2017 IEEE 24th International Conference on High Performance Computing (HiPC)

Designing Registration Caching Free High-Performance MPI Library with Implicit On-Demand Paging (ODP) of InfiniBand*

Mingzhe Li, Xiaoyi Lu, Hari Subramoni, and Dhabaleswar K. (DK) Panda
Department of Computer Science and Engineering
The Ohio State University
Email: {limin, luxi, subramon, panda}@cse.ohio-state.edu

***Abstract*—Modern high-performance communication runtime systems have taken advantage of advanced features on high-performance networks (e.g. InfiniBand) to deliver optimal performance. High-performance communication over InfiniBand typically requires the communication buffers to be registered first. However, buffer registration and deregistration are costly operations, which leads to performance degradation if they happen frequently. To hide this overhead, many existing communication runtime choose to design a high-performance registration cache to reduce the number of buffer registrations, but such type of designs still need some amount of buffers to be registered and cached, which leads to multiple issues such as performance overhead, high memory consumption for bookkeeping, and code complexity for maintaining the registration cache. To solve these issues, a recently introduced feature for InfiniBand called Implicit On-Demand Paging (ODP) is getting momentum. This feature enables one process to register its complete memory address space for I/O accesses. To fully take advantage of Implicit-ODP, it is critical to fully understand the behavior and benefits of Implicit-ODP on InfiniBand and performance/memory trade-offs it presents. This paper first presents an analysis of the Implicit-ODP feature and studies its basic performance with InfiniBand verbs-level micro-benchmarks. Then, we describe the design tradeoffs with Implicit-ODP and the various optimizations at MPI runtime. We propose and design communication protocols that can leverage the Implicit-ODP feature at the MPI level. The experimental results at the micro-benchmark level and application level show that our proposed design can deliver comparable performance to the existing pin-down scheme, while it does not need registration cache in the MPI runtime. To the best of our knowledge, this is the first work to study and analyze the Implicit-ODP feature and design a registration caching free MPI library with it.**

I. Introduction

RDMA-enabled high-performance interconnects such as InfiniBand (IB) [1] and multi-/many-core systems have been the key drivers for high-performance computing systems. Most runtime systems have been designed to take advantage of the RDMA technology for scientific applications to deliver optimal performance [2], [3], [4]. More importantly, the kernel bypass feature with these high performance interconnects is able to maximize the computation and communication overlapping in applications to optimize performance.

The Message Passing Interface (MPI) is the dominant programming model for parallel scientific applications. To enable IB RDMA communication, MPI stacks typically need to handle the procedures of registering and deregistering communication buffers, which are expensive operations due to memory pinning and unpinning overhead. Existing designs in MPI runtime to reduce the pinning/unpinning overhead require communication buffers to be pinned in memory until the finalization phase. These pin-down buffers could limit the amount of memory resources for computation in scientific applications. With modern multi-/many-core platforms that are evolving very rapidly with 32/64 cores per node, the total size of pin-down buffers is expected to be higher compared to current generation system. However, per core memory resource is expected to be decreased. This makes the memory contention between computation and communication even worse. Thus, it is very important to reduce the memory usage of the runtime and size of pin-down buffers.

Many optimizations have been proposed to reduce the memory footprint of MPI stacks on such systems. Several different protocols like Extended Reliable Connection and Dynamic Connected Transport have been used by MPI stacks to reduce the memory footprint used for communication, while delivering comparable or better performance [5], [6]. However, all these designs still need to pin-down communication buffers, such as pinned eager buffers for small message transfers and pinned user buffers for large message transfers. These pin-down buffers may consume a lot of memory resources which leads to limited memory resources for computation in scientific applications.

A. Motivation

Mellanox has introduced a new feature named On-Demand Paging that alleviates much of the shortcomings of pin-down memory. With this feature, communication buffers with RDMA operations are no longer needed to be pinned in memory. These buffers can be automatically paged in when the HCA needs them and paged out when the kernel needs to swap them. This feature provides an alternative solution for RDMA communications and has a big potential to increase the amount of swappable memory for computation. ODP could be divided into two subclasses. One subclass is called Explicit-ODP which has been explored in our previous study [7]. The other subclass is called Implicit-ODP which is the focus of this work.

* This research is supported in part by National Science Foundation grants #CNS-1419123, #CNS-1513120, #ACI-1450440, and #CCF-1565414.

0-7695-6326-0/17/$31.00 ©2017 IEEE
DOI 10.1109/HiPC.2017.00017

TABLE I: Summary of Pin-down, Explicit-ODP, and Implicit-ODP

Supported Features	Pin-down	Explicit-ODP	Implicit-ODP(this work)
IB Verbs Send	√	√	√
IB Verbs Recv	√	√	√
IB Verbs RDMA Write	√	√	×
IB Verbs RDMA Read	√	√	×
Pin/Unpin Communication Buffer	√	×	×
Dynamically Page in/out	×	√	√
Pre-fetch Page	×	√	√
Register Whole Address Space	×	×	√

Table I shows the supported features with Pin-down, Explicit-ODP, and Implicit-ODP schemes. Compared to the Pin-down scheme, both ODP-based schemes could dynamically page in or page out communication buffers. This gives the operating system the flexibility to manage communication buffers. Compared to the Explicit-ODP scheme, Implicit-ODP could register the whole address space, this implies that the Host Channel Adapter (HCA) could directly send/recv any memory buffer of one process. However, current generation Implicit-ODP scheme does not support RDMA read/write yet which is widely used by MPI stacks for large message transfers [8], [9].

Both Pin-down and Explicit-ODP scheme require memory buffers for communication to be explicitly registered. Previous research [10] shows that memory registration is an expensive operation. The cost and overhead of memory registration dramatically degrade the performance of RDMA and increase network latency in the critical data path of I/O operations. To hide this overhead, many existing communication runtimes choose to incorporate a high-performance registration cache [10] in the memory manager and it needs to be carefully designed to achieve optimal performance. Most MPI stacks have been using this registration cache design on RDMA-enabled interconnects. But, such type of designs still needs some amount of buffers to be registered and cached, which leads to multiple issues such as performance overhead, high memory consumption for bookkeeping, and code complexity for maintaining the registration cache. With the Implicit-ODP feature, this registration cache design is no longer needed as no registration is required for specific communication buffers.

The Implicit-ODP feature opens up many new research opportunities. This motivates us to **design a registration caching free high-performance MPI library with Implicit-ODP**. To achieve this goal, several challenges and questions both at IB verbs-level and MPI level need to be addressed to solve this problem:

- What are the performance characteristics and behavior of Implicit-ODP at IB verbs-level?
- What is the overhead of dynamically paging in/out communication buffers?
- How can we design MPI level protocols using Implicit-ODP to efficiently deliver performance?
- Can registration cache be totally removed from MPI library without performance degradation?

B. Contribution

Implicit-ODP enables a process to register its complete address space, so any communication buffer could be directly accessed by the HCA. In order to efficiently exploit this feature, one needs to analyze and understand the behavior and performance characteristics of Implicit-ODP with both performance and memory metrics. In this paper, we tackle the above challenges and perform an extensive study and analysis of the behavior of Implicit-ODP at IB verbs-level, then using the extracted knowledge we propose and study alternative approaches to efficiently design MPI level support for the Implicit-ODP feature. To summarize, this paper makes the following contributions:

- Present the Implicit-ODP feature and analyze its basic performance characteristics with IB verbs-level micro-benchmarks
- Identify performance bottlenecks when naively integrating Implicit-ODP with MPI stacks
- Propose alternative schemes (Implicit-ODP based eager protocol, zero-copy rendezvous protocol) and designs to overcome the identified bottlenecks related with Implicit-ODP at MPI level
- Propose high-performance communication protocols without registration cache
- Demonstrate the benefits and impact of our designs on applications performance

Experimental evaluations show that our proposed designs are able to achieve comparable performance as current pin-down scheme. To the best of our knowledge, this is the first research work that studies and proposes registration caching free high-performance MPI communication designs with Implicit-ODP on InfiniBand clusters.

The rest of the paper is organized as follows. Section II presents the necessary background. Section III presents IB verbs-level performance evaluations with the Implicit-ODP feature. Section IV shows details of our proposed designs. Section V describes our detailed performance evaluation. Related work is presented in Section VI. Finally, we conclude in Section VII with some description of possible future work.

II. Background

In this section, we briefly discuss existing pin-down scheme used for communications over RDMA channels. After that, we give an overview of the ODP feature.

A. Pin-down Scheme

Communication buffers with RDMA operations need to be registered with an HCA using system calls. The Pin-down scheme has been used for IB hardware atomics and RDMA operations with contiguous or noncontiguous buffers [7], [11], [12], [13]. After that, applications could post I/O operations directly to the HCA that translates the I/O virtual address to physical address. With current technology, this is achieved by pinning the communication buffers during the registration calls. Pin-down scheme basically ensures that these memory regions are locked and can not be swapped by the OS kernel.

Figure 1 gives an overview of the pin-down scheme. From the figure, we can see that the physical memory is broadly divided into two parts. One part is only used for communication that has been pinned in memory, which can not be swapped by the OS kernel. The other part is not pinned into memory and could be swapped by the OS kernel. If scientific applications use different buffers for communication and computation phases, given a fixed size of physical memory, the more communication buffers are pinned in memory, the fewer memory resources could be used for computation.

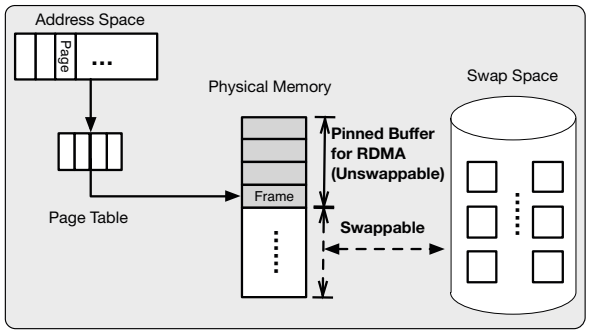


Fig. 1: Pin-down Scheme

B. On-Demand Paging

ODP is a new feature introduced by Mellanox to register communication buffers for RDMA operations. ODP allows the communication buffers automatically be paged in when the HCA needs them and paged out when the OS kernel needs to swap them. With this feature, no communication buffers are required to be pinned in memory, so it gives users the flexibility to swap these buffers. Figure 2 gives an overview of the ODP scheme. We can see that the OS kernel could swap all communication buffers. Compared with the pin-down scheme that requires all pinned buffers fitting into physical memory, ODP allows the registered memory regions to be larger than the physical memory size.

ODP can be further divided into 2 sub-classes: Explicit and Implicit ODP. Explicit ODP means that applications explicitly specify the memory regions used with ODP. Implicit ODP means that applications specify its complete address space used with ODP. In this paper, we focus on Implicit-ODP.

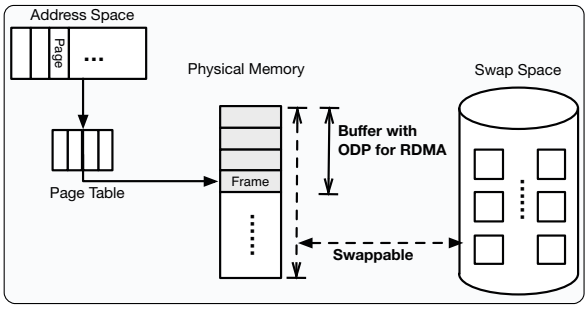


Fig. 2: On-Demand Paging (ODP) Scheme

C. Page Fault and Invalidation with ODP

In this subsection, we describe how page fault and page invalidation are handled by ODP shown in Figure 3. Both CPU and HCA are involved in handling page fault and invalidation events.

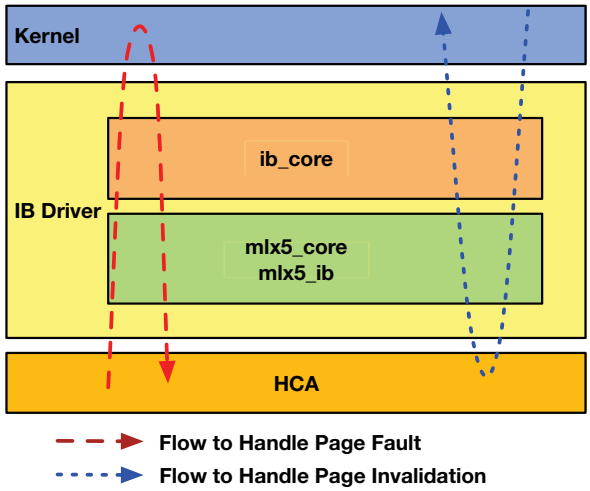


Fig. 3: Page Fault and Invalidation Flow

On IB networks, users post I/O virtual address to the HCA, which handles the translation from virtual address to physical address, for RDMA operations. This implies that when the HCA performs RDMA operations, it needs mapping information from virtual addresses to physical addresses. However, when a new memory region is created, it is possible that this memory region has not been loaded into memory nor the mapping exists in the HCA. In this case, when the HCA attempts to access this memory region, a page fault event will be generated by the HCA, which will be handled by the IB driver and OS kernel. When a page fault happens at the HCA, it will generate a page fault event to the IB driver. The IB driver needs to handle these page fault events in three steps. In the first step, it searches for the registered memory regions associated with the I/O virtual address which the HCA attempted to access. In the second step, it finds the pages for this registered memory region. In the third step, it brings these pages into memory and maps these pages to the HCA. After that, the driver notifies the HCA with the new mapping information. When the mapping is updated in the HCA, suspended communications start to proceed.

Page Invalidation happens when the OS kernel needs to release some pages that have been registered by the HCA. When this happens, an invalidation notification event is generated by the OS kernel and sent to the IB driver. After the IB driver receives this notification, it will look for intersecting pages and notify the HCA about the invalidation request. The HCA flushes the hardware caches to invalidate these pages. Only after flushing the HCA's page table caches, the OS kernel is allowed to swap these pages. With page invalidation events, the OS kernel could swap registered memory regions similar as other memories in process's address space.

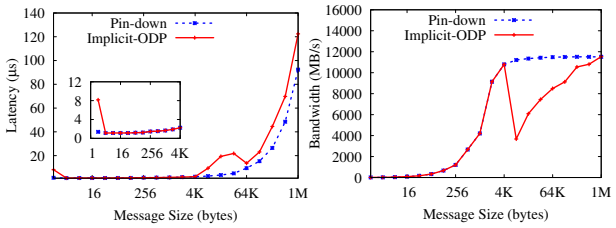
D. Page Pre-fetching

Page pre-fetching is designed to warm up new mappings by bringing memory regions into memory and updating mapping from virtual address to physical address in the HCA. Prefetch operation is an asynchronous verbs-level call and it is a best effort hint. Pages pre-fetched by this verbs-level call is not guaranteed to remain resident in the memory. They could be swapped out when the kernel wants to reclaim these pages.

III. Understanding Implicit-ODP Performance Characteristics at IB verbs-level

In this section, we use IB verbs-level point-to-point latency and bandwidth benchmarks to show and analyze the performance one can get with the Implicit-ODP feature.

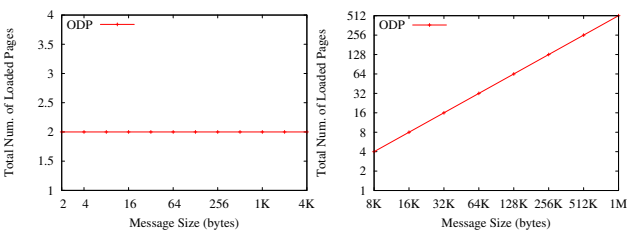
A. Performance Evaluation with IB verbs-level Point-to-Point Benchmarks



(a) Point-to-Point Latency (b) Point-to-Point Bandwidth

Fig. 4: IB Verbs-level Latency and Bandwidth with Implicit-ODP

We evaluate verbs-level uni-directional latency and bandwidth performance with the Pin-down and Implicit-ODP schemes. We use two verbs-level benchmarks to do the evaluations. In both benchmarks, the memory buffer for communication is allocated at the beginning of the program and freed at the end of each run. For each message size, we run the ping-pong tests in a loop of 1,000 iterations without any warm-up iterations. Figure 4(a) shows the verbs-level ping-pong latency results with the Pin-down scheme and the Implicit-ODP scheme. From the results, we could see that the Implicit-ODP scheme has a big degradation at 2 byte message sizes as well as message sizes larger than 8K bytes. The bandwidth benchmark results in Figure 4(b) shows the same trend.



(a) Small Messages (b) Large Messages

Fig. 5: Number of Pages Updated by Page Fault Events

On our systems, the page size is 4K bytes that means for message sizes from 2 - 4K bytes, all the communications only use one page on each side. When the HCA tries to access the communication buffer in the very first iteration, it leads a page fault which causes the performance degradation. To validate our explanation for performance degradations, we gathered some statistics from the HCA regarding the total number of pages loaded by page fault events for each message size shown in Figure 5. To get these numbers, we modify the latency benchmark to allocate separate buffers for each message size. We count the number of page faults for each message size. Figures 5(a) and 5(b) show the number of page faults happening with message sizes from 2 bytes to 1M bytes. In our latency benchmark that has 1,000 iterations for each message size, only two page faults are happening. One page fault at the sender side and the other one at the receiver side. These page faults only happen at the first iteration to warm up buffers and update mapping in the HCA. So the performance degradation at 2 bytes message sizes comes from a page fault event. For message sizes larger than 4K bytes, more pages are needed for each memory buffer which leads to increased number of page faults.

B. Page Fault Overhead

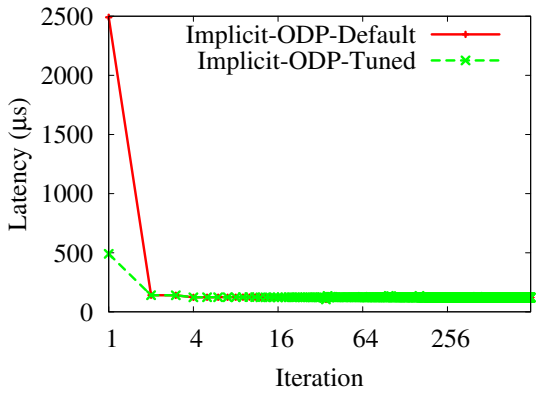


Fig. 6: Per Iteration Latency with 1M bytes Message Size

To measure the time of handling a page fault, we run the latency benchmark with 1M bytes message sizes and reports the latency in each iteration. Figure 6 shows the experimental results. We see that the latency does not change after the first iteration. However, the time spent in the first iteration is significantly higher than that in other iterations. We see that the latency in first iteration and second iteration is 2,500us and 150us, respectively. It means that a page fault with 1M bytes message size take more than 2,000 us to handle.

Based on [14], [15], when page fault occurs on the receive, the receiver sends an RNR NACK packet to the sender and sender re-issues the operation after RNR NACK timeout. Receiver does not notify sender explicitly that page was swapped in, so if RNR timeout is too long on the sender, the reported latency is longer than what is needed to swap the page in. We tried different RNR NACK timeout values, with smaller RNR NACK timeout value, we see the time to handle page fault is significantly reduced which is shown as Implicit-ODP-Tuned in Figure 6.

IV. Proposed Designs

In this section, we first give an overview of MVAPICH2 [16] for MPI point-to-point primitives with Pin-down and Explicit-ODP schemes on IB clusters. With that knowledge, we propose three designs to take advantage of Implicit-ODP in MVAPICH2. We choose MVAPICH2 for our implementation. But the designs are generic and applicable to other MPI libraries as well

A. Overview of MVAPICH2 on IB Clusters

Figure 7 gives an overview of MVAPICH2 on IB clusters. The boxes with dotted lines are proposed designs in this paper.

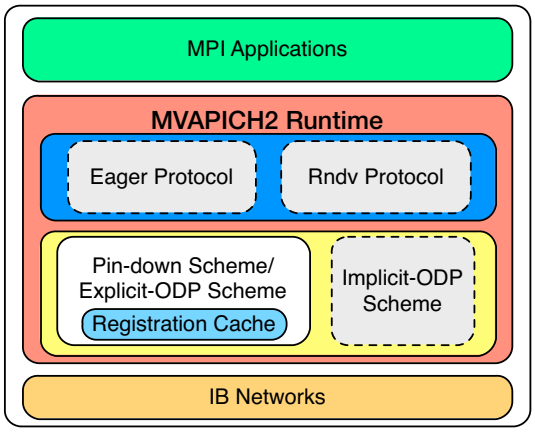


Fig. 7: Overview of MVAPICH2 on IB Clusters

1) Existing Pin-down and Explicit-ODP Schemes: Eager protocol: MVAPICH2 uses a bounce buffer design by simply copying the user buffer to a pre-registered bounce buffer which could be directly read/write by the HCA. Since the bound buffer is pre-registered, the overhead of registering is not incurred in critical paths. The eager protocol only involves cost and issue overhead.

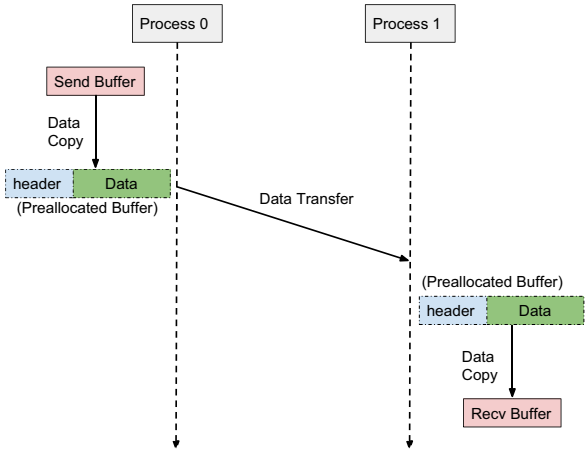


Fig. 8: Eager Protocol with Implicit-ODP

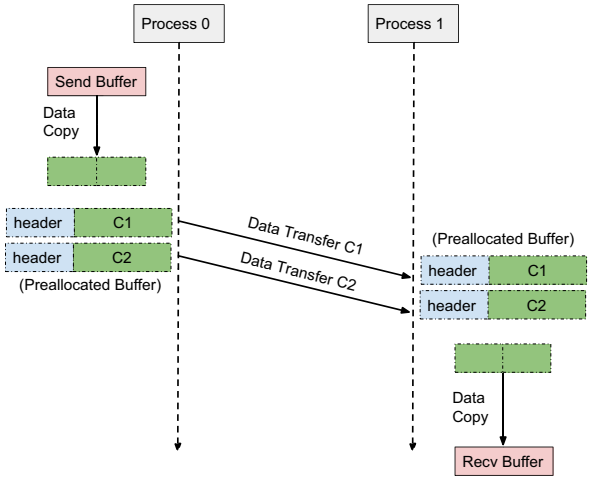


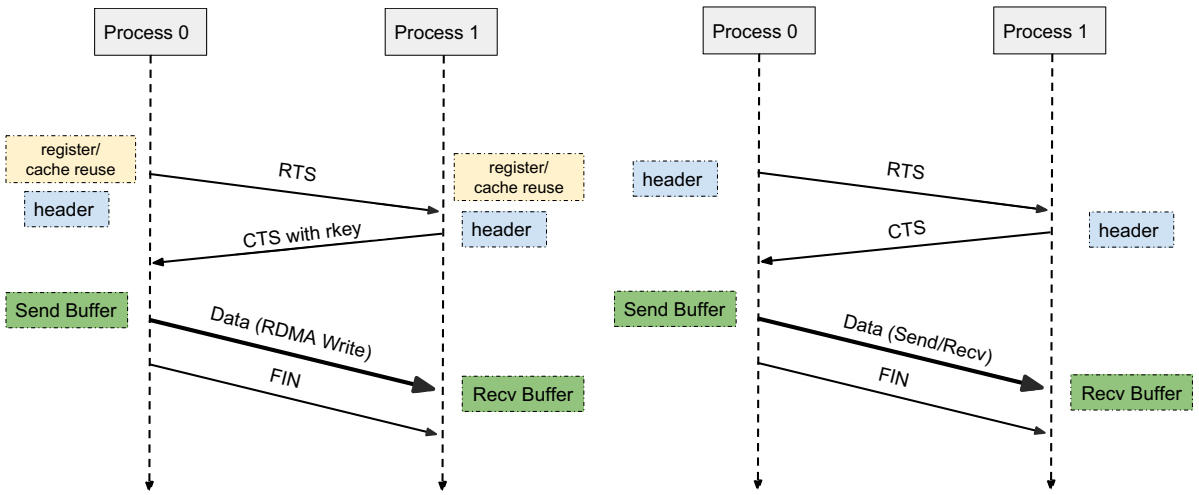
Fig. 9: Packetized Implicit-ODP based Design for Rendezvous Protocol

RDMA-based Rendezvous Protocols: The typical protocol used with RDMA write has three steps: (1) The sender registers its send buffer or reuses saved information in registration cache. Then, it sends a control message to the receiver. (2) The receiver registers its receive buffer or reuses saved information in registration cache. After that, it replies with a control message to notify the sender that the receive buffer is ready for RDMA operation. (3) The sender starts to directly write data to the receive buffer, and a finish message is sent to the receiver after the payload transfer is completed. Figure 10(a) shows how this protocol works. This is the best performing communication protocol used in MVAPICH2 which could directly write data from sender buffer to receive buffer. Rendezvous protocol with Explicit-ODP has these steps plus some pre-fetch operation in appropriate places in the library to avoid page faults [7].

2) Proposed Implicit-ODP Scheme: We propose three designs inside the MPI library to take advantage of the Implicit-ODP feature. The first design is using Implicit-ODP for small messages. The second is a chunking based design with Implicit-ODP for large messages. The third is a zero-copy based design for large messages with Implicit-ODP. With Implicit-ODP, communication buffers are paged in when they are needed by the HCA and paged out when the OS needs to swap them. This behavior will lead to page faults in critical paths. How to efficiently resolve page faults has been well explored in our previous study [7]. We are reusing the same design to pre-fetch communication buffers to avoid page faults.

B. Impact of Implicit-ODP on Registration Cache

All communication protocols with the Pin-down and Explicit-ODP scheme need to register and deregister memory buffers for communication. In order to alleviate the overhead of memory registration and deregistration, registration cache scheme is used for all communication protocols. However, registration cache has to be carefully designed and managed inside an MPI library. With the Implicit-ODP feature, since the complete address space of a process can be registered, there is



(a) Existing RDMA Write-based Zero-Copy Communication Protocol (b) Proposed IB Send-/Recv-based Zero-Copy Communication Protocol

Fig. 10: Comparisons of Existing and Proposed Zero-Copy Based Designs for Rendezvous Protocol

no need to explicitly register/deregister communication buffers. In all our Implicit-ODP based designs, we register the complete address space of a process in MPI Init time. As a result, we totally get rid of the registration cache in our Implicit-ODP based design.

C. Eager Protocol with Implicit-ODP

For short message transfers with Pin-down and Explicit-ODP scheme, wherein the payload size is smaller than the eager threshold, the header and message are copied directly into the pre-registered intermediate buffer. Using IB send primitives with eager protocol, the header and the message are transferred to the target process. With the Implicit-ODP scheme, the complete address space is registered, those pre-registered buffers are not required for the eager protocol. The HCA could directly send from the send buffer. However, to send out a small message, the library needs to send both the packet header and payload to the receiver, so the receiver could match the message and put into the corresponding receive buffer. There are two design alternatives to achieve it. One design is allocating a new buffer when an eager message needs to be sent out. At the sender side, the user buffer and header are copied into this new buffer and sent out. At the receiver side, the receiver copies the data from an intermediate buffer to the receive buffer. The issue with this design is that the overhead of buffer allocation and free is added to the critical path. The other design is pre-allocating user-defined intermediate buffers at MPI Init time and use them for the eager protocol. With this design, the overhead of memory allocation and free could be removed from the critical path. In this paper, we go for the second design alternative. This is illustrated in Figure 8.

D. Packetized Implicit-ODP based Design for Rendezvous Protocol

For large messages, most MPI stacks utilize a zero-copy based communication protocol which directly transfers data from a sender buffer to a receiver buffer. This zero-copy based protocol is based on RDMA write/read which is not supported yet with Implicit-ODP. So we propose a packetized rendezvous protocol for large messages using IB send/recv primitives. The message is divided into smaller chunks and communicated to target process using IB send primitive. Each chunk contains a header and data to help order the message chunks on the target process. At the target process, the chunked messages are received into pre-allocated buffers and re-assembled into the receive buffer. Figure 9 gives an overview of this design.

E. Zero-Copy Implicit-ODP based Design for Rendezvous Protocol

With the packetized design, each message needs to be copied from/to user buffer to/from intermediate buffer both at the sender and receiver. The extra copy incurs significant overhead for large messages. We propose a zero-copy based design using IB send/recv primitives. The header information and the message size information are sent in the form of a 'Request-To-Send'(RTS) message to the target process. On receipt of the header, the progress engine on the target process directly post the user receiver buffer with IB recv primitive and send back a 'Clear-To-Send'(CTS) message. Through this negotiation, the receiver side can guarantee that the receive buffer is ready for receiving data. When the sender receives the CTS packet, the message is directly sent from the send buffer to the receive buffer. Once the entire message has been received into the receive buffer, the posted MPI recv is completed and returned. The zero-copy transfer scheme is designed to avoid the copy overhead with large messages. This is illustrated in Figure 10(b).

Most MPI stacks prepost intermediate buffers to receive control and eager messages from peer processes. One challenge with our proposed zero-copy communication protocol is that when the receive buffer is posted, there is no guarantee that the sender data will be written into this receive buffer instead of pre-posted intermediate buffers. To ensure that the data will be directly received in user buffer, each process will maintain two

separate queue pairs (QP). The first set of QPs is used to send eager messages and control messages. The second set of QPs is used only for zero-copy based rendezvous protocol. With this design, the data is sent and received through the second set of QPs.

The RTS/CTS message used here is to avoid the page fault overhead. Once RTS/CTS messages are received, we issue IB verbs pre-fetch operation to warm up the sender and receiver buffer for upcoming data transfers. Without this RTS/CTS handshake, page faults will incur in critical path and add significant overhead.

V. Performance Evaluation

In this section, we describe our experimental setup and evaluate the performance of our MPI runtime design with both MPI micro-benchmarks and applications, respectively. We present the application level evaluation results using NAS Parallel Benchmarks, AWP-ODC, and Graph500 with a varied number of MPI processes.

A. Experimental Setup

Since the ODP feature is a cutting-edge technology, no major supercomputing centers have this kind of configuration yet. So all our experiments are conducted on a local cluster. The cluster consists of 16 compute nodes with Intel Broadwell series of processors using Xeon Dual fourteen-core processor nodes operating at 2.4 GHz with 132 GB RAM. Each node is equipped with MT4115 EDR HCAs (100 Gbps data rate). The operating system used is CentOS Linux release 7.2.1511 (Core), with kernel version 3.10.0-327.10.1.el7 and Mellanox OpenFabrics version 3.4-2.0.0. Table II presents the description of the legends we use in this section.

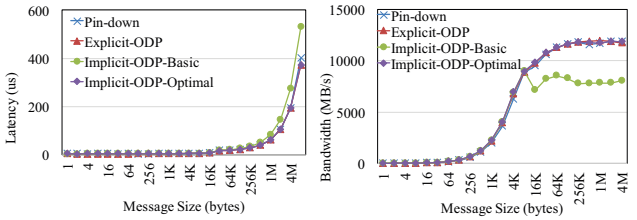
TABLE II: Legend Description

Legend	Description
Pin-down	Pin-down based design
Explicit-ODP	Explicit-ODP based design
Implicit-ODP-Basic	Proposed eager design and packetized rendezvous design
Implicit-ODP-Optimal	Proposed eager design and zero-copy rendezvous design

B. MPI Point-to-Point Micro-benchmarks

In this subsection, we evaluate the influence of our proposed designs on communication performance with OSU micro-benchmarks [17]. Figure 11 shows the MPI level point-to-point uni-directional latency and bandwidth performance numbers with existing Pin-down, Explicit-ODP, and proposed Implicit-ODP schemes.

In Figure 11(a), we can see that all schemes achieve the same performance for message sizes smaller than 16K bytes. This is because all different schemes just involve the same copy and issue overhead. However, for message sizes larger than 16K bytes, Implicit-ODP-Basic scheme starts to perform worse than other schemes. For example, the point-to-point latency with 256K bytes message size is 40.54us, 40.49us, 51.36us, and 40.04us, respectively. Compared to the



(a) MPI Point-to-Point Latency (b) MPI Point-to-Point Bandwidth

Fig. 11: MPI Point-to-Point Latency and Bandwidth Experimental Results

Pin-down scheme, the Implicit-ODP-Basic scheme has a 20% performance degradation. This overhead is because of copying data from/to user buffers to/from intermediate buffers. With our proposed zero-copy rendezvous protocol that removes the extra memory copies, the Implicit-ODP-Optimal scheme is able to achieve the same performance as the Pin-down scheme.

Figure 11(b) shows the point-to-point bandwidth performance with different schemes. We could see that the Implicit-ODP-Optimal scheme is able to improve the bandwidth significantly over the Implicit-ODP-Basic scheme. The bandwidth with 256K bytes message size for Pin-down, Explicit-ODP, Implicit-ODP-Basic, and Implicit-ODP-Optimal is 11782.91Mbps, 11794.57Mbps, 7773.62Mbps, and 11782.62Mbps, respectively. Compared to the Implicit-ODP-Basic scheme, the Implicit-ODP-Optimal scheme is able to improve the bandwidth by 30%.

C. Application Results

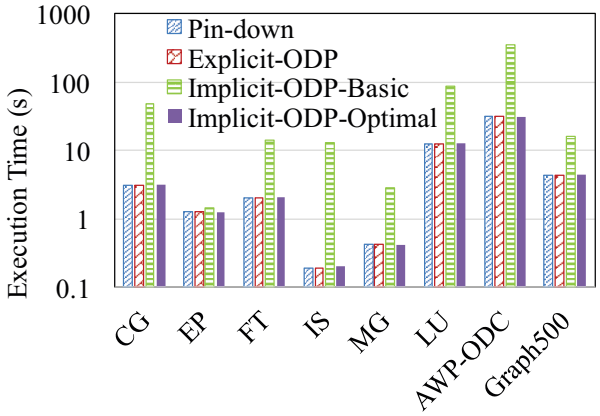
We evaluate our proposed designs with NAS Parallel Benchmarks, Graph500 and AWP-ODC. We evaluate these applications with varied system sizes. We report the total execution time and pin-down buffer sizes of application with existing and proposed schemes.

1) NAS Parallel Benchmarks: The NAS Parallel Benchmarks (NPB) are a set of programs that are designed to be typical of several MPI applications. We evaluate using the Class 'B'. We run CG, EP, FT, IS, MG, LU, and SP with 128 processes and 256 processes in the experiments. Since the number of processes for the SP benchmark needs to be a square (1, 4, 9...), so the performance numbers are missing with 128 processes. Figure 12(a) and Figure 12(b) show the total execution time of each benchmark with 128 processes and 256 processes, respectively. In Figure 12(a), we can see that Implicit-ODP-Optimal scheme always perform better than the Implicit-ODP-Basic scheme. There are two reasons for it. The first reason is that the Implicit-ODP-Optimal scheme could remove the extra copy overhead at both sender and receiver. The other reason is that the Implicit-ODP-Optimal scheme has less number of page faults compared to the Implicit-ODP-Basic scheme. For LU benchmark with 128 processes, the total execution time for Pin-down, Explicit-ODP, Implicit-ODP-Basic, Implicit-ODP-Optimal is 12.59s, 12.6s, 87.79s, and 12.59, respectively. Compared to the Implicit-ODP-Basic scheme, the Implicit-ODP-Optimal scheme could reduce the total execution time by 7X.

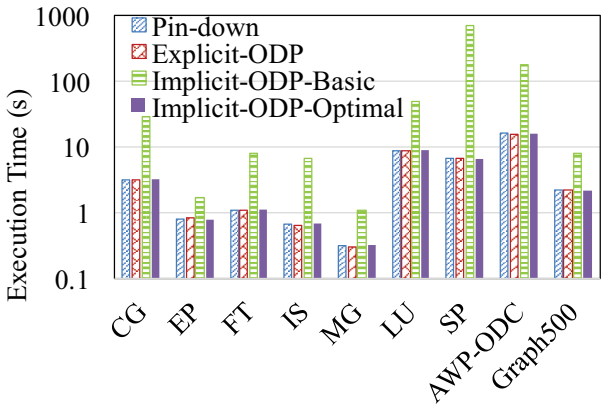
Figure 12(b) shows the NAS benchmark results with 256 processes. For the CG benchmark, the total execution time for the Pin-down, Explicit-ODP, Implicit-ODP-Basic, Implicit-ODP-Optimal scheme is 3.18s, 3.1s, 28.29s, and 3.18s, respectively. The Implicit-ODP-Optimal scheme is able to achieve the same performance as Pin-down and Explicit-ODP scheme, whereas the Implicit-ODP-Basic scheme has a significant performance degradation.

2) AWP-ODC: AWP-ODC is an elastic wave propagation code which simulates the dynamic rupture and wave propagation that occurs during an earthquake. It mainly uses non-blocking MPI primitives. We evaluate its performance with 128 processes (process grid: 8x4x4) and 256 processes (process grid: 8x8x4) running with input size (512x512x512). The performance results are shown in Figure 12(a) and Figure 12(b). We can see that the Implicit-ODP-Optimal scheme is able to maintain same performance as the Pin-down and Explicit-ODP schemes. For the problem size of 512x512x512 with 64 and 128 processes, compared with the Implicit-ODP-Basic scheme, the Implicit-ODP-Optimal scheme is able to reduce the execution time by 11X and 12X, respectively.

3) Graph500: Graph500 [18] is a Breadth First Search (BFS) benchmark, which reports the BFS traversal for a given input graph size. It has a random-access communication pattern where root processes per node receive random messages from other processes. We run this application with scale 26 and edgefactor 16. The experimental results are shown in Figures 12(a) and 12(b). From the results, we could see that the Implicit-ODP-Basic scheme performs worse than the Pin-down scheme. The performance degradation comes from page fault events and copy overhead in packetized-based design. Our proposed Implicit-ODP-Optimal scheme is able to remove the copy overhead as well as reduce the number of page faults to deliver comparable performance as the pin-down scheme.



(a) Performance of Applications with 128 Processes



(b) Performance of Applications with 256 Processes

Fig. 12: Application Level Experimental Results

D. Benefits of Implicit-ODP

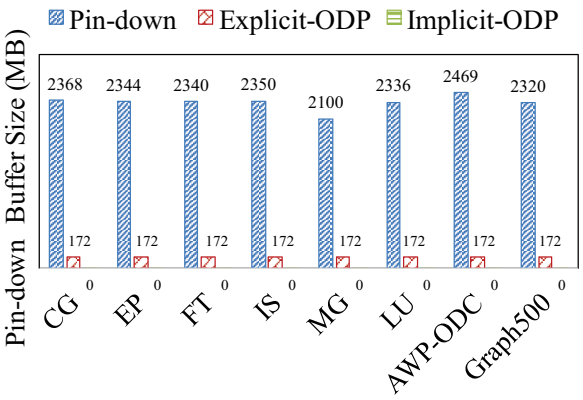
We also measure the pin-down buffer sizes with the Pin-down, Explicit-ODP, Implicit-ODP schemes at application level. The results are shown in Figures 13(a) and 13(b) with 128 processes and 256 processes, respectively. With the Pin-down scheme, all the communication buffers are pinned in memory until the end of each run. So it has the largest pin-down buffer size. For the Explicit-ODP scheme, only the preallocated buffers are pinned in memory. With our proposed Implicit-ODP based scheme, no communication buffer or preallocated buffers are pinned in memory. All the communication buffers are paged in when the HCA needs them and swapped out when the operating system reclaims them. Our proposed Implicit-ODP-Optimal scheme could maintain the same performance as the pin-down and Explicit-ODP scheme while removing all pin-down buffers.

Another benefit with our Implicit-ODP based schemes is removing the registration cache design which is used in almost all MPI stacks on IB networks. MPI stacks need to maintain an internal data structure to avoid repeatedly registering the same communication buffer.

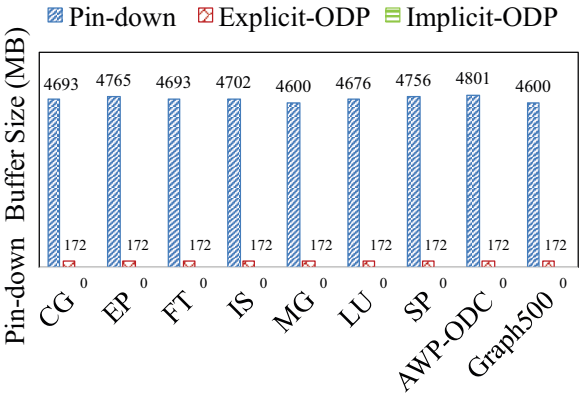
Last but not least, the number of registration calls with Implicit-ODP could be reduced significantly. Compared to the Pin-down and Explicit-ODP scheme which needs to register all 'cold' communication buffers, Implicit-ODP scheme only needs one registration call to register the whole address space in the beginning of a run.

VI. Related Work

Many research works have been proposed to reduce the memory footprint of MPI libraries over InfiniBand clusters. Koop, et al. proposed [19] a connection-less Unreliable Datagram (UD) based design to reduce the connection memory usage. Rashti, et al. proposed [20] implementation of iWARP over UD and accessed its potential benefits for HPC applications. Subramoni, et al. explored [6] designs based on shared connection to reduce the memory consumption. Shipman, et al. proposed using different Shared Receive Queue (SRQ) for different data sizes to improve utilization of communication buffers on RC [21], [22]. A dynamic approach that uses both RC and UD protocols based on the communication pattern of the application has also been designed [23]. Erimli [24] used a method based on multiply-linked lists to share memory for WQEs. Perache, et al. [25] extended MPC framework providing an MPI library to reduce the overall memory consumption of MPI applications. All these studies are able to reduce the



(a) Pin-down Buffer Size of Applications with 128 Processes



(b) Pin-down Buffer Size of Applications with 256 Processes

Fig. 13: Application Level Pin-down Buffer Size

memory footprint of MPI libraries. However, they still need to pin-down communication buffers for data transfers.

Cray Aries [26] network uses I/O Memory Management Unit (IOMMU) to translate virtual-to-physical addresses for data transfers. Meiko CS-2 [27] uses a protocol processor to deal with address translation and data transfer. Hong et al. [28] proposed an algorithm to reduce cache conflict misses. UNet-MM [29], an extension U-Net, stores address translation in a translation cache on the network interface. Misses in the translation cache are handled by the host OS which pins virtual pages and installs their translations on the network interface. Petrini et al. [30] described (Quadrics interconnection network) QsNet that extends the native operating system in the nodes with a network operating system and specialized hardware support in the network interface. QsNet does not require memory to be registered before communication. MPICHQsNet and Open MPI were used to provide support over Quadrics [31]. Since the Quadrics network is very old, so we do not have access to such systems and evaluate its performance.

Woodall et al. [32] described a pipeline protocol by overlapping the dynamic registration and deregistration of memory buffers with data transfer. Li et al. [33] proposed a memory registration strategy that hides the cost of dynamic memory management by offloading all dynamic memory registration/deregistration requests to a dedicated memory management helper thread. Bao et al. [34], [35], [36] proposed a compile-time approach to CPU frequency and core count selection to optimize energy. Our previous studies [7] have explored the Explicit-ODP feature for MPI point-to-point and one-sided communication. In contrast to the previous work, this work replaced existing Pin-down and Explicit-ODP scheme with the Implicit-ODP scheme for data transfers. We also proposed designs in this paper to improve the performance of transferring large messages with ODP.

VII. Conclusion and Future Work

In this paper, we presented the Implicit-ODP feature and evaluated its performance characteristics with IB verbs-level micro-benchmarks. We proposed registration caching free Implicit-ODP based designs for MPI point-to-point communications. We showed that directly reusing existing communication protocol for Implicit-ODP led to significant performance degradation compared to the Pin-down scheme. We then proposed a zero-copy communication protocol which delivered the same performance as existing Pin-down and Explicit-ODP schemes. Furthermore, our proposed Implicit-ODP based schemes were able to maintain performance without registration cache, which typically is a complex component in MPI stacks. We systematically evaluated proposed schemes with MPI level micro-benchmarks, NAS Parallel Benchmark, Graph500, and AWP-ODC applications. The experimental results at the micro-benchmark level show that the Implicit-ODP based scheme in MVAPICH2 is able to deliver comparable performance as existing pin-down scheme without any pin-down buffers. In future work, we plan to evaluate our designs at scale when the ODP feature is getting adopted in major supercomputing centers. As HPC cloud is gaining momentum [37], [38], [39], we also plan to evaluate the impact of the proposed design on the emerging HPC cloud in the future. In addition, one-sided programming model [40], [41], [42] has been shown to benefit applications with irregular communication patterns. we also plan to extend our design for one-sided programming model and applications.

References

[1] InfiniBand Trade Association, http://www.infinibandta.com.

[2] M. Luo, M. Li, A. Venkatesh, X. Lu, and D. K. Panda, "Upc on mic: Early experiences with native and symmetric modes," in *Proceedings of the 7th International Conference on Partitioned Global Address Space Programming Models*, ser. PGAS '13, 2013.

[3] J. Lin, K. Hamidouche, X. Lu, M. Li, and D. K. Panda, "High-performance coarray fortran support with mvapich2-x: Initial experience and evaluation," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, May 2015.

[4] S. Potluri, K. Kandalla, D. Burdeey, M. Li, and D. K. Panda, "Efficient intranode desgins for openshmem on multicore clusters," in *Proceedings of the 6th International Conference on Partitioned Global Address Space Programming Models*, ser. PGAS '12, 2012.

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

[6] H. Subramoni, K. Hamidouche, A. Venkatesh, S. Chakraborty, and D. K. Panda, "Designing MPI Library with Dynamic Connected Transport (DCT) of InfiniBand: Early Experiences," in *International Supercomputing Conference (ISC)*, 2014.

[7] M. Li, K. Hamidouche, X. Lu, H. Subramoni, J. Zhang, and D. K. Panda, "Designing MPI Library with On-demand Paging (ODP) of Infiniband: Challenges and Benefits," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '16, 2016.

[8] J. Liu, J. Wu, and D. K. Panda, "High Performance RDMA-Based MPI Implementation over InfiniBand," in *17th Annual ACM International Conference on Supercomputing*, June 2003.

[9] M. Li, H. Subramoni, K. Hamidouche, X. Lu, and D. K. Panda, "High Performance MPI Datatype Support with User-Mode Memory Registration: Challenges, Designs, and Benefits," in *2015 IEEE International Conference on Cluster Computing*, Sept 2015.

[10] H. Tezuka, F. O. Carroll, A. Hori, and Y. Ishikawa, "Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication," in *Proceedings of 12th Int. Parallel Processing Symposium*, March 1998.

[11] M. Li, S. Potluri, K. Hamidouche, J. Jose, and D. K. Panda, "Efficient and Truly Passive MPI-3 RMA Using InfiniBand Atomics," in *Proceedings of the 20th European MPI Users' Group Meeting*, ser. EuroMPI '13, 2013.

[12] M. Li, K. Hamidouche, X. Lu, J. Zhang, J. Lin, and D. K. Panda, "High Performance OpenSHMEM Strided Communication Support with InfiniBand UMR," in *2015 IEEE 22nd International Conference on High Performance Computing (HiPC)*, Dec 2015.

[13] M. Li, K. Hamidouche, X. Lu, J. Lin, and D. K. D. Panda, *High-Performance and Scalable Design of MPI-3 RMA on Xeon Phi Clusters*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015.

[14] S. Raindel, H. Eran, L. Liss, and N. Bloch, "Look-ahead Handling of Page Faults in I/O Operations," Dec. 16 2014, uS Patent 8,914,458. [Online]. Available: https://www.google.com/patents/US8914458

[15] N. Bloch, S. Raindel, H. Eran, and L. Liss, "Use of Free Pages in Handling of Page Faults," Jun. 3 2014, uS Patent 8,745,276. [Online]. Available: https://www.google.com/patents/US8745276

[16] D. K. Panda, K. Tomko, K. Schulz, and A. Majumdar, "The MVAPICH Project: Evolution and Sustainability of an Open Source Production Quality MPI Library for HPC," in *Int'l Workshop on Sustainable Software for Science: Practice and Experiences, held in conjunction with Int'l Conference on Supercomputing (SC '13)*, November 2013.

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

[18] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, "Introducing the Graph 500," *Cray User Group*, 2010.

[19] M. J. Koop, S. Sur, Q. Gao, and D. K. Panda, "High Performance MPI Design using Unreliable Datagram for Ultra-scale InfiniBand Clusters," in *ICS '07: Proceedings of the 21st annual international conference on Supercomputing*. New York, NY, USA: ACM, 2007, pp. 180–189.

[20] M. J. Rashti, R. E. Grant, A. Afsahi, and P. Balaji, "iWARP Redefined: Scalable Connectionless Communication over High-speed Ethernet," in *High Performance Computing (HiPC), 2010 International Conference on*. IEEE, 2010, pp. 1–10.

[21] G. M. Shipman, T. S. Woodall, R. L. Graham, A. B. Maccabe, and P. G. Bridges, "InfiniBand Scalability in Open MPI," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006, pp. 10–pp.

[22] G. M. Shipman, R. Brightwell, B. Barrett, J. M. Squyres, and G. Bloch, "Investigations on InfiniBand: Efficient Network Buffer Utilization at Scale," in *Proceedings, Euro PVM/MPI*, Paris, France, October 2007.

[23] M. J. Koop, T. Jones, and D. K. Panda, "MVAPICH-Aptus: Scalable High-performance Multi-transport MPI over InfiniBand," in *IPDPS'08*, 2008, pp. 1–12.

[24] B. Erimli, "Arrangement in an InfiniBand Channel Adapter for Sharing Memory Space for Work Queue Entries using Multiply-linked Lists," Mar. 18 2008, uS Patent 7,346,707.

[25] H. J. Marc Perache, Patrick Carribault, "MPC-MPI: An MPI Implementation Reducing the Overall Memory Consumption," in *Proceedings, Euro PVM/MPI*, Helsinki, Finland, October 2009.

[26] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Cray XC Series Network," Tech. Rep., 2012.

[27] J. Beecroft, M. Homewood, and M. McLaren, "Meiko cs-2 interconnect elan-elite design," *Parallel Computing*, vol. 20, no. 10, pp. 1627 – 1638, 1994.

[28] C. Hong, W. Bao, A. Cohen, S. Krishnamoorthy, L.-N. Pouchet, F. Rastello, J. Ramanujam, and P. Sadayappan, "Effective padding of multidimensional arrays to avoid cache conflict misses," *SIGPLAN Not.*, Jun. 2016.

[29] M. Welsh, A. Basu, and T. von Eicken, "Incorporating memory management into user-level network interfaces," Ithaca, NY, USA, Tech. Rep., 1997.

[30] F. Petrini, W.-c. Feng, A. Hoisie, S. Coll, and E. Frachtenberg, "The Quadrics Network: High-Performance Clustering Technology," *IEEE Micro*, vol. 22, no. 1, pp. 46–57, Jan. 2002. [Online]. Available: http://dx.doi.org/10.1109/40.988689

[31] W. Yu, T. S. Woodall, R. L. Graham, and D. K. Panda, "Design and Implementation of Open MPI over Quadrics/Elan4," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, April 2005, pp. 96a–96a.

[32] T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, "High Performance RDMA Protocols in HPC," in *Proceedings of the 13th European PVM/MPI User's Group Conference on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, ser. EuroPVM/MPI'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 76–85. [Online]. Available: http://dx.doi.org/10.1007/11846802_18

[33] D. Li, K. W. Cameron, D. S. Nikolopoulos, B. R. d. Supinski, and M. Schulz, "Scalable Memory Registration for High Performance Networks Using Helper Threads," in *ACM International Conference on Computing Frontiers Supercomputing (CF11)*, Ischia, Italy, May 2011.

[34] W. Bao, C. Hong, S. Chunduri, S. Krishnamoorthy, L.-N. Pouchet, F. Rastello, and P. Sadayappan, "Static and dynamic frequency scaling on multicore cpus," *ACM Trans. Archit. Code Optim.*, Dec. 2016.

[35] W. Bao, S. Tavarageri, F. Ozguner, and P. Sadayappan, "Pwcet: Power-aware worst case execution time analysis," in *2014 43rd International Conference on Parallel Processing Workshops*, Sept 2014, pp. 439–447.

[36] W. Bao, S. Krishnamoorthy, L.-N. Pouchet, F. Rastello, and P. Sadayappan, "Polycheck: Dynamic verification of iteration space transformations on affine programs," *SIGPLAN Not.*, vol. 51.

[37] J. Zhang and X. Lu and D. K. Panda, "High-Performance Virtual Machine Migration Framework for MPI Applications on SR-IOV enabled InfiniBand Clusters," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Orlando, USA, May 2017.

[38] J. Zhang, X. Lu, J. Jose, M. Li, R. Shi, D. K. Panda, "High Performance MPI Library over SR-IOV Enabled InfiniBand Clusters," in *Proceedings of International Conference on High Performance Computing (HiPC)*, Goa, India, December 17-20 2014.

[39] J. Zhang and X. Lu and D. K. Panda, " High Performance MPI Library for Container-based HPC Cloud on InfiniBand Clusters ," in *The 45th International Conference on Parallel Processing (ICPP '16)*, Philadelphia, USA, August 2016.

[40] M. Li, X. Lu, K. Hamidouche, J. Zhang, and D. K. Panda, "Mizan-rma: Accelerating mizan graph processing framework with mpi rma," in *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*, Dec 2016, pp. 42–51.

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

[42] M. Li, J. Lin, X. Lu, K. Hamidouche, K. Tomko, and D. K. Panda, "Scalable MiniMD Design with Hybrid MPI and OpenSHMEM," in *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*, ser. PGAS '14, 2014.