

# Efficient One-Copy MPI Shared Memory Communication in Virtual Machines

Wei Huang, Matthew J. Koop, Dhableswar K. Panda

*Network-Based Computing Laboratory*  
*Department of Computer Science and Engineering*  
*The Ohio State University, Columbus, OH 43210*  
huanwei@cse.ohio-state.edu  
koop@cse.ohio-state.edu  
panda@cse.ohio-state.edu

**Abstract**—Efficient intra-node shared memory communication is important for High Performance Computing (HPC), especially with the emergence of multi-core architectures. As clusters continue to grow in size and complexity, the use of Virtual Machine (VM) technologies has been suggested to ease the increasing number of management issues. As demonstrated by earlier research, shared memory communication must be optimized for VMs to attain the native-level performance required by HPC centers.

In this paper, we enhance intra-node shared memory communication for VM environments. We propose a one-copy approach. Instead of following the traditional approach used in most MPI implementations, copying data in and out of a pre-allocated shared memory region, our approach dynamically maps user buffers between VMs, allowing data to be directly copied to its destination. We also propose a grant/mapping cache to reduce expensive buffer mapping cost in VM environment. We integrate this approach into MVAPICH2, our implementation of MPI-2 library. For intra-node communication, we are able to reduce the large message latency in VM-based environments by up to 35%, and increase bandwidth by up to 38% even as compared with unmodified MVAPICH2 running in a native environment. Evaluation with the NAS Parallel Benchmarks suite shows up to 15% improvement.

## I. INTRODUCTION

Modern Virtual Machine (VM) technology has been considered a solution to many management difficulties appearing on today's computing systems of increasing size and complexity [8], [16]. However, in many performance-critical scenarios, such as High Performance Computing (HPC), adoption of VM technology depends heavily on having very minimal performance overhead to achieve these management flexibilities. Benefits of VM-based HPC include on-line maintenance, fault tolerance, performance isolation, OS customization, etc. On the other hand, while recent advances such as Xen [7] and VMM-bypass I/O [15] have proved the feasibility of VM-based environment for HPC, various issues remain to further reduce the overhead of virtualization.

Shared memory communication is an area in need of additional enhancement for VM environment. In cluster environments, shared memory communication allows computing processes within the same computing node to communicate through shared memory pages that are mapped into the address space of both processes. To send a message, the sender process

copies the data into the shared region and the receiver copies the data out of the shared region to the user buffer. Especially with multi-core systems dominating the market, it is an efficient alternative to network loopback for intra-node communication. As a result, most publicly-available implementations of MPI, the *de facto* standard for parallel programming, support intra-node shared memory communication.

In a VM-based environment, shared memory communication is significantly more complicated, as the computing processes may be hosted in separate VMs (OSes) and page mapping across OSes is not natively supported. Our previous work [10] addressed this limitation by an Inter-VM Communication library (IVC), which maps memory pages between computing processes in separate VMs through the Xen grant table device [23]. Having successfully achieved close-to-native performance, IVC demonstrated the potential of a VM-based HPC solution with both competitive performance and management advantages. However, these solutions are still not optimal. For example, the size of the shared memory region is limited. Thus, it may take multiple copies to send large messages, wasting CPU cycles to coordinate between sender and receiver. It should be noted, however, that this problem exists not only for IVC, but for the aforementioned MPI shared memory communication in native environments as well. Thus, the challenge is how to remove this bottleneck and increase performance.

In this paper, we take further advantage of the Xen grant table device and propose an efficient one-copy protocol for shared memory communication between VMs. We take a similar approach as kernel-assisted one-copy communication in native environment [13]. We dynamically establish shared mappings of user buffers to allow receiver process to directly copy data from sender buffers. There are additional complexities, however. For example, mapping pages between VMs is a costly operation. Furthermore, the number of pages that can be mapped between VMs are limited based on the Xen implementation. Thus, we also propose an Inter-VM Grant/Mapping Cache (IGMC) framework to reduce the mapping cost while using limited resources. By integrating our design in MVAPICH2 [17], our implementation of MPI-2 library, we are able to achieve up to 15% improvement in

several NAS Parallel Benchmarks [19] when compared with original MVAPICH2 in a native environment over eight-core Clovertown [11] systems. Benefits on system management together with the further enhanced performance through our design make VM-based solutions for HPC become not only feasible, but attractive. Besides for HPC, the paradigm proposed in this paper is applicable to other applications domains as well, including VM-based data-centers, etc.

The rest of the paper is organized as follows: we start with a brief introduction of shared memory communication in both native and VM environments in Section II. Then, we present the design of our one-copy protocol in Sections III and IV, and carry out performance evaluation in Section V. Lastly, we discuss related work in Section VI and conclude the paper in Section VII.

## II. SHARED MEMORY INTRA-NODE COMMUNICATION

In this section, we look at intra-node shared memory communication in both native and virtualized computing environment.

### A. Intra-node Communication in Native Environment

Modern HPC clusters are typically built from computing nodes with multi-core architecture. Thus, parallel applications running on clusters will exchange data among computing processes through both inter- and intra-node communication. MPI is a commonly used standard for parallel programming. Computing processes of a MPI application are identified by a unique rank and typically run on a dedicated core. With the number of cores per computing node increasing, there is a larger chance that processes with adjacent ranks will be hosted within the same computing node. Previous study [4] reveals that many parallel applications tend to communicate more often among adjacently ranked processes, emphasizing the importance of intra-node communication.

Although intra-node communication can be trivially achieved through network loopback, most current MPI implementations support intra-node communication through userspace memory copy mechanisms. These copy-based mechanisms are generally considered to be more efficient. Here the key idea is to setup a shared memory region that can be accessed by both communicating processes. These shared regions can be created through system calls. After a shared memory region is created, the sender process can copy data into the shared region, while the receiver polls to detect data arrival and copies the data to corresponding receive buffers. Different MPI implementations, such as MVAPICH [17], Open MPI [21], and MPICH2-Nemesis [1], may have additional optimizations, but they all follow the same paradigm.

### B. Intra-node Communication in Virtual Machines

In a VM-based environment, computing processes within the same computing node can be either hosted in one VM, or separate VMs. If processes are hosted in the same VM, the intra-node communication can be realized with no difference

as the aforementioned shared memory scheme. Hosting processes in separate VMs, however, can be more beneficial for several reasons. First, the management activities in VM-based environment, including VM migration, are at the granularity of each single VM. Hosting individual computing process in different VMs allows the management flexibility at a finer grain. Further, OS customization and simplification is one of the main advantage of VM-based computing [16]. In extreme cases, the OSes can be so simplified that they do not necessarily understand multiple processors/cores. Hosting one computing process per VM is then the only choice in this case.

Once the computing processes are in separate VMs, creating a shared memory region becomes a non-trivial task. Figure 1 shows the architecture of a physical machine hosting Xen, which is a popular open source VMM distribution [7]. Above the Xen hypervisor (the VMM) are the Xen domains (VMs) running *guest OS* instances<sup>1</sup>, which host computing processes. Since the computing processes are now in separate guest OSes, shared memory regions can no longer be easily created through system calls. Using this VM-based environment, however, allows guest OSes to be migrated to other physical machines, bringing additional system management flexibility. Note that an additional special domain 0 (dom0) is typically not used to host computing processes. Dom0 runs software to create/destroy/migrate guest VMs, and also handles the device access requests from the guest VMs.

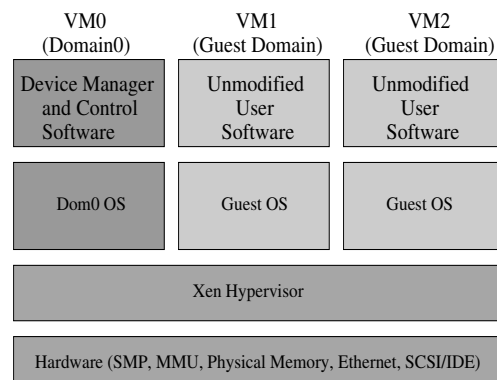


Fig. 1. Structure of the Xen Virtual Machine Monitor (VMM)

In our previous work [10], we proposed Inter-VM Communication (IVC), which creates the shared memory regions through the Xen grant table device [23]. To allow sharing, a guest VM first generates a reference handle through the grant table device for each memory page that it needs to share with another VM. The reference handles are passed to the remote domain, which uses the handles to map the pages into its local address space through the help of the Xen VMM. Based on this mechanism, several memory pages can be mapped from one guest VM to another to create a shared memory region. With this shared memory region, the intra-node communication can happen in a similar manner as in

<sup>1</sup>In this paper, domain, VM, and guest OS are interchangeable.

native computing environment described above.

The drawback of the userspace memory copy approach (in both native environments and IVC), however, is that it requires two copies to send data – one copy by the sender and one by the receiver process, which requires careful coordination. The shared memory regions are also created with limited size. Thus, when sending large messages, it is very likely that shared memory region will not be large enough to fit all the data. The sender and the receiver will then have to coordinate to finish the send with multiple rounds of copies. Such coordination lies in the communication critical path, which can affect communication progress and waste CPU cycles. We also refer to the user space memory copy approach as the *two-copy* approach in this paper.

### III. ONE-COPY PROTOCOL FOR INTER-VM SHARED MEMORY COMMUNICATION

In this section we present a new protocol for efficient inter-VM shared memory communication. The key idea of our design is to dynamically map the user buffer of the sender to the address space of the receiver. With this mapping in place the data can be directly copied into the user buffer of the receiver, saving one copy from the traditional userspace memory copy (two-copy) approach described in the last section.

Figure 2 shows an overview of the proposed protocol. It illustrates the process to send an MPI message between two computing processes using the two-copy approach proposed in [10] as well as our new one-copy approach. These two computing processes are hosted in separate VMs on the same physical host. The two-copy approach, as illustrated in Figure 2(a), exchanges the message through a shared memory region. This shared memory region is created with the help of the IVC kernel driver and the Xen VMM. The sender copies the data to the shared region (step 1 and 2), the receiver detects the data through polling, and copies the data to the receiver user buffer (step 3). After the copy completes, the receiver acknowledges the message (step 4) so that the sender can reuse the shared memory region.

To reduce the number of copies, we directly map the sender user buffer to the address space of the receiver. Detailed steps are labeled in Figure 2(b) and referred to next. Once the MPI library gets a send request, it grants access to the memory pages containing the sender user buffer through the IVC kernel driver (step 1). Unlike the previous method, instead of copying the data to the shared memory region, it only copies the aforementioned reference handles (step 2). The MPI library at the receiver side discovers the send requests (step 3) and maps the user buffer of the sender to its own address space using the grant references from the sender (step 4 and 5). With the mapping in place, the receiver directly copies the data to the destination user buffer (step 6). Once the copy completes, the receiver unmaps the sender pages and notifies the sender so that the sender can revoke the grant of the user buffers (step 7).

Granting page access to remote domains at the sender side and mapping user buffers at the receiver side are both privi-

leged operations that require the involvement of guest kernels as well as the VMM. Thus, they are computationally costly operations. Given this startup overhead, we only use the one-copy protocol to send large messages, with small messages still transferred using the traditional two-copy protocol described in Figure 2(a).

To further hide the sender-side cost from the receiver we used a pipelined send protocol. Instead of granting access to the entire sender buffer and then copying the reference handles to the shared memory region, the sender buffer is divided into multiple chunks of memory pages. Each time access is granted, one chunk of memory pages and the grant references are immediately sent to the receiver. The receiver then can discover the message and begin the transfer sooner. In this way, the cost of granting page accesses can be totally hidden except for the first chunk. Note that grants do not need to be revoked immediately after the receiver finishes copying one chunk, but can be delayed until the whole message is finished. This eliminates the need to coordinate between the sender and the receiver while sending the message. Another reason to use a pipeline mechanism is that the number of pages that can be mapped from another domain is limited. Thus, if an application needs to send an exceptionally large message, a pipelined send is the only option. It is to be noted that such case happens rarely since the limit is at least tens of Mega-bytes.

The pipeline size (number of pages per chunk) must be carefully chosen. It is inefficient to use too small of a chunk because there are high startup overheads to grant page access at the sender side and map pages at the receiver side. Larger chunk size allows us to amortize these costs. Figure 3 shows the cost to grant/map chunks of various numbers of pages and the per page cost. We observe that mapping the pages at the receiver side is the most costly operation and a 16-page pipeline size allows us to achieve a reasonable efficiency (grant/map cost per page) to map the pages. This value can be adjusted for different platforms as needed.

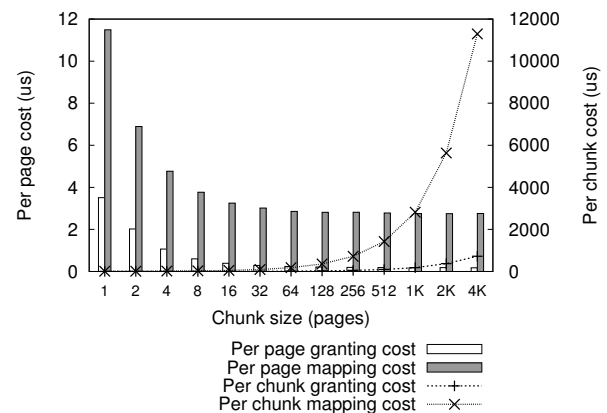


Fig. 3. Cost of granting/mapping pages

Once the pipeline size is determined, we also force the start address of each chunk to be aligned with the pipeline size (64KB aligned in our case) except for the first chunk.

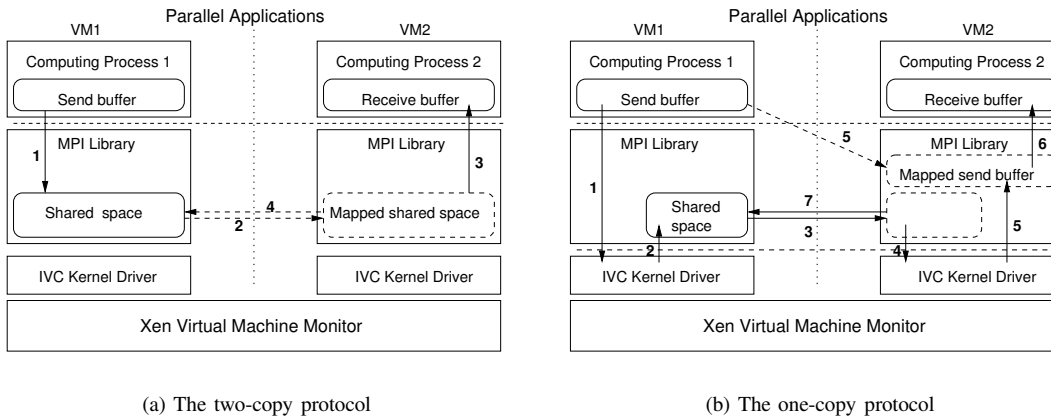


Fig. 2. Protocols to send MPI messages (details explained in Section III)

As we will see in the next section, this greatly improves the grant/mapping cache hit ratio.

#### IV. IGMC: INTER-VM GRANT/MAPPING CACHE

A pipelined send protocol can only hide the cost of granting page access at the sender. At the receiver side, however, mapping pages and copying data are both blocking operations and cannot be overlapped. Unfortunately, as we have observed in Figure 3, the receiver mapping is a more costly operation due to heavier VMM involvement than the sender page granting.

To address the high mapping cost, we propose an Inter-VM Grant/Mapping Cache (IGMC). The objective is to keep page chunks mapped at the receiver as long as possible. Then, if the sender user buffer is re-used later, it does not need to be mapped again and the data can directly be copied. In the area of high performance interconnects, similar ideas have been proposed to reduce the registration cost for zero-copy network communication [9]. Despite the similarity, IGMC is much more complicated due to two reasons:

- To keep a page mapping at the receiver, the sender must also keep the corresponding page granted as well. Thus, IGMC consists of two separate caches: a *grant cache* at sender, which caches all the page chunks that are granted, and a *mapping cache* at the receiver, which caches the mapping of those chunks.
- There are upper limits on both the number of pages that can be granted at the sender as well as the number of pages that can be mapped at the receiver. Thus, cache eviction may occur at either the sender or receiver side.

The caching logic complexity at both sides is hidden within IGMC, simplifying its use. After the IGMC receives a request to grant a chunk of pages at the sender side, the grant cache is searched for entries that contain the requested page chunk as identified by the start buffer address and the length of the chunk. Since a pipelined send protocol is used, most entries will be of the pipeline chunk size, and the start address of those chunks will be aligned. This allows us to achieve higher hit ratios without complicated mechanisms to expand/merge cache

entries. An additional complexity is that the buffer addresses used to index the caches are user space virtual addresses, which can be backed up by different physical pages in different references, causing false hit. We can use two approaches to address this issue. First, the memory pages are pinned within the OS by the sender until the grants are revoked, so that they will not be swapped out while the receiver is copying the data. Second, we can use `mallopt2` to ensure that virtual addresses are not reused when the computing process calls `free` and `malloc` operations. Thus, we can ensure that the same virtual addresses always point to the same data during application life time.

Each chunk of pages granted at the sender is assigned a unique key. At step 3 in Figure 2(b), this key is sent along with the reference handles to the receiver. The mapping cache at the receiver uses this key to determine if the corresponding chunk is already mapped or not.

Using the described protocol the grant cache and the mapping cache will work perfectly given unlimited resources to grant/map pages. Unfortunately, only a limited number of pages can be granted or mapped at one time in Xen. Thus, before creating any new entries in the grant or mapping caches, unused entries may need to be evicted to free resources if the limit is being approached. While identifying unused entries can be detected through reference counters, evicting cache entries requires coordination from both the grant and mapping caches.

##### A. Evicting Cache Entries

We limit the number of entries in both the grant cache and the mapping cache<sup>3</sup>. Since each entry corresponds to a chunk of up to the pipeline size, we also have a limit on the maximum number of pages that can be granted/mapped at a single time. IGMC behaves more elegantly in this way by avoiding granting/mapping failure at the guest OS/VMM

<sup>2</sup>More advanced mechanisms including intercepting all memory management calls can be used here to avoid false hits. We will not discuss these schemes since they are beyond the scope of this paper.

<sup>3</sup>This limit is set to be 8,192 pages in our implementation.

level. Limiting grant cache entries may not be needed in Xen, because granting access to a page only consumes few bytes of bookkeeping information in the Xen hypervisor. Thus, this is less of a concern compared to the limit on the maximum allowed mapped pages. However, we do not want to limit ourselves to such Xen-specific features; we instead design our protocol be applicable to other VMM implementations as well. An additional reason to limit the number of entries in the grant cache (at the sender) is that if we run out of resources at the receiver and fail to map the pages, the receiver will have to notify the sender to fallback to the two-copy approach. It is much more efficient for the sender to detect imminent failure so the two-copy approach can be directly selected.

Before evicting grant cache entries and revoking access to the corresponding memory page chunks, those page chunks must not be mapped at the receiver side. If the chunk is still mapped, the sender has to notify the receiver to unmap it before revoking the grant. Thus, at the sender side, we must be able to track which chunks are still in the receiver mapping cache, as well as notify the receiver to unmap a specific chunk. To achieve this, we allocate two bitmaps in the shared memory region, as shown in Figure 4. Each of them contains the same number of bits as the maximum number of allowed entries in the grant cache. The sender notifies the receiver to unmap a chunk by setting the corresponding bit in the control bitmap. The receiver will poll on the control bitmap in every library call. Once the receiver discovers the  $i$ th bit in the control bitmap is set, it unmaps the entry with the key  $i$ . The second bitmap, the mapping bitmap, is set/cleared by the receiver mapping cache. A set bit in the mapping bitmap indicates that the receiver is still keeping the map of the corresponding chunk.

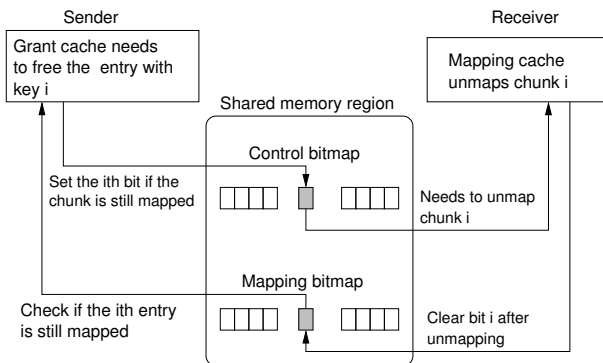


Fig. 4. Exchanging control information through bitmaps

At the receiver side, the mapping cache will evict entries when it is running out of available resources to map new page chunks, or when it is advised by the sender to do so. In either case, the corresponding bits in the mapping bitmap must be cleared to notify the sender that it no longer is maintaining the mapping of the specified chunk.

### B. Improving the Cache Hit Ratio

Since it is very costly to map/unmap pages, optimizing the cache hit ratio will be an important issue. We choose

to use LRU currently to manage both the grant cache and the mapping cache, since it is relatively simple to implement. Using simple LRU replacement, however, may result in poor hit ratio in some cases. There are numerous studies in literature to improve the cache hit ratio, such as LRU/k [20], 2Q [14], LIRS [12], etc. We plan to implement 2Q as it admits only re-visited cache entries into the main LRU queue, which can effectively reduce the impact of cache entries that are only referred once. Since the grant cache and the mapping cache can interfere with each other (i.e., before an entry can be removed from the grant cache, the corresponding entry must be removed from the mapping cache), other cache replacement algorithms which take more access history information into account could be more effective. However, the actual impact of different cache replacement algorithms is beyond the scope of this paper and is left for future study.

### C. Fall-back Mechanism

No matter how hard we try to optimize the cache replacement algorithm, there could always be few applications having poor access patterns, which causes a low cache hit ratio. This is true especially if the application working set (user buffers involved in intra-node communication) is too large to fit in the cache. In this case, instead of frequently mapping/unmapping process, which hurts the performance, it is more efficient to fall back to the original two-copy based shared memory communication. The decision on falling-back is made based on the hit ratio of the more costly mapping cache at receivers. Each process will track the overall hit ratio and the communication (receive) frequency to each sending peer. If the overall hit ratio is under a certain threshold for a long period of time (i.e., several hundred messages), the process will decide a group of sending peers that communicate the least often and start to use two-copy protocol with these senders. To notify the senders, the process will raise a flag at the shared memory region so that the corresponding sender will send messages using traditional two-copy protocol. We treat each peer differently since the communication pattern could be different among each peers. It also allows the flexibility to fall-back on some of the peers to reduce the application working set. In this case, there is a good chance that the user buffers used in communication to the rest of the peers can fit in the IGMC cache. It is to be noted that the fall-back cases will not happen often especially for larger scale applications, where only part of the communication is intra-node.

The two-sided fall-back mechanism will also be used for handling non-contiguous datatype communication. This is because “holes” in data buffers may cause inefficiency to grant/map whole all buffer pages. Additionally, the complexity of passing data layout information to the receiver may also overshadow the benefits of one-copy protocol.

## V. EVALUATION

In this section we evaluate our one-copy approach. We integrate the one-copy protocol into MVAPICH2 [17], and compare the design with our earlier work, a two-copy based

inter-VM shared memory communication protocol. We also compare with unmodified MVAPICH2 running in a native Linux environment. We first demonstrate the improvements on latency and bandwidth benchmarks. Then we present evaluation results using the NAS Parallel Benchmark Suite [19].

### A. Experimental Setup

The experiments are carried out on a Intel Clovertown system. The computing node is equipped with dual-socket quad-core Xeon E5345 2.33GHz and 6 GB of memory. We run Xen-3.1 with the 2.6.18 kernel on the computing node for VM-based environments. We launch up to eight VMs on the node. Each of VMs will be assigned a dedicated core when running CPU intensive jobs. Each VM consumes 512MB of memory. Even though the modified MPI is certainly capable of running multi-node applications, our experiment is within one node since the focus of this paper is intra-node communication. As demonstrated by our earlier work [5], [10], benefits shown by single-node study can be effectively propagated to multi-node tasks due to the importance of intra-node communication. For native environment evaluation, we run RHEL 5 with the 2.6.18 kernel. We evaluate three configurations in this section:

- **IVC-one-copy** - Our new one-copy based scheme in a VM-based environment;
- **IVC-two-copy** - Previous inter-VM shared memory communication in VM environment, as proposed in [10], using a two-copy method;
- **Native** - Unmodified MVAPICH2 running in native environment, using a two-copy method.

### B. Micro-benchmarks Evaluation

Figures 5 and 6 are the MPI level latency and bandwidth. In our current design, the new one-copy scheme is only used for messages larger than 64KB. Any smaller messages will be sent using the old two-copy schemes (IVC-two-copy), so we focus on messages larger than 64KB. Our benchmarks repeatedly send and receive using the same user buffer. Thus, our one copy scheme can greatly benefit from the grant/mapping cache. Also, with the mappings cached, the one-copy scheme simply copies data between the same buffers when sending/receiving from the same position. The performance, therefore, is significantly better due to CPU cache effects as compared with two-copy schemes. Here the IVC-one-copy configuration is able to achieve up to 6.5GB/s bandwidth (in-cache copy) and reduce the latency by up to 75% compared with the native environment.

One-copy approach shows significant improvement when the same buffer is used for communication. Unfortunately, real applications will not repeatedly send data from the same location without even writing into the buffer. Thus, to remove the CPU cache effects and evaluate the performance more accurately, we modify the benchmarks to send and receive messages into a larger user buffer pool in a cyclic pattern. After each iteration of send/receive, both sender and receiver increase the user buffer address so that no two consecutive operations will use the same buffer. Once they reach the end

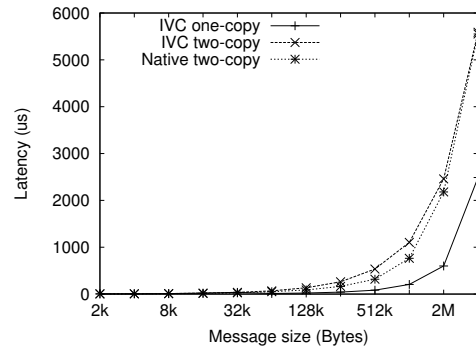


Fig. 5. MPI latency

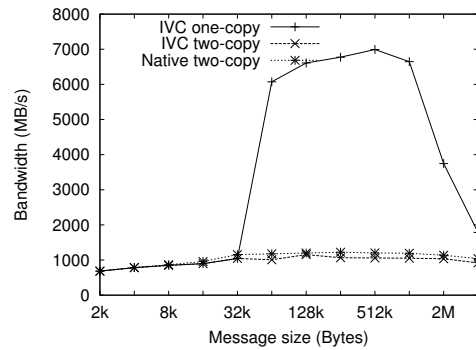


Fig. 6. MPI bandwidth

of the buffer pool, however, they will start again from the head of the pool. Figures 7 and 8 show the latency and bandwidth using a 16MB pool, which is much larger than the L2 cache on the system (4MB). Since the size of the grant/mapping cache in our design is chosen to be 8K pages, the VM-one-copy scheme will still benefit from it. In this case, the one-copy scheme is able to reduce the latency by up to 35% and increase the bandwidth by up to 38% compared with the native configuration.

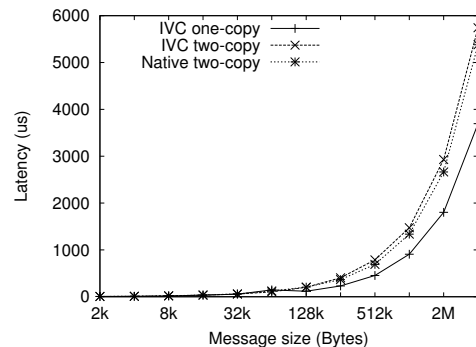


Fig. 7. MPI latency with cyclic access pattern

If we increase the size of the user buffer pool to 40MB, we will exceed the available entries in the grant/mapping cache. The grant/mapping cache will be forced to evict entries. Those entries, unfortunately, contain the grant/mapping of

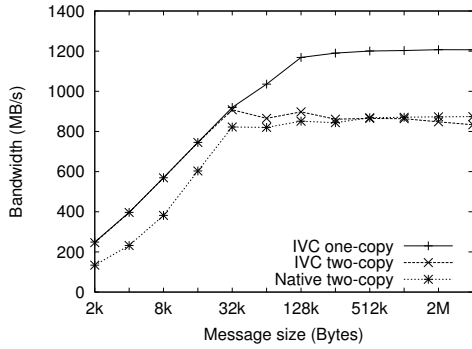


Fig. 8. MPI bandwidth with cyclic access pattern

the user pages that will be re-used later. In this case, the one-copy based scheme is penalized by the high cost of grant/mapping pages in the VM environment. As we can see in Figures 9 and 10, the one-copy scheme gets up to 39% and 31% worse on latency and bandwidth, respectively, compared with native environment. This is expected due to the mapping cost illustrated in Figure 3. This example demonstrates the importance of optimizing the cache hit ratio and dynamically falling back to two-copy scheme if too many grant/mapping cache misses are observed. If we fall-back to the two-copy based scheme, we will still have the native-level performance, delivered by the IVC-two-copy configuration.

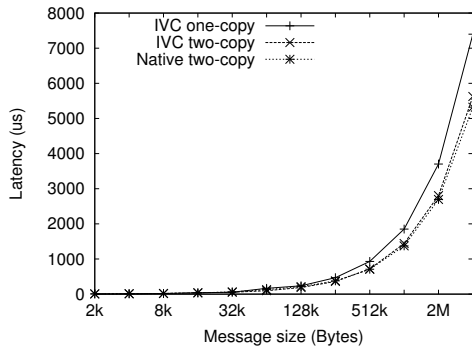


Fig. 9. MPI latency (mapping cache miss)

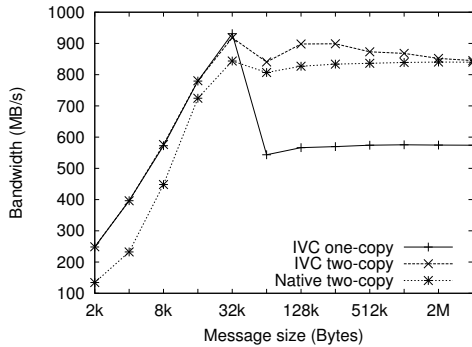


Fig. 10. MPI bandwidth (mapping cache miss)

TABLE I  
COMMUNICATION PATTERN OF NPB (CLASS A)

	> 64KB (count)	> 64KB (volume)	Comm. rate
IS	56.4%	99.9%	497.4 MB/s
CG	0.0%	0.0%	111.5 MB/s
LU	1.60%	67.2%	32.0 MB/s
MG	14.0%	78.5%	66.7 MB/s
BT	73.3%	90.7%	8.6 MB/s
SP	98.8%	100.0%	26.6 MB/s
FT	52.5%	100.0%	211.1 MB/s
EP	0.0%	0.0%	0.0 MB/s

### C. NAS Parallel Benchmarks

In this section, we evaluate our one-copy approach on NAS Parallel Benchmarks (NPB) [19]. This is a popular parallel benchmarks suite containing computing kernels typical of various fluid dynamics scientific applications.

NPB contains multiple benchmarks, each of which have different communication patterns. Table I summarizes the communication characteristics of NPB assuming it is running on eight processes (maximum for one dual-socket Quad-core system). Here we show the communication intensity in terms of the volume of messages sent per second, and the percentage of large MPI messages (64KB, the threshold above which messages will be sent using one-copy approach).

Figure 11 presents the performance comparison between the Native, VM-two-copy and VM-one-copy configurations. As the figure shows, our one-copy approach is able to meet or exceed the performance of both Native and VM-two-copy configurations for all benchmarks. In particular, for IS a 15% improvement over Native and 20% improvement over VM-two-copy is achieved. Other benchmarks, including LU and MG also show a 8%/5% improvement and 6%/7% improvement, respectively over Native and VM-two-copy.

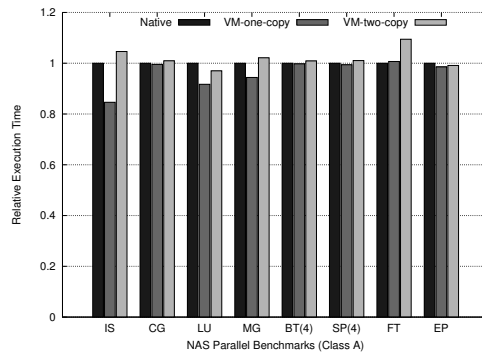


Fig. 11. Relative performance for NPB

Both message sizes as well as message patterns effect the performance using the different mechanisms. For IS, 56% of messages and 99% of message volume is for messages greater than 64KB. SP shows a similar pattern, with over 98% of messages greater than 64KB. Despite these similarities, the performance improvement for SP using our one-copy design is minimal as compared to IS. LU and MG, with significant large message transfer (67% and 79% of volume), but less

than IS or SP, show higher performance using the one-copy mechanism. This can be attributed to a pattern where latency is of increased importance or better processor cache utilization since a one-copy transfer does not pollute the sender cache as occurs with a traditional two-sided copy technique.

Though our MPI is able to run across multiple nodes, the results are shown on a single computing node since the focus of the paper is intra-node communication. On a larger scale, where part of the communication will be inter-node, applications could benefit less as shown in Figure 11. However, simulation of communication patterns on a larger cluster as shown in Figure 12 suggests that intra-node communication is still very important even when applications span across multiple nodes - some applications tend to communicate more frequently between neighbors. We show the percentage of intra-node communication for NPB on 16, 64, and 256 processes (assuming each computing node has 8 cores) and observe that intra-node communication can be up to 40% even on 256 processes.

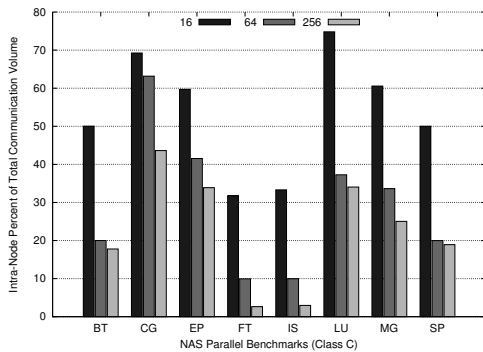


Fig. 12. Percentage of intra-node communication for larger scale run with NPB

## VI. RELATED WORK

In this paper, we discussed an efficient one-copy protocol for shared memory communication in VM environments. In literature, there are many continuing research efforts to make shared memory communication efficient in native environments. These efforts take different approaches, including user space memory copy, such as proposed by Buntinas et al. [3] and Chai et al. [5], kernel-assisted memory mapping [13], etc. In [2], Buntinas et al. has a more detailed discussion on various mechanisms for data transfer in SMP machines. Even though a kernel-assisted mapping scheme is a one-copy scheme as well, all these approaches require the computing processes be hosted in the same OS, which is not applicable to VM environments.

The grant/mapping cache discussed in this paper takes a similar approach as proposed by Tezuka et al. [9]. The work is similar to ours as it avoids costly kernel/hardware privileged operations by caching the results of previous entries. Our work is much more complex in the way that grant and mapping caches must coordinate to ensure correct caching results. This coordination, however, will be needed only when evicting IGMC entries and may not be in communication critical path.

Thus, application performance will not be affected by the additional complexity in most cases.

VMs have been a popular research topic in recent years and have been deployed in industry, such as VMware VirtualCenter [22] and Xen Enterprise [23]. Our work is specific towards a VM-based environment for HPC. VM technologies have the potential to greatly benefit HPC applications, which is thoroughly analyzed by Mergen et al. [16].

## VII. CONCLUSIONS AND FUTURE WORK

In this paper, we present a one-copy protocol for inter-VM shared memory communication. We dynamically map the sender user buffer to the address space of the receiver. As compared with traditional two-copy approaches, which are most commonly used, our approach saves the cost of copying data from the sender buffer to the shared memory space. Because grant/mapping operations in VM environments are expensive, we also propose a grant/mapping cache to reduce this cost. We incorporate our scheme into the MVAPICH2 software package. With our design, we show the large message latency in VM-based environment can be reduced by up to 35% and increase bandwidth up to 38% even as compared with unmodified MVAPICH2 running in a native environment. Evaluation with the NAS Parallel Benchmarks suite also shows up to 15% improvement. With advantages of VM technology for system management proposed by related research and industrial efforts including [16], [18], [6], [23] and the enhanced performance through our design, a VM-based environment will become an attractive solution for HPC.

In the future, we plan to further enhance our design in multiple aspects. First, we will evaluate more MPI applications and study cache replacement algorithms to optimize the hit ratio of the grant/mapping caches, as this is a core part for achieving low overhead communication. Second, we plan to explore more intelligent designs for multi-core architecture. For example, the shared memory regions can be shared across multiple VMs, instead of only between two peers, which brings opportunity to optimize collective operations. Finally, our current design is integrated with MPI. We plan to design APIs which can be used by other applications in VM environments, which will benefit a larger pool of applications.

## ACKNOWLEDGMENT

We would like to thank Dr. Jiuxing Liu and Dr. Bulent Abali from IBM T. J. Watson Research Center for the valuable discussion and suggestions at the early stage of this work.

This research is supported in part by an IBM PhD Scholarship, and by the following grants and equipment donations to the Ohio State University: Department of Energy's Grant #DE-FC02-06ER25749 and #DE-FC02-06ER25755; National Science Foundation grants #CNS-0403342 and #CCF-0702675; grant from Wright Center for Innovation #WCI04-010-OSU-0; grants from Intel, Mellanox, Sun, Cisco, and Linux Network; and equipment donations from Apple, AMD, IBM, Intel, Microway, Pathscale, Silverstorm and Sun.

## REFERENCES

- [1] Argonne National Laboratory. <http://www-unix.mcs.anl.gov/mpi/mpich/>.
- [2] D. Buntinas, G. Mercier, and W. Gropp. Data Transfers Between Processes in an SMP System: Performance Study and Application to MPI. In *International Conference on Parallel Processing*, Columbus, OH, Aug. 2006.
- [3] D. Buntinas, G. Mercier, and W. Gropp. The design and evaluation of Nemesis, a scalable lowlatency message-passing communication subsystem. In *Proceedings of CCGrid 2006*, Singapore, 2006.
- [4] L. Chai, Q. Gao, and D. Panda. Understanding the Impact of Multi-Core Architecture in Cluster Computing: A Case Study with Intel Dual-Core System. In *Proceedings of CCGrid 2007*, Rio de Janeiro, Brazil, May 2007.
- [5] L. Chai, A. Hartono, , and D. K. Panda. Designing High Performance and Scalable MPI Intra-node Communication Support for Clusters. In *The IEEE International Conference on Cluster Computing (Cluster 2006)*, Barcelona, Spain, Sept. 2006.
- [6] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *Proceedings of 2nd Symposium on Networked Systems Design and Implementation*, 2005.
- [7] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the Art of Virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, Oct. 2003.
- [8] R. Figueiredo, P. Dinda, and J. Fortes. A Case for Grid Computing on Virtual Machines. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS), May 2003.*, 2003.
- [9] H. Tezuka et al. Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication. In *Proceedings of the 12th International Parallel Processing Symposium*, 1998.
- [10] W. Huang, M. Koop, Q. Gao, and D. Panda. Virtual Machine Aware Communication Libraries for High Performance Computing. In *Proceedings of SC'07*, Reno, NV, November 2007.
- [11] Intel Corporation. Quad-Core Intel Xeon processor 5300 series product brief. <http://www.intel.com/>.
- [12] S. Jiang and X. Zhang. LIRS: An Efficient Low Inter-Reference Recency Set Replacement Policy to Improve Buffer Cache Performance. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Marina Del Rey, CA, 2002.
- [13] H.-W. Jin, S. Sur, L. Chai, and D. K. Panda. Design and Performance Evaluation of LiMIC (Linux Kernel Module for MPI Intra-node Communication) on InfiniBand Cluster. In *International Conference on Parallel Processing*, Sept. 2005.
- [14] T. Johnson and D. Shasha. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. In *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, San Francisco, CA, USA, 1994.
- [15] J. Liu, W. Huang, B. Abali, and D. K. Panda. High Performance VMM-Bypass I/O in Virtual Machines. In *Proceedings of USENIX '06*, Boston, MA, 2006.
- [16] M. F. Mergen, V. Uhlig, O. Krieger, and J. Xenidis. Virtualization for High-Performance Computing. *SIGOPS Oper. Syst. Rev.*, 40(2):8–11, 2006.
- [17] MVAPICH Project Website. <http://mvapich.cse.ohio-state.edu>.
- [18] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott. Proactive Fault Tolerance for HPC with Xen Virtualization. In *Proceedings of the 21st Annual International Conference on Supercomputing (ICS'07)*, Seattle, WA, June 2007.
- [19] NASA. NAS Parallel Benchmarks. <http://www.nas.nasa.gov/Software/NPB/>.
- [20] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K Page Replacement Algorithm For Database Disk Buffering. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1993.
- [21] OpenMPI. <http://www.open-mpi.org/>.
- [22] VMware. <http://www.vmware.com>.
- [23] XenSource. <http://www.xensource.com/>.