Host-Assisted Zero-Copy Remote Memory Access Communication on InfiniBand

V. Tipparaju[†] G. Santhanaraman[‡] J. Nieplocha[†] D.K. Panda[‡]

† Pacific Northwest National Laboratory
‡ Ohio State University

Abstract

The remote memory access (RMA) is becoming an increasingly important communication model due to its excellent potential for overlapping communication and computations and achieving high performance on modern networks with RDMA hardware such as Infiniband. RMA plays a vital role in supporting the emerging global address space languages and management of advanced distributed data structures. This paper describes how remote memory access communication (RMA) can be implemented efficiently over InfiniBand based on the 'zero-copy' approach. The capabilities not offered directly by the Infiniband verb layer can be implemented efficiently using the novel host-assisted approach while achieving zero-copy communication and supporting a high degree of overlapping computations and communication. For contiguous case we are able to achieve a small message latency of 7.44µs and a peak bandwidth of 730 MB/s for 'put' and a small message latency of 15µs and a peak bandwidth of 689 MegaBytes for 'get'. These numbers are almost as good as the performance of the native VAPI layer. For the noncontiguous case, with our host assisted approach, we can support close to the peak bandwidth that was achieved for the contiguous data. We also demonstrate the superior tolerance of host-assisted data-transfer operations to CPU intensive tasks due to minimum host involvement in our approach as compared to the traditional host-based approach. Our implementation also supports a very high degree of overlap of computation and communication. 99% overlap for contiguous and up to 95% for non contiguous in case of large message sizes were achieved. Finally, the NAS MG and parallel matrix multiplication benchmarks were used to validate effectiveness of our approach, and demonstrated excellent overall performance.

1. Introduction

The demand for computer cycles in scientific simulation is growing faster than the processor speed described by Moore's law. To mitigate the impact of this trend, parallel systems employ increasingly large numbers of processors. At the same time, the gap between processor, memory, and network speed is not improving but getting worse. Even to sustain the scalability and performance levels of current leading scientific applications, progress needs to be made in implementation of the user-level communication protocols. First, zero-copy communication protocols are of increased importance because they remove memory performance factor from the communication performance model and help avoid wasting the valuable and limited memory bandwidth of the compute nodes. The limited memory bandwidth is often pointed out as a major issue affecting application efficiency in current systems based on commodity processors [1]. Second, the ability to overlap communication with computation as a simple and well understood latency-hiding mechanism is essential for addressing the growing gap between the network and processor speed. Memory copies used internally to implement the user-level communication protocols require host involvement and thus reduce the potential for effective overlapping nonblocking communication with computation. Because zero-copy protocols do not require memory copies, they are a more attractive approach for supporting latency hiding through nonblocking communication.

In this paper, we are focusing on the remote memory access (RMA) communication model. RMA offers several desirable properties such as the lack of explicit coordination between sender and receiver and simplified flow control (does not involve tag matching or handling or early message arrivals). RMA is well suited for zero-copy nonblocking implementation. Current communication networks offer increasing levels of support for RMA communication. The RMA model has been available in the user-level communication libraries such as SHMEM, MPI-2 1-sided, ARMCI, and Global Arrays. It is also the preferred communication model for implementing the emerging global address space languages such as UPC [2] or CAF [3]. We are working on advancing ARMCI, a portable RMA library used as a part of the run-time system developed by the Center for Programming Models for Scalable Parallel Computing project (www.pmodels.org) sponsored by the U.S. Department of Energy. In particular, the current goal is to provide efficient communication capabilities that could be used for latency hiding and reducing communication overhead in language- and library- based programming models and for devising implementation techniques that enhance the overall application performance.

The cost-effectiveness and performance of InfiniBand makes this technology a very attractive network for commodity clusters. This paper evaluates the performance and capabilities of InfiniBand in the area of RMA communication. It describes how to harness the InfiniBand verbs layer to implement RMA efficiently while addressing the requirements of the user-level protocols by implementing ARMCI one-sided RMA on top of InfiniBand. In addition, we describe a novel implementation approach called host-assisted zero-copy RMA. It can be used to implement the missing RMA capabilities in the network communication protocols while achieving zero-copy communication and maximizing the potential for overlapping communication with computation. In the context of InfiniBand, this approach has been used for noncontiguous RMA communication, which has limited support in the InfiniBand verbs standard. This has been accomplished using nonblocking scatter-gather point-to-point messaging interfaces of the Mellanox VAPI layer and a special helper thread. For other networks with even more limited support for RMA (e.g., VIA), this technique can be used to efficiently implement RMA Get protocol on top of RMA Put while minimizing host involvement and preserving zero-copy processing.

The effectiveness of these techniques has been evaluated across two different platforms with InfiniBand interconnect. For the contiguous case, we are able to achieve a small message latency of 7.44µs and a peak bandwidth of 730 MegaBytes for 'Put' and a small message latency of 16µs and a peak bandwidth of 689 MegaBytes for 'Get'. For the non contiguous case with the host based approach we achieved close to the peak bandwidth and very close to the contiguous case. The proposed host-assisted approach delivered superior tolerance to CPU intensive tasks because of the minimal host involvement. Our implementation of RMA protocols supports up to 99% and 95% overlap for contiguous and noncontiguous operations respectively for large message sizes. The benefits of this approach were demonstrated at the application level in the context of the NAS MG benchmark and in the dense matrix multiplication.

The paper is organized as follows. Section 2 provides an overview of RMA communication. Section 3 describes InfiniBand architecture and its capabilities. In Section 4, we present the implementation of basic RMA capabilities over InfiniBand and evaluate their performance. Section 5 describes our novel host-assisted protocol and demonstrates its performance benefits in the context of noncontiguous data communication. An application-level performance evaluation is presented in Section 6. Our conclusions are offered in Section 7.

2. RMA Communication

Remote memory operations offer an intermediate programming model between message passing and shared memory. This model combines some advantages of shared memory, such as direct access to shared/global data, and the message-passing model, namely the control over locality and data distribution. Certain types of shared memory applications can be implemented using this approach. In some other cases, remote memory operations can be used as a high-performance alternative to message passing. Many such applications are characterized by irregular data structures and dynamic or unpredictable data access patterns. MPI-2 offers one version of remote memory operations with two specific variations—*active* and *passive target* one-sided communication. Other versions are found in vendor specific interfaces such as LAPI on the IBM SP, RDMA on the Hitachi SR-8000, MPlib on the Fujitsu VPP-5000, and in other portable interfaces such as ARMCI [4] or SHMEM [5]. Differences between these models can be significant in terms of progress rules and semantics, and they can affect performance. MPI-2 offers a model closely aligned with traditional message passing and includes high-level concepts such as windows, epochs, and distinct progress rules for passive and active target communication. A recent paper [6] points out the MPI-2 model is not optimal for implementing global address space languages due to excessive synchronization and its progress rules.

In ARMCI, we are focusing on a low-level interface and simpler progress rules motivated by the existing hardware support for remote memory operations on the current systems. The library is intended to be used as a run-time system for other programming models such as Global Arrays [7], Co-Array Fortran [8] or UPC compilers, or even as a portable SHMEM library [9]. Compared to the well known Cray SHMEM one-sided interface [5], ARMCI places more focus on noncontiguous data transfers that correspond to data structures in scientific applications (e.g., sections of multidimensional dense or sparse arrays). Such transfers can be optimized, thanks to the noncontiguous data interfaces available in the ARMCI data transfer operations—multi-strided and generalized UNIX I/O vector interfaces.

Some networks and native communication interfaces on these networks do not have direct support for all the RMA operations offered by the portable interfaces discussed above. Other networks have a rich functionality set but introduce substantial performance compromises. For example, IBM LAPI [10], an active message library for the IBM SPs, does support contiguous and noncontiguous RMA but is not copy-free and requires use of the host CPU on both sides of data transfer. As the memory copies degrade performance and host CPU resources are taken away from the application, this approach usually has an adverse affect on the overall application performance and scalability. To maximize application performance, it is important to avoid data movement and protocol processing on the remote side as much as possible. RMA models that require explicit synchronization might incur overhead on the part of the application running on the remote side. For example, the MPI-2 one-sided operations involve synchronization between the source and the destination for every one-sided operation via a fence, a lock, or dedicated Post-Wait coordination in the active target mode [11, 12], at least in principle.

Despite the progress in networking technologies, the gap between processor speed and network (especially latency) has been increasing. As a result of this trend, the ability to overlap communication with computation through the use of nonblocking communication is becoming critical. Given the simplicity of its communication model (source and destination for the data transfer explicitly known, no send/receive tag and buffer matching, no early message arrival processing), RMA offers increased opportunities for designing implementations that provide a high degree of overlap between communication and computations. In addition to reducing or eliminating the data movement on the remote side, the quality implementation of the nonblocking RMA should return the control to the user program as soon as possible, giving the application a chance to make progress on computations while the communication is being completed by the underlying network hardware.

3. Overview of InfiniBand

InfiniBand is a recently developed interconnect technology that has been rapidly becoming popular in the commodity clusters. The InfiniBand architecture is an industry standard introduced by the InfiniBand Trade Association and has been proposed as the next-generation interconnect for I/O and inter-process communication. The InfiniBand architecture defines a system area network (SAN) for connecting multiple independent processor platforms, I/O platforms, and I/O devices. It uses scalable switched serial links to design clusters and servers that can offer high bandwidth and low latency. A 4x HCA link allows for a bandwidth of up to 10 Gb/s. In an InfiniBand network, nodes are connected to the IBA fabric using channel adapters. The inter-processor communication is handled by the host channel adapters (HCA) installed on the processing nodes. The I/O nodes are connected to the fabric through target channel adapters (TCA). The IBA hardware offloads much of the I/O communications operation from the OS and CPU, thus eliminating traditional communication overhead. Further, each channel adapter may have one or more ports for use as multiple paths to provide reliability. HCA, TCA, switch, routers, and a subnet manager form the five primary components of an InfiniBand fabric.

Unlike VIA, InfiniBand architecture does not specify an API even though it does incorporate many of the concepts of VIA. IBA defines a semantic interface called Verbs that configures, manages, and operates a HCA. The communication verbs are based on queue pairs. InfiniBand supports both channel (send/receive) and memory (RDMA) communication semantics. These operations are initiated by posting work queue requests on the send or receive queues. The completion of a work request is reported through completion queues (CQs). The host communication buffers have to be registered because the HCA uses DMA operation to send from or receive into these buffers.

VAPI is the Verbs implementation provided by Mellanox Technologies for HCAs. The VAPI interface closely follows the verbs specification of the IBA, and several VAPI functions are mapped directly from the corresponding verbs functionality. In addition to basic send/receive and RDMA read/write, they provide scatter/gather operations as well as atomic operations, thus supporting several essential interfaces of the RMA communication. With the communication capabilities provided by the new InfiniBand VAPI as an example, one also can implement some of the RMA capabilities that are not directly provided by the underlying communication layer.

4. Implementing the Basic RMA Capabilities over InfiniBand

A mismatch between user-level RMA interfaces and InfiniBand requirements is related to the virtual memory. The native RDMA write and read operation on InfiniBand can address only so-called registered memory for both sides of the data transfer. Memory registration involves locking pages in physical memory, which can be quite costly. In addition, the amount of memory that can be registered/locked is limited. This constraint has a profound impact on the implementation strategies of user-level RMA on this network. Three techniques or strategies address the requirement for registered memory in InfiniBand: 1) on-demand dynamic memory registration and deregistration (or lazy deregistration) as a part of the data transfer; 2) copying data via preallocated registered memory buffers (we refer to this as the host-based/buffered technique); and 3) providing the user with a memory allocation interface that allocates registered memory underneath.

The first technique is potentially more attractive, as it provides zero-copy data transfers and eliminates the need for data copy present in the second strategy. However, it does not always lead to superior performance, as the memory registration operations can be relatively expensive. Figure 2 shows on log-linear scales the performance of memory registration operations (registration and deregistration calls combined) in InfiniBand compared to the bandwidth of the memory copy operation. The assembly-coded memory copy uses Intel IA32 MMX registers, write combining, and prefetching instructions.

We implemented all three strategies described above. Because of the cost of memory registration on InfiniBand, Strategy 1 is not very competitive. An enhancement to the second technique is to divide data into chunks and pipeline the memory copy and nonblocking communication so that they overlap. Based on the message size, the message transmission/reception can be broken into smaller requests; a copy of one part of the request can be overlapped with the transmission of another piece, as described in [14]. Figure 1 shows the steps involved in a host-based/buffered protocol.

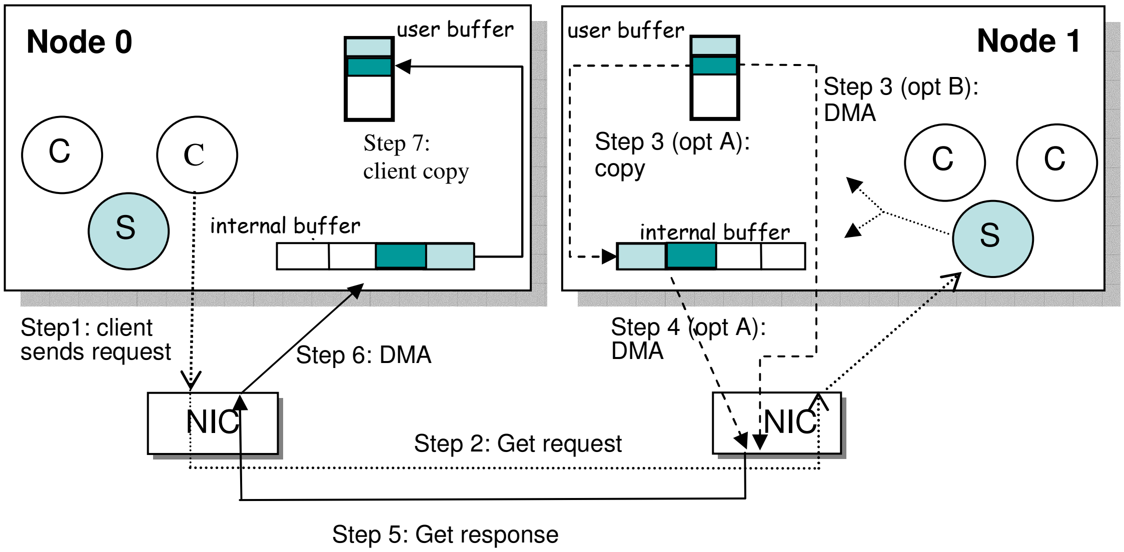

Figure 1: Host-based pipelined Get protocol

To enable implementations of all three strategies to coexist, special care is needed. First, the user can optionally call the provided memory allocator interface, which attempts to allocate registered memory. Because the memory is registered/pinned on a page basis, the memory is allocated from the operating systems in potentially larger chunks and managed by a portable K&R malloc code. In addition, there is a table of the registered chunks with the address range and *VAPI memory key* information. If the requested amount of memory can be allocated but not registered, the appropriate entry in the table is not added. When placing an RMA data transfer call, the user does not have to be concerned about whether memory on either/both sides of the data transfer has been registered. We simply compare the specified address range to the entries in the table; by increasing the granularity of the memory segments, we can limit the number of entries in the tables and the associated verification cost, if the specified address range fits in an entry in the table, we can use the InfiniBand zero-copy RDMA Read/Write protocol directly. Otherwise, depending on the size of the message, either Strategy 1 or 2 (described above) is used.

4.1 Performance of basic put/get operations

We used two different platforms for evaluating the efficiency of our implementation and analyzing the performance of different protocols. The first (henceforth referred to as cluster-1) is a dual processor -GHz Itanium2 cluster with Mellanox A1 "Cougar" cards. The second one (referred to as cluster-2) is a 32-node dual processor Pentium IV cluster with Mellanox A1 cards. The cluster-2 was used only in the comparison of NAS MG benchmark because a bigger configuration was necessary for understanding the impact of these protocols on application benchmarks.

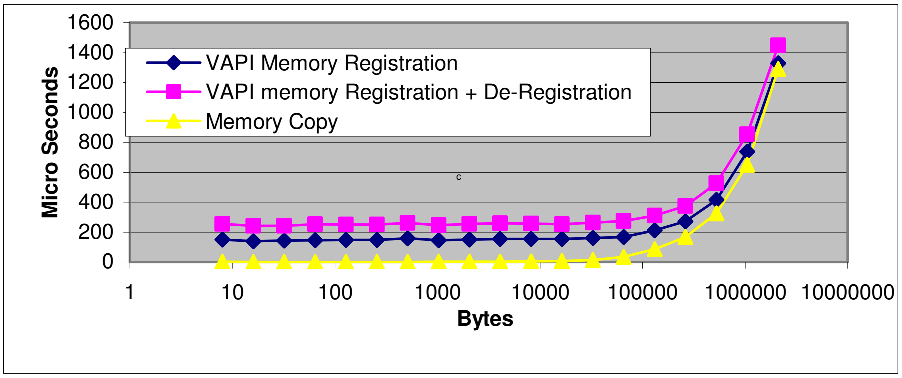

Figure 2: Cost of memory registration and deregistration with the Mellanox VAPI interface as compared to the memory copy bandwidth on Itanium-2 1000MHz processor.

Figures 3 and 4 show the performance of zero-copy contiguous ARMCI Get and Put operations. Figure 3 compares the bandwidths of ARMCI Put operation with MPI send/receive, Mellanox VAPI RDMA Put, and Mellanox VAPI send/receive. For computing Mellanox VAPI bandwidth, a performance test "perf_main" provided by Mellanox was used. This Mellanox test chains multiple RDMA's in computing bandwidth and hence doesn't end up computing the actual average point to point bandwidth. ARMCI_Put bandwidth however seems slightly lower but is very representative of what an application using ARMCI put can expect as it computes an average of the actual point to point bandwidth. For the MPI bandwidth, a nonblocking send/receive-based test was used [13]. Figure 4 shows the ARMCI Get bandwidth as compared to the Mellanox VAPI RDMA get operation. It can be seen from the figures that ARMCI operations have been implemented with very little overhead. We obtained a peak bandwidth of 730 MBps for a Put operation and 689 MBps for a get operation. It should be noted here that unlike MPI send/receive, ARMCI Put / Get are based on a less restrictive one-sided communication model.

Figure 5 compares performance of zero-copy and the two copy-based implementations: host-based/buffered (baseline) and pipelined. The pipelined version uses the scheme we had developed earlier [14]. It relies on dividing the data into multiple, variable-sized chunks and exploits the nonblocking RDMA Read/Write communication to overlap memory copies on the client and server side with data transmission. To improve performance for smaller requests, the chunk size is adaptively chosen to maximize the concurrency between memory copies and data transmission operations on both sides involved in the data transfer.

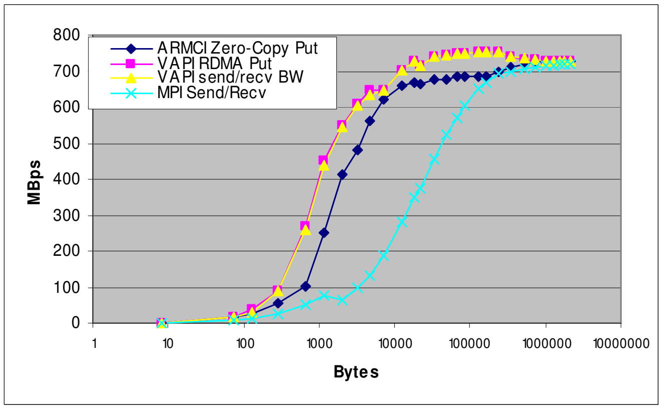

Figure 3: ARMCI Put Bandwidth in comparison to Raw VAPI bandwidth and MPI

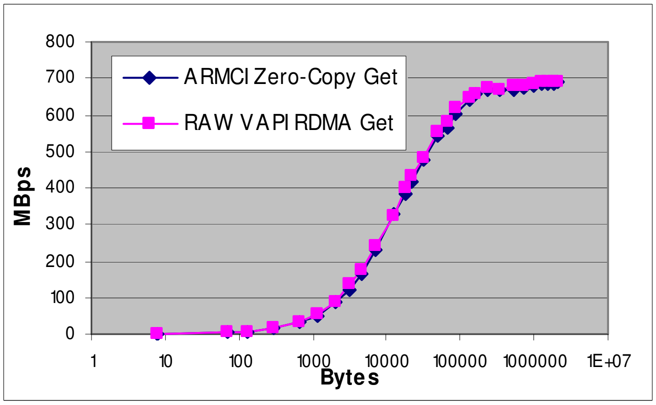

Figure 4: ARMCI Get Bandwidth in comparison to RAW VAPI get bandwidth

Although the pipelined version delivers good performance, it relies on the remote host participation in the data transfer. Therefore, the numbers presented in Figure 5 do not reflect the operational regimes in actual applications where remote CPU is involved in computations. We designed a test to measure the impact of a remote host CPU engaged in calculations and found that it does, in fact, bring performance down in the copy-based (host-based/buffered and host-based/pipelined) implementations (Figure 6). As expected, the performance of the zero-copy version is virtually immune to the remote host activities, whereas the performance of copy-based protocols is seriously degraded.

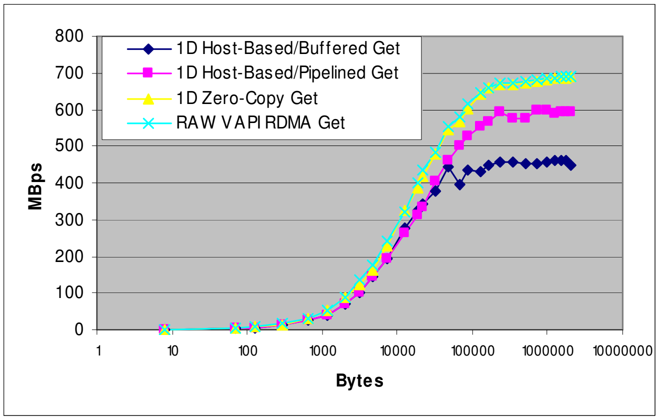

Figure 5: Bandwidth comparison for different protocols supporting the contiguous get data transfers with remote side idle.

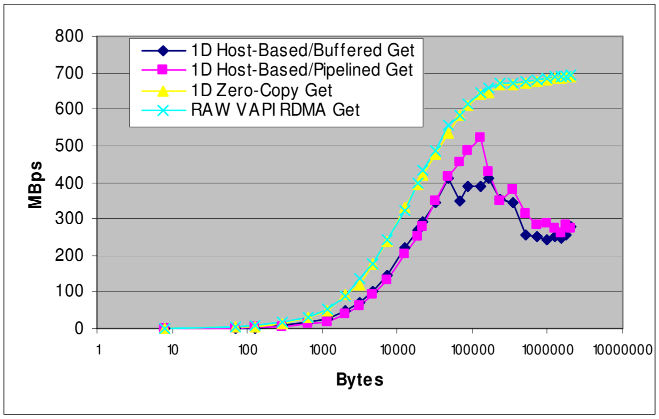

Figure 6: Bandwidth comparison for different protocols supporting the contiguous get data transfers with remote side busy.

4.2 Overhead and overlap

One of the key design requirements is reducing the implementation overhead over the VAPI layer. Another one is to maximize the potential for overlap between user computations and nonblocking communications. At the initiation of a call, the most appropriate protocol for efficient transmission of that message is selected based on the message size. The receiver thread on the remote side has the capability to select either a polling or blocking mode of operation, depending on the processing resources available on the system. Latency of our implementation is very comparable to the lowest attainable latencies of the VAPI layer. This is due to 1) direct use of RDMA capabilities whenever possible and 2) fast access to the registered memory information to determine if the current operation can directly use RDMA. The latencies of ARMCI Put/Get and MPI are contrasted with the VAPI level latencies obtained from the Mellanox VAPI layer in Table-1.

ARMCI Put	MPI	VAPI RDMA Put	VAPI RDMA Get	ARMCI Get
7.44	8.03	5.84	15	16

Table-1: ARMCI, MPI and VAPI latencies

The overlap attainable by contiguous ARMCI operations is very close to that obtained with the Mellanox VAPI layer (see Figure 7). It is close to 99% for large messages, which means that 99% of the total data transfer time can be overlapped by the application to do the computation. In comparison to the zero-copy get, the host-based buffered protocol offers relatively less overlap because, as a part of a wait operation in the critical path, data needs to be copied. For the Itanium-2 systems, the percentage of total time that was not overlapped was almost same as the percentage of total time it took for posting a descriptor and polling for a completed descriptor (it is measured as 8 microseconds on average in cluster-1 used for our tests). Thus the overlap attainable at the ARMCI level for all the message sizes in the zero-copy case is almost equal to the maximum amount of overlap attainable directly at the VAPI level.

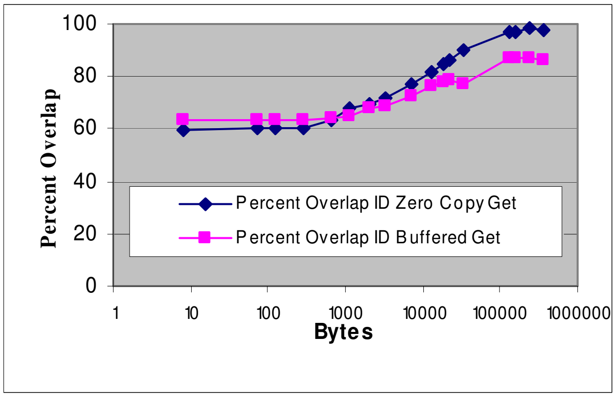

Figure-7(a): Percentage overlap of Zero Copy and Host-Based Buffered Get 1D get

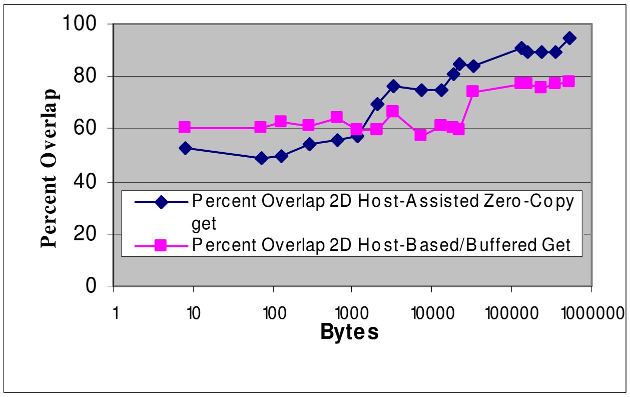

Figure-7(B): Percentage overlap of Zero Copy and Host-Based Buffered 2D Get

4.3 Atomic operations

The atomic Read-Modify-Write (RMW) operation is a very useful primitive for implementing mutual exclusion, shared task counters (e.g., in dynamic load balancing), and more complex synchronization operations. Unlike MPI-2 that offers no support for RMW, ARMCI offers *atomic swap*' and *atomic fetch and add* for both intermediate and long data types as part of the RMW interface. We used two techniques for implementing these operations. The first technique is server-based; the second technique bypasses the server thread by using VAPI RDMA atomic calls.

For the server-based implementation, the client sends the request message to the data-server thread on the remote side [14]. The data server thread executes the operation on behalf of the requesting process and sends the result back to the client. To achieve atomicity, the server needs to lock the local memory, perform the operation, and unlock the memory before sending the result. In the second implementation, we exploit the atomic operations provided by the VAPI interface to enable a faster RMW operation. The atomic operations provided by VAPI are *atomic fetch and add* and *atomic compare and swap* with an operand size of 64 bits, which restricts their usefulness. The VAPI atomic operations are one-sided and do not involve host overhead on the remote node. The operation can be completed either through polling or through event-based notification, which involves registering a function handler to notify completion. In our implementation, we use the polling-based approach for performance considerations. The requesting process posts an *atomic fetch and add* and then polls the send completion queue for the completion of the atomic operation.

We compared the two implementations of ARMCI RMW for the atomic fetch and add operation. The results show that the VAPI atomic-based implementation cuts down the latency of the RMW operation by about 23%, reducing it from 22.1 µs for the server-based implementation to 17.1 µs in the RDMA implementation, in addition to eliminating host involvement on the remote node.

5. Host-Assisted Zero-Copy RMA

The IBA verbs layer has some inadequacies in providing support for all the RMA capabilities required by applications. We attempted to address its lack of support in providing one-sided noncontiguous strided and vector data transfers. A simple way of addressing the lack of support for one-sided data transfer between a strided/vector source and a strided/vector destination is to maintain a contiguous buffer on both the local and the remote side and move data using this contiguous buffer. This approach requires heavy involvement on both the local and remote sides in moving the data between the buffer and the noncontiguous source or destination. Another approach that can be used here is to do multiple contiguous transfers for each contiguous chunk. This approach is zero-copy but may require the initiator of the request to spend some time in processing the multiple contiguous requests it has to initiate for every noncontiguous request. In addition, handling flow control issues like the number of outstanding requests allowed might adversely affect performance. We introduced a host-assisted zero-copy method to address the problems inherent in both the approaches described above.

To leverage the advantages of the host-assisted zero-copy approach in Mellanox VAPI, memory on both sides must be registered. The user is not expected to either explicitly register memory or keep track of this information. Instead, as described in Section 4, we maintain and parse our high-granularity global memory information table to determine if the memory on both sides is registered. The host-assisted approach requires partial involvement of a remote host to complete operations. We refer to the representative on the remote side that assists in the completion of the operation as a “helper” thread. The helper thread initiates an operation and hence requires minimal remote-side CPU involvement. This is very similar to the ARMCI *data server thread* [4, 14] and the dispatcher thread in the IBM LAPI. The significant difference is that the helper thread does not copy any data and does not wait on an operation it issued to complete.

With this helper thread as an assistant to complete the operation on the remote side, we describe the implementation details of contiguous and noncontiguous one-sided Get and Put operations. We demonstrate the benefits of this approach by contrasting its performance with the traditional host-based/buffered approach and by showing the performance of these protocols on a few application benchmarks in Section 6.

5.1 Implementation of Get operation for Contiguous Data

Because the Mellanox VAPI implements the RDMA Read operation, contiguous get can be done using the RDMA Read operation when both source and destination memories are registered. On networks that do not support RDMA Read or have limited or unoptimized RDMA Read, implementation of this scheme is inherently simple. The client doing the get operation sends a request to the helper thread on the remote node. The helper thread, upon receiving a get request, initiates a RDMA put and then would resume polling/blocking. As a result, request processing overhead on the remote host is very little. We verified the effectiveness of this approach on the Myrinet GM 1.6 [21]. The ARMCI Get latency dropped from 27μs to 24μs by using the host-assisted approach. Get bandwidth improved from 198MB for copy based protocol to 237MBps for host-assisted zero-copy contiguous get, there by showing that this approach is not just high performance but also adaptive to the platforms that don’t have support for RDMA read.

5.2 Implementation of Get Operation for Noncontiguous Data

Because a noncontiguous data transfer would involve transfer of multiple segments of data, our strategy is to use the scatter/gather message passing feature provided by IBA to achieve the *zero-copy transfer*. Using that feature, we can send /receive multiple data segments as a single message by posting a single scatter/gather descriptor. Two types of scatter/gather message-passing operations defined in IBA VAPI are 1) Gather-Send (which requires the noncontiguous data being sent to be represented as a Gather-Send descriptor) and 2) Scatter-Receive (which requires the noncontiguous destination for the receive to be specified in a Scatter-Receive descriptor format).

In a host-assisted zero-copy Put, the source sends a request to the remote side the helper thread processes the request, converts the vector/stride information in the request into a VAPI Receive-Scatter descriptor, posts the descriptor, and sends an acknowledgment to the requesting process, indicating that it has posted the necessary receive descriptor. On receiving this acknowledgment, the source process posts a Gather-Send from the VAPI Gather-Send descriptor it created while waiting for an acknowledgment from the helper thread. This directly delivers the data to the destination memory without the overhead of any intermediate copies. Although the explicit acknowledgment might seem like an overhead for large messages, when the copying cost starts to dominate, this approach performs better. It should be enabled only for multidimensional Put operations when the first stride or the size of each contiguous segment is large.

For a host-assisted zero-copy Get (Figure 8), the source node posts a Scatter-Receive descriptor to receive the vector/strided data and then sends a request to the remote host with the remote stride/vector information. The helper thread on the remote host receives the request and then posts a corresponding VAPI Gather-Send by converting the stride/vector information in the request message into a VAPI Gather-Send descriptor. The implementation of this protocol prompted us to address a number of design issues.

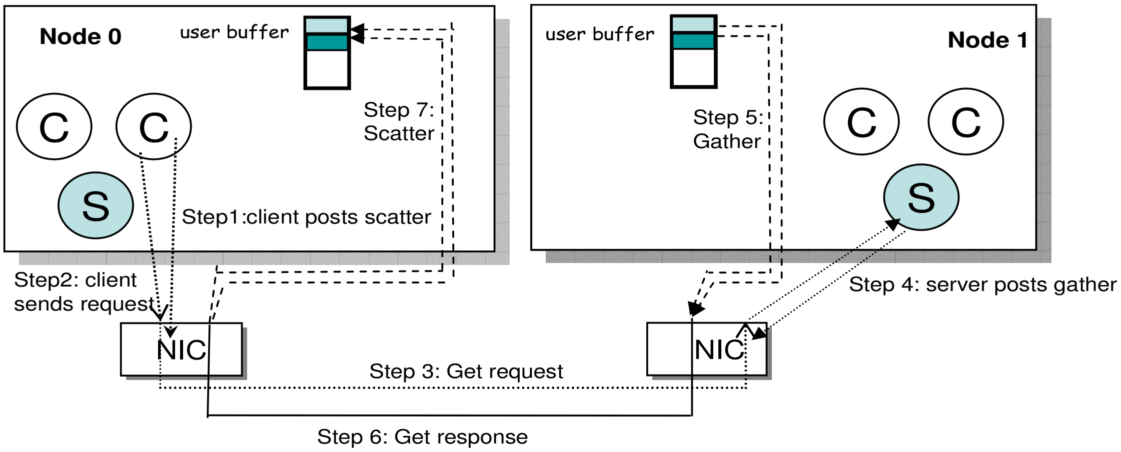

Figure 8: Host assisted get protocol for noncontiguous data

Limit on Scatter/Gather Entries per Descriptor: The strided put/get operations can be used to transfer sections of multidimensional arrays. Each dimension of the array can support any number of data segments. However, the IBA implementation puts an upper limit of 60 on the number of scatter/gather entries that can be allowed per Scatter-Receive or Gather-Send descriptor. Hence, for large messages, the maximum scatter/gather entry limit requires us to extend the above approach. Because we can have only 60 scatter/gather entries in a descriptor, our solution is to break our message into chunks of up to 60 data segments and post a gather send/scatter receive for each one of them. Posting a send/receive is a nonblocking operation in IBA and takes only a very short time (a microsecond on Itanium 1GHz), so the overhead in posting multiple gather descriptors is not significant. In the case of Strided Get, the client posts multiple scatter receives and then sends the request. At the remote side, the helper thread processes the request and posts multiple gather sends. A similar approach has been followed for implementing the noncontiguous puts.

Resource Allocation: At the client level, memory needs to be allocated and maintained to create a scatter/gather descriptor from a strided/vector request. Unlike VIA, VAPI copies the posted descriptor on to the NIC and hence does not require ud to keep the descriptor until the request has been completed. At the NIC level, the number of scatter/gather entries must be decided at the initialization phase. The larger the scatter gather list, the larger the amount of memory allocated per descriptor on the NIC. To investigate the effect of this on the performance of the operation, we conducted experiments to measure the change in latency with increasing number of scatter/gather entries. We determined that the overhead for having 60 scatter gather entries in a descriptor instead of 1 is not significant (< 1 micro sec) and hence we could afford to set the scatter/gather limit to the maximum allowed value of 60.

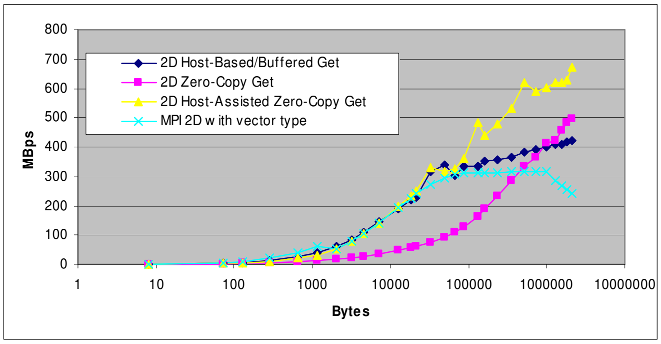

Figure 9: Bandwidth comparison for different protocols supporting the contiguous get data transfers with remote side idle.

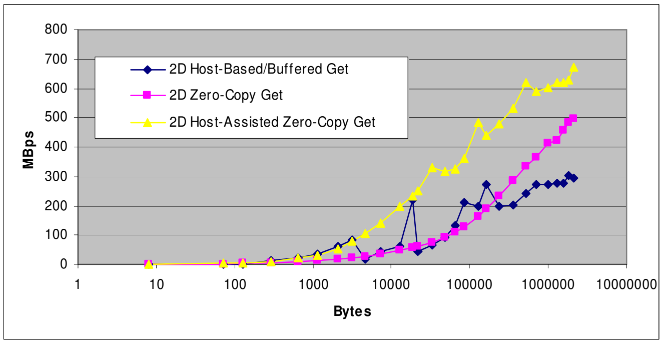

Figure 10: Bandwidth comparison for different protocols supporting the contiguous get data transfers with remote side busy.

5.3 Experimental Results of Host-Assisted Approach

We compared the performance of the different methods described above not just to contrast the host-assisted zero copy with the other two implementations but also to show the importance of using multiple protocols in achieving a sustained good performance. Figure 9 shows the performance of noncontiguous ARMCI operations. It compares the performance of host-based/buffered get and host-assisted zero-copy get operations with MPI for two-dimensional data. **Zero-Copy 2D get** in Figures 9 and 10 represents the approach discussed earlier in this section where a noncontiguous Get operation is implemented on top of multiple contiguous RDMA Get operations, one for each contiguous segment. For this test, ARMCI 2D data is represented using the strided data format [4] and in MPI using the strided data type. It is clear that the host-assisted zero copy implementation performs much better and more significantly so when the first dimension is large.

An advantage of using host-assisted zero copy can be determined by measuring the effect on protocol performance when the remote side is doing a CPU-intensive operation. Unlike the zero-copy approach, host-assisted zero-copy requires some host involvement in initiating data transfer. This is more representative of the impact these protocols may have on an application than mere measurement of communication bandwidth/latency. Figure 10 shows the performance difference between the buffered and host-assisted zero-copy protocols when the remote side is doing a CPU-intensive operation. In comparison to Figure 9, it is very clear that the performance of the host-assisted zero-copy protocol has not been affected at all by the CPU-intensive operation on the other side while the performance of the buffered Get protocol dropped very significantly. This clearly shows the very low overhead this protocol imposes on the remote-side CPU.

5.4 Overhead and Overlap

Another significant advantage of this protocol is the amount of overlap it can provide in nonblocking operations. Because the implementation does not involve any data movement in call initiation or call completion, the amount of overlap possible is much higher than that for the other protocols. This can be clearly seen in Figure 7(b), which compares the amount of overlap attainable with host-based/buffered and host-assisted protocols for a noncontiguous data transfer for various square noncontiguous chunks of data.

6. Experimental Evaluation: NAS MG and Matrix Multiplication

Reported performance numbers for RMA operations often misrepresent the actual impact of the protocol used to implement the one-sided operation on an application. A significant issue that comes to light in actual application performance in the case of one-sided operations is the ability of the operation to make progress with minimal to no remote host involvement. Unlike message passing, efficiently implemented one-sided RMA operations have the potential to complete without significant or explicit remote host involvement. In our previous work [22], we have attempted to show the advantages of the RMA programming model in comparison to a more synchronous two-sided communication model, MPI. Here the emphasis is not just on the advantages of this communication model but also on the way one-sided communication is implemented on InfiniBand, including the ability to offload the processing on the remote CPU by as much as possible so that the CPU can be more efficiently utilized by the application for computation. We used two different benchmarks, representing a sample of algorithms used in scientific computing: 1) Multigrid (MG) kernel benchmarks from the NAS suite and 2) dense matrix multiplication.

6.1 NAS MG as a benchmark for evaluating implementation efficiency

The NAS parallel benchmarks are a set of programs designed at the NASA Numerical Aerodynamic Simulation (NAS) program, originally to evaluate supercomputers. They mimic the computation and data movement characteristics of large scale computations. NAS parallel benchmark suite consists of five kernels (EP, MG, FT, CG, IS) and three pseudo applications (LU, SP, BT) programs. Our starting point was NPB 2.4 [15] implementation written in MPI and distributed by NASA. We modified NAS MultiGrid (MG) to replace point-to-point blocking and nonblocking message-passing communication calls with ARMCI one-sided RMA communication. The NAS-MG MultiGrid benchmark solves Poisson's equation in 3D using a multigrid V-cycle. The multigrid benchmark carries out computation at a series of levels and each level of the V-cycle defines a grid at a successively coarser resolution. This implementation of MG from NAS is said to approximate the performance a typical user can expect for a portable parallel program on a distributed memory computer.

NAS benchmarks are categorized into different *classes*. Their classes vary from each other in the problem size used and the number of iterations for which the solver is run. For Class A, a smaller problem size with the least iteration, the ARMCI host-based/buffered code outperforms the original message-passing implementation by 8% to 22%. The performance advantage here is because of the less restrictive programming model ARMCI uses, which allows progress without explicit remote host involvement. For the Host-based/buffered approach, the remote-side CPU is still involved in copying the data between the buffer and destination; this shows up in the overall application time. By using the zero-copy approach in ARMCI, an improvement of 14% to 25% is obtained over message-passing implementation on the benchmarks. For Class B, with the same problem size as Class A but more iterations, the ARMCI host-based/buffered approach outperforms the original message-passing implementation by 5% to 19% (see Figure 11). Again, as above, by using a zero copy ARMCI implementation, a 14% to 27% improvement can be seen over the original message-passing implementation.

6.2 Matrix Multiplication

SUMMA is a highly efficient, scalable implementation of common matrix multiplication algorithm proposed by van de Geijn and Watts [16]. The MPI version is the SUMMA code developed by its authors, which is modified to use more efficient matrix multiplication dgemm routines from Intel math libraries on Itanium rather than equivalent C code distributed with SUMMA. For comparing with the RMA version, we used the algorithm implemented using ARMCI RMA in Global Arrays. The matrix in the Global Arrays implementation of ARMCI is decomposed into blocks and distributed among processors with a two-dimensional block distribution. Each submatrix is divided into chunks. Overlapping is achieved by issuing a call to get a chunk of data while computing the previously received chunk. The minimum chunk size was 128 for all runs, which was determined empirically. The maximum chunk size was determined dynamically, depending on memory availability and the number of processors.

Experiments with matrix multiplication were run by varying the matrix size and the number of processors. The first three lines labeled in both the graphs in Figure12 represent three different approaches to implement multi-dimensional RMA in ARMCI. The host-assisted zero-copy approach was introduced in Section 5. The host-based/buffered approach and zero-copy approaches were discussed at the beginning of Section 5. The host-based/buffered approach involves two copies, one on each side; the zero-copy approach involves multiple contiguous sends for each noncontiguous message. The computations were done on four nodes with two processes each. The left side in Figure 12 is for square matrices with sizes varying from 128 to 2000. The right side in Figure 12 is for a rectangular matrix where the second dimension is set to 512 and the first dimension varies from 128 to 2000. For the square matrix (Figure 12, left), in comparison to MPI, the ARMCI host-based/buffered approach outperforms the message-passing implementation by up to 44% whereas the host-assisted zero-copy approach, because of its negligible overhead on the remote processor, outperforms the message-passing implementation by 18% to 80%.

7. Conclusions and Future Work

This paper described how the RMA communication model can be implemented efficiently over InfiniBand. The capabilities not offered directly by the InfiniBand verb layer such as noncontiguous RMA were implemented efficiently through the novel host-assisted approach to support the zero-copy communication. In addition, a high degree of overlapping computations and communication was demonstrated. The benchmarks used in the study showed effectiveness of the RMA implementation on InfiniBand and the importance of zero-copy nonblocking protocols for hiding latency in the interprocessor communication. When reimplemented to use RMA, the NAS MG and parallel matrix multiplication benchmarks when reimplemented to use RMA, achieved superior performance over their MPI counterparts. Our current approach uses the InfiniBand Reliable Connection mode, which ensures ordered delivery of messages. However, other modes such as Reliable Delivery could be investigated to evaluate the tradeoffs between the lack of message ordering in this mode and potentially increased performance.

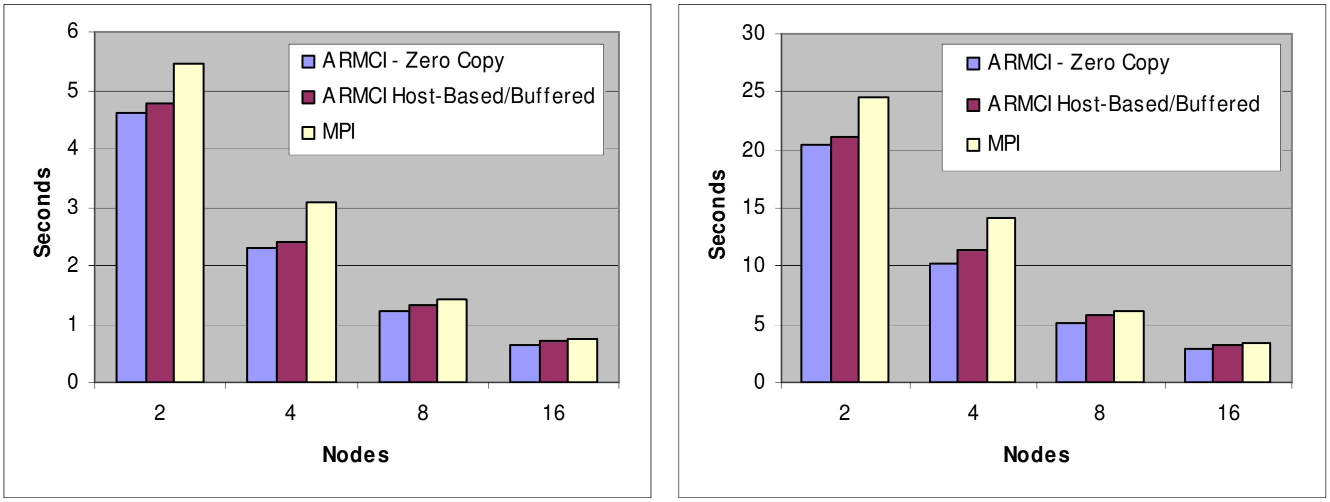

Figure-11 : Performance of NAS MG using contiguous data transfers. Left: Class A and Right: Class B

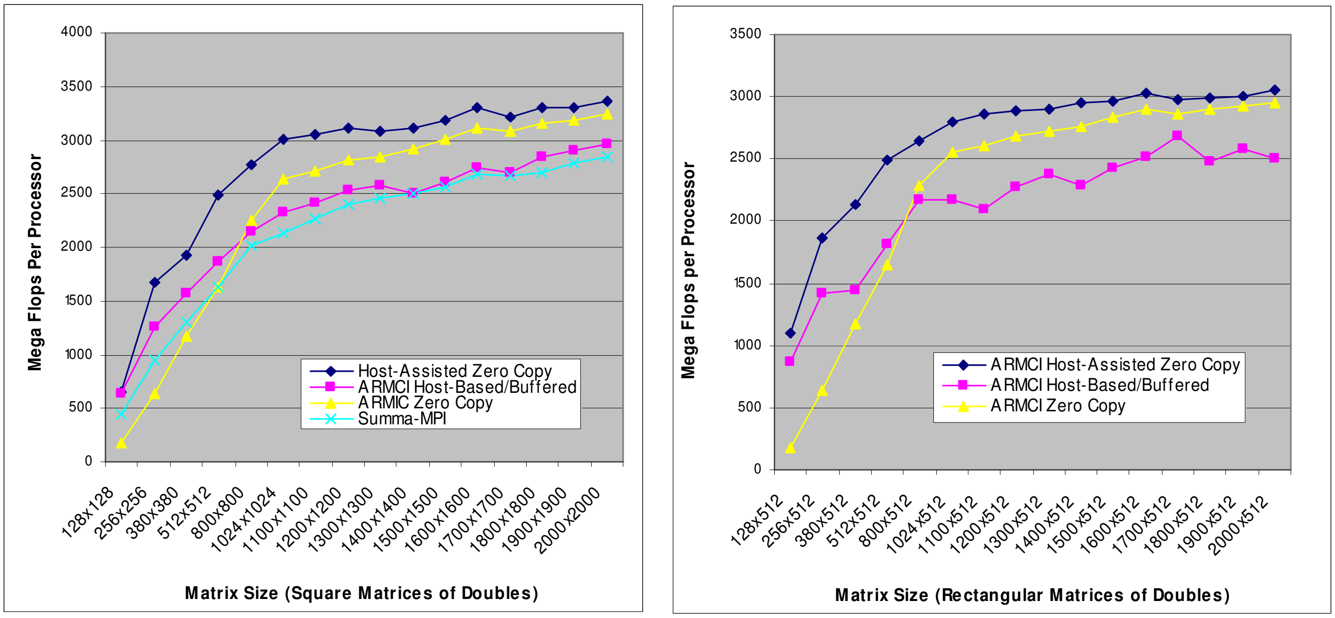

Figure-12 : Performance of Matrix Multiplication. Left: Square Matrices, and Right: Rectangular Matrices

8. References

[1] Christopher Lazou, “NEC SX-6 - two times more cost efficient than IBM P4”, HPCWire, 09.26.03.

[2] Parry Husbands, Costin Iancu and Katherine Yelick, “A performance analysis of the Berkeley UPC compiler” *Proc. 17th annual international conference on Supercomputing,* 2003.

[3] C. Coarfa, Y. Dotsenko, J. Eckhardt, J. Mellor-Crummey, “Co-Array Fortran Performance and Potential: An NPB Experimental Study”, 16th International Workshop on Languages and Compilers for Parallel Computing.2003.

[4] J. Nieplocha and B. Carpenter, “ARMCI: A Portable Remote Memory Copy Library for Distributed Array Libraries and Compiler Run-time Systems”, *Proc. RTSPP IPPS/SDP*, 1999.

[5] R. Bariuso, Allan Knies, SHMEM's User's Guide,; Cray Research,, SN-2516, 1994.

[6] Dan Bonachea and Jason Duell, *“Problems with using MPI 1.1 and 2.0 as compilation targets for parallel language implementations”*, 2nd Workshop on Hardware/Software Support for High Performance Scientific and Engineering Computing, SHPSEC-PACT03.J.

[7] J. Nieplocha, RJ Harrison, and RJ Littlefield, *“Global Arrays: A portable `shared-memory' programming model for distributed memory computers”*, Proc. Supercomputing'94, 1994.

[8] R. Numrich, J.K. Reid, *“Co-Array Fortran for parallel programming”*, ACM Fortran Forum, 17(2):1-31, 1998.

[9] K. Parzyszek, J. Nieplocha, and R.A. Kendall, “A generalized portable SHMEM library for high performance computing”, *Proc. Parallel and Distributed Computing and Systems PDCS*, 2000.

[10] G.H. Shah, J. Nieplocha, J. Mirza, C. Kim, R. K. Govindaraju, K. J. Gildea, R. Harrison, C. A. Bender, *“LAPI: A Low Level Communication Interface on the IBM RS/6000 SP: Experience and Performance Evaluation”*, Proc. IPPS’98. 1998.J. Nieplocha, V. Tipparaju, J. Ju, and E. Apra, “One-sided communication on Myrinet”, *Cluster Computing*, 6, 115-124, 2003.

[11] W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir. *“MPI –The Complete Reference, volume 2, The MPI Extensions”*, MIT Press, 1998.

[12] Jesper Larsson Traff, Hubert Ritzdorf, and Rolf Hempel. *“The implementation of MPI2 one-sided communication for the NEC SX-5”,* In Proceedings of Supercomputing 2000.

[13] http://nowlab.cis.ohio-state.edu/projects/mpi-iba/mpi_bandwidth.c

[14] Nieplocha, V. Tipparaju, A. Saify, and D. Panda, “Protocols and Strategies for Optimizing Remote Memory Operations on Clusters”, *Proc. Communication Architecture for Clusters Workshop of IPDPS*, 2002.

[15] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, *The NAS parallel benchmarks*, Tech. Rep. RNR-94-007, NASA Ames Research Center, March 1994.

[16] R. Van de Geijn and J. Watts, *SUMMA: Scalable Universal Matrix Multiplication Algorithm.* Concurrency: Practice and Experience, 9: 255-74, 1997.

[17] J. B. White and S. W. Bova, “Where’s the overlap? Overlapping communication and computation in several popular mpi implementations”, *Proceedings of the Third MPI Developers’ and Users’ Conference*, March 1999.

[18] C. Bell, D. Bonachea, Y. Cote, J. Duell, P. Hargrove, P. Husbands, C. Iancu, M. Welcome, and K. Yelick, “An evaluation of current high-performance networks”, in *Proc. 17th IPDPS*, 2003.

[19] C. Bell and D. Bonachea, “A New DMA Registration Strategy for Pinning-Based High Performance Networks”, *Proc. CAC’03*, 2003.

[20] Center for Programming Models for Scalable Parallel Computing, www.pmodels.org.

[21] Myricom, The GM Message Passing System, 10/16/1999.

[22] V.Tipparaju, M. Krishnan, J. Nieplocha, G. Santhanaraman D. K. Panda, *“Exploiting Nonblocking Remote Memory Access Communication in Scientific Benchmarks on Clusters”*, in Proc. HiPC’03.