Unifying UPC and MPI Runtimes: Experience with MVAPICH

Jithin Jose Miao Luo Sayantan Sur Dhabaleswar K. Panda
Department of Computer Science and Engineering, The Ohio State University
{jose, luom, surs, panda}@cse.ohio-state.edu

Abstract

Unified Parallel C (UPC) is an emerging parallel programming language that is based on a shared memory paradigm. MPI has been a widely ported and dominant parallel programming model for the past couple of decades. Real-life scientific applications require a lot of investment by domain scientists. Many scientists choose the MPI programming model as it is considered low-risk. It is unlikely that entire applications will be re-written using the emerging UPC language (or PGAS paradigm) in the near future. It is more likely that parts of these applications will be converted to newer models. This requires that underlying implementation of system software be able to support both UPC and MPI simultaneously. Unfortunately, the current state-of-the-art of UPC and MPI interoperability leaves much to be desired both in terms of performance and ease-of-use.

In this paper, we propose "Integrated Native Communication Runtime" (INCR) for MPI and UPC communications on InfiniBand clusters. Our library is capable of supporting both UPC and MPI communications simultaneously. This runtime is based on the widely used MVAPICH (MPI over InfiniBand) Aptus runtime, which is known to scale to tens-of-thousands of cores. Our evaluation reveals that INCR is able to deliver equal or better performance compared to the existing UPC runtime - GASNet on InfiniBand verbs. We observe that with UPC NAS benchmarks CG and MG (class B) at 128 processes, we outperform current GASNet implementation by 10% and 23%, respectively.

1. Introduction

Modern high-end computing (HEC) systems allow scientists and engineers to tackle grand challenge problems in their respective domains and make significant contributions to their fields. Examples of such problems include astro-physics, earthquake analysis, weather prediction, nanoscience modeling, multiscale and multiphysics modeling, biological computations, computational fluid dynamics. The Message Passing Interface (MPI) is a very widely used parallel programming model. Scientific applications rely heavily on the performance and portability offered by it. Although MPI has been widely used for the past couple of decades, recently, there is an effort to improve programmer productivity for parallel applications. The hypothesis is that for several classes of applications, especially irregular applications, shared memory programming is much easier than explicit message passing. Additionally, language-based parallelism constructs provide greater flexibility of expression, analysis by software tools and optimization opportunities. For example, unneeded barriers in an application can be syntactically analyzed by tools and compiled away, eliminating the need for programmers to manually analyze dependencies and removing barriers manually. Often programmers tend to insert more barriers than strictly required for safety.

Unified Parallel C (UPC) [27] is an emerging language for parallel programming, that is based on extensions to ISO C99. It provides parallel iterations and remote memory access among a host of other features. UPC is a member of the Partitioned Global Address Space (PGAS) family of languages. Several flavors of UPC are available, along with different implementations of UPC compiler and runtime on a variety of architectures including commodity clusters and leadership class machines such as the Blue Gene/P. The Berkeley UPC implementation [19] is one of the open-source and popular implementations that support high-performance interconnects such as Myrinet, InfiniBand and Quadrics through the portable GASNet runtime [10]. Even though UPC is an emerging language, there are not many real-life scientific applications using it as their primary parallel programming model. Real life scientific applications require a lot of investment by domain scientists for design and verification, and their life cycles are in decades. Since MPI is standardized, adopted and portable, it provides a low-risk implementation choice. It is unlikely that entire applications will be re-designed using UPC in the near future. It is more likely that parts of these applications (primarily written in MPI) will be converted to UPC, resulting in a "hybrid" application.

InfiniBand [14] is an industry standard interconnection technology that has gained wide acceptance and is rapidly becoming one of the dominant interconnection technologies in HEC domain. Accordingly, the size of InfiniBand clusters is growing as well, with the Nebulae cluster in China having 120,640 cores. Design and implementation of highly scalable InfiniBand runtime is not a trivial task as the software interface provided by *verbs* is at a very low-level. Using verbs, various buffer management, connection management, message coalescing, and reliability mechanisms need to be explored [15–18, 26].

Thus, there are two emerging trends: need for hybrid programming support and the need for extreme scalability on InfiniBand clusters. In order to realize the vision of highly scalable hybrid computing, the following questions need to be answered:

1. Can a communication library be designed for UPC that provides excellent performance and scalability on very large InfiniBand clusters?
2. Can this unified communication library support both UPC and MPI communications in a manner that does not degrade performance for either?

In this paper, we aim to answer the above questions. We have extended the MVAPICH-Aptus [17] runtime, which is known to be one of the most scalable InfiniBand runtime, to support UPC com-

PGAS '10 Oct 12-15, New York City.

munications. MVAPICH/MVAPICH2 [23] are very widely used MPI libraries, supporting MPI-1 and MPI-2 standards specifically designed for InfiniBand. It is currently being used by more than 1,185 organizations in 59 countries. This extended MVAPICH library is capable of acting as a new GASNet communication conduit. Since these extensions are made to an MPI library, they do not interfere in any way with the existing MPI communication infrastructure nor affect MPI communication performance (MPI applications simply ignore the extensions). Our evaluation reveals that this new extended MVAPICH conduit not only able to match the existing GASNet InfiniBand conduit in both microbenchmarks and application-level evaluation; but also to improve scalability significantly. We observe that with UPC NAS benchmarks CG and MG (class B) at 128 processes, the new design outperforms the current GASNet implementation by 10% and 23%, respectively.

The rest of the paper is organized as follows. In Section 2, we describe background material for this work along with a discussion of related work. We contrast our work with existing software stacks and research. Then, in Section 3, we describe our design in detail. We present our evaluations in Section 4. Finally, we conclude and illustrate future directions in Section 5.

2. Background and Related Work

In this Section, we describe associated background work that is relevant to the contributions of this paper. We also cover other recent and related work in this area. We first start with the lowest InfiniBand layer. Then, we describe recent work in designing scalable communication techniques for MPI on InfiniBand. After that, we review work in UPC and the GASNet communication system. Finally, we compare and contrast our work with recent related efforts to develop a common communication subsystem for multiple programming models.

2.1 InfiniBand Overview

InfiniBand [14] is an industry standard switched fabric that is designed for interconnecting nodes in HEC clusters. It is a high-speed, general purpose I/O interconnect that is widely used by scientific computing centers world-wide. The recently released TOP-500 rankings in June 2010 reveal that more than 40% of the computing systems use InfiniBand as their primary interconnect. The yearly growth rate of InfiniBand in the TOP500 systems is pegged at 30%, indicating a strong momentum in adoption. It is also gaining ground in the commercial domain with the recent convergence around RDMA (Remote Direct Memory Access) over Converged Enhanced Ethernet (RoCE).

The InfiniBand specification clearly demarcates the duties of hardware (such as Host Channel Adapters (HCAs)) and software. Upper-level software uses an interface called *verbs* to access the functionality provided by HCAs and other network equipment (such as switches). The verbs interface is a low-level communication interface that follows the Queue Pair model. Communication end-points are required to establish a queue pair between themselves. Each queue pair has a certain number of work queue elements. Upper-level software places a work request on the queue pair that is then processed by the HCA. When a work element is completed, it is placed in the completion queue. Upper level software can detect completion by polling the completion queue.

Additionally, there are different types of Queue Pairs based on the type of transport used. There are Reliably Connected (RC) queue pairs that provide reliable transmission (retransmissions after packet losses are performed by the HCA). These RC queue pairs need to be established uniquely for each communicating pair. This implies an $O(n^2)$ memory usage (for a system with N processes). Another type of queue pair is the Unreliable Datagram (UD). This queue pair type does not provide reliable transmission although it has a significant memory advantage – only one UD QP is capable of communicating with all remote processes. Thus, the memory usage of UD QP is $O(n)$ (for a system with N processes).

Yet another feature of InfiniBand is Remote Direct Memory Access (RDMA). This feature allows software to remotely read memory contents of another remote process without any software involvement at the remote side. This feature is very powerful and can be used to implement high-performance communication protocols.

As indicated above, the verbs software layer is very low-level, and the responsibility of communication protocols, buffer management mechanisms, connection management techniques etc. are left to the upper level software. The following Section 2.2 describes some recent work in designing scalable mechanisms using these verbs. Not all features are mentioned in this section and readers are encouraged to peruse the InfiniBand specification [14] for more details.

2.2 Scalable MPI Design over InfiniBand: MVAPICH-Aptus

MPI has been the dominant parallel programming model for the past couple of decades. It has been widely ported and several open-source implementations have been made available. It has also achieved very good performance and scalability. As a result, all modern super-computers support it. During the past several years, InfiniBand has made great inroads into the HPC domain. Clusters using InfiniBand have been growing in size rapidly. One of the largest clusters is the Nebulae cluster in China with 120,640 cores. As indicated in the previous section, InfiniBand offers a low-level *verbs* interface with several types of Queue Pairs, with varying levels of services. This enables upper-level software, such as MPI implementations, to design flexible and high-performance connection management, buffer management, coalescing strategies, etc.

MVAPICH/MVAPICH2 [23] are high-performance implementations of MPI-1 and MPI-2 interfaces on InfiniBand, iWARP and RoCE (RDMA over Converged Ethernet). The internal design of MVAPICH has been systematically designed to achieve very good scalability by exploiting various InfiniBand features, such as Unreliable Datagrams [16], Shared Receive Queues (SRQ) [26], and eXtended Reliable Connections (XRC) [18] along with connection management strategies such as On-demand connections and buffering strategies for message coalescing [15] to improve memory efficiency. All of these optimizations have been combined into one unified runtime, called MVAPICH-Aptus [17]. To the best of our knowledge, this is the most scalable runtime on InfiniBand that offers high-performance and is open-source. Our team is in the process of bringing this framework into the MVAPICH2 library, so that MPI-2 applications can also leverage this scalable runtime.

2.3 UPC, PGAS and GASNet Communication System

Unified Parallel C (UPC) [27] is an emerging parallel programming language that aims to increase programmer productivity and application performance by introducing parallel programming and remote memory access constructs in the language. UPC is based on the Partitioned Global Address Space (PGAS) programming model. The PGAS programming model allows programmers to view a distributed memory supercomputer as a global address space, that may be partitioned to improve performance. There are several other PGAS programming languages, namely X10 [7], Chapel [6] and HPF [21], along with Global Address Space libraries such as Global Arrays [11].

The runtime implementations of UPC have been demonstrated to be scalable and provide very good performance to end applications through fine-grained remote memory accesses [5] and improved communication overlap [24]. In particular, the Blue Gene implementation of UPC, developed by IBM, has been demonstrated to be highly scalable [3]. In this paper, we focus on the InfiniBand

design and implementation of UPC through the popular GASNet communication library [10]. GASNet for InfiniBand clusters has only been mildly optimized with Firehose [4], which is a strategy to manage remote keys (required for RDMA). It is to be noted that as compared to the exhaustive scalability and optimization work carried out to scale MVAPICH/MVAPICH2 up to the level of 81,920 cores (of Pleiades at NASA) and 62,976-core (of Ranger at TACC), not much recent work has taken place in the InfiniBand implementation of GASNet. In particular, even on-demand connections and shared receive queues, which are considered to be fundamental requirements to scale InfiniBand runtime, are missing in the current GASNet implementation. In this paper, we present our strategy to unify the runtime of MVAPICH and GASNet in such a way that the InfiniBand communication schemes are unified between them in an efficient manner. Thus, our work *adds* major scalability features to the GASNet implementation on InfiniBand.

2.4 Common Communication Subsystems for Multiple Programming Model Support

High-performance communication libraries for HEC systems have historically been very closely tied with their corresponding programming models and their implementations. This has been due to the fact that each communication layer has been optimized heavily to suit the particular programming model and its requirements. Some libraries, such as ARMCI [12] are deployed along with MPI in a side-by-side manner to enable Global Arrays applications that occasionally use utility functions from MPI. There have been some efforts to unify the communication subsystems, such as CCS by Buntinas, et. al [9]. It is to be noted that even though the CCS layer had the potential to support multiple programming models, it did not offer the best performance as compared to GASNet (please refer to Fig. 2 of [9]), especially for short messages. Also, the authors of [9] did not evaluate whether CCS will be suitable for MPI itself as a communication layer or not. Additionally, there was no evaluation of UPC or GASNet on top of CCS. In this paper, we describe a truly unified communication library which provides the best performance to *both* UPC and MPI libraries in conjunction with thorough evaluation of the entire UPC stack through our new communication runtime.

The Low-level Application Programming Interface (LAPI) [13], is a unified communication layer that is known to support both MPI and pure LAPI communications. The LAPI layer is available only on IBM systems and is not available on general commodity cluster systems with InfiniBand. In this respect, it can potentially support GASNet implementation over LAPI along with MPI over LAPI. We have not yet found any publicly available document describing the feasibility of this approach. Additionally, publically available documents, such as [1] describe the limitations of LAPI supporting both MPI and LAPI communications. In particular, it mentions that in the event of concurrent communications, the DMA Receive Buffer may be too small to contain packets from both LAPI and MPI, and packets may be dropped, impairing performance. Further, the `MP_CSS_INTERRUPT` variable, that allows incoming messages to cause interrupts and message progress, is available to MPI only. LAPI communications cannot be progressed using this mechanism. We believe that the MVAPICH-Aptus design is not constrained by these limitations. First, DMA buffer space is shared among all communication contexts. Second, MVAPICH-Aptus follows a "low-water-mark" based flow control mechanism. In this mode, interrupts do not need to be generated or each incoming message; rather only *one* interrupt is generated when DMA buffer space is low. Thus, we believe that this is a much more flexible buffer management strategy compared to the one described for LAPI.

The MPI-2 RMA interface has been criticized before for its inadequacy as a compiler target [8]. The MPI community is actively discussing the upcoming MPI-3 standard. The MPI-3 RMA Working Group [25] is attempting to completely revamp the MPI-2 RMA semantics in such a way that supporting PGAS languages becomes simpler at the MPI level. However, these discussions are a long way from inclusion in the formal standard. Our work is intended to influence these discussions by providing critical data about a high-performance UPC implementation in an existing MPI stack.

3. INCR: Design Requirements and Solutions

In this section, we discuss the design requirements and solutions for the implementation of our Integrated Communication Runtime (INCR) for InfiniBand clusters.

3.1 Design Requirements

The major design requirement of our work is to enable *simultaneous* MPI and UPC communications without imposing any performance penalties on either MPI or UPC. That is, using our runtime, UPC programs should get identical or better performance than currently available software (and the same for MPI applications). Another design requirement is to eliminate buffer waste inside the communication library. For example, we envision that the solution should not require separate communication resources, such as connections, buffers etc. for supporting simultaneous communications.

3.2 Approach

Currently, there exist several approaches that can be deployed on commodity InfiniBand clusters. We have outlined some of these approaches in Figure 1. Each implementation option is referred to by the name given below the stack.

The left-most alternative demonstrates that pure-MPI operations can work through the MPI standard implementations of MVAPICH on InfiniBand. The second alternative, marked "GASNet-IBV" indicates the currently available GASNet implementation on OpenFabrics verbs. This implementation is on the lowest software layer made available by InfiniBand vendors. Currently, the two approaches "MPI" and "GASNet-IBV" can be combined together to support simultaneous communications of hybrid MPI and UPC applications. However, this approach suffers from two main drawbacks: (a) Communication progress of both UPC and MPI are separate, and it is possible to deadlock UPC and MPI by not progressing in their respective communications [2] and (b) It wastes communication resources, since both MPI and UPC allocate their own connection and buffer resources.

Another approach is to use the "GASNet-MPI" stack for both MPI and UPC communications. This mode has the added advantage of progressing both MPI and UPC communications simultaneously. The primary disadvantage of this mode is that there is a mismatch of Active Messages (a fundamental design point of GASNet), and the MPI-1 point-to-point primitives. Due to this mismatch, there is a performance penalty imposed on most operations. As discussed in Section 2, MPI-2 RMA is not a viable target for compilers [8]. Thus, there is a performance penalty associated with the "GASNet-MPI" stack.

In our design, presented in the rightmost stack, "GASNet-INCR", we extend the existing MVAPICH-Aptus runtime to support native active messages. In addition, we design a new communication conduit for GASNet that supports this interface. Using this approach, both UPC and MPI communication resources are shared. Further, we have the benefit of utilizing several *common* communication related optimizations that have been designed for MPI over the past several years (described in Section 2) for UPC communications as well.

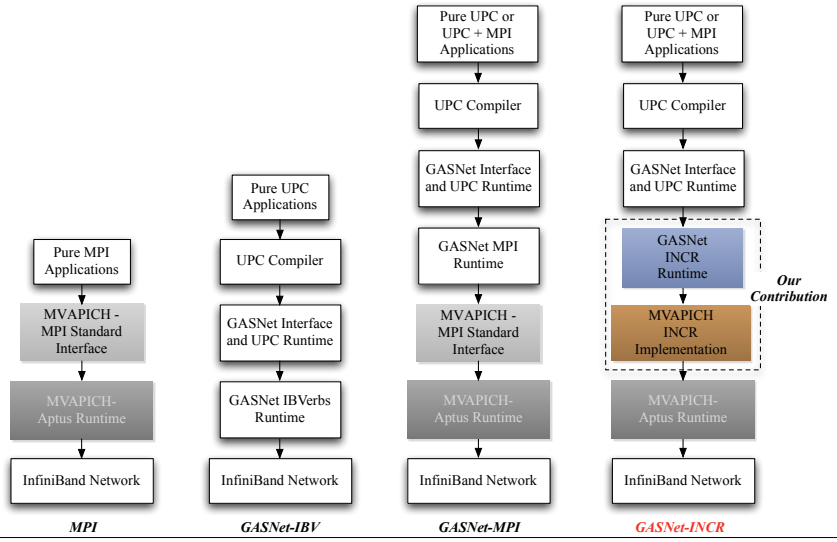


Figure 1. An overview of various communication stack options available for MPI and UPC applications and our contribution

3.3 Internal Implementation Details

Our design requirements are to enable the highest possible performance for both GASNet and MPI communications. We made the following changes to implement MVAPICH-INCR.

Packet Headers: MPI communications stacks are required to carry message-matching information triplet ({`src, tag, context`}), to enable the tag-matching semantics of MPI. This represents an overhead of 12 bytes per packet. Additionally, packets are required to carry information for credits (such as the last received packet) for flow-control. Overall, the minimum MPI packet header is 20 bytes. We have decided that this overhead is too heavy for GASNet. Thus, we have added extra packet types in the MVAPICH-Aptus implementation. Accordingly, the packets used by GASNet messages are handled completely separately from the MPI packets. This approach allows us to share common optimizations from the MPI communications layer, while providing less overhead for GASNet communications. Our GASNet-INCR has only four bytes of extra header on top of GASNet-specific fields. We believe that in the future this can be improved upon as well.

Extended Interfaces: As indicated earlier in this Section, we have extended the communication implementation of the MVAPICH-Aptus runtime to support native Active Messages. In addition, we have also implemented the *extended* GASNet interfaces for natively supporting RDMA. Below is a short description of the interface.

Active Messaging Interfaces: These are the new active messaging functions that are implemented inside MVAPICH-Aptus.

1) `int incr_send_short_am_no_args(uint32_t dest, uint32_t token, uint16_t handler)`

This function implements sending very short messages without any arguments to a remote destination. This is an optimization over the short message functions which need to send arguments. Since the handler at the remote end expects no arguments, the space for arguments in the message packet can be optimized out.

2) `int incr_send_short_am_with_args(uint32_t dest, uint32_t token, uint16_t handler, va_list argptr, uint8_t numargs)`

This function implements sending short messages to remote destinations. Short active messages do not carry any data payload.

3) `int incr_send_medium_msg(uint16_t dest, uint16_t token, uint8_t handler, void * source_addr, uint32_t nbytes, va_list argptr, uint8_t numargs)`

This function implements sending medium size messages with data payload. The data payload can be sent over a bounce buffer implementation that is implemented using native RDMA [20]. It is to be noted that using MVAPICH-Aptus, RDMA connections are only opened in an on-demand basis and not an all-to-all manner during communication startup.

4) `int incr_send_long_msg(uint32_t dest, uint32_t token, uint16_t handler, void * source_addr, uint32_t nbytes, uint32_t lkey, va_list argptr, uint8_t numargs, void * dest_addr, uint32_t rkey, incr_handle_t *handle_ptr)`

This function implements sending large messages. Since large messages may take a long time to send, this interface call supports returning a handle to the communication request. Using this handle, this call may be used in a non-blocking manner. Long messages are always sent using InfiniBand RDMA, with reliable connections (RC) established in an on-demand fashion.

Extended Remote Memory Interfaces: This interface implements the GASNet extended interface. These focus on efficient remote memory access. These functions currently do not return any error status, i.e. all errors in remote memory accesses are considered fatal.

1) `void incr_inline_put (uint32_t dest, void *rem_addr, void *local_addr, size_t nbytes, uint32_t rkey, incr_handle_t *handle_ptr)`

This function sends a very small (< 128 bytes) data payloads directly to remote memory using RDMA. This utilizes the InfiniBand "inline" send operation, which reduces sender side overhead of DMAs. As long as send work queue elements (WQEs) are available, this function completes the put on returning.

2) `void incr_put (uint32_t dest, void *rem_addr, void * local_addr, size_t nbytes, uint32_t rkey, incr_handle_t * handle_ptr, uint32_t lkey)`

This function is used for remote put. The payload data is put directly using RDMA write. If no send work queue elements (WQEs) are available, then this data can be buffered internally (decision made based on message size), and the call returns immediately. To test for completion of the data transfer, the handle needs to be checked upon. Since the call is implemented using RDMA put, an RC connection is used (set up in an on-demand fashion).

3) `void incr_put_bulk (uint32_t dest, void *rem_addr, void *local_addr, size_t nbytes, uint32_t rkey, incr_handle_t * handle_ptr, uint32_t lkey)`

This function is also used for remote put. This is for supporting the GASNet put_bulk interface, in which the local memory is guaranteed to be untouched until the put operation is completed, so that data need not be buffered internally if the work queue elements (WQEs) are not available.

4) `void incr_get (uint32_t dest, void *rem_addr, void * local_addr, size_t nbytes, uint32_t rkey, uint32_t lkey, incr_handle_t *handle_ptr)`

This function is used for remote get operations. This maps directly to RDMA Read operation, which uses RC connection (set up in an on-demand fashion).

4. Experimental Results

In this section, we compare performance evaluation results of the three approaches: GASNet-IBV, GASNet-MPI and GASNet-INCR described in section 3.2. The existing implementation of GASNet on OpenFabrics verbs is called GASNet-IBV and the implementation on MPI is called GASNet-MPI. GASNet-INCR, which is our contribution in this paper, is the GASNet implementation over extended MVAPICH-Aptus runtime. MVAPICH-1.1 was used as the base MPI version for this work.

GASNet consists of core APIs and extended APIs. The core API interface is a narrow interface based on the Active Message paradigm. Extended APIs provide a richly expressive and flexible interface that provides medium and high-level operations on remote memory and collective operations [10]. GASNet-MPI has only the core APIs implemented whereas GASNet-IBV and GASNet-INCR have the extended APIs for one sided put and get operations. As mentioned in Section 2, the GASNet-MPI implementation cannot utilize the MPI2-RMA interface for put and get due to several restrictions and is limited to only using the core APIs. We used Berkeley UPC version 2.10.2 for our experimentation. GASNet was configured with `--enable-pshm`, which makes use of shared memory for intra-node communication for all network conduits.

We compared these three designs from different angles. These include microbenchmark-level performance evaluation, scalability analysis based on memory footprint (memory consumption of each process as the total number of processes increase), and performance analysis of different NAS benchmarks using these conduits. Detailed analysis of these performance evaluation is presented in the following sections.

4.1 Experimental Platform

We used three different clusters for our experimental evaluations. Cluster A consists of four Intel Nehalem machines equipped with ConnectX QDR HCAs. Each node has eight Intel Xeon 5500 processors organized into two sockets of four cores each clocked at 2.40 GHz with 12 GB of main memory. Cluster B consists of 32 Intel Clovertown based systems equipped with ConnectX DDR HCAs. Each node in this cluster has eight Intel Xeon processors, organized into two sockets of four cores each clocked at 2.33 GHz with 6 GB of main memory. Cluster C consists of eight AMD Barcelona hosts. Each node has four sockets each with a Quad-Core AMD Opteron 8350 2GHz Processor with 512KB L2 cache and 2 MB L3 cache per core. Each node has a Mellanox MT25418 dual-port ConnectX HCA. RedHat Enterprise Linux Server 5 was used on all machines along with OFED version 1.4.2. We used Cluster A for the microbenchmark-level experiments, Cluster B for scalability analysis and Cluster C for application level evaluations.

4.2 Microbenchmark Level Evaluation

We chose three representative benchmarks for the microbenchmark-level performance analysis. These include the performance evaluation of UPC calls, `upc_memput`, `upc_memget` and `bupc_memput_signal`. The `upc_memput` call writes the specified amount of data bytes to the remote side. Similarly, `upc_memget` fetches the specified amount of data bytes from remote side. `bupc_memput_signal` is one of the UPC extensions proposed by Berkeley Lab. It performs the same data movement semantics as that of `upc_memput`, and updates the specified semaphore on the remote side. The update of semaphore on the remote side signals the global completion of data movement. We used Cluster A for microbenchmark performance evaluation. In this section we present results for inter-node communication. Since we configured GASNet with `--enable-pshm`, all three appraches GASNet-INCR, GASNet-MPI, GASNet-IBV will perform exactly the same inside a node. Therefore, we do not need to discuss intra-node performance in this section.

In `upc_memput` and `upc_memget` microbenchmarks, `upc_memput` and `upc_memget` were called in the sender thread for message sizes varying from one byte to two megabytes. The receiver thread waits on a barrier. Time taken for the upc call for each of these message sizes is reported. We have split the latency results into two graphs, one showing the results for small payload sizes (1byte to 2K bytes) and other one for large payload sizes (4KB to 2MB). Microbenchmark results are shown in Figures 2 and 3.

The results indicate that the performance of GASNet-INCR is very similar to that of the high performance GASNet-IBV. Since GASNet-MPI does not have the GASNet extended APIs, `upc_memget` and `upc_memput` calls are translated into active messages These active messages are exchanged using MPI send/recv calls. This involves not only the overhead of translating into active messages, but also the extra MPI headers that these messages have to carry. But for GASNet-IBV and GASNet-INCR, `upc_memput` and `upc_memget` calls are translated into one sided RDMA put/get operations. This explains the huge performance difference between GASNet-MPI and the other two GASNet implementations.

We used a ping-pong test in `bupc_memput_signal` microbenchmark. The sender thread calls `bupc_memput_signal` and then calls `bupc_sem_wait`. The receiver thread calls `bupc_sem_wait` and then calls `bupc_memput_signal`. The `bupc_sem_wait` call causes the thread to wait until the semaphore gets incremented. The `bupc_sem_wait` call in the receiver thread gets unblocked when the data movement initiated by the sender thread is over. Similarly, the semaphore in the sender thread gets unblocked when the data movement initiated by receiver thread is over. The test was also done for message sizes varying from one byte to two megabytes. The one-way latency numbers are reported. Here also, the latency

numbers are shown in two different graphs for fine-grain analysis. The observed performance results of `bupc_memput_signal` benchmark is similar to that of `upc_memget` and `upc_memput` benchmarks. The results are shown in Figure 4.

The microbenchmark performance results indicate that the GASNet-INCR delivers similar performance as that of GASNet-IBV conduit, and it even outperforms GASNet-IBV conduit for small payload sizes in `bupc_memput_signal` microbenchmark. GASNet-MPI performs worse in all microbenchmarks.

4.3 Scalability Evaluation

In order to analyze the scalability aspect of the GASNet-INCR design, we conducted memory scalability tests. In these tests, we study how the overall memory consumption of an individual process changes as the total size of the process group increases. The UPC (and MPI) programming models are "fully connected", i.e. at any instant any thread can send a message to any other thread, and vice-versa. Therefore, it is incumbent on the runtime to either a) establish communication channels across threads before the application attempts communication, or b) provide an out-of-band technique by which communication can be set up on-demand. Since communication channels consume memory, establishing them consumes memory from the system. Another aspect of memory consumption by communication channels, is that of the choice of low-level transport. As indicated in Section 2.1 and Section 2.2, unless the design of the InfiniBand runtime is done using scalable transports, such as Unreliable Datagram and eXtended Reliable Connections, increasing process group size results in negative impact on memory requirement scalability.

We used a simple UPC hello world program in this experiment. We measured the memory footprint of the process, with number of connections (processes) ranging from 16 to 256. The memory footprint analysis results are shown in Figure 5. We used Cluster B for scalability analysis.

It was observed that for a 256 process UPC hello world execution, each process consumed about 227 MB in case of GASNet-INCR conduit, 265 MB in case of GASNet-IBV conduit and 237 MB in case of GASNet-MPI conduit. It can be noticed that, with increase in number of connections, memory footprint increases almost linearly for GASNet-IBV conduit, where as memory footprint of GASNet-INCR and GASNet-MPI conduits remain almost constant. The low memory footprint of GASNet-INCR and GASNet-MPI conduits is because it uses the MVAPICH-Aptus runtime, which is known to scale to tens of thousands of cores. Please refer to Section 2.2 for more details on the scalability features in MVAPICH-Aptus.

Analyzing the gradient of the scalability performance results, it can be noted that, for GASNet-IBV, the memory footprint increases by around 128KB with every additional process, where as the memory footprint of GASNet-INCR and GASNet-MPI conduit remains constant. If we extrapolate the memory footprint for 10,000 processes, then memory footprint of GASNet-IBV will be around 1.4 GB per process, whereas the memory footprint of GASNet-INCR process will be around 250 to 300 MB per process. This shows that our design is highly scalable.

4.4 MPI Application Evaluation

In this section, we present experimental results that compare pure MPI performance between MVAPICH-1.1 (which uses Aptus runtime) and MVAPICH-INCR (which includes Aptus runtime, but with INCR extensions). The results are presented in Figure 6. In this figure we see the performance of MVAPICH-INCR normalized to that of MVAPICH-1.1, i.e. performance of MVAPICH-1.1 is designated to be 1. Based on the experimental results, we can see that MVAPICH-INCR performs virtually identically to MVAPICH-1.1. This result is expected as the extensions to support GASNet-INCR do not interfere with the MPI functionality. With this, we can conclude that the INCR library does not degrade MPI performance.

4.5 UPC Application Level Evaluation

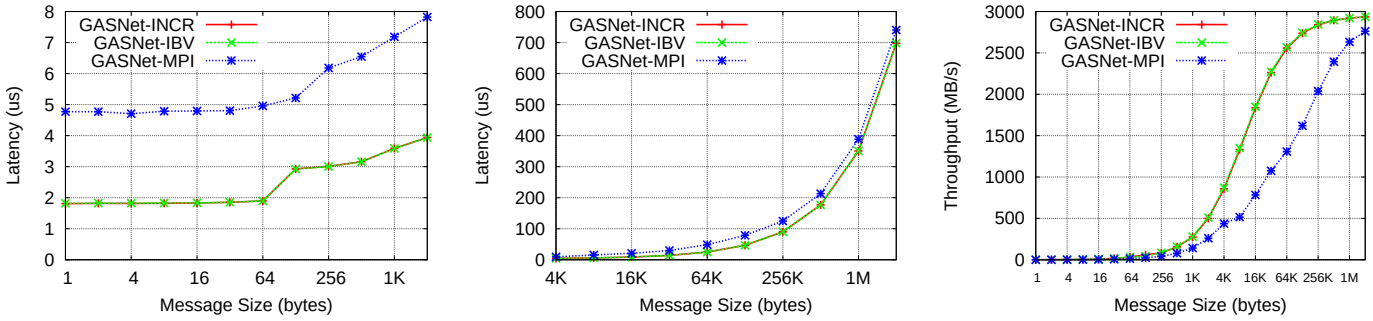
To evaluate our GASNet-INCR design, we analyzed its performance with the NAS benchmarks [22] written in UPC. The version of the UPC NPB Benchmark Suite used was 2.4. This version is distributed along with Berkeley UPC version 2.10.2 and can be found under `upc-tests/NPB2.4`. Among the NAS benchmarks, we chose to focus on Conjugate Gradient benchmark (CG), 3-D FFT PDE benchmark (FT) and the Multi-Grid benchmark (MG), for our analysis. These benchmarks were run for different problem sizes (class B and C) with 64 and 128 processes. We chose not to present experimental data for the NAS EP benchmark as it has very little communication. We used cluster C for these experiments.

The CG benchmark has the smallest problem size (75,000 for Class B and 150,000 for Class C) among the three benchmarks and has a relatively more frequent communication pattern. The communication calls from the UPC version of the NAS benchmarks mainly consist of point-to-point calls, but many of them are due to the fact that the benchmarks do not utilize UPC collectives. Rather, they implement collectives based on point-to-point operations. The execution times for these different NAS benchmarks are shown in Figure 7.

The performance results observed for the CG and MG benchmarks are indicated in Figures 7(a) and 7(b). For 128 process MG and CG benchmark, GASNet-INCR performs 23% and 11% faster as compared to GASNet-IBV conduit, respectively. One of the main difference in point-to-point performance (since the benchmark does not use collectives), is that GASNet-INCR implements a weak flow control method that is receiver driven, as opposed to sender keeping track of remote credits. This results in better utilization of concurrency in the network. Both GASNet-IBV and GASNet-INCR outperform GASNet-MPI. As can be seen from the microbenchmark results, this is due to poorer performance of the MPI conduit as it is trying to simulate active messages. The Class C version of the benchmarks generate larger volume of data, which makes the runtime more dependent on bandwidth of the network, and the differences between the three conduits is less pronounced.

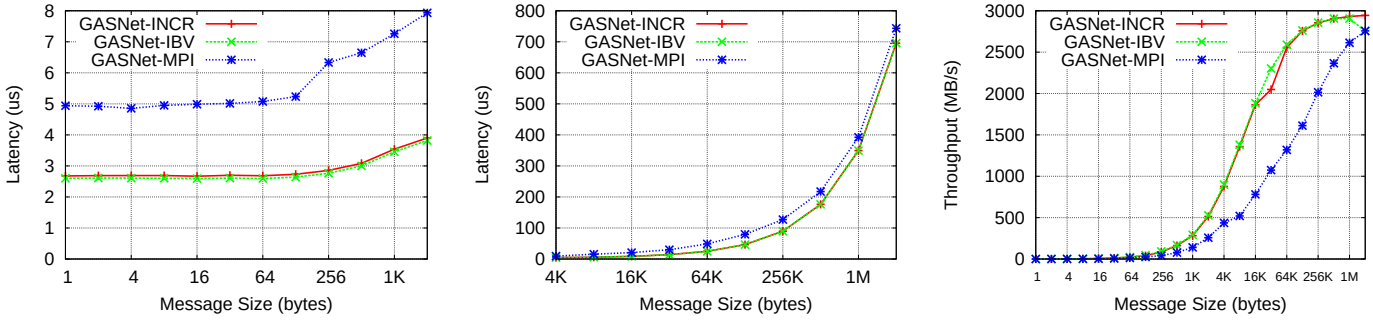
We experimented with two versions of the FT benchmark, one unmodified (i.e. from the UPC NAS 2.4 distribution), and the second was slightly modified to reduce network hot-spotting. We modified it to remove an artificial limitation of the benchmark. The performance results for the FT benchmark are shown in Figures 7(c) and 7(d). Figure 7(c) shows the performance of the unmodified version, whereas Figure 7(d) shows the performance of the modified version. We note that the modified version performs significantly better than the unmodified version, e.g. the modified version performs 21% better than the unmodified version for FT Class C at 128 processes using GASNet-IBV conduit. The time spent in FT is dominated by the Alltoall phase, where distributed matrices are transposed. The unmodified version performs the transpose using the pattern: `for(i=0;i<NTHREADS;i++) { upc_memget(i); }`

Obviously, this results in hot-spotting threads as all threads try to simultaneously access memory from thread 0, then thread 1 and so on. The modified version simply changes the access pattern to distribute it better between the threads: `for(i=0;i<NTHREADS;i++) { upc_memget(i+MYTHREAD%NTHREADS); }` It is evident from the results that even in the modified version of FT, GASNet-INCR is able to provide equal or better performance (3% better for FT, Class C on 128 processes).



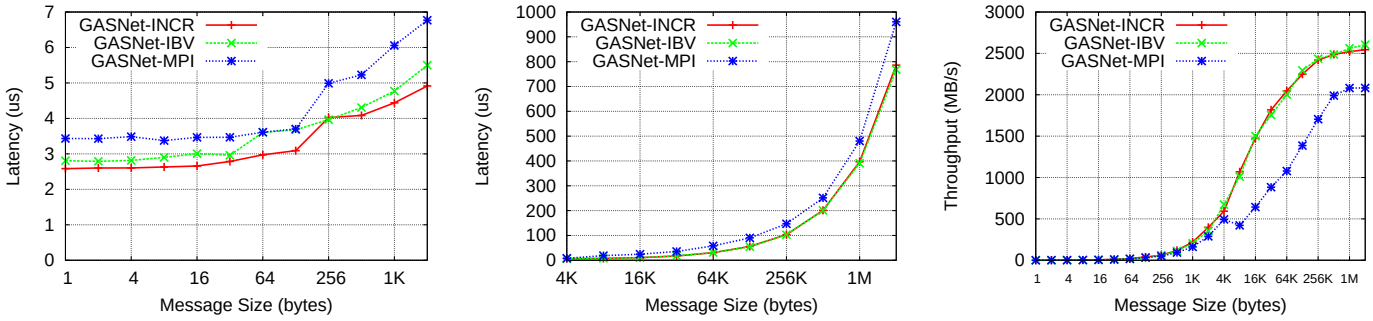
(a) Memput latency: Small (b) Memput latency: Large (c) Memput bandwidth

Figure 2. Memput performance



(a) Memget latency: Small (b) Memget latency: Large (c) Memget bandwidth

Figure 3. Memget performance



(a) bupc Memput latency: Small (b) bupc Memput latency: Large (c) bupc Memput bandwidth

Figure 4. Berkeley UPC (signal) memput performance

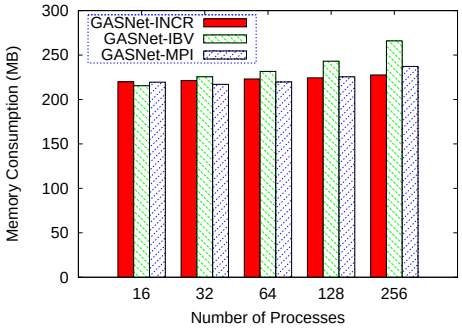


Figure 5. Memory consumption with increasing process group size

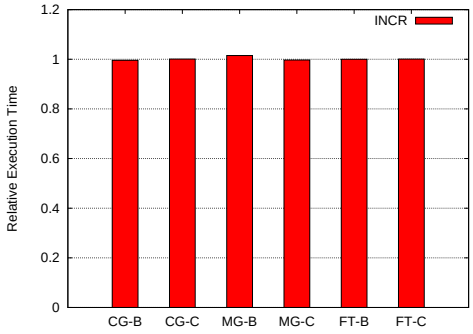
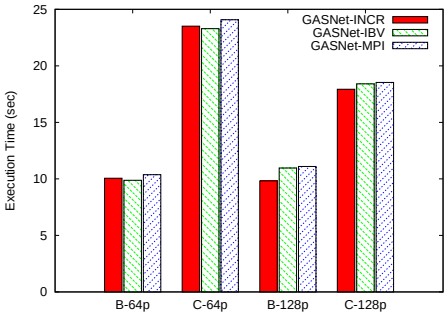
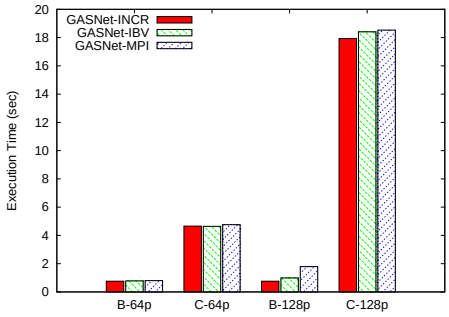


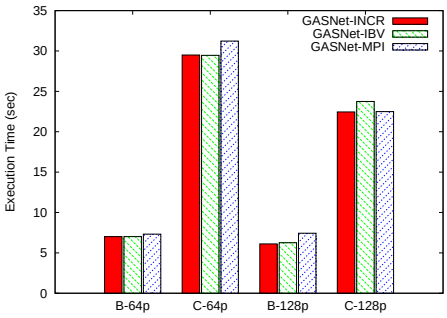
Figure 6. Performance comparison of INCR with MVAPICH-1.1 with pure MPI NAS Benchmarks on 128 processes



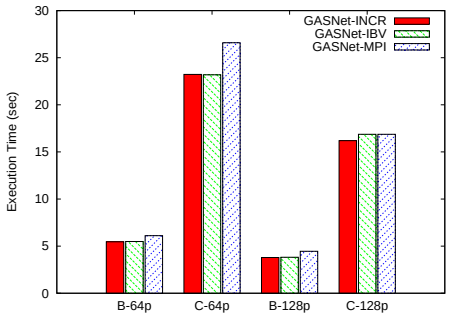
(a) CG Class B and C (64 procs and 128 procs)



(b) MG Class B and C (64 procs and 128 procs)



(c) FT Class B and C (64 procs and 128 procs)



(d) Modified FT Class B and C (64 procs and 128 procs)

Figure 7. Performance of pure UPC NAS Benchmarks

5. Conclusions and Future Work

During the past few years, there has been an increasing interest in improving programmer productivity for scientific applications. UPC and other PGAS languages have emerged as alternatives to the still popular MPI programming model. At the same time, InfiniBand technology is being widely deployed and very large clusters, consisting of many tens-of-thousands of cores are being deployed for conducting ground-breaking science. In this context, there is a need for a communication library that supports both UPC and MPI programming models, provides excellent performance for them and is highly scalable. In this paper, we have presented our designs for Integrated Native Communication Runtime (INCR) for MPI and PGAS on InfiniBand clusters. Our design and evaluation reveal that we are able to achieve our primary design objectives. For UPC NAS benchmarks CG and MG (class B) at 128 processes, GASNet-INCR outperforms the GASNet-IBV runtime by 10% and 23%, respectively. Further, memory scalability analysis reveals that GASNet-INCR is highly scalable.

In the future, we aim to continue working in this direction. In this paper we presented performance results for UPC using our communication runtime. We plan to extend our evaluation to hybrid MPI and UPC applications. Our team is involved in optimizations of several well-known applications from the San Diego Supercomputer Center and Texas Advanced Computing Center. We plan to investigate optimization opportunities by using hybrid programming models in those applications. Our aim is to enable generalized PGAS support in INCR in the future.

Acknowledgments

This research is supported in part by U.S. Department of Energy grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; National Science Foundation grants #CNS-0403342, #CCF-0702675, #CCF-0833169 and #CCF-0916302; grants from Intel, Mellanox, Cisco, QLogic, and Sun Microsystems; Equipment donations from Intel, Mellanox, AMD, Appro, Chelsio, Dell, Fujitsu, Fulcrum, Microway, Obsidian, QLogic, and Sun Microsystems. We would like to thank Mr. Jonathan Perkins and Mr. Mark Arnold of The Ohio State University for extending help in setting up the experimental systems on a short notice. We would also like to thank the anonymous reviewers for their helpful suggestions and constructive criticism.

References

[1] Parallel Environment for AIX 5L MPI Programming Guide. (v4.1.1): Using MPI and LAPI in the same program.

[2] Berkeley UPC User's Guide: Mixing UPC and C/C++/MPI/FORTRA. `http://upc.lbl.gov/docs/user/interoperability.shtml`.

[3] C. Barton, C. Casçaval, G. Almási, Y. Zheng, M. Farreras, S. Chatterjee, and J. N. Amaral. Shared memory programming for large scale machines. In *PLDI '06: Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation*, pages 108–117, New York, NY, USA, 2006. ACM. ISBN 1-59593-320-4. doi: http://doi.acm.org/10.1145/1133981.1133995.

[4] C. Bell and D. Bonachea. A New DMA Registration Strategy for Pinning-Based High Performance Networks. In *Workshop on Communication Architecture for Clusters (CAC), in conjunction with Int'l Symposium on Parallel and Distributed Computing (IPDPS)*, 2003.

[5] C. Bell and D. Bonachea and R. Nishtala and K. Yelick. Optimizing Bandwidth Limited Problems Using One-Sided Communication and Overlap. In *Int'l Symposium on Parallel and Distributed Computing (IPDPS)*, 2006.

[6] B. Chamberlain, D. Callahan, and H. Zima. Parallel Programmability and the Chapel Language. *Int. J. High Perform. Comput. Appl.*, 21(3):291–312, 2007. ISSN 1094-3420. doi: http://dx.doi.org/10.1177/1094342007078442.

[7] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar. X10: an object-oriented approach to non-uniform cluster computing. In *OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 519–538, New York, NY, USA, 2005. ACM. ISBN 1-59593-031-0. doi: http://doi.acm.org/10.1145/1094811.1094852.

[8] D. Bonachea and J. Duell. Problems with using MPI 1.1 and 2.0 as Compilation Targets for Parallel Language Implementations. *Intl. J. High Performance Computing and Networking*, 1(1/2/3):91–99, 2004.

[9] D. Buntinas and W. Gropp. Designing a Common Communication Subsystem. In *Euro PVM/MPI Conference*, 2005.

[10] Editor: Dan Bonachea. GASNet specification v1.1. Technical Report UCB/CSD-02-1207, U. C. Berkeley, 2008.

[11] Environmental Molecular Sciences Laboratory and Pacific Northwest National Laboratory. The GA Toolkit. `http://www.emsl.pnl.gov/docs/global/`.

[12] Environmental Molecular Sciences Laboratory, Pacific Northwest National Laboratory. Aggregate Remote Memory Copy Interface (ARMCI). `http://www.emsl.pnl.gov/docs/parsoft/armci/`.

[13] IBM. Message Passing Interface and Low-Level Application Programming Interface (LAPI). `http://www-03.ibm.com/systems/software/parallel/index.html`.

[14] InfiniBand Trade Association. InfiniBand Industry Standard Specification. `http://www.infinibandta.org/`.

[15] M. Koop, T. Jones, and D. K. Panda. Reducing Connection Memory Requirements of MPI for InfiniBand Clusters: A Message Coalescing Approach. In *7th IEEE Int'l Symposium on Cluster Computing and the Grid (CCGrid07)*, May 2007.

[16] M. Koop, S. Sur, Q. Gao, and D. K. Panda. High Performance MPI Design using Unreliable Datagram for Ultra-Scale InfiniBand Clusters. In *21st ACM International Conference on Supercomputing (ICS07)*, June 2007.

[17] M. Koop, T. Jones, and D. K. Panda. MVAPICH-Aptus: Scalable High-Performance Multi-Transport MPI over InfiniBand. In *IEEE Int'l Parallel and Distributed Processing Symposium (IPDPS 2008)*, April 2008.

[18] M. Koop, J. Sridhar, and D. K. Panda. Scalable MPI Design over InfiniBand using eXtended Reliable Connection. In *IEEE Int'l Conference on Cluster Computing (Cluster 2008)*, September 2008.

[19] Lawrence Berkeley National Laboratory and University of California at Berkeley. Berkeley UPC - Unified Parallel C. `http://upc.lbl.gov/`.

[20] J. Liu, J. Wu, and D. K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. In *In 17th Annual ACM International Conference on Supercomputing (ICS)*, 2003.

[21] D. Loveman. High performance Fortran. *Parallel Distributed Technology: Systems Applications, IEEE*, 1(1):25 –42, feb. 1993. ISSN 1063-6552. doi: 10.1109/88.219857.

[22] National Aeronautics and Space Administration (NASA). NAS Parallel Benchmarks. `http://www.nas.nasa.gov/Resources/Software/npb.html`.

[23] Network-Based Computing Laboratory. MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE. `http://mvapich.cse.ohio-state.edu/`.

[24] R. Nishtala and P. Hargrove and D. Bonachea and K. Yelick. Scaling Communication-Intensive Applications on BlueGene/P Using One-Sided Communication and Overlap. In *Int'l Symposium on Parallel and Distributed Computing (IPDPS)*, 2009.

[25] Remote Memory Access Working Group. MPI3 RMA Wiki. `https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/RmaWikiPage`.

[26] S. Sur, L. Chai, H.-W. Jin, and D. K. Panda. Shared Receive Queue Based Scalable MPI Design for InfiniBand Clusters. In *IEEE Int'l*

Parallel and Distributed Processing Symposium (IPDPS 2006), April 2006.

[27] UPC Consortium. UPC Language Specifications, v1.2. Technical Report LBNL-59208, Lawrence Berkeley National Lab, 2005.