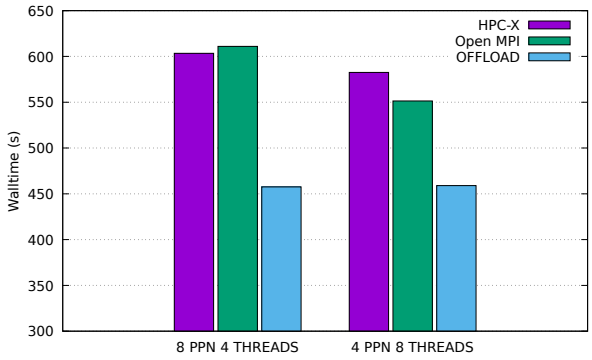



Figure 9: Octopus total wall time with traditional and offloaded Allgatherv implementations on 32 nodes for time-dependent simulations of the Adenine benchmark (100 time steps).

PPN and eight TPP. For each configuration, the traditional host implementation of collectives from Open MPI was used as the baseline, and the BlueField-3 offloaded collective implementation with UCC was run to demonstrate any performance improvements. In the offloaded runs, four service processes were launched on each DPU, and each service process was allowed four concurrent sends and receives. Figure 9 shows that our new DPU offloaded implementation is able to improve Octopus performance over HPC-X by 24% for the eight PPN four TPP configuration. The total communication time (time spent in the MPI library) is approximately 40% of the walltime in the non-offloaded case (245.99s) and 31% in the offloaded case (138.45s). Octopus performance improves by 17% over Open MPI in the four PPN eight TPP configuration.

Table III shows the breakdown of the average cost per process for all MPI functions accumulating more than 5 seconds per process. From this, one can see that the DPU-offloaded MPI_Allgatherv was improved 35% in the eight PPN four TPP configuration and 25% in the four PPN eight TPP configuration, respectively. This seems to be in contradiction to Figure 5, where the performance, measured by the OSU MPI_Allgatherv benchmark, of the host and DPU-offloaded MPI_Allgatherv looked similar. However, the reason is simple, as DPU-offloaded collectives better handle late-arrival or straggler processes. Across the 308 offloaded Allgatherv calls, approximately 9% of the messages received at service processes were for ranks that had not yet arrived at the collective. Furthermore, for 5.9% of the messages received, no ranks for that host had arrived at the collective, resulting in a delay in the servicing of those messages. Across Allgatherv calls with late arrivals, the late arriving processes arrived between 30–100ms later than the earliest arriving process (with that late arrival time making up approximately 15–50% of the total Allgatherv execution time for that collective). Finally, we note that by improving the performance of a single collective, the performance of other communication routines also improves. The majority of the improvement is in communication calls that immediately follow the offloaded Allgatherv (the Alltoall immediately following an Allgatherv shows 3.4–5.9x speedup, while a later Alltoall shows no speedup).

Table III: Octopus process average communication timing breakdown for 100 time steps of the adenine benchmark running on 32 nodes.

| Function | 8P4T HOST | 8P4T OFF. | 4P8T HOST | 4P8T OFF. |
|---|---|---|---|---|
| All Communication | 245.99 | 138.15 | 230.66 | 142.37 |
| MPI_Allgatherv | 126.38 | 82.74 | 102.17 | 76.26 |
| MPI_Waitall | 57.60 | 32.02 | 73.65 | 41.40 |
| MPI_Alltoall | 37.89 | 13.87 | 30.03 | 17.45 |
| MPI_Allreduce | 7.44 | 4.96 | 2.88 | 4.05 |
| Other | 16.68 | 4.56 | 21.93 | 3.21 |

## V. Related Work

### A. In-network Computing Frameworks

Programming frameworks for offloading computation onto SmartNICs have recently become an area of research focus. The sPIN framework [38] enables the creation of SmartNIC-based *handlers* that are designed to perform line-rate packet processing with handler selection performed based on the contents of the packet headers. The INCA programmable NIC framework is built upon *triggered operations* and a tag field within packets that enables a series of tag-specific in-transit operations to a packet as it enters or exits the NIC [39]. Construction of a pipeline of packet processing operations, called chained operations, is also supported by the PANIC programming framework, which adds explicit support for multi-tenancy and dynamic resource allocation to improve both offload performance and network utilization under load [40]. The MIMOSA framework described in this paper most closely follows the asynchronous progress principle and memory semantics described by RaDD [41]. Offloaded operations within MIMOSA are similar to the Mercury framework's nonblocking RPCs [42] while the mechanisms used to perform reads and writes to on-host data structures without host CPU intervention are related to the PRISM RDMA read/write extensions [43], but extended to enable SmartNIC DPUs to modify local and remote host memory directly. More recently, the Two-chains [44] and Three-chains [45] frameworks have extended traditional active message capabilities to perform execution of RDMA binary code payloads on remote SmartNICs.

### B. Offloads for Communication Libraries

The abstract interface for collective communication algorithms provided by MPI has enabled the development of dedicated hardware and software for improving the performance of collectives. The Quadrics network provided acceleration for the collective barrier synchronization and broadcast algorithms using a set of specialized hardware multicast primitives with supporting system software [46]. Offloads for the collective broadcast algorithm leveraging in-switch multicast primitives have demonstrated latency improvements and near-linear scaling for small message broadcasts [47], [48].

More specifically, efforts using NIC-based processors have explored performing barrier message origination entirely within the network card to reduce the number of transactions

between the CPU and network interface [49]. The Cray XC40 network provided support for performing the collective reduction operators within compute elements inside the Aries NIC [50]. SmartNICs supporting general program offloading have been demonstrated to improve communication latency and reduce host CPU load for the nonblocking versions of allgather and broadcast collectives [6].

## VI. Discussion

This paper describes strategies for offloading collective communication work to BlueField DPUs. It describes offloaded implementations of the (blocking) MPI_Allgatherv and (nonblocking) MPI_Ialltoallv long-message protocols, which are prominently used by the Octopus and P3DFFT++ applications. An analysis of these collective operations offloaded to BlueField-3 DPU is provided, demonstrating the benefits of this strategy to improve application performance.

Standard MPI benchmarks have been used to individually characterize the performance of these collective operations. The latency of the DPU-offloaded MPI_Allgatherv was shown to be similar to that of the host-only implementation, although the latter incurs extra work associated with coordination of the DPU SPs and handling requests to read data from host ranks that are not ready. These latencies also scale linearly, as expected, with process count. However, the real benefits of using the offloaded MPI_Allgatherv algorithm appear more clearly when used in the context of the Octopus application, with the performance of the four-process, eight-thread, 32-node job improving by 17% when offloading the collective operation to DPU. While almost 9% of the ready-to-read requests arrive at SPs that are unable to service these requests immediately, the SP implementation is able to find other MPI processes that are ready for this data in 3% of these cases. The remaining cases are serviced later as additional MPI processes enter the collective operation. Such a load imbalance is typical for applications, but this offloaded approach is effective in mitigating the effects of this imbalance on application run time. Similar improvements in performance have also been seen with other applications; however, space constraints limit the results presented here.

The benefits of having a truly asynchronous agent progressing nonblocking collective operations are demonstrated with the BlueField-offloaded MPI_Ialltoallv algorithm. The OSU overlap benchmark shows that for messages of 16 KB and above, over 99% of the time spent in execution of the collective operation can be overlapped with computation. This is reflected in an improvement of 14–42% in P3DFFT's transfor time for 1024 processes.

In developing offload solutions that are advantageous not only for nonblocking, but also for blocking collectives, we created a set of tenets to guide collective offload work. First, service processors (SPs) must be able to act as peers to traditional host processors. SPs must also be able to directly access both local and remote host memory to avoid extra data copying. Load balancing across SPs and MPI host processes can be critical. Finally, looking to the future, due to the acceleration possibilities created by DPUs, it is critical to allow additional specialization within communication libraries.

## References


[1] L. Johnsson and G. Netzer, "The impact of Moore's Law and loss of Dennard scaling: Are DSP SoCs an energy efficient alternative to x86 SoCs?" *Journal of Physics: Conference Series*, vol. 762, no. 1, p. 012022, oct 2016. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/762/1/012022

[2] T. Schneider, T. Hoefler, R. E. Grant, B. W. Barrett, and R. Brightwell, "Protocols for fully offloaded collective operations on accelerated network adapters," in *2013 42nd International Conference on Parallel Processing*. Lyon, France: IEEE, 2013, pp. 593–602.

[3] G. Sabin and M. Rashti, "Security offload using the SmartNIC, a programmable 10 Gbps Ethernet NIC," in *2015 National Aerospace and Electronics Conference (NAECON)*. Dayton, OH, USA: IEEE, 2015, pp. 273–276.

[4] A. Kaufmann, S. Peter, N. K. Sharma, T. Anderson, and A. Krishnamurthy, "High performance packet processing with FlexNIC," *SIGPLAN Not.*, vol. 51, no. 4, p. 67–81, Mar 2016. [Online]. Available: https://doi.org/10.1145/2954679.2872367

[5] NVIDIA, "BlueField Data Processing Units," https://www.nvidia.com/en-us/networking/products/data-processing-unit/, 2023, accessed 2023-04-04.

[6] N. Sarkauskas, M. Bayatpour, T. Tran, B. Ramesh, H. Subramoni, and D. K. Panda, "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)*. Bengaluru, India: IEEE, 2021, pp. 388–393.

[7] M. Bayatpour, N. Sarkauskas, H. Subramoni, J. Maqbool Hashmi, and D. K. Panda, "BluesMPI: Efficient MPI non-blocking Alltoall offloading designs on modern BlueField Smart NICs," in *High Performance Computing*, B. L. Chamberlain, A.-L. Varbanescu, H. Ltaief, and P. Luszczek, Eds. Cham: Springer International Publishing, 2021, pp. 18–37.

[8] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W.-K. Su, "Myrinet: a gigabit-per-second local area network," *IEEE Micro*, vol. 15, no. 1, pp. 29–36, 1995.

[9] F. Petrini, W. chun Feng, A. Hoisie, S. Coll, and E. Frachtenberg, "The Quadrics network: high-performance clustering technology," *IEEE Micro*, vol. 22, no. 1, pp. 46–57, 2002.

[10] R. Brightwell, K. Pedretti, K. Underwood, and T. Hudson, "SeaStar interconnect: Balanced bandwidth for scalable performance," *IEEE Micro*, vol. 26, no. 3, pp. 41–57, 2006.

[11] S. Araki, A. Bilas, C. Dubnicki, J. Edler, K. Konishi, and J. Philbin, "User-space communication: A quantitative study," in *SC '98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*. Orlando, FL, USA: IEEE, 1998, pp. 18–18.

[12] C. Kreibich, A. Warfield, J. Crowcroft, S. Hand, and I. Pratt, "Using packet symmetry to curtail malicious traffic," 2005.

[13] A. Currid, "TCP offload to the rescue: Getting a toehold on TCP offload engines—and why we need them," *Queue*, vol. 2, no. 3, p. 58–65, may 2004. [Online]. Available: https://doi.org/10.1145/1005062.1005069

[14] J. Hypolite, J. Sonchack, S. Hershkop, N. Dautenhahn, A. DeHon, and J. M. Smith, "DeepMatch: Practical deep packet inspection in the data plane using network processors," in *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 336–350. [Online]. Available: https://doi.org/10.1145/3386367.3431290

[15] J. Kim, I. Jang, W. Reda, J. Im, M. Canini, D. Kostić, Y. Kwon, S. Peter, and E. Witchel, "LineFS: Efficient SmartNIC offload of a distributed file system with pipeline parallelism," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, ser. SOSP '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 756–771. [Online]. Available: https://doi.org/10.1145/3477132.3483565

[16] K. Taranov, B. Rothenberger, A. Perrig, and T. Hoefler, "sRDMA – efficient NIC-based authentication and encryption for remote direct memory access," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. Virtual: USENIX Association, Jul. 2020, pp. 691–704. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/taranov

[17] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H. K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D. A. Maltz, and A. Greenberg, "Azure accelerated networking: SmartNICs in the public cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 51–66. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/firestone

[18] Netronome, "Agilio CX SmartNICs," https://www.netronome.com/products/agilio-cx/, 2023, accessed 2023-04-04.

[19] NVIDIA, "NVIDIA Mellanox Innova-2 Flex," https://www.nvidia.com/en-us/networking/ethernet/innova-2-flex/, 2023, accessed 2023-04-04.

[20] Fungible Networks, "Fungible F1 Data Processing Units," https://www.hc32.hotchips.org/assets/program/conference/day2/HotChips2020_Networking_Fungible_v04.pdf, 2020, accessed on 2022-11-29.

[21] Broadcom, "Broadcom Stingray SmartNIC Accelerates Baidu Cloud Services," https://www.broadcom.com/company/news/product-releases/53106, 2020, accessed 2023-04-04.

[22] NVIDIA, "NVIDIA DOCA SDK," https://developer.nvidia.com/networking/doca, 2023, accessed 2023-04-04.

[23] Unified Communication Framework Consortium, "Unified Collective Communication," https://openucx.github.io/ucc/api/v1.2/pdf/ucc.pdf, 2023.

[24] P. Shamis, M. G. Venkata, M. G. Lopez, M. B. Baker, O. Hernandez, Y. Itigin, M. Dubman, G. Shainer, R. L. Graham, L. Liss *et al.*, "UCX: an open source framework for HPC network APIs and beyond," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, IEEE. Santa Clara, CA, USA: IEEE, 2015, pp. 40–43.

[25] J. L. Träff, A. Ripke, C. Siebert, P. Balaji, R. Thakur, and W. Gropp, "A pipelined algorithm for large, irregular all-gather problems," *The International Journal of High Performance Computing Applications*, vol. 24, no. 1, pp. 58–68, 2010. [Online]. Available: https://doi.org/10.1177/1094342009359013

[26] A. Kang, J. Traff, R. Al-Bahrani, A. A., and A. Choudhary, "Scalable algorithms for MPI intergroup Allgather and Allgatherv," *Parallel Computing*, vol. 85, pp. 220–230, 2019.

[27] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, September 2004, pp. 97–104.

[28] NVIDIA, "NVIDIA HPC-X 2.11 documentation," https://docs.nvidia.com/networking/display/hpcxv211, 2023, accessed 2023-04-06.

[29] Ohio State University, "OSU micro-benchmarks," https://mvapich.cse.ohio-state.edu/benchmarks, 2023, accessed 2023-04-04.

[30] D. Pekurovsky, "P3DFFT: A framework for parallel computations of fourier transforms in three dimensions," *SIAM Journal on Scientific Computing*, vol. 34, no. 4, pp. C192–C209, 2012.

[31] D. Pekurovsky, "A portable framework for multudimensional spectral-like transforms at scale," in *Proceedings of the 2021 Improving Scientific Software Conference*, 2021.

[32] D. Pekurovsky, "P3DFFT++ github repository," https://github.com/sdsc/p3dfft.3, 2023.

[33] K. Czechowski, C. Battaglino, C. McClanahan, K. Iyer, P.-K. Yeung, and R. Vuduc, "On the communication complexity of 3D FFTs and its implications for exascale," in *Proceedings of the 26th ACM international conference on Supercomputing*, 2012, pp. 205–214.

[34] C. McClanahan, K. Czechowski, C. Battaglino, K. Iyer, P. Yeung, and R. Vuduc, "Prospects for scalable 3D FFTs on heterogeneous exascale systems," in *ACMIEEE conference on supercomputing, SC*, 2011.

[35] A. Castro, H. Appel, M. Oliveira, C. Rozzi, X. Andrade, F. Lorenzen, M. Marques, E. Gross, and A. Rubio, "Octopus: a tool for the application of time-dependent density functional theory," *Phys. Stat. Sol. B*, vol. 243, pp. 2465–2488, 2006.

[36] N. Tancogne-Dejean, M. J. T. Oliveira, X. Andrade, H. Appel, C. H. Borca, G. Le Breton, F. Buchholz, A. Castro, S. Corni, A. A. Correa, U. De Giovannini, A. Delgado, F. G. Eich, J. Flick, G. Gil, A. Gomez, N. Helbig, H. Hübener, R. Jestädt, J. Jornet-Somoza, A. H. Larsen, I. V. Lebedeva, M. Lüders, M. A. L. Marques, S. T. Ohlmann, S. Pipolo, M. Rampp, C. A. Rozzi, D. A. Strubbe, S. A. Sato, C. Schäfer, I. Theophilou, A. Welden, and A. Rubio, "Octopus, a computational framework for exploring light-driven phenomena and quantum dynamics in extended and finite systems," *The Journal of Chemical Physics*, vol. 152, no. 12, p. 124119, 03 2020. [Online]. Available: https://doi.org/10.1063/1.5142502

[37] The Octopus development team, "Octopus Version 11," https://www.octopus-code.org/documentation/13/releases/octopus-11/, 2021, accessed 2023-12-12.

[38] T. Hoefler, S. Di Girolamo, K. Taranov, R. E. Grant, and R. Brightwell, "SPIN: High-performance streaming processing in the network," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3126908.3126970

[39] W. Schonbein, R. E. Grant, M. G. F. Dosanjh, and D. Arnold, "INCA: In-network compute assistance," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356153

[40] J. Lin, K. Patel, B. E. Stephens, A. Sivaraman, and A. Akella, "PANIC: A High-Performance programmable NIC for multi-tenant networks," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. Virtual: USENIX Association, Nov. 2020, pp. 243–259. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/lin

[41] R. E. Grant, W. Schonbein, and S. Levy, "RaDD runtimes: Radical and different distributed runtimes with SmartNICs," in *2020 IEEE/ACM Fourth Annual Workshop on Emerging Parallel and Distributed Runtime Systems and Middleware (IPDRM)*. Atlanta, GA, USA: IEEE, 2020, pp. 17–24.

[42] J. Soumagne, D. Kimpe, J. Zounmevo, M. Chaarawi, Q. Koziol, A. Afsahi, and R. Ross, "Mercury: Enabling remote procedure call for high-performance computing," in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*. Indianapolis, IN, USA: IEEE, 2013, pp. 1–8.

[43] M. Burke, S. Dharanipragada, S. Joyner, A. Szekeres, J. Nelson, I. Zhang, and D. R. K. Ports, "PRISM: Rethinking the RDMA interface for distributed systems," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, ser. SOSP '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 228–242. [Online]. Available: https://doi.org/10.1145/3477132.3483587

[44] M. Grodowitz, L. E. Peña, C. Dunham, D. Zhong, P. Shamis, and S. Poole, "Two-Chains: High performance framework for function injection and execution," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. Portland, OR, USA: IEEE, 2021, pp. 377–387.

[45] W. Lu, L. E. Peña, P. Shamis, V. Churavy, B. Chapman, and S. Poole, "Bring the BitCODE-moving compute and data in distributed heterogeneous systems," in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*. Heidelberg, Germany: IEEE, 2022, pp. 12–22.

[46] F. Petrini, S. Coll, E. Frachtenberg, and A. Hoisie, "Hardware- and software-based collective communication on the Quadrics network," in *Proceedings IEEE International Symposium on Network Computing and Applications. NCA 2001*. Cambridge, MA, USA: IEEE, 2001, pp. 24–35.

[47] J. Liu, A. Mamidala, and D. Panda, "Fast and scalable MPI-level broadcast using InfiniBand's hardware multicast support," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.* Santa Fe, NM, USA: IEEE, 2004, pp. 10–.

[48] T. Hoefler, C. Siebert, and W. Rehm, "A practically constant-time MPI broadcast algorithm for large-scale InfiniBand clusters with multicast," in *2007 IEEE International Parallel and Distributed Processing Symposium*. Long Beach, CA, USA: IEEE, 2007, pp. 1–8.

[49] W. Yu, D. Buntinas, R. Graham, and D. Panda, "Efficient and scalable barrier over Quadrics and Myrinet with a new NIC-based collective message passing protocol," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.* Santa Fe, NM, USA: IEEE, 2004, pp. 182–.

[50] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Cray XC series network," 2012.