2016 PGAS Applications Workshop

OpenSHMEM NonBlocking Data Movement Operations with MVAPICH2-X: Early Experiences*

Khaled Hamidouche, Jie Zhang, and Dhabaleswar K. (DK) Panda
Department of Computer Science and Engineering
The Ohio State University
Email: {hamidouche.2, zhang.2794, panda.2}@osu.edu

Karen Tomko
Ohio Supercomputer Center
Email: ktomko@osc.edu

Abstract—**PGAS models with a lightweight synchronization and shared memory abstraction, are seen as a good alternative to the Message Passing model for irregular communication patterns. OpenSHMEM is a library based PGAS model. OpenSHMEM 1.3 introduced Non-Blocking data movement operations to provide better asynchronous progress and overlap. In this paper, we present our experiences in designing Non-Blocking Put and Get operations on InfiniBand systems. Using the MVAPICH2-X runtime, we present the alternative designs for intra-node and inter-node operations. We also present a set of new benchmarks to analyze the latency, message rate performance, and communication/computation overlap benefits. The performance evaluation shows 7X improvement in the message rate. Furthermore, using a 3D-Stencil based application kernel, we assess the benefits of OpenSHMEM Non-Blocking extensions. We show 50% and 28% improvement on 27 and 64 processes, respectively.**

Index Terms—**Computers and information processing, Computer science, Programming, Parallel Programming**

I. Introduction

Data-driven applications pose challenges associated with load balancing and exhibit irregular communication patterns. These issues are harder to address with a traditional message-passing programming (MPI) paradigm. The Partitioned Global Address Space (PGAS) programming models are seen as an attractive alternative approach to MPI and are believed to improve programmability and performance of such applications. PGAS models can be classified as: 1) Language-based like Unified Parallel C (UPC) [32], Titanium [9], and Co-array Fortran (CAF) [23] which have been undergoing standardization for over a decade now. 2) Library-based like OpenSHMEM [1] where a recent effort has been put together by the community to standardize the API for the different vendor specific implementations of SHMEM. The OpenSHMEM standard is gaining attention as it allows existing codes that were written using vendor-specific SHMEM APIs to be made platform-independent with minimal effort. It is also seen as an alternative to PGAS languages for designing new applications.

This research is supported in part by National Science Foundation grants #OCI-1148371, #CCF-1213084, #CNS-1419123 and #CCF-1565414.

The OpenSHMEM model provides an API for memory allocation and management as well as data movement operations and synchronizations including point-to-point one-sided, and collective operations. Traditionally, the data movement operations have been blocking. Blocking communication means that the buffer can be reused once we return from the call. This requires the runtime to ensure the transmission of the buffer (or perform an internal copy) before returning the control to the application. Hence it limits the potential for computation/communication overlap. On the other hand, researchers have shown that blocking communication (e.g. in MPI) wastes CPU cycles [10]. Non-blocking communication provides an efficient alternative method of designing applications where independent computation is allowed to overlap the communication. Hence, to alleviate these limitations, the latest OpenSHMEM 1.3 Specification [1] introduced Non-blocking put/get data movement operations.

High performance networks with RDMA capabilities provide a hardware level support to efficiently and asynchronously perform the data movement. Several implementations of OpenSHMEM including the UH reference implementation [25], MVAPICH2-X [22], [26], OpenMPI [24] and HPC-X [11] provide efficient support for blocking operations on InfiniBand networks.

A. Motivation

The primary motivation for Non-Blocking (NBI) operations is to maximize the overlap while restricting the communication latency to acceptable levels. While most implementations for blocking data movement are optimized for latency, non-blocking operations add overlap as a new metric to measure their performance and efficiency. Furthermore, different alternatives can be exploited to design OpenSHMEM communication and synchronization operations on multi-core nodes [6], [29] taking advantage of shared memory and kernel-based channels.

In this paper, we investigate designs of Non-blocking (NBI) data movement operations introduced with the OpenSHMEM 1.3 specification. Designed and build on top of the MVAPICH2-X runtime, which is a high-performance implementation of OpenSHMEM [17], we propose efficient designs for *shmem_put*_nbi* and *shmem_get*_nbi* operations for both intra-node and inter-node configurations.

To summarize, we make the following key contributions in this paper.

- Propose high-performance designs and implementations of OpenSHMEM NBI operations on top of the MVAPICH2-X library.
- Propose and implement new NBI benchmarks for evaluating OpenSHMEM 1.3 NBI operations in a standardized manner.
- Design communication kernels including 3D stencil and alltoall patterns using OpenSHMEM.
- Demonstrate the benefits and impact of OpenSHMEM NBI operations on both latency and overlap metrics.

The rest of the paper is organized as follows. Relevant background information is provided in Section II. Design and implementation details of OpenSHMEM NBI operations are discussed in Section III. Proposed NBI benchmarks and communication kernels for evaluating overlap and latency of NBI operations in OpenSHMEM are described in Section IV. Comprehensive performance evaluation is presented in Section VI. Section VII outlines related work. Section VIII contains the conclusion and future work.

II. Background

In this section, we provide the necessary background information for this paper.

A. PGAS Models and OpenSHMEM:

In Partitioned Global Address Space (PGAS) programming models, each Processing Element (PE) has access to its own private local memory and a global shared memory space. The locality of the global shared memory is well defined. Such a model allows for better programmability through a simple shared memory abstraction while ensuring performance by exposing data and thread locality. SHMEM (SHared MEMory) [30] is a library-based approach to realize the PGAS model and offers one-sided point-to-point communication operations, along with collective and synchronization primitives. SHMEM also offers primitives for atomic operations, managing memory, and locks. There are several implementations of the SHMEM model that are customized for different platforms. However, these implementations are not portable due to minor variations in the API and semantics. OpenSHMEM [25] aims to create a new, open specification to standardize the SHMEM model to achieve performance, programmability, and portability.

B. Data Movement Operations in OpenSHMEM:

The OpenSHMEM Specification [25] defines several types of communication operations — data transfer, atomics, and collective communication operations. In this section, we focus on data movement operations.

Data transfer operations defined in OpenSHMEM 1.2 consist of `shmem_put` and `shmem_get`, and their variants. The source/ destination address of data transfer operations can either be in the symmetric heap or symmetric static memory, as defined in the OpenSHMEM specification. `shmem_put` writes the local data to the corresponding data objects of the target process. `shmem_get` fetches the data from a remote process and stores it in the local data object. These operations are blocking which means that after returing from the call the buffer can be reused. In other words the application does not get control back until it is safe to reuse the buffer. On the other hand, high performance networks with their RDMA-capabilities, offer asynchronous data movement, in order to provide opportunities for computation/communication overlap. To take advantage of such features, the OpenSHMEM 1.3 specification introduced support for Non-Blocking variants for the data movement operations (`shmem_put_nbi` and `shmem_get_nbi`). These operations have the same semantics and parameters as the blocking versions except that the buffer cannot be reused until the completion of the operation. To ensure completion, the user is required to use synchronization operations such as `shmem_quiet` or `shmem_barrier`. `shmem_quiet` ensures the completion of all previously issued non-blocking operations.

C. MVAPICH2-X OpenSHMEM runtime

MVAPICH2-X [22], [26] (in short MV2-X) provides a unified high-performance communication runtime, UCR, that supports both MPI and PGAS programming models on InfiniBand clusters. It enables developers to port parts of large MPI applications that are suited for the PGAS programming model. This minimizes the development overheads that have been a substantial deterrent in porting MPI applications to PGAS models. The unified runtime also delivers superior performance compared to using separate MPI and PGAS libraries by optimizing the use of network and memory resources [13], [14]. MV2-X provides support for the reference implementation of OpenSHMEM. Currently, it is based on the University of Houston's v10.h OpenSHMEM reference implementation.

III. Design of Non-blocking Data Movement Operations for OpenSHMEM

In this section, we will describe the alternative designs and schemes to provide efficient Non-Blocking operations for both intra- and inter-node configurations.

A. Efficient Intra-node Designs

Using the UCR layer of MVAPICH2-X, we analyze two alternatives to provide efficient intra-node non-blocking operations. For multi-core architectures, most of the OpenSHMEM runtimes including MVAPICH2-X provide a shared memory channel. In order to provide a Zero-copy based intra-node communication, depending on the message size and the placement of the heap allocation (shared memory or local memory), two schemes can be used:

- **Shared Memory:** For a configuration where the heap is mapped to shared memory, the origin process performs a direct copy.

- **Cross Memory Attach (CMA):** For a configuration where the heap is not in shared memory, ie the heap is allocated in local memory for each process, MVAPICH2-X uses the CMA kernel support to provide a zero-copy design. For instance, the CMA based design is useful when the heap is very lage and cannot be mapped in shared memory. During the allocation (shmem_malloc), the processes map the data and exchange the CMA handles. Using these handles, processes can copy data directly. However, CMA-based design has overhead for small messages, hence MV2-X falls back to traditional shared memory design for small messages.

As the above schemes are zero-copy based, they are blocking by nature. While they provide a very low latency, their overlap is limited. In order to provide overlap, we propose a third scheme, where we force the intra-node communication to use the inter-node scheme. This is known as a LoopBack-based (LB) scheme. In order to be able to perform an IB-based loopback, the intra-node processes are required to perform a memory registration and exchange the r_key and l_key. Online registration and caching are possible, however as it adds overhead on the critical path, we move the exchange of the IB keys to the allocation phase. Thus in addition to exchanging the CMA handles, intra-node processes register and exchange the IB keys, during the allocation phase.

While the selection between CMA and Shared memory channels is automatically performed by MV2-X depending on tuning parameters, the selection of the loopback design to achieve better overlap requires intervention by the users. In non-blocking operations, as overlap is the primary goal, the LB scheme is set to be the default.

We note that similar to using CMA, one can use other kernel-based modules like Knem [21] and Limic2 [12]. Furthermore, unlike CMA, these modules offer asynchronous copy schemes that can provide better overlap with minimal latency. We are currently working on providing such asynchronous support with Limic kernel and MVAPICH2-X. However, CMA has the advantage that it is production-ready as it is part of the Linux kernel.

B. Efficient Inter-node Designs

In order to design Non-Blocking operations on InfiniBand, the main task is ensuring completion. In other words, how to enhance *shmem_quiet*? Indeed, as IB operations are asynchronous, the *shmem_put_nbi* and *shmem_get_nbi* operations consist only of posting the IB operation. After creating the Work Element, the process just posts the operation. To ensure the completion of the operation, we need to poll on the completion queue. However, in a unified runtime, like MVAPICH2-X, the polling for completion is performed by the progress engine. The progress engine is common to the different operations at the higher level, it can be for an MPI, OpenSHMEM or UPC operation. Therefore, a completion of a non-blocking put operation can be pulled out by an MPI operation and not the Quiet operation. Hence, we need to keep track of the nonblocking operations and mark their completion on the fly.

Two alternative schemes can be used to keep track and mark the completion of the operations:

- List-based: Each time a non-blocking operation is performed, we create a corresponding request and push it to a list. When the progress engine completes the operations and detects that it is an OpenSHMEM Non-Blocking operation (we introduce a new flag on the packet to mark the operation as OpenSHMEM NBI operation), it removes the corresponding request from the list. During a Quiet operation, we poke the progress engine until the list is empty.
- Counter-based: As the semantic of Quiet operation is to ensure the completion of all previously issued operations, a simple counter can be used. When issuing an NBI operation, the counter is incremented. When processing a completion, the progress engine decreases the value of the counter. During the Quiet operation, we poke until the counter is zero.

To avoid an allocation and free of the request in the critical path, we chose the counter-based scheme for our implementation.

IV. Extending OpenSHMEM OSU Micro-Benchmarks for Non-Blocking Operations

In order to provide a standard way to evaluate the performance of NBI operations, we have extended the OSU Micro-Benchmarks suite (OMB) with new benchmarks. The extensions include:

Latency Benchmark: This is a simple extension to the existing blocking version. We simply change the blocking operation to its NBI counterpart, followed by a Quiet operation.

Message Rate Benchmark: We introduce two new benchmarks that evaluate the message rate for put and get operations, respectively. The benchmarks do a set (window) of NBI operations in a loop, for the same message size and then perform a Quiet operation to ensure the completion of all of the NBI operations included in the window, where the window operation count is 64. The benchmarks report the number of messages sent per second.

Overlap Benchmark: In addition to latency, communication/computation overlap is one of the main metrics to evaluate an NBI interface and support. To do so, following the same approach as the OMB MPI-3 NBC benchmark [2], we propose a new benchmark to measure the overlap for put and get operations. The benchmarks start by estimating the communication time by performing an NBI operation followed by a Quiet. Then use this time in a dummy compute function that is overlapped with the communication. It measures the time to initiate an NBI operation, followed by the dummy compute and then a Quiet operation. In addition to the communication time (latency), the benchmark reports the NBI initialization overhead, waiting time and the overlap achieved.

V. Designing 3D Stencil and All-to-All Communication Kernel using NBI operations

To understand and demonstrate the impact of the OpenSHMEM NBI operations, we have redesigned a 3D Stencil communication kernel using the new OpenSHMEM NBI interface. Further to mimic a non-blocking collective operation, we have implemented an all-to-all benchmark which measures overlap between the communication and computation phases.

Stencil communication patterns are common in many HPC applications like MILC [5] and AWP-ODC [7]. In the 3D Stencil kernel, each process communicates with six neighbors in a 3D grid configuration. This pattern mimics also the non-contiguous data movement behavior. The benchmark measures the time to perform the six *shmem_put_nbi* or *shmem_get_nbi* operations followed by a *shmem_quiet*. A barrier is performed and we report the average time on all processes.

FFT-based applications like P3DFFT [8] and PSDNS rely heavily on all-to-all exchange. Blocking all-to-all operations severely limit their performance, hence researchers try to overlap the communication with the computation. To do so, we propose to implement such overlap pattern using OpenSHMEM NBI operations. One challenge while using one-sided operations is to ensure the notification of the completion at remote target. To do so, we can either rely on:

- Local Completion: In a network like InfiniBand, the local completion ensures remote completion as well. Indeed, the hardware ACK is sent by the remote HCA once it receives the data. In this case, one can use Quiet to ensure local completion.
- Memory Polling: Alternatively and for a more generic solution, one can use memory polling to wait for a value to get updated. This involves using a specific flag which is updated by a subsequent put and using the *shmem_wait* operation to detect the completion of this put. This scheme relies on the ordering semantics of the network.

In the proposed kernel, each process posts (N-1) NBI put operations to the N-1 processes using the source and destination buffer. Then the processes start the computation while the communication is progressed by the network in the background. At the end of the computation phase, each process calls *shmem_quiet* to ensure the local completion of all the put operations.

VI. Performance Evaluation

This section describes the experimental setup used to conduct our performance evaluation. An in-depth analysis of the results is also provided to correlate design motivations and observed behavior. All results reported here are averages of multiple (10) runs to discard the effect of system noise.

A. Experimental Setup

For the evaluation, we used the TACC Stampede [31] system. Stampede is a 10 PFLOPS (PF) Dell Linux Cluster based on 6,400+ Dell PowerEdge server nodes, each outfitted with 2 Intel Xeon E5 (Sandy Bridge) processors, an Intel Xeon Phi Coprocessor, and FDR InfiniBand interconnected technology.

Our evaluation compares the blocking version of MVAPICH2-X 2.2RC1, with the proposed NBI interface. As the reference implementation with 1.3 support is not available yet, we explicitly create all the infrastructure to support the 1.3 specification.

For intra-node configuration, as mentioned in Section III, we provide a shared memory-based design (referred as *NBI_SHM*) for latency and a LoopBack-based design (*NBI_LB*) for overlap. Similarly for blocking Put operation, MVAPICH2-X has optimizations to provide some asynchronous progress. It uses an MPI-like Eager protocol where it uses an internal buffer and marks the completion as soon as the copy is done. We note that this optimization is available only for Put operation. With the message rate benchmark, for a fair evaluation, we compare the NBI design with both blocking versions default and optimized, referred as OPT.

B. Latency, Message Rate, and Overlap Evaluation

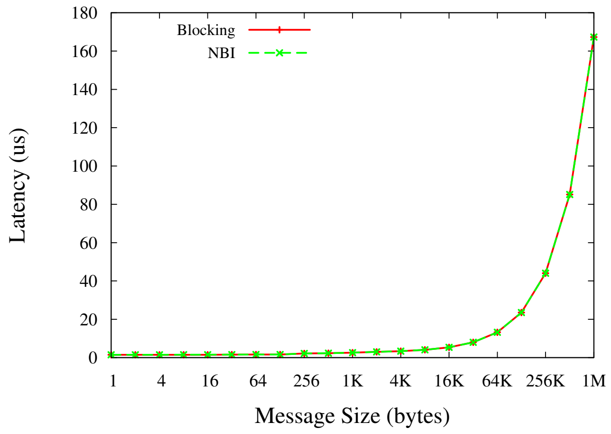
Latency

Figures 1(a) and 1(b) compare the latency of blocking and non-blocking Put and Get operations, respectively, on inter-node configuration. We can clearly see that both versions have the same performance. This experiment confirms that the NBI operations do not add any overhead to the inter-node communications.

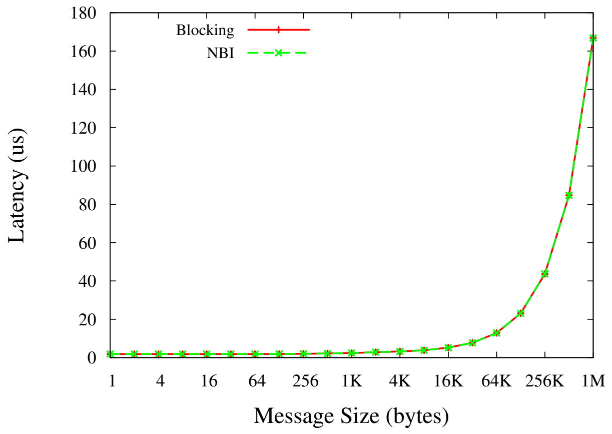
For intra-node configuration, the situation is quite different as depicted in Figure 2. The LB-based design has an overhead of 40% compared to the blocking and SHM-based (shared memory) design, for large messages. However, SHM and Blocking exhibit the same performance as they are both using shared memory channel. For small message size, blocking operation exhibits better latency than both NBI designs. The reason is due to the software overhead of calling Quiet operation. Indeed, in blocking, the operation is marked as complete as soon as the copy completes, while for NBI the semantics require calling Quiet, which enters the progress engine and checks the different channels.

Message Rate

We evaluated the throughput using the message rate benchmarks. For inter-node configuration, The NBI operations show significant improvement compared to blocking operations. Figure 3(b) shows 7X improvement for Get operation with small-medium message size. *shmem_getmem_nbi* operation is able to deliver 3.4M message per second. These benefits come from the asynchronous behavior which allows posting several back-to-back operations to the network. For large message size, as the benchmark will be bound by the IB bandwidth both versions show the same performance. For Put operation, compared to default blocking operation, we see 5X improvement. However, compared to the optimized blocking version, we see between 7% and 10% degradation as shown in Figure 3(a). Similarly, Figure 4 shows the benefits of NBI_SHM for message rate with put and get operations.

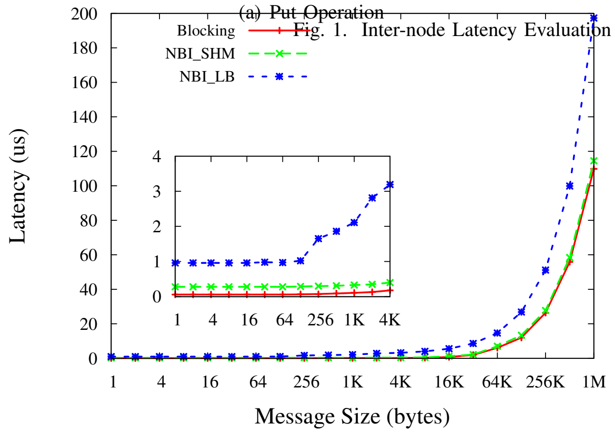


(a) Put Operation

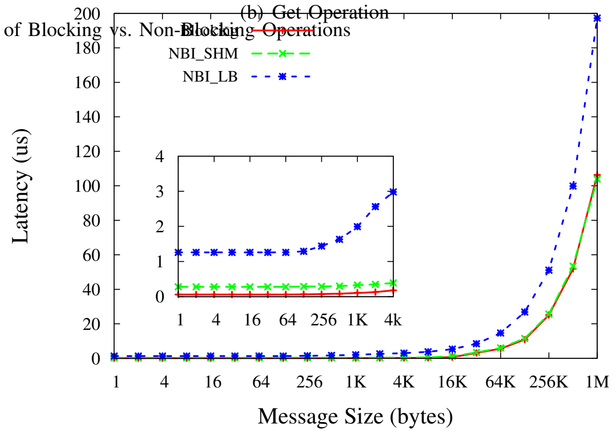


(b) Get Operation

Fig. 1. Inter-node Latency Evaluation of Blocking vs. Non-Blocking Operations

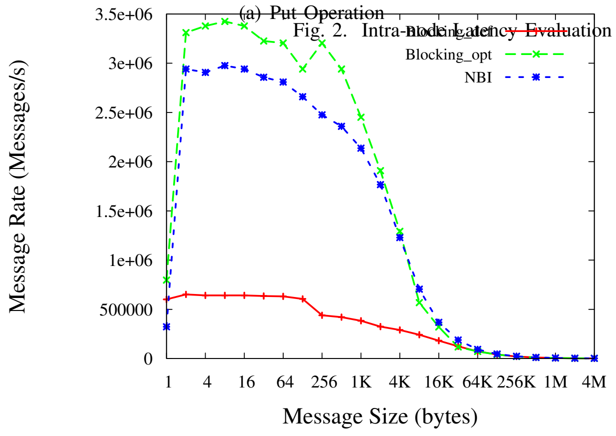


(a) Put Operation

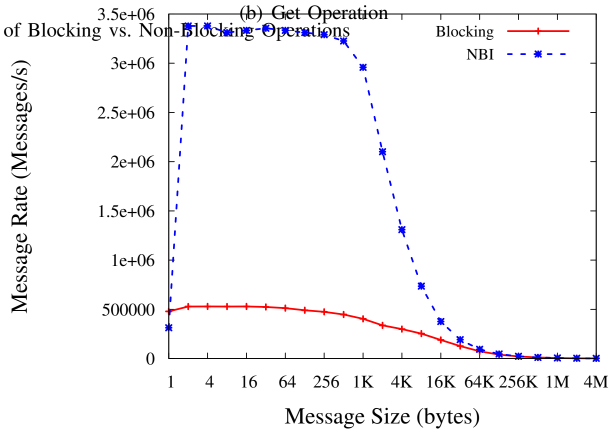


(b) Get Operation

Fig. 2. Intra-node Latency Evaluation of Blocking vs. Non-Blocking Operations



(a) Put Operation



(b) Get Operation

Fig. 3. Inter-node Message Rate (Throughput) Evaluation of Blocking vs. Non-Blocking Operations

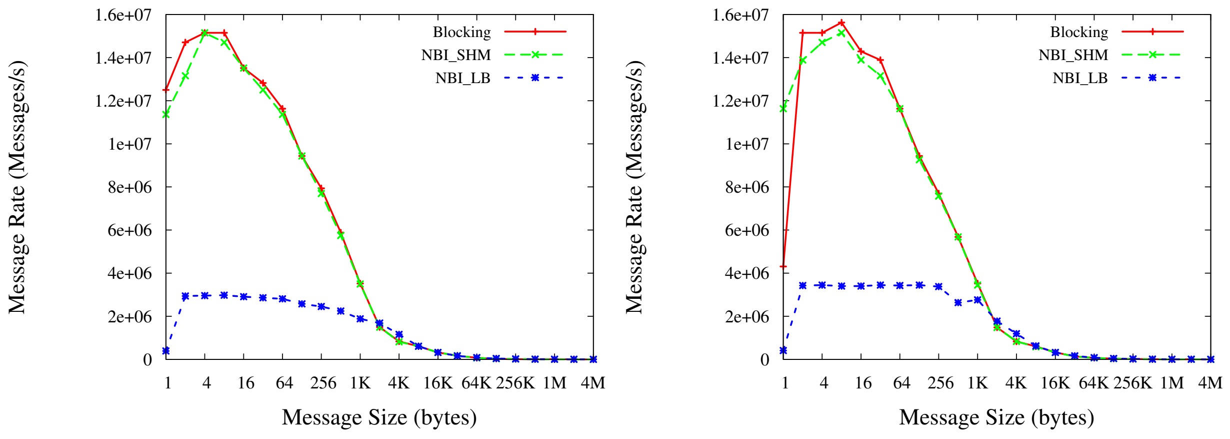
Indeed the overhead of NBI–LB scheme on latency, impacts its message rate performance.

Overlap

As the numbers are similar between Put and Get operations, in order to avoid repetition in the observations, we show only Get numbers onward. Figure 5(a) shows the percentage of computation/communication overlap achieved using Get operation for inter-node configuration. NBI interface shows 100% overlap while blocking operation, as expected, is showing 0%. For intra-node configuration, similar trend is observed in Figure 5(b). LB-based design delivers 100% overlap while SHM and Blocking provide almost no overlap. From Figures 2(b) and 5(b), we infer a reverse relationship between overlap and latency for intra-node operation due to the shared memory based blocking design. As mentioned in Section III, an alternative kernel-based design can be used to have asynchronous progress. Such a design can deliver good results for both latency and overlap. This capability and its evaluation are part of future work.

C. *Communication Kernel Evaluation*

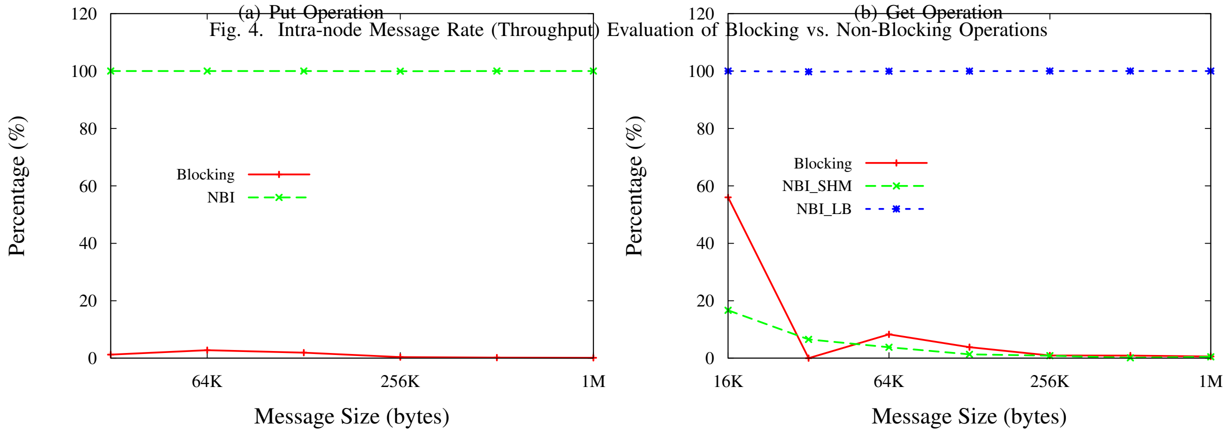
Figure 6 shows the performance of the 3D Stencil benchmark using Get operation on 8, 27, 64 and 125 cores.



(a) Put Operation

(b) Get Operation

Fig. 4. Intra-node Message Rate (Throughput) Evaluation of Blocking vs. Non-Blocking Operations



(a) Inter-node

(b) Intra-node

Fig. 5. Computation/Communication Overlap Evaluation with Get Operation

As mentioned earlier, NBI operations, in addition to the computation/communication overlap, offers an opportunity for communication/communication overlap for small message size. Figure 6(a) depicts 50%, 30% and 15% improvement of the NBI operations for a 512Bytes messages on 27, 64 and 125 cores, respectively. A similar trend is highlighted by Figure 6(b) for 2K message size.

In addition to the 3D Stencil kernel, in order to assess the computation/communication overlap potential of the NBI semantics, we evaluated a kernel that mimics FFT behavior by overlapping an all-to-all exchange with some dummy computation. Table I summarizes the evaluation. First, we observe that overlap is perfect. Second, while the Init time is negligible, it is increasing as the system size increases. We attribute this to software overhead and the concurrent access of processes to the HCA to post the RDMA operations.

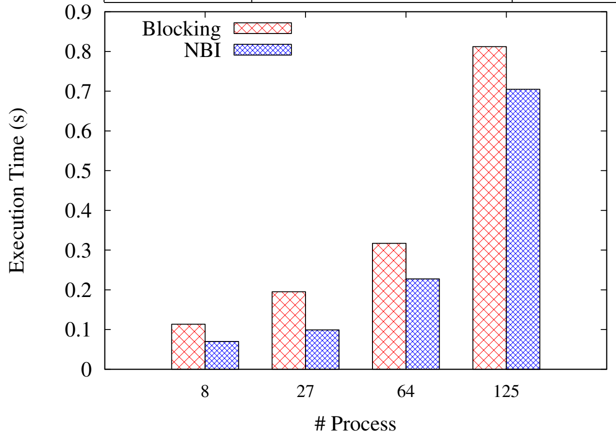
VII. Related Work

There are various implementations of the OpenSHMEM specification from the HPC community and system vendors. Yoon et al. developed a portable OpenSHMEM library called GSHMEM [33] which is built solely on the GASNet communication middleware from UC Berkeley and is based on the v1.0 draft of the OpenSHMEM specification. Sandia National Laboratory provides an open source implementation of the specification for the Portals [4] network stack. Portals is a low-level network data movement layer and programming interface to support higher-level one-sided and two-sided interfaces. The Portals 4 specification includes support for PGAS programming languages. Brightwell et al. proposed an intra-node implementation of OpenSHMEM, which uses operating system virtual address space mapping capabilities to provide efficient intra-node operations. The OpenSHMEM reference implementation [1] also uses GASNet as the communication subsystem. Using the GASNet InfiniBand conduit, these OpenSHMEM implementations enable OpenSHMEM communication on InfiniBand networks. In addition to the implementations outlined above, there are SHMEM implementations from system vendors, such as SGISHMEM [30], GPSHMEM [18], HP-SHMEM and IBMSHMEM.

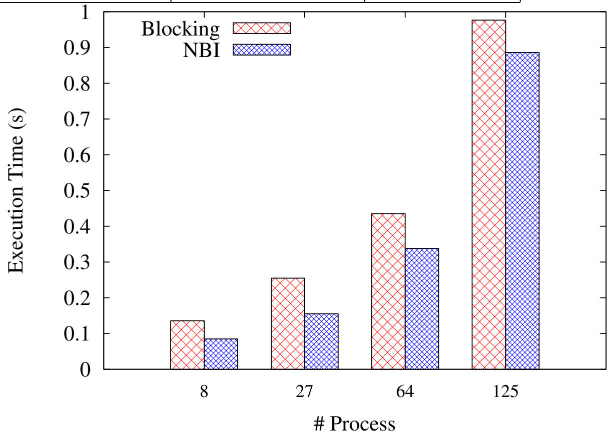
Several studies have explored and taken advantage of the scalability and programmability that OpenSHMEM offers. Pophale et al. [28] presented an implementation of OpenSHMEM-based NAS Parallel Benchmarks. Jose et al. [16] presented a hybrid MPI+OpenSHMEM version of the Graph500 benchmark. Jose et al. [15] proposed a high-performance design of sorting using MPI with simple extensions to OpenSHMEM communication. Lin et al. [19] proposed a group of alternative hybrid MPI+OpenSHMEM designs to improve the performance of the classification and regression algorithm k-NN on a large-scale environment with InfiniBand.

TABLE I
OVERLAP COMMUNICATION KERNEL

# Processes	Comm_init Time (sec)	Computation Time (sec)	Wait Time (sec)	Overlap (%)
16	0.001	2.8	0.0003	99.64
32	0.002	2.8	0.0001	99.28
64	0.004	2.8	0.0003	98.54
128	0.009	2.8	0.0002	96.78



(a) Small Message Size



(b) Medium Message Size

Fig. 6. Execution time of the 3D Stencil Communication Kernel

Non-blocking operations have been included in the current MPI Standard, MPI-3 [20]. Various microbenchmark level [3] and application level studies for the non-blocking operations have been published. The benefits of overlapping computation and communication have been highlighted at different large and small scales. Open-source MPI libraries like MPICH2, OpenMPI and MVAPICH2 offer full support for all non-blocking operations defined by the MPI specification. Similarly, the non-blocking collective operations for OpenSHMEM have been first proposed in [27] by Poole et al. However, the paper presented only the concept of OpenSHMEM non-blocking collectives with little design and/or implementation details. This paper focuses on detailed designs, implementations, and performance impacts of the new non-blocking point-to-point operations introduced by the OpenSHMEM 1.3 specification.

VIII. Conclusion and Future Work

In this paper, we have presented an initial study of the non-blocking Put/Get operations in the new OpenSHMEM specification. We have presented alternative designs for intra- and inter-node configurations. We have implemented these designs using the MVAPICH2-X unified runtime and we have highlighted the impact of these extensions on latency and overlap metrics. Using different communication patterns, we have demonstrated the efficiency of the designs and the impact of trade-offs between latency and overlap. Our evaluation shows no overhead on inter-node latency and a small overhead for intra-node latency. For both configurations, the NBI operations deliver perfect overlap. Further, in addition to the computation/communication overlap, the asynchronous progress on NBI operations provides high throughput. We showed a 7X improvement on the message rate with the NBI Get operation vs. the traditional blocking Get. For Stencil communication kernels, we showed 50% and 28% improvement compared to blocking operations, on 27 and 64 processes, respectively.

In the future, in addition to providing Limic-based asynchronous copy, we plan also to apply the proposed schemes and demonstrate their impact on the execution time of libraries and applications like P3DFFT to leverage the overlap benefits and AWPODC to take advantage of Stencil optimizations.

References

[1] OpenSHMEM. http://www.openshmem.org/.
[2] A. A. Awan, K. Hamidouche, C.-H. Chu, and D. K. Panda. A Case for Non-blocking Collectives in OpenSHMEM: Design, Implementation, and Performance Evaluation using MVAPICH2-X. In *OpenSHMEM and Related Technologies. Experiences, Implementations, and Technologies: Second Workshop, OpenSHMEM 2015*, Annapolis, USA, 2015.
[3] A. A. Awan, K. Hamidouche, A. Venkatesh, J. Perkins, H. Subramoni, and D. K. Panda. GPU-Aware Design, Implementation, and Evaluation of Non-blocking Collective Benchmarks. In *Proceedings of the 22nd European MPI Users' Group Meeting*, EuroMPI '15, Bordeaux, France, 2015. ACM.
[4] B. Barrett, R. Brightwell, S. Hemmert, K. Pedretti, K. Wheeler, and K. Underwood. Enhanced Support for OpenSHMEM Communication in Portals. In *IEEE Annual Symposium on High Performance Interconnects*, 2011.
[5] C. Bernard and M. O. et al. Studying Quarks and Gluons on MIMD Parallel Computers. *Int. Journal of High Performance Computing Applications, 5(4):61-70*, 1991.
[6] R. Brightwell and K. Pedretti. An Intra-Node Implementation of OpenSHMEM Using Virtual Address Space Mapping. In *PGAS*, 2011.
[7] Y. Cui, R. Moore, K. Olsen, A. Chorasia, P. Maechling, B. Minister, S. Day, Y. Hui, J. Zhu, A. Majumdar, and T. Jordan. Enabling Very Large Earthquake Simulations on Parallel Machines. In *Lecture Notes in Computer Science*, 2007.

[8] 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.
[9] P. N. Hilfinger, D. Bonachea, D. Gay, S. Graham, B. Liblit, G. Pike, and K. Yelick. Titanium Language Reference Manual. Technical report, Berkeley, CA, USA, 2001.
[10] T. Hoefler, J. Squyres, W. Rehm, and A. Lumsdaine. A Case for Non-blocking Collective Operations . In *Frontiers of High Performance Computing and Networking . ISPA 2006 Workshops, Lecture Notes in Computer Science*, volume 4331/2006, pages 155–164, 2006.
[11] HPC-X. http://www.mellanox.com/page/hpcx_overview.
[12] H. W. Jin, S. Sur, and D. K. Panda. LiMIC: Support for High-Performance MPI Intra-Node Communication on Linux Cluster. In *In Proceedings of the 2005 International Conference on Parallel Processing (ICPP)*, 2005.
[13] J. Jose, K. Kandalla, M. Luo, and D. Panda. Supporting Hybrid MPI and OpenSHMEM over InfiniBand: Design and Performance Evaluation. In *Parallel Processing (ICPP), 2012 41st International Conference on*, 2012.
[14] J. Jose, M. Luo, S. Sur, and D. K. Panda. Unifying UPC and MPI Runtimes: Experience with MVAPICH. In *PGAS*, 2010.
[15] J. Jose, S. Potluri, H. Subramoni, X. Lu, K. Hamidouche, K. Schulz, H. Sundar, and D. K. Panda. Designing Scalable Out-of-core Sorting with Hybrid MPI+PGAS Programming Models. In *PGAS*, 2014.
[16] J. Jose, K. T. Sreeram Potluri, and D. K. Panda. Designing Scalable Graph500 Benchmark with Hybrid MPI+OpenSHMEM Programming Models. In *International Supercomputing Conference (ISC)*, 2013.
[17] J. Jose, J. Zhang, A. Venkatesh, S. Potluri, and D. K. Panda. A Comprehensive Performance Evaluation of OpenSHMEM Libraries on InfiniBand Clusters. In *Proceedings of the First Workshop on OpenSHMEM and Related Technologies. Experiences, Implementations, and Tools - Volume 8356*, OpenSHMEM 2014, pages 14–28, Annapolis, MD, USA, 2014. Springer-Verlag New York, Inc.
[18] K. Parzyszek. Generalized Portable Shmem Library for High Performance Computing. In *Doctoral Thesis. UMI Order Number: AAI3105098.*, Iowa State University, 2003.
[19] J. Lin, K. Hamidouche, J. Zhang, X. Lu, A. Vishnu, and D. K. Panda. Accelerating k-NN Algorithm with Hybrid MPI and OpenSHMEM. In *OpenSHMEM and Related Technologies. Experiences, Implementations, and Technologies: Second Workshop, OpenSHMEM 2015*, Annapolis, USA, 2015.
[20] Message Passing Interface Forum. http://www.mpi-forum.org/.
[21] S. Moreaud, B. Goglin, D. Goodell, and R. Namyst. Optimizing MPI Communication within Large Multicore Nodes with Kernel Assistance. In *CAC 2010: The 10th Workshop on Communication Architecture for Clusters, held in conjunction with IPDPS 2010*, Atlanta, GA, Apr. 2010.
[22] MVAPICH2-X: Unified MPI+PGAS Communication Runtime over OpenFabrics/Gen2 for Exascale Systems. http://mvapich.cse.ohio-state.edu/.
[23] R. Numrich and J. Reid. Co-Array Fortran for Parallel Programming. Technical Report Technical Report RAL-TR-1998-060, Rutheford Appleton Laboratory, 1998.
[24] OpenMPI: Open Source High Performance Computing. http://www.open-mpi.org/.
[25] OpenSHMEM. http://openshmem.org/.
[26] D. K. Panda, K. Tomko, K. Schulz, and A. Majumdar. The MVAPICH Project: Evolution and Sustainability of an Open Source Production Quality MPI library for HPC. In *Workshop on Sustainable Software for Science: Practice and Experiences, held in conjunction with Int'l Conference on Supercomputing (WSSPE)*, 2013.
[27] S. Poole, P. Shamis, A. Welch, S. Pophale, M. G. Venkata, O. Hernandez, G. Koenig, T. Curtis, and C.-H. Hsu. Openshmem extensions and a vision for its future direction. In *Proceedings of the First Workshop on OpenSHMEM and Related Technologies. Experiences, Implementations, and Tools - Volume 8356*, OpenSHMEM 2014, pages 149–162, Annapolis, MD, USA, 2014. Springer-Verlag New York, Inc.
[28] S. Pophale, H. Jin, S. Poole, and J. Kuehn. OpenSHMEM Performance and Potential: A NPB Experimental Study. In *Proceedings of the 1st Conference on OpenSHMEM Workshop*, Oct 2013.
[29] S. Potluri, K. Kandalla, D. Bureddy, M. Li, and D. K. Panda. Efficient Intranode Desgins for OpenSHMEM on Multicore Clusters. In *PGAS*, 2012.
[30] Silicon Graphics International. SHMEM API for Parallel Programming. http://www.shmem.org/.
[31] TACC Stampede Cluster. http://www.xsede.org/resources/overview.
[32] UPC Consortium. UPC Language Specifications, v1.2. Technical Report LBNL-59208, Lawrence Berkeley National Lab, 2005.
[33] C. Yoon, V. Aggarwal, V. Hajare, A. D. George, and M. B. III. GSHMEM: A Portable Library for Lightweight, Shared-Memory, Parallel Programming. In *PGAS*, 2011.