Cooperative Rendezvous Protocols for Improved Performance and Overlap

S. Chakraborty, M. Bayatpour, J. Hashmi, H. Subramoni, and D. K. Panda
Department of Computer Science and Engineering, The Ohio State University
{chakraborty.52, bayatpour.1, hashmi.29, subramoni.1, panda.2}@osu.edu

***Abstract*—With the emergence of larger multi-/many-core clusters and new areas of HPC applications, performance of large message communication is becoming more important. MPI libraries use different rendezvous protocols to perform large message communication. However, existing rendezvous protocols do not take the overall communication pattern into account or make optimal use of the Sender and the Receiver CPUs. In this work, we propose a cooperative rendezvous protocol that can provide up to 2x improvement in intra-node bandwidth and latency for large messages. We also propose designs to dynamically choose the best rendezvous protocol for each message based on the overall communication pattern. Finally, we show how these improvements can increase the overlap of intra-node communication and computation with inter-node communication and lead to application level benefits at scale. We evaluate the proposed designs on three different architectures - Intel Xeon, Knights Landing, and OpenPOWER against state-of-the-art MPI libraries including MVAPICH2 and Open MPI. Compared to existing designs, the proposed designs show benefits of up to 19% with Graph500, 16% with CoMD, and 10% with MiniGhost.**

***Index Terms*—MPI, Rendezvous Protocols, HPC**

I. Introduction

Modern High-Performance Computing (HPC) systems allow scientists and engineers to tackle grand challenges in their respective domains and make significant contributions to their fields. The design and deployment of such ultra-scale systems is fueled by the increasing use of multi-/many-core environments (Intel Xeon, Xeon Phi, and IBM OpenPOWER architectures).

The Message Passing Interface (MPI) [1] is the de-facto standard programming model for developing portable and high-performance parallel scientific applications. Many MPI applications spend a significant portion of their overall runtime inside the MPI library [2–4]. Thus, overall performance of the application is closely coupled with the performance characteristics of the MPI library. To obtain the best performance, application developers utilize a variety of communication primitives such as point-to-point, collective, and one-sided provided by MPI. Furthermore, applications often use non-blocking operations to overlap the cost of computation with communication and reduce the overall execution time [5–7].

For point-to-point communication, MPI libraries broadly use two schemes — "Eager" and "Rendezvous" [8]. The eager protocol generally uses a set of known, pre-registered/pre-allocated buffers to communicate asynchronously with peer processes and is typically used for small messages. On the other hand, rendezvous protocols use handshaking to avoid extra copies and are used for large messages. Over the years, researchers have proposed different rendezvous protocols [9–11] to improve communication performance or overlap in specific scenarios. However, none of the protocols proposed in literature fully utilize all the available resources to efficiently progress the communication operation. This leads to sub-optimal resource utilization and sub-par performance. To make matters worse, MPI libraries often use the same protocol for all communication operations without considering the nature of the higher level operation, the overall communication pattern, and the impact it has on lower level hardware and software resources. These constraints limit the applicability of the available rendezvous communication protocols in literature to specific situations. This lack of a "silver bullet" means that many of these designs need to be tuned differently based on the application and system characteristics. Researchers have proposed methods for static tuning [12, 13] or trace-driven analysis [14, 15] to mitigate this issue. However, the usability of these approaches is limited by the need to frequently re-tune or generate new traces for different applications, or even different sizes of the input or the job size for the same application. Hence, it is imperative for modern MPI libraries to become more introspective and dynamic to be able to cater to a wide array of applications and hardware systems.

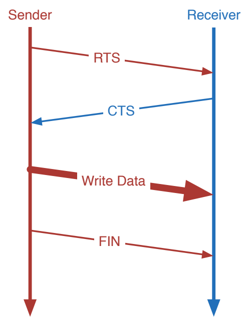
In this paper, we propose novel "cooperative" and "dynamic" rendezvous communication protocols that can adapt to application needs in real time by being cognizant of 1) the needs of the higher level communication operation in terms of performance and overlap, 2) overall communication pattern, and 3) the impact these operations have on lower level hardware and software resources. We define a process as being "cooperative" if it satisfies at least one of the following aspects: 1) shares its resources with other processes to improve the performance of data transfer, 2) shares local information about communication primitives or resource utilization with other processes, or 3) takes local actions to become more responsive to communication requests from other processes. The scale of this cooperative behavior can also vary — it can be between just the sender and the receiver, among processes within the same node, or among processes spread across multiple nodes. To the best of our knowledge, this is the first study to explore cooperative rendezvous protocols that can dynamically adapt to the application's needs. Experimental

This research is supported in part by National Science Foundation grants #CNS-1513120, #ACI-1450440 and #CCF-1565414.

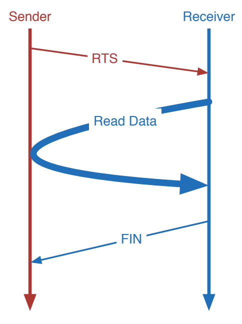
evaluations show that our proposed family of protocols can select appropriate protocols at a per-message granularity and offer improved performance and overlap for point-to-point and collective operations as well as overall application execution time. With the proposed designs, we were able to improve the runtime of Graph500, CoMD, and MiniGhost by 19%, 16%, and 10% respectively.

A. *Overview of Rendezvous Protocols in MPI*

With increasingly dense nodes with multi-/many-core architectures and larger systems, application scientists are trying to solve larger problems. For many applications, this translates to an increase in the size of the messages being communicated. For large messages, the cost of copying the data is the dominant factor for both transfers within the same node and across different nodes. MPI libraries use a variety of rendezvous protocols to minimize this copying cost. Broadly, these protocols can be divided into two schemes — write-based and read-based. Figure 1 shows two such protocols commonly used by MPI libraries. In these protocols, the sender and the receiver perform a handshake to exchange information about the buffers using RTS (Request-to-Send) and CTS (Clear-to-Send) packets. After the handshake is performed, the sender writes the data to the receiver's buffer in case of RPUT or the receiver reads the data from the sender buffer in case of RGET. In most high-performance MPI libraries, RDMA write or read operation is used if the sender and receiver are located on different nodes. In case of intra-node transfers, kernel-assisted single-copy mechanisms such as CMA [16], XPMEM [17], or KNEM [18] are used to transfer the data. Once the buffer is copied, a FIN packet is used to signal completion to the peer. Some researchers have also proposed Receiver-initiated protocols [11] where the receiver initiates the handshake by sending an RTR (Ready-to-Receive) packet to the sender. The same read or write mechanisms are used to transfer the data. However, most MPI libraries do not use such receiver-initiated rendezvous protocols since they require extra complexity to ensure correct matching when 'ANY_SOURCE' receive operations are used [11, 19, 20].



(a) Write-based (RPUT)



(b) Read-based (RGET)

Fig. 1: Commonly used rendezvous protocols in MPI. The actual data movement can be based on kernel-assisted single-copy mechanisms (e.g. CMA, XPMEM, KNEM) for intra-node transfers and RDMA Read/Write operations for inter-node transfers.

B. *Experimental Setup*

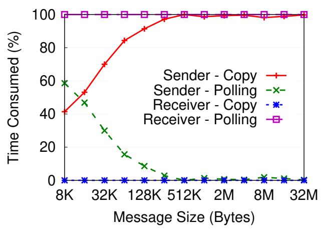
We used three different architectures commonly used in modern HPC systems for our experiments and evaluations. Table I lists the detailed hardware specification for the these systems. The initial experiments and the proposed designs were implemented in the MVAPICH2-2.3rc1 MPI library [21]. The OSU Microbenchmarks Suite [22] (OMB-5.4.2) was used for the synthetic benchmarks. Further details about the microbenchmark and application level evaluations can be found in Sections IV and V.

TABLE I: Hardware specification of the clusters used

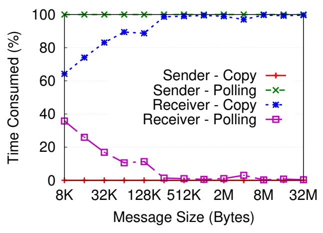
Specification	**Xeon**	**Xeon Phi**	**OpenPOWER**
Processor Family	Intel Broadwell	Knights Landing	IBM POWER-8
Processor Model	E5 v2680	KNL 7250	PPC64LE
Clock Speed	2.4 GHz	1.4 GHz	3.4 GHz
No. of Sockets	2	1	2
Cores Per Socket	14	68	10
Threads per Core	1	4	8
RAM (DDR)	128GB	96GB	256GB
Interconnect	IB-EDR(100G)	IB-EDR(100G)	IB-EDR(100G)
Compiler	gcc-7.2.0	gcc-7.2.0	gcc-4.8.5
OS/Kernel	RHEL7.4/3.10.0	RHEL7.4/3.10.0	RHEL7.2/3.10.0

C. *Motivation*

For inter-node rendezvous transfers, the copying of data is handled by the HCA while the CPU only needs to handle the handshake/control messages. However, for intra-node transfers, either the sender or the receiver CPU needs to copy the data. Figures 2(a) and 2(b) show the CPU usage profile for an intra-node, blocking, point-to-point transfer using the RPUT and the RGET protocols respectively on the Broadwell system.



(a) RPUT



(b) RGET

Fig. 2: Sender and receiver side CPU usage with different rendezvous protocols on Broadwell. The CPU performing the copy cannot perform computation at the same time. Polling represents idle time and can potentially be overlapped.

Figure 2(a) shows that the RPUT protocol is primarily driven by the sender process. The sender polls until it receives the CTS from the receiver and then writes the data to the receiver's buffer. As the message size increases, the time spent in copying becomes more dominant since the time taken for exchanging the control messages is not affected by the size of the data. It also shows that the receiver process does not participate in the actual movement of the data. Similarly, in case of RGET, the receiver process performs the actual copy and the sender spends most of its time polling. Clearly, the existing rendezvous protocols do not make efficient use of all the available resources.

In addition to raw performance, the overlap of communication and computation is also critical for application performance. MPI applications use non-blocking communication primitives such as MPI_Isend/MPI_Irecv to overlap the compute phase with communication. However, for intra-node transfers, both the application compute as well as the communication (copying the data) are done by the CPU. Hence, the amount of overlap can be increased by reducing the amount of polling and using it for computation. As Figure 2 shows, RPUT offers good overlap on the receiver side but none on the sender side. Conversely, RGET offers good sender-side overlap but no receiver-side overlap. However, offloading the copy operations to DMA engines or the HCA can free up the CPU and provide good overlap for both the sender and the receiver. Clearly, using a statically selected rendezvous protocol for all transfers cannot account for these different scenarios and may lead to sub-optimal performance and overlap. These observations bring us to the first challenge: **How can the sender and the receiver cooperate with each other and share resources to improve the performance and overlap of rendezvous transfers?**

The choice of the rendezvous protocol also depends on the overall communication pattern. For example, a one-to-all communication would be faster with the RGET protocol compared to RPUT since it distributes the copy operations among all the non-root processes (receivers) instead of the root (sender) performing all the copies. Similarly, an all-to-one communication would benefit from using RPUT instead of RGET. However, the sender and receiver may not be aware of the overall communication pattern and the most suitable rendezvous protocol based only on local information. This brings us to the second challenge: **How can multiple MPI processes cooperate with each other to discover the overall communication pattern and dynamically adapt the communication strategy accordingly?**

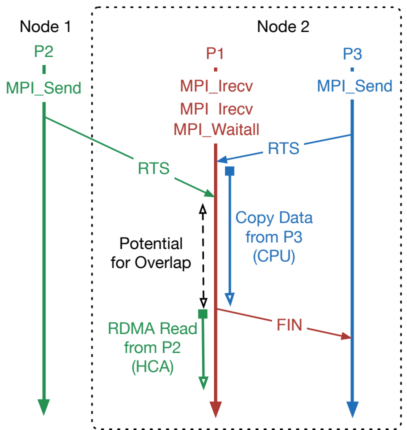


Fig. 3: Example of Lost Overlap Potential. It can affect both intra-node and inter-node transfers.

Since intra-node and inter-node communications are progressed by different agents (CPU and the HCA), ideally they should always achieve perfect overlap. However, rendezvous transfers require the exchange of initial handshake messages which must be processed by the CPU before the actual data movement can take place. However, since intra-node transfers require the CPU to perform blocking copies, the CPU may be unable to process these handshake messages for a long period of time for large messages. Figure 3 shows an example of such a scenario. P1 receives an RTS message from an intra-node peer P3 and proceeds to perform the copy operation from P3. P1 is able to process the RTS from another peer P2 and initiate the RDMA read from P2 only after it has finished this copy operation. Thus, although these two transfers from P2 and P3 could be perfectly overlapped, P1 does not see any overlap of these two communications. This observation leads us to the next challenge: **What designs can be employed to improve the overlap of intra-node and inter-node communication and overall application performance?**

D. Contributions

In this paper, we tackle these challenges and propose different designs that rely on processes cooperating at local and global scale to improve the performance, overlap, and overall communication progress. We also show how these designs can be combined to develop a truly dynamic and adaptive design that is applicable to different communication patterns across applications. To the best of our knowledge, no scholarly work or MPI implementation has proposed similar dynamic and cooperative designs for rendezvous protocols. To summarize, the main contributions of this paper are:

- Study the impact of different rendezvous protocols on intra-node communication performance and overlap.
- Propose new rendezvous protocols that take advantage of cooperation between the sender and receiver to improve performance and overlap.
- Propose new designs to enable cooperation among multiple processes to discover and dynamically adapt to varying communication patterns.
- Demonstrate the effectiveness and scalability of proposed designs using microbenchmarks and representative applications on different architectures.

II. Designing Dynamic and Cooperative Rendezvous Protocols

In this section, we describe the proposed dynamic and cooperative rendezvous protocols. Broadly, this section is divided into three parts. The first part considers cooperation between the sender and the receiver, the second part focuses on cooperation among intra-node peers, and the third section describes cooperation among processes on different nodes.

A. Designs based on Sender/Receiver Cooperation

1) Improving Point-to-point Communication Performance: Based on the results shown in Figure 2, we propose a new rendezvous protocol that uses cooperation between the sender and the receiver to accelerate the movement of the data. This protocol is referred to as the **COOP-p2p** protocol. Figure 4 shows a high-level overview of this new protocol. Broadly, it combines the RPUT and RGET protocols described in

Section I-A. The protocol is initiated when the sender sends an RTS (Ready-to-Send) packet to the receiver. This packet contains the address of the send buffer and any additional information required to access it. When it is matched with a recv operation on the receiver side, the receiver responds with a CTS (Clear-to-Send) packet which contains the address and other information about the recv buffer. After sending the CTS, the receiver proceeds to copy half of the data from the send buffer to the recv buffer. The sender also copies the other half of the data once it receives the CTS. These copies can be performed using any kernel-assisted copy mechanism such as CMA, XPMEM, or KNEM. Since these two copies are being done by two different CPU cores, they can progress in parallel. Once the copy operation is finished, both the sender and the receiver send a FIN packet to mark the completion. Since the FIN packet may arrive before or after finishing the local copy operation, the send/recv operation is marked as completed only when both the copy operations is finished and the FIN from the remote process has been received.

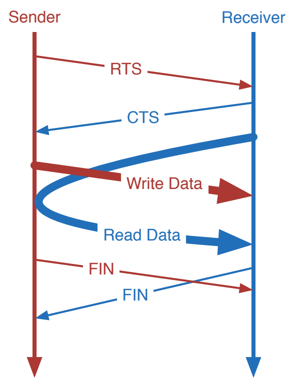


Fig. 4: Design of the COOP-p2p Protocol

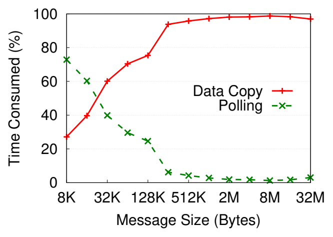


Fig. 5: Sender CPU usage

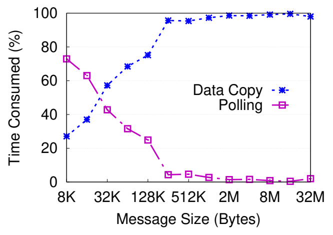
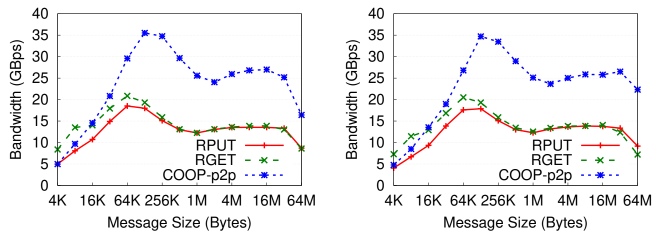


Fig. 6: Receiver CPU usage

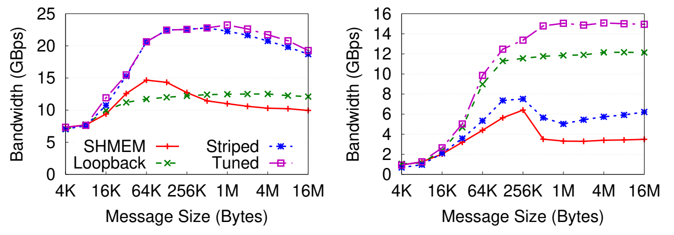
Figures 5 and 6 show the CPU utilization on the sender and the receiver side while using the COOP-p2p protocol on the Broadwell system. Both CPU cores participate equally in the data movement, and the overall time spent in polling and similar overheads is reduced. Figure 7 compares the performance of the COOP-p2p protocol against RPUT and RGET protocols on Broadwell. For messages smaller than 32KB, the overhead from the additional FIN packet and performing two fragmented copies outweigh the benefits of the increased parallelism. For larger messages, the COOP-p2p protocol outperforms others and delivers up to 2x improvement in unidirectional bandwidth for both intra-socket and inter-socket transfers. However, the COOP-p2p protocol performs similar to RGET for bidirectional bandwidth since both the sender and the receiver CPUs are already copying data and there is no spare CPU capacity left to take advantage of.

2) Offloading Point-to-point Communication for Overlap: In the past, researchers have explored the strategy of offloading expensive copy operations to hardware DMA engines to keep the CPU free for computation [23]. Unfortunately, this additional DMA hardware is not present in many modern architectures such as Knights Landing and OpenPOWER. To make our proposed designs more generic and applicable to a wider range of systems, we utilize the DMA engine present in the HCA to offload copy operations. This approach of using the HCA to progress intra-node transfers using loopback can leave the CPU free for computation. This approach can provide close to 100% overlap for both the sender and the receiver, at the cost of increasing the load on the HCA. However, if both the CPU and the HCA are available to progress the communication (e.g., in case of blocking send/recv), using only one of them is not the optimal use of resources. To ensure that both the CPU and HCA are used in the transfer, a large message can be “striped” across both channels.



(a) Intra-Socket Bandwidth (b) Inter-Socket Bandwidth

Fig. 7: Intra-node bandwidth on Broadwell with different Rendezvous protocols. COOP-p2p shows upto 2x better unidirectional bandwidth compared to RPUT or RGET.



(a) Broadwell (b) KNL

Fig. 8: Unidirectional Bandwidth with the proposed design where both the CPU and the HCA can drive the data transfer.

Figures 8(a) and 8(b) show the unidirectional bandwidth achieved between two intra-node processes using this approach on Broadwell and KNL equipped with one EDR HCA. The bandwidth of the loopback channel (shown as ‘Loopback’) is similar (100 Gbps) on both architectures since it is mostly determined by the HCA. However, the bandwidth of the CPU-driven channel (shown as ‘SHMEM’) is much higher on Broadwell compared to KNL due to the faster clock-speed and more complex instruction/data pipelines. ‘Striped’ refers to the approach of dividing up the message into two equal parts and sending them through the two different channels at teh same time. On Broadwell, the memory-copy bandwidth of the CPU and the HCA are comparable and hence this approach achieves almost twice the bandwidth compared to using the CPU alone. However, since the CPU copy performance on KNL is significantly slower compared to the HCA, the 'Striped' design offers poor performance. To obtain the best performance, the ratio of the message communicated using loopback (HCA) and shared memory (CPU) must be varied based on the message size and the architecture. The optimal ratio can be calculated by the following equation: $R_{optimal} = Bandwidth_{HCA}/Bandwidth_{CPU}$. The performance of this design is shown as 'Tuned' in Figures 8(a) and 8(b) and referred to as **COOP-hca**. Note that in this section we only consider a single pair of communicating processes. In case of many concurrent transfers, the HCA can get easily overwhelmed since it has only a single DMA engine. Further, using the HCA for intra-node communication can also slow down the inter-node communication due to the additional load. More advanced cooperative designs to mitigate these issues are described in Section II-B3.

B. Global Cooperation based Rendezvous Protocols

An MPI application typically performs multiple concurrent point-to-point operations as well as collective communication. None of the existing rendezvous protocols, including the basic cooperative design proposed in Section II-A1 is optimal for all different communication patterns. For example, in a one-to-all communication pattern. using RPUT or COOP-p2p would lead to a load-imbalance at the root process. Similarly, for an all-to-one communication, using RGET leads to sub-optimal performance. Hence, the MPI library must consider the overall communication pattern and select the appropriate protocol accordingly. In the next few sections, we describe how multiple processes can cooperate in order to create more advanced dynamic protocols.

1) Communication Primitive Based Approach: The current MPI standard provides no portable way for an application to indicate the overall communication pattern other than using collective operations. Similarly, an application cannot also specify the its expected computation time. However, the MPI library can infer useful information from the actual MPI calls used. For example, when an MPI process uses a blocking send or receive operation, it does not expect to spend any time in computation and its resources (e.g, the CPU) can be used fully to progress that transfer. On the other hand, if a process calls a non-blocking operation, it is reasonable to assume that the process expects to overlap some computation or perform multiple such operations. In fact, most MPI collectives are internally implemented on top of individual point-to-point transfers in this fashion; where blocking and non-blocking point-to-point operations are used to express the dependency or the lack of it between individual communications. For example, MPI_Scatter is usually implemented using multiple non-blocking sends at the root process and a single blocking receive at the non-root processes.

Based on this information, the MPI library can decide which protocol is appropriate for a given transfer. For example, if a sender uses an MPI_Send and the receiver uses MPI_Irecv, the RPUT protocol is selected since it ensures that the sender will perform the copy operation while the receiver will be able to get more overlap. Table II shows the different MPI point-to-point communication primitives and the appropriate protocol for them. For MPI_Send and MPI_Isend, the rules are self-explanatory. MPI_Ssend is functionally equivalent to a blocking MPI_Send. Since MPI_Bsend has an attached buffer, it can return once the data is copied from the user buffer to the intermediate buffer. Since MPI_Rsend can only be called after the receiver operation has been posted, a receiver initiated protocol (RTS) or RPUT is optimal. This approach also works well for most collective operations. For example, in the MPI_Scatter scenario described above, the proposed design correctly selects the RGET protocol. We refer to this design as **COOP-coll**.

TABLE II: Decision Table for Dynamic Protocol Selection

Operation	**MPI_Recv**	**MPI_Irecv**
MPI_Send	COOP	RPUT
MPI_Isend	RGET	COOP
MPI_Bsend	RGET	RPUT
MPI_Ssend	COOP	RPUT
MPI_Rsend	COOP	RTR/RPUT

Since all three protocols considered here are sender-initiated, the sender cannot predict whether a Recv or an Irecv will match any given Send. Thus, the sender cannot unilaterally decide which protocol to use and must receive this information from the receiver. To avoid introducing additional synchronization, the sender sets an additional bit in the RTS packet to indicate whether the operation is a Send or an Isend. Since the format of these control packets are defined by the MPI implementation, this information can be added independent on the network. Once the RTS arrives at the receiver, it is matched with a Recv or an Irecv operation and the decision tree is used to determine the correct protocol. In case of RGET, the receiver performs the read operation and sends the FIN packet to the sender upon completion. In case of RPUT or COOP, the receiver sends the protocol information to the sender using the CTS packet. This protocol guarantees that the sender and the receiver will always decide on the same protocol for a given message without introducing any additional control messages.

2) Load-Balancing Rendezvous Design: While the decision tree-based design described in Section II-B1 works well in most scenarios, it cannot select the correct protocol in certain situations. For example, in case of random or stencil-based communication where the sender uses MPI_Isend and the receiver uses MPI_Irecv, both the sender and the receiver CPU could be used to copy the data. Also, depending on the communication pattern and random skews, one of the CPUs could have more copies to perform than the other. Thus, there is no clear way to predict which rendezvous protocol would be most beneficial in this scenario. To address this challenge, we propose a design to dynamically select the rendezvous protocol based on the "load" of the sender and the receiver. During initialization, a shared memory window is created that can hold one counter for each MPI process on the node. Each counter can be incremented or decremented atomically and holds the number of copy operations each process is expected to perform. These counters are used to determine which rendezvous protocol is going to be used for each transfer. In general, the protocol is selected such that the peer with the smaller counter will perform the actual copy. Once the protocol is selected, the counters for the sender and the receiver are updated accordingly. Since both the sender and the receiver are responsible for copying half of the data in the COOP-p2p design, both their counters are incremented accordingly. The counters are decremented once the copy operation is completed. This design ensures that the load-imbalance between communicating processes is reduced and is referred to as **COOP-load**.

3) Adaptive Offload-based Design: Section II-A2 described how the sender and the receiver process can offload the communication to the HCA to improve performance and overlap. However, the bandwidth of the HCA is limited by the PCI-e bandwidth and is much lower than the system memory bandwidth. Thus, the HCA can quickly get overloaded by multiple concurrent transfers and cause performance degradation. To maximize the HCA utilization while avoiding oversubscription, we propose an adaptive offload-based design that uses additional information exported by the intra-node peers. To achieve this, a shared memory region similar to the one used in the COOP-load design, described in Section II-B1, is used. A shared counter is incremented when a large RDMA read/write operation is posted to the HCA, and decremented when the operation is completed. This counter keeps track of both intra-node (loopback) and inter-node communication requests and indicates the current load on the HCA. If it is higher than a predefined threshold, no intra-node communication is offloaded to the HCA. We also evaluated using InfiniBand hardware counters but our evaluations showed that these counters are not updated instantaneously and cannot be used to determine the instantaneous load on the HCA.

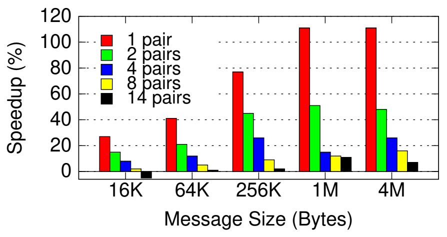


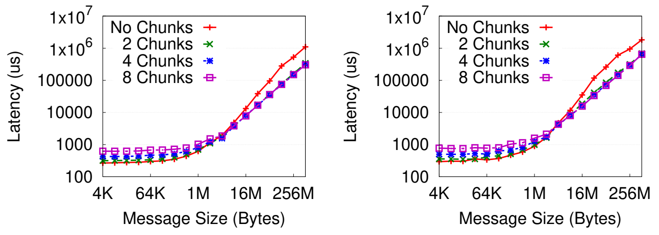
Fig. 9: Speedup of Multi-Pair Bandwidth on Broadwell through using the Adaptive Offload Design. Speedup achieved decreases with increasing number of processes per node.

However, even this adaptive design results in diminishing returns as the number of processes per node increases. Furthermore, since the HCA has to progress both inter-node and intra-node communications with this approach, it can increase the overall communication time significantly for applications running a large number of processes on each node or performing dense communication patterns. This trend can be seen in Figure 9, which compares the speedup obtained from adaptively using the loopback channel in conjunction with shared memory. With increasing number of communicating processes per node, the HCA bandwidth gets saturated quickly and the improvements are reduced. The improvements beyond 100% for 2MB and 4MB messages are due to the loopback being faster than CPU based copies for very large messages.

C. Cooperative Overlap of Intra-/Inter-node Communication

While the selection of the correct protocol can improve the performance of intra-node transfers, this does not mitigate the issue that intra-node copies are blocking in nature. Hence, if the CPU is progressing a large message intra-node transfer, processing of control messages can be delayed until the copy is completed. This can delay a remote process as well as the copying process from achieving maximum overlap. An example of such a scenario was shown in Figure 3. Similar to this situation with a remote process, it can prevent another process on the same node from starting its own transfer as well. This reduction in number of concurrent intra-node transfers or loss of overlap in intra-node and inter-node communication may lead to sub-optimal performance.

In order to alleviate this issue, we allow the MPI library to process incoming control messages even when it is performing some intra-node operation. While threads can be used for this purpose, they can reduce the performance on fully-subscribed systems due to context switching overhead and lock contention issues [24]. Thus, we adopt a chunked copy scheme where large intra-node messages are split into smaller chunks internally. This enables the MPI library to enter the progress engine and process incoming inter-node messages more frequently for improved overlap. We evaluate the performance of this design with a pipelined broadcast algorithm typically used for large messages. As Figure 10 shows, increasing the number of chunks decreases the performance for messages smaller than 4 MB. However, for larger messages, the chunked design enables more overlap and improved the overall performance of the operation. We refer to this design as **COOP-chunked**.



(a) 8 Nodes, 128 Processes (b) 16 Nodes, 256 Processes

Fig. 10: Performance of a Ring-based Broadcast algorithm on Broadwell with the COOP-chunked design

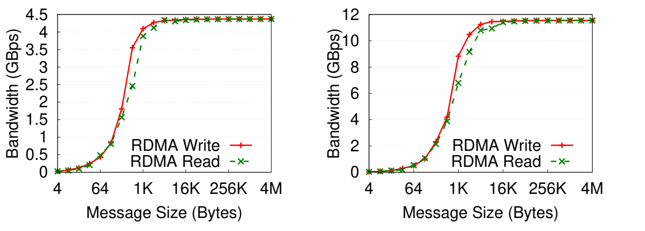
D. Hybrid Cooperative Rendezvous Protocol

Out of the different proposed designs described in the previous sections, some designs are compatible or complementary while some designs are applicable to specific scenarios. For example, the COOP-coll and COOP-load designs use the COOP-p2p protocol as part of the dynamic protocol selection. Similarly, the COOP-chunked design can be applied to any of the existing rendezvous protocols such as RPUT and RGET as well. Based on the individual characteristics of the proposed designs, we combine them intelligently into a single dynamic scheme, which we refer to as **COOP-hybrid**. This design internally uses the previously mentioned designs as applicable. For example, the COOP-coll protocol is used when the application calls Send/Irecv or Isend/Recv, but the COOP-load is used when the application calls Isend/Irecv. Similarly, the hybrid design triggers the COOP-chunked design or the COOP-hca design only when the messages are larger than a certain threshold and the system is under-subscribed. We use this protocol for our application level evaluations in Section V.

III. Applicability of Cooperative Rendezvous Designs to Inter-node Communication

A. Point-to-point communication

The reader may have noticed that we have used the COOP-p2p protocol only within the same node and not across different nodes. A natural question, therefore, is whether the proposed design can be applied to inter-node transfers. However, our evaluations show that the COOP-p2p design does not offer performance improvement for inter-node transfers. This can be explained by the performance trends of current generation HCAs, as shown in Figure 11. On both FDR and EDR HCAs, messages larger than 1 MTU (4KB) can saturate the link bandwidth using a single transfer. Thus, performing multiple concurrent transfers for medium and large messages does not yield any additional benefit. For messages smaller than 4KB, the synchronization overhead from using rendezvous is too high and eager protocols are preferred. However, as the HCAs become faster, it is likely that multiple concurrent transfers would be required to saturate the link and the COOP-p2p design would perform better. We plan to reevaluate these designs when HDR InfiniBand HCAs capable of 200 Gbps become available.

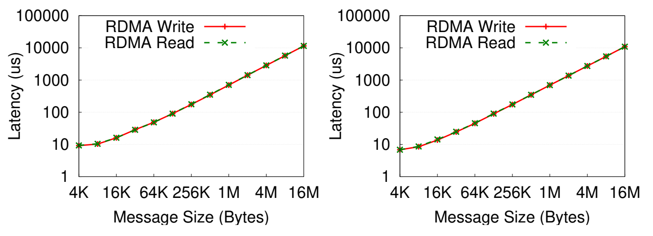


(a) FDR, 56 Gbps (b) EDR, 100 Gbps

Fig. 11: IB-verbs level bandwidth for EDR and FDR HCAs on Broadwell. A single large message transfer can saturate the bandwidth, removing the requirement for using the COOP-p2p protocol.

B. Collective communication

To evaluate the impact of selecting RDMA-read or RDMA-write based protocols on collective communication performance, we compare the latency of one-to-all and all-to-one communication patterns using these protocols. Figure 12 shows the results of these experiments on 8 nodes connected with EDR and running a single process per node to avoid any intra-node communication. As we can see, even for collective communication, the difference in performance of RDMA-read and RDMA-write based protocols is negligible. This again highlights the fact that for inter-node communication the network link is the bottleneck. Consequently, for medium to large messages where rendezvous protocols are applicable, n parallel RDMA-reads from a single node performs similar to n sequential writes. Thus, the COOP-coll design described in Section II-B1 is unlikely to show benefits for inter-node collective communication.



(a) One-to-all (b) All-to-one

Fig. 12: Impact of Rendezvous protocol selection on inter-node transfers on Broadwell+EDR system. RDMA Read vs. Write has negligible impact on the collective performance.

IV. Performance Evaluation - Microbenchmarks

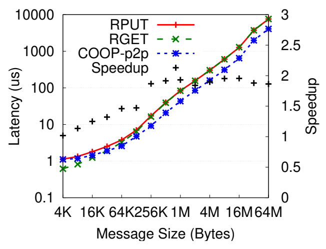
In this section we compare the performance of our proposed designs against the existing protocols used in the MVAPICH2-2.3rc1 MPI library [21] using the OSU Micro Benchmarks suite (OMB) [22]. The average of five runs with 1,000 iterations each are reported. For synthetic benchmarks, the run-to-run variation was minor and ommitted for clarity. Experiments other than point-to-point benchmarks were run using full-subscription of physical cores (28, 64, and 20 processes per node on Broadwell, KNL, and OpenPOWER, respectively). More details about the different platforms used can be found in Section I-B.

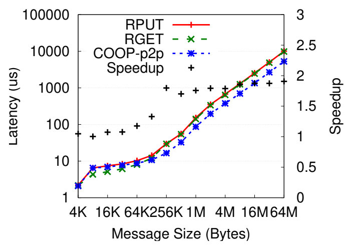
A. Point-to-Point Benchmarks

Figure 13 compares the intra-socket latency of the proposed COOP-p2p protocol against existing protocols such as RPUT and RGET. "Speedup" refers to the improvement obtained over the default RPUT protocol. As shown in Figures 13(a) and 13(c), the COOP-p2p design reduces the latency of point-to-point transfers by up to 2x on Broadwell and OpenPOWER architectures. On KNL, inter-tile latency is improved by up to 2x (shown in Figure 13(b)) while intra-tile latency is improved by up to 1.75x. This difference is due to the fact that two adjacent KNL cores in the same tile shares the same L2 cache while each Xeon core has its own L2 cache. Similar improvements were obtained for bandwidth, as previously shown in Figure 7 for the Broadwell architecture. We do not present the graphs for the other architectures due to space constraints.

B. Collective Benchmarks

As described in Section II-B1, the COOP-coll design selects the best rendezvous protocol based on the overall communication pattern. To evaluate the efficacy of this design, we


(a) Broadwell


(b) KNL

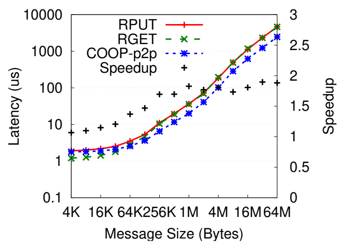
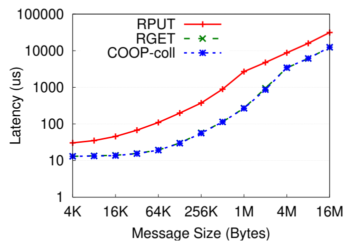
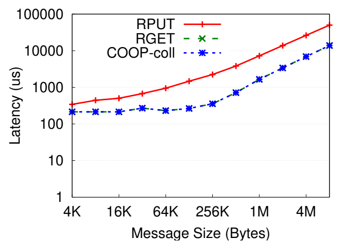

(c) OpenPOWER

Fig. 13: Intra-socket latency with different protocols on different architectures. COOP-p2p performs upto 2x better than RPUT and RGET.


(a) Broadwell, 28 processes


(b) KNL, 64 processes

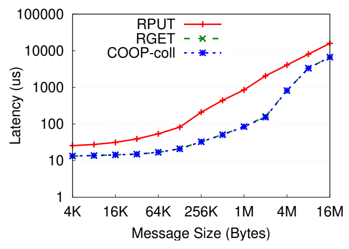
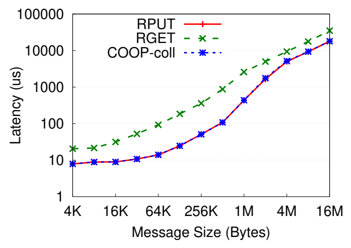
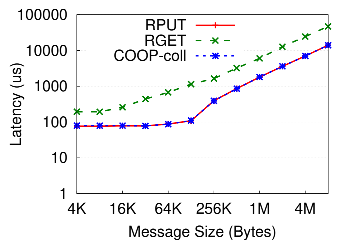

(c) OpenPOWER, 20 processes

Fig. 14: Performance of one-to-all communication pattern on different architectures


(a) Broadwell, 28 processes


(b) KNL, 64 processes

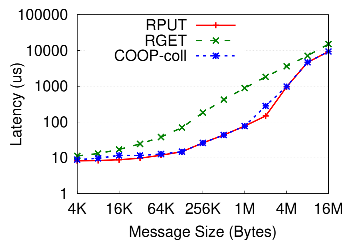

(c) OpenPOWER, 20 processes

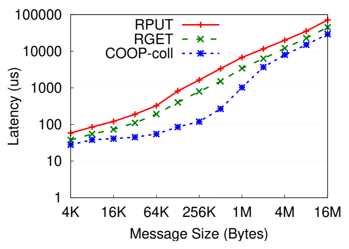
Fig. 15: Performance of all-to-one communication pattern on different architectures

compare it against the existing protocols in three different communication patterns. Figure 14 compares the performance of statically selected RPUT and RGET against the proposed cooperative design for one-to-all communication using MPI_Scatter. Since with RGET all the processes perform a single copy compared to the root performing all the copies with RPUT, RGET significantly outperforms RPUT in this scenario. Similarly, for the all-to-one communication pattern using MPI_Gather, RPUT performs better than RGET, as shown in Figure 15. Figure 16 shows the performance of the MPI_Reduce operation using the Reduce-Scatter-Gather algorithm. This algorithm is commonly used for large messages and uses both one-to-all and all-to-one communication patterns. Due to this, neither RPUT nor RGET can provide the best performance in both the phases. However, the proposed COOP-coll design is able to dynamically select the best protocol based on the communication pattern and outperforms both. Note that the percentage improvements are smaller as the numbers shown here also include the time taken for the actual reduction operation, which is unaffected by the rendezvous protocol. These experiments show that the proposed design is able to detect and react to different communication patterns and provide the best performance.

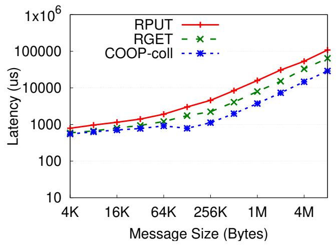
C. *Stencil Benchmark*

Stencil codes update an array of elements by running a kernel in an iterative manner. In each iteration, according to a fixed pattern, new data is received and is used in the kernel to update the elements. In the 3D-Stencil benchmark, a 7 point stencil is used, which requires exchanging data with six neighboring processes. Figure 17(a) shows the communication latency reported with 1MB messages for the different number of processes with 28 processes per node. The default design based on RPUT is compared against the cooperative load-balancing design (COOP-load) described in Section II-B2. As shown here, the cooperative design outperforms the default design by up to 10% and scales well with increasing number of processes.

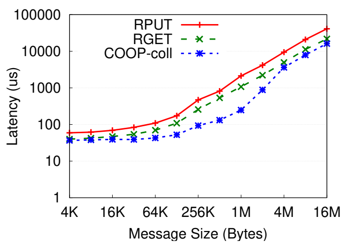
To further evaluate the impact of the COOP-load design, we compare the number of kernel-assisted copy operations performed by different ranks on a 64 process run with 1 MB message size and 8 processes per node. Although each process performs the same number of communication (ignoring the boundary processes), the amount of intra-node communication is not the same across all the processes. This is due to each process having a mix of intra-node and inter-node peers. This trend results in a load-imbalance in the processes, as depicted



(a) Broadwell, 28 processes

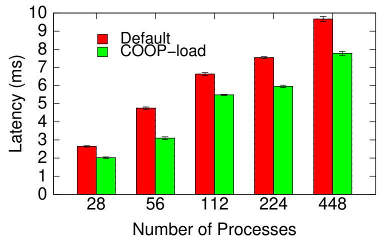


(b) KNL, 64 processes

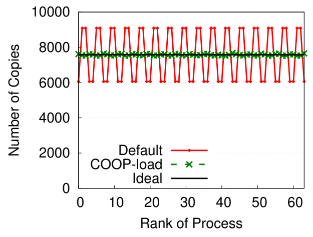


(c) OpenPOWER, 20 processes

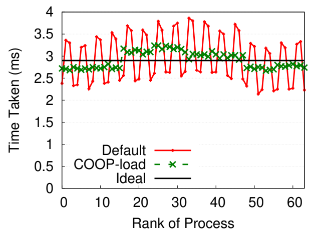
Fig. 16: Performance of Reduce with Reduce-Scatter+Gather algorithm on different architectures



(a) Communication latency with 1MB messages



(b) Number of copy operations performed by each rank



(c) Time spent copying by each rank

Fig. 17: Performance and analysis of 3dstencil benchmark on Broadwell. COOP-load reduces the load-imbalance and overall execution time by dynamically selecting the rendezvous protocol for each transfer.

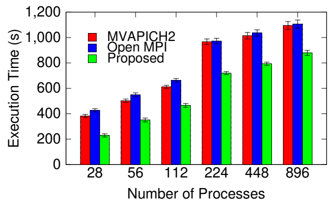
in Figure 17(b). Similarly, Figure 17(c) shows the imbalance in total time taken by the copy operations across processes, which results in under-utilization of the CPU resources and sub-optimal performance. However, the proposed COOP-load design is able to dynamically detect and mitigate this load imbalance by selecting different rendezvous protocols for different messages, improving the overall performance. "Ideal" represents the best achievable performance if all the future transfers and the time consumed by them is known ahead of time. It is calculated by averaging the number of copies and the time spent in copying across all ranks. The proposed design comes very close (within $\pm 1\%$) to this best case scenario in terms of number of copies. In terms of time spent in copying, individual copies can take more or less time due to skews and system noise, leading to a slightly larger difference ($\pm 6\%$) between the proposed design and the theoretical best. Note that Figures 17(b) and 17(c) show the outcome of a single run to highlight the difference between theoretical and actual performance. Averaging multiple runs would hide these differences since they are randomly distributed across ranks for each run.

V. Performance Evaluation - Applications

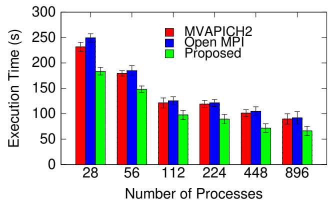
We evaluate three HPC applications - Graph500, CoMD, and MiniGhost to show the impact of the proposed designs on overall application performance. These experiments were done on the Broadwell cluster with full subscription (28 processes per node). The proposed COOP-hybrid protocol described in Section II-D is compared against MVAPICH2-2.3rc1 and Open MPI v3.1.0 [25]. Average and standard deviation of the execution time over five runs are reported. Relative improvements are calculated compared to MVAPICH2. More details about problem sizes and other parameters can be found in the Artifact Description.

A. Graph Processing: Graph500

The Graph500 Benchmark [26, 27] has two kernels, the first kernel is responsible for generating a large undirected graph, and the second kernel performs a Breadth-first search. Graph500 is a data-intensive application and is commonly used to benchmark the performance of many HPC clusters. We use the reference implementation of the Graph500 benchmark for our evaluations.



(a) Weak Scaling



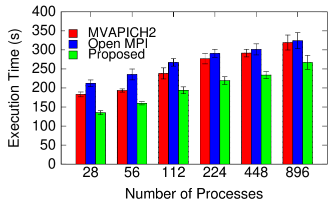
(b) Strong Scaling

Fig. 18: Execution time of Graph500 on Broadwell

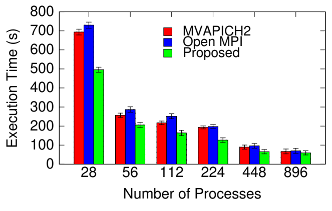
Figure 18(a) shows the results of the weak-scaling experiments with Graph500 on Broadwell. The number of vertices was increased proportional to the increasing number of nodes, varying from 2^{25} to 2^{30}. The default edge factor of 16 was used. Compared to 1095 seconds and 1106 seconds taken by MVAPICH2 and Open MPI respectively, the proposed COOP-hybrid design reduces the execution time to 879 seconds, representing an improvement of 19% at 896 processes. The strong scaling experiment also showed similar improvements, as illustrated in Figure 18(b).

B. Molecular Dynamics: CoMD

CoMD [28] was developed at the Sandia National Laboratories as a part of the Exascale Computing Project (ECP) proxy applications. CoMD implements various classical molecular dynamics algorithms and workloads, and serves as a proxy for the SPaSM [29] application. Figures 19(a) and 19(b) show the performance of CoMD with weak and strong scaling parameters, respectively. With weak scaling, the execution time of CoMD on 896 processes is improved by 16% from 319 seconds with MVAPICH2 and 324 seconds with Open MPI to 267 seconds. For the strong scaling experiment, we observed up to 11% benefit with 896 processes.



(a) Weak Scaling

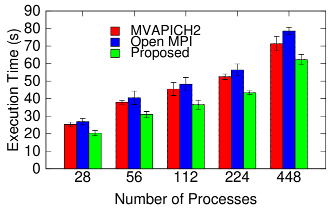


(b) Strong Scaling

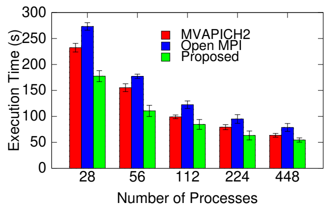
Fig. 19: Execution time of CoMD on Broadwell

C. Halo-Exchange: MiniGhost

MiniGhost [30] is part of the Mantevo applications suite [31]. It represents a three-dimensional nearest-neighbor halo-exchange communication pattern present in many HPC applications such as CTH [32]. Our evaluation was performed using a 2-dimensional 5-point stencil and the default BSPMA method which exchanges information with the neighbors using large messages. As shown in Figure 20(a) and 20(b), MiniGhost shows improvements of 10% and 12% in overall execution time with 448 processes with weak and strong scaling respectively. Since MiniGhost is a stencil application, it shows similar benefits as the 3DStencil kernel as shown in Section IV-C.



(a) Weak Scaling



(b) Strong Scaling

Fig. 20: Execution time of MiniGhost on Broadwell

VI. Related Work

A number of studies have been conducted to improve the performance of MPI point-to-point communication that is based on rendezvous protocols. Small et al. [14] proposed a set of protocols based on the process arrival patterns and timing of the Isend/Irecv and Wait calls. However, their proposed design relies on application traces to select which protocol should be used. Compared to this, our work relies on the communication primitives used and considers the CPU utilization of different processes to dynamically adapt to the application requirements. This makes our proposed designs more practical and applicable to a wider variety of applications and systems. Kernel-assisted rendezvous transfer mechanisms and intra-node collectives have been explored by many researchers as well [33–37].

Subramoni et al. [38] proposed a dynamic design which adapts to the communication pattern of each communicating pair during the runtime by personalizing the eager threshold for each pair. During the runtime, some pairs can decide to increase the threshold and use eager protocol for larger message sizes to minimize the progress time. Takagi et al. [39] focuses on optimizing the processor-device communications which in the conventional rendezvous protocols is conducted using PCI bus as a link to poll for completion of RDMA transfer. There have been several research studies to take advantage of asynchronous progress engine to increase the overlap of communication and computation [40–43]. Gu et al. [15] complements [14] by proposing a trace-driven protocol customization based on protocols defined in [14] to allow the appropriate protocol to be selected for each communication in a static manner. Other researchers have explored methods to tune different parameters of the MPI library in an offline manner to maximize the performance [12, 13, 44–46]. However, unlike these, our proposed design dynamically selects the best protocol during the runtime and does not need any offline tuning.

VII. Conclusion and Future Work

In this paper, we investigated the performance and overlap characteristics of intra-node rendezvous protocols and identified several inefficiencies in existing designs. To tackle these inefficiencies, we proposed a set of cooperative and dynamic rendezvous protocols that allow sender and receiver processes to share resources and information in order to improve the performance and overlap of point-to-point transfers. We also proposed cooperative designs that take into account the overall communication pattern and dynamically adapt to it. The proposed designs led to better load balancing among participant processes and improved overlap of intra-node and inter-node communication. With the proposed designs, we were able to improve the intra-node latency and bandwidth of large messages by up to 2x on three different architectures including Intel Xeon, Xeon Phi (KNL), and OpenPOWER. The proposed designs also showed significant improvement in the performance of different collective operations on these platforms. We evaluated the proposed designs on a set of representative applications including Graph500, CoMD, and MiniGhost and obtained improvements of up to 19%, 16%, and 10% respectively. Going forward, we plan to evaluate the proposed designs on a more diverse set of applications on larger scale clusters. We also plan to add support for MPI_T control variables (CVARs) to allow applications and external tools to provide additional hints for dynamic selection of rendezvous protocols.

References

[1] "MPI-3 Standard Document," http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf, 2012.

[2] R. Rabenseifner, "Automatic Profiling of MPI Applications with Hardware Performance Counters," in *European Parallel Virtual Machine/Message Passing Interface Users Group Meeting*. Springer, 1999, pp. 35–42.

[3] K. B. Ferreira, P. Bridges, and R. Brightwell, "Characterizing Application Sensitivity to OS Interference using Kernel-level Noise Injection," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 2008, p. 19.

[4] HPC Advisory Council, http://www.hpcadvisorycouncil.com/best_practices.php.

[5] T. Hoefler, P. Gottschling, A. Lumsdaine, and W. Rehm, "Optimizing a Conjugate Gradient Solver with Non-blocking Collective Operations," *Parallel Computing*, vol. 33, no. 9, pp. 624–633, 2007.

[6] T. Hoefler, P. Gottschling, and A. Lumsdaine, "Leveraging Non-blocking Collective Communication in High-performance Applications," in *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*. ACM, 2008, pp. 113–115.

[7] K. Kandalla, H. Subramoni, K. Tomko, D. Pekurovsky, S. Sur, and D. K. Panda, "High-performance and Scalable Non-blocking All-to-all with Collective Offload on InfiniBand Clusters: a Study with Parallel 3D FFT," *Computer Science-Research and Development*, vol. 26, no. 3-4, p. 237, 2011.

[8] J. Liu, J. Wu, and D. K. Panda, "High Performance RDMA-based MPI Implementation over InfiniBand," *International Journal of Parallel Programming*, vol. 32, no. 3, pp. 167–198, 2004.

[9] S. Sur, H.-W. Jin, L. Chai, and D. K. Panda, "RDMA Read based Rendezvous Protocol for MPI over InfiniBand: Design Alternatives and Benefits," in *Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 2006, pp. 32–39.

[10] M. J. Rashti and A. Afsahi, "Improving Communication Progress and Overlap in MPI Rendezvous Protocol over RDMA-enabled Interconnects," in *High Performance Computing Systems and Applications, 2008. HPCS 2008. 22nd International Symposium on*. IEEE, 2008, pp. 95–101.

[11] Rashti, Mohammad J and Afsahi, Ahmad, " A Speculative and Adaptive MPI Rendezvous Protocol over RDMA-enabled Interconnects," *International Journal of Parallel Programming*, vol. 37, no. 2, pp. 223–246, 2009.

[12] S. Pellegrini, R. Prodan, and T. Fahringer, "Tuning MPI Runtime Parameter Setting for High Performance Computing," in *Cluster Computing Workshops (CLUSTER WORKSHOPS), 2012 IEEE International Conference on*. IEEE, 2012, pp. 213–221.

[13] G. E. Fagg, J. Pjesivac-Grbovic, G. Bosilca, T. Angskun, J. Dongarra, and E. Jeannot, "Flexible Collective Communication Tuning Architecture Applied to Open MPI," in *Euro PVM/MPI*, 2006.

[14] M. Small and X. Yuan, "Maximizing MPI Point-to-point Communication Performance on RDMA-enabled Clusters with Customized Protocols," in *Proceedings of the 23rd International Conference on Supercomputing*, ser. ICS '09. New York, NY, USA: ACM, 2009, pp. 306–315. [Online]. Available: http://doi.acm.org/10.1145/1542275.1542320

[15] Z. Gu, M. Small, X. Yuan, A. Marathe, and D. K. Lowenthal, "Protocol Customization for Improving MPI Performance on RDMA-Enabled Clusters," *International Journal of Parallel Programming*, vol. 41, no. 5, pp. 682–703, Oct 2013.

[16] Vienne, Jerome, "Benefits of Cross Memory Attach for MPI libraries on HPC Clusters," in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*. ACM, 2014, p. 33.

[17] K. Pedretti and B. Barrett, "XPMEM: Cross-Process Memory Mapping."

[18] Goglin, Brice and Moreaud, Stephanie, "KNEM: A Generic and Scalable Kernel-assisted Intra-node MPI Communication Framework," *Journal of Parallel and Distributed Computing*, vol. 73, no. 2, pp. 176–188, 2013.

[19] J. A. Zounmevo and A. Afsahi, "An Efficient MPI Message Queue Mechanism for Large-scale Jobs," in *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*. IEEE, 2012, pp. 464–471.

[20] M. Bayatpour, H. Subramoni, S. Chakraborty, and D. K. Panda, "Adaptive and Dynamic Design for MPI Tag Matching," in *Cluster Computing (CLUSTER), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–10.

[21] "MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE," http://mvapich.cse.ohio-state.edu/, 2018.

[22] Network Based Computing Laboratory, "OSU Micro-Benchmarks," http://mvapich.cse.ohio-state.edu/benchmarks, 2018.

[23] K. Vaidyanathan, L. Chai, D. K. Panda, and W. Huang, "Efficient Asynchronous Memory Copy Operations on Multi-core Systems and I/OAT," in *2007 IEEE International Conference on Cluster Computing(CLUSTER)*, vol. 00, 09 2007, pp. 159–168. [Online]. Available: doi.ieeecomputersociety.org/10.1109/CLUSTR.2007.4629228

[24] A. Amer, H. Lu, Y. Wei, P. Balaji, and S. Matsuoka, "MPI+Threads: Runtime Contention and Remedies," in *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP 2015. New York, NY, USA: ACM, 2015, pp. 239–248. [Online]. Available: http://doi.acm.org/10.1145/2688500.2688522

[25] Open MPI : Open Source High Performance Computing, http://www.open-mpi.org, 2018.

[26] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, "Introducing the Graph 500," *Cray Users Group (CUG)*, 2010.

[27] M. Li, X. Lu, S. Potluri, K. Hamidouche, J. Jose, K. Tomko, and D. K. Panda, "Scalable Graph500 design with MPI-3 RMA," in *Cluster Computing (CLUSTER), 2014 IEEE International Conference on*. IEEE, 2014, pp. 230–238.

[28] (2018) CoMD: Classical Molecular Dynamics Proxy Application. [Online]. Available: hhttps://github.com/ECP-copa/CoMD

[29] K. Kadau, T. C. Germann, and P. S. Lomdahl, "Large-scale molecular-dynamics simulation of 19 billion particles," *International Journal of Modern Physics C*, vol. 15, no. 01, pp. 193–201, 2004.

[30] R. F. Barrett, C. T. Vaughan, and M. A. Heroux, "MiniGhost: A Miniapp for Exploring Boundary Exchange Strategies Using Stencil Computations in Scientific Parallel Computing," *Sandia National Laboratories, Tech. Rep. SAND*, vol. 5294832, 2011.

[31] Heroux, Michael A and Doerfler, Douglas W and Crozier, Paul S and Willenbring, James M and Edwards, H Carter and Williams, Alan and Rajan, Mahesh and Keiter, Eric R and Thornquist, Heidi K and Numrich, Robert W, "Improving performance via mini-applications," *Sandia National Laboratories, Tech. Rep. SAND2009-5574*, vol. 3, 2009.

[32] E. Hertel, R. Bell, M. Elrick, A. Farnsworth, G. Kerley, J. McGlaun, S. Petney, S. Silling, P. Taylor, and L. Yarrington, "CTH: A Software Family for Multi-dimensional Shock Physics Analysis," in *Shock Waves@ Marseille I*. Springer, 1995, pp. 377–382.

[33] S. Moreaud, B. Goglin, R. Namyst, and D. Goodell, "Optimizing MPI Communication within Large Multicore Nodes with Kernel Assistance," in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–7.

[34] T. Ma, G. Bosilca, A. Bouteiller, and J. J. Dongarra, "Locality and Topology Aware Intra-node Communication among Multicore CPUs," in *European MPI Users' Group Meeting*. Springer, 2010, pp. 265–274.

[35] S. Chakraborty, H. Subramoni, and D. K. Panda, "Contention-Aware Kernel-Assisted MPI Collectives for Multi-/Many-Core Systems," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept 2017, pp. 13–24.

[36] T. Ma, G. Bosilca, A. Bouteiller, B. Goglin, J. M. Squyres, and J. J. Dongarra, "Kernel Assisted Collective Intra-node MPI Communication among Multi-core and Many-core Cpus," in *Parallel Processing (ICPP), 2011 International Conference on*. IEEE, 2011, pp. 532–541.

[37] T. Ma, G. Bosilca, A. Bouteiller, and J. Dongarra, "HierKNEM: An Adaptive Framework for Kernel-assisted and Topology-aware Collective Communications on Many-core Clusters," in *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. IEEE, 2012, pp. 970–982.

[38] H. Subramoni, S. Chakraborty, and D. K. Panda, "Designing Dynamic and Adaptive MPI Point-to-Point Communication Protocols for Efficient Overlap of Computation and Communication," in *High Performance Computing*. Cham: Springer International Publishing, 2017, pp. 334–354.

[39] M. Takagi, Y. Nakamura, A. Hori, B. Gerofi, and Y. Ishikawa, "Revisiting Rendezvous Protocols in the Context of RDMA-capable Host Channel Adapters and Many-core Processors," in *Proceedings of the 20th European MPI Users' Group Meeting*, ser. EuroMPI '13. New York, NY, USA: ACM, 2013, pp. 85–90. [Online]. Available: http://doi.acm.org/10.1145/2488551.2488571

[40] C. Keppitiyagama and A. Wagner, "Asynchronous MPI Messaging on Myrinet," in *Parallel and Distributed Processing Symposium., Proceedings 15th International*. IEEE, 2001, pp. 8–pp.

[41] R. Kumar, A. R. Mamidala, M. J. Koop, G. Santhanaraman, and D. K. Panda, "Lock-free Asynchronous Rendezvous Design for MPI Point-to-point Communication," in *European Parallel Virtual Machine/Message Passing Interface Users Group Meeting*. Springer, 2008, pp. 185–193.

[42] S. Majumder, S. Rixner, and V. S. Pai, "An Event-driven Architecture for MPI libraries," in *The Los Alamos Computer Science Institute Symposium*, 2004.

[43] V. Tipparaju, G. Santhanaraman, J. Nieplocha, and D. K. Panda, "Host-assisted Zero-copy Remote Memory Access Communication on InfiniBand," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, 2004, p. 31.

[44] A. Sikora, E. César, I. Comprés, and M. Gerndt, "Autotuning of MPI Applications using PTF," in *Proceedings of the ACM Workshop on Software Engineering Methods for Parallel and High Performance Applications*. ACM, 2016, pp. 31–38.

[45] A. Pimenta, E. Cesar, and A. Sikora, "Methodology for MPI Applications Autotuning," in *Proceedings of the 20th European MPI Users' Group Meeting*. ACM, 2013, pp. 145–146.

[46] S. Benkner, F. Franchetti, H. M. Gerndt, and J. K. Hollingsworth, "Automatic Application Tuning for HPC Architectures," in *Dagstuhl Reports*, vol. 3, no. 9. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

Appendix

A. Abstract

The artifact contains a software package that we have developed for this work. The software implements the Cooperative rendezvous designs described in the paper on top of the open-source MPI library MVAPICH2. This software can also use Mallanox InfiniBand network adapters.

B. Description

1) Check-list (artifact meta information):

- **Algorithms: Hybrid Cooperative Rendezvous Protocol**
- **Program: COOP MPI library (based on MVAPICH2-2.3rc1)**
- **Binary: C executables for 3DStencil, Graph500, CoMD, MiniGhost, and OSU Micro Benchmarks**
- **Run-time environment: RHEL7, CentOS7**
- **Hardware: Intel Xeon / Xeon Phi / OpenPOWER CPU, InfiniBand Network**
- **Output: Text output of the different applications**
- **Experiment workflow: Download the tarball and install the optimized MPI library. Build the benchmarks and applications with the optimized MPI library. Then run tests with different modes and process counts**
- **Experiment customization: Change the input file of the application under test and set the desired mode of optimization**
- **Publicly available?: Yes**

2) How software can be obtained (if available): The proposed designs will be made available as part of the MVAPICH2 MPI library. It can be obtained from the following URL: `http://mvapich.cse.ohio-state.edu/downloads/`

3) Hardware dependencies: As evidenced by the experiments, the proposed designs are independent of the CPU and the network architecture and can work on a wide array of systems. While in this paper we only evaluate InfiniBand networks, the designs can be easily extended to other RDMA capable networks such as Intel Omni-Path.

4) Software dependencies: As the proposed designs are implemented on top of the MVAPICH2 MPI library, its software dependencies such as InfiniBand verbs library (libibverbs), CMA, XPMEM, etc. are also inherited by the provided software.

C. Installation

Instructions to install and setup the MPI library can be found in the included README. More details can be found in the Useguide located at the following URL: `http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.3-userguide.html`

After installing the COOP MPI library, the applications need to be installed against this library. To do so, mpicc and mpicxx flags in the Makefile of these applications should be pointed to the installation of the MPI libraries. After that, these applications need to be configured and compiled. For example, the OSU Micro-Benchmarks suite is available for download at: `http://mvapich.cse.ohio-state.edu/benchmarks/`

It should be configured and installed as follows:

```
$ ./configure --prefix=path/to/install
CC=mpicc CXX=mpicxx $ make; make install
```

D. Experiment workflow

For these experiments, a script will be available that sets the proper environment variables to enable the optimization that one would like to use by following commands:

```
$ ./run_test.sh /path/to/app/binary
-protocol <RPUT/RGET/AUTO>
```

Other combinations can be tested similarly.

E. Evaluation and expected result

As the results of the jobs are redirected to sdtout, one can find the results in the output file provided by the job scheduler. The results file contains the latency numbers for a different set of the experiments. Graph500 reports performance numbers as traversed edges per second (TEPS). CoMD and MiniGhost generates `.yaml` files that contains detailed performance statistics. Total execution time can also be compared using the Linux `time` binary. In general, one can expect to see equal or better performance with the hybrid protocol compared to other choices such as RPUT or RGET.

F. Experiment customization

Here are the inputs used for our evaluations of the applications used in the paper:

Graph500:

```
$ $MPI/bin/mpirun_rsh -np $NPROCS
-hostfile $HOSTFILE $ENV graph500_simple
<SIZE>
```

Graph500 uses strong scaling by default. There are two relevant parameters. Number of vertices are calculated as 2^{SIZE}. NUmber of vertices are calculated from the edge factor. We used the default edge factor of 16. For weak scaling, SIZE was varied from 25 to 30+ based on the number of processes.

CoMD:

```
$ $MPI/bin/mpirun_rsh -np $NPROCS
-hostfile $HOSTFILE $COMMON $ENV CoMD-mpi
-e -i $NPX -j $NPY -k $NPZ -x $NX -y $NY
-z $NZ --nSteps 5000
```

CoMD uses strong scaling by default. NY and NZ was fixed to 100 while NX was set as 100*Nodes to achieve weak scaling. The included example potential files (Cu_u6.eam, Cu01.eam.alloy) were used.

MiniGhost:

```
$ $MPI/bin/mpirun_rsh -np $NPROCS
-hostfile $HOSTFILE $COMMON MiniGhost.x
--npx $NPX --npy $NPY --npz $NPZ --nx
$NX --ny $NY --nz $NZ --percent_sum
0 --num_vars 1000 --stencil 21
--report_diffusion 0 --report_perf 0
--num_tsteps 100 --num_spikes 1
```

MiniGhost uses weak scaling by default. NPX was chosen as the number of nodes. NPY and NPZ were fixed as 7 and 4 to match with 28 ppn. Parameter size $< 160, 70, 40 >$ was used for strong scaling experiments.