

# Non-threaded and Threaded Approaches to MultiRail Communication with uDAPL

Jie Cai <sup>#</sup>, Alistair P. Rendell <sup>\*</sup>, Peter E. Strazdins <sup>+</sup>

*School of Computer Science, the Australian National University  
Canberra ACT 0200 Australia*

<sup>#</sup> Jie.Cai@anu.edu.au

<sup>\*</sup> Alistair.Rendell@anu.edu.au

<sup>+</sup> Peter.Strazdins@anu.edu.au

## Abstract

*uDAPL is a portable and platform independent communication library that provides RDMA as well as send/recv operations. Some well-known software has attempted to take advantage of uDAPL's portability, such as Open MPI, MVAPICH2, Intel MPI, and Cluster OpenMP. However, the network bandwidth limitation can still be a bottleneck for applications using these software. Engaging a "Multirail" network is a method to by-pass this. In this paper, we design a non-threaded and a threaded approach to improve the performance of uDAPL over multirail configured clusters. The two approaches are evaluated on an InfiniBand cluster with different multirail configurations. The results show that the threaded approach improves 33% and 148% of the uni-directional bandwidth on the multi-port and the multi-HCA configured network respectively, and the non-threaded approach improves ~90% of the uni-directional bandwidth on the multi-HCA configured network. A similar improvements is achieved for the bi-directional bandwidth.*

## 1. Introduction

The user-level direct access programming library (uDAPL) defined by the DAT Collaborative attempts to provide a network, architecture and operating system independent interface to applications for remote direct memory access (RDMA) communications [1]. This potentially allows applications to seamlessly use different networks as the underlying transport with minimal effort. Nowadays, some well-known communication libraries and applications are trying to take advantage of the better portability of uDAPL, such as MVAPICH2 [2], [3], Intel MPI Library [4], Open MPI [5], and Cluster OpenMP [6], [4].

InfiniBand is an emerging networking technology supporting low latency, high bandwidth and RDMA communications. At the end of 2008, 36% of the top 50 supercomputers listed in TOP500 [7] are using InfiniBand as their interconnection network. However, even with InfiniBand, network bandwidth can still become the performance bottleneck for

some applications, especially for clusters built with multi-core machines [8]. A well-known method to overcome the bandwidth limitation is to use "multirail networks" [9], [8], [10]. However, none of those attempts solved the problem with a generic/portable solution; either the socket application programming interface (API) or the InfiniBand Verb API was targeted. Furthermore, only a single process was engaged to handle communications over both rails for those attempts.

Hence, a question, "Can significant performance improvement be achieved through a portable and platform independent communication library?", arises. As some InfiniBand network interface cards, known as host channel adapters (HCA), provide dual-ports, there are some different ways to configure a multirail network. Therefore, another question, "What is the best way to pursue the performance improvement on different configured multirail networks?", arises as well. In this paper, we will answer both questions. To explore these, a set of uDAPL multirail benchmarks will be designed. Two different design approaches, threaded and non-threaded, are developed. The two approaches will be evaluated on an InfiniBand cluster with different multirail configurations.

The rest of this paper will be managed in five sections. In section 2 and 3, background knowledge of uDAPL and multirail networks will be introduced. In section 4, design issues of multirail uDAPL benchmarks for the two different approaches will be discussed, and illustrated. The benchmarks will be evaluated on an InfiniBand cluster in section 5 followed by conclusions and future work in section 6.

## 2. uDAPL

The Direct Access Programming Library (DAPL) was released by DAT Collaborative [11] as an attempt to provide a portable set of API for all RDMA networks. DAPL contains a kernel and user space specification in the C programming language, which are kDAPL and uDAPL respectively. User programs use uDAPL for generic RDMA applications [1].

Abstract concepts are represented as objects in the uDAPL specifications. For example, an Interface Adapter (IA) is the

object used to represent a RDMA network interface <sup>1</sup>. An Endpoint (EP) is the local part of a connection that supports the posting of data transfer operation (DTO) requests and receives. A Public Service Point (PSP) is a persistent service point advertised to other hosts to support connections. An event is a structure or record that is delivered to the consumer through an Event Dispatcher to provide notifications, such as DTO completions, connections, asynchronous errors and user events. An Event Dispatcher (EVD) is the object that queues these events for the consumer. A Protection Zone (PZ) of an IA defines protection for the local and remote memory access by DTOs. It is also a mechanism to associate an EP with a registered local memory region (LMR) and a remote memory region (RMR). These objects are related to one another in an ownership hierarchy. The IA object is the root of this hierarchy [1].

uDAPL supports reliable connections using a client-server connection model. The passive side of a connection creates a service point, and the active side issues a Connect Request (CR) through an EP to the service point at the passive side. Four kinds of DTOs are supported: send and receive, RDMA write and read. All of these operations are non-blocking. The successful return of a DTO does not mean that the data transfer is completed. The RDMA DTO initiator needs to wait for an event to acknowledge the DTO completion, and the remote side needs to be notified with a message send/rcv after the RDMA DTO <sup>2</sup>. The transport ordering rules of uDAPL guarantees: (1) the receive operations on a connection must be completed in the order of posting of their corresponding sends and (2) each RDMA write operation posted on a connection prior to a send operation must have its data payload delivered to the target memory region prior to the completion of the receive operation matching that send [1].

As the default setting for uDAPL, the consumer is notified by an event through an EVD once a DTO is completed. However the DTO completion event can be suppressed when the consumer specifies a suppress flag for the data transfer. If the consumer wishes to know when the DTO is completed, the consumer needs to issue an event waiting operation, and waits for the EVD to deliver the corresponding event. Otherwise, the completion suppress flag can be used for saving this busy-waiting time. Generally, uDAPL employs an event-driven communication/notification model.

### 3. Multirail Configured InfiniBand

Multirail refers to a network containing multiple physical connections or “rails” [9]. Utilizing multirail for cluster com-

1. DAT recognizes different ports on a same InfiniBand Host Channel Adapter (HCA) as different network interfaces or devices.

2. The InfiniBand extension RDMA write with the immediate data function does not require a message send to notify the completion of DTO. However, the remote side still needs to issue a receive operation to receive the 64 bytes in-line data.

munications can significantly improve network bandwidth and communication efficiency.

As some current InfiniBand host channel adapters (HCA) contain multiple ports, the InfiniBand multirail network can be configured in different ways, as shown in figure 1 [8].

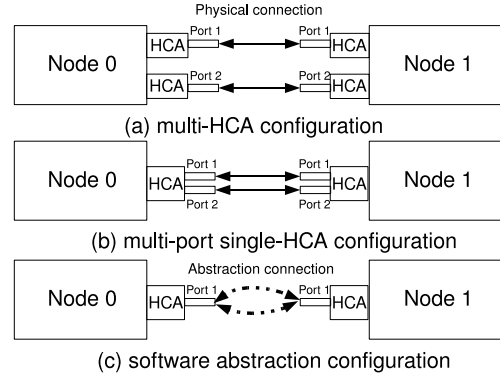


Figure 1. Different ways to configure a InfiniBand multirail network [8].

As illustrated in figure 1, there are three different configurations of InfiniBand multirail networks. There are multiple HCAs employed for each node at configuration (a). For configuration (b) a single HCA with multiple ports (SHMP) is employed for each node. Configuration (c) establishes a multirail network at an abstract level via software design. For configurations (a) and (b), physical “rails” are actually established, which is the real “multirail”. For configuration (c), only abstraction sub-channels are established on a single physical connection.

There are different theoretical bandwidth upper bounds for different multirail configurations. In general, the theoretical bandwidth for configuration (a) can be formulated as follows:

$$B_a = N_r \text{Min}(B_{IB}, B_{HCA}, B_{host}) \quad (1)$$

In equation (1),  $B_a$  stands for theoretical bandwidth for configuration (a),  $N_r$  stands for number of rails,  $B_{IB}$  stands for InfiniBand speed,  $B_{HCA}$  stands for HCA PCI speed, and  $B_{host}$  stands for host machine PCI speed. For example, if we use two 4x DDR x8 PCI-e 1.0 InfiniBand HCAs with a node equipped with PCI-e 2.0 slots,  $B_{IB} = 2.0\text{GB/s}$  [12],  $B_{HCA} = 2.0\text{GB/s}$ , and  $B_{host} = 4\text{GB/s}$ . Hence,  $B_a = 4\text{GB/s}^3$ .

For configuration (b), the theoretical bandwidth could be formulated as equation (2).

$$B_b = \text{Min}(N_r B_{IB}, B_{HCA}, B_{host}) \quad (2)$$

Taking the same example as above, the theoretical bandwidth for configuration (b) is limited by HCA PCI speed, which is  $2.0\text{GB/s}$ .

3. 8B/10B encoding has been used for both PCI-e 2.0 or lower and InfiniBand data transmission.

For configuration (c), the theoretical bandwidth is shown in equation (3).

$$B_c = \text{Min}(B_{IB}, B_{HCA}, B_{host}) \quad (3)$$

Taking the same example as above, the theoretical bandwidth for configuration (c) is limited by either  $B_{IB}$  or  $B_{HCA}$ , which is  $2.0GB/s$ .

Based on the above analysis, generally, configuration (c) has the lowest theoretical peak bandwidth. When  $N_r B_{IB}$  is larger than  $B_{HCA}$ ,  $B_{HCA}$  becomes the limitation. In this case, although the theoretical peak bandwidth of the configuration (c) may be as the same as the configuration (b), software and hardware contention on configuration (c) has more effect [8]. Therefore, in this paper, we will only consider the first and second configurations of InfiniBand multirail networks.

## 4. Micro-Benchmarks

Since the improvement on bandwidth is the major advantage brought by “multirail networks”, we have developed two different approaches, threaded and non-threaded, to achieve the bandwidth improvement on the multirail networks. The corresponding benchmarks have been implemented for each approach. RDMA operations are utilized to transfer the primary data in the benchmarks<sup>4</sup>. Design issues related to these benchmarks and the details of their implementation will be discussed in the following subsections.

### 4.1. Design Issues

As we mentioned in section 2, DAPL recognizes different ports as different DAT InfiniBand devices even if those ports are physically located in the same adapter. Therefore, we can utilize one Interface Adapter (IA) object for each device. The benchmark for the single-rail configuration will just open a single IA per node, while the benchmarks for the multirail configurations will need to open multiple IAs per node.

Correspondingly, other objects, such as protection zone (PZ), endpoint (EP), public service point (PSP), event dispatcher (EVD) and local/remote memory region (LMR/RMR), will need to be created for each IA. LMRs/RMRs for different IAs can be registered to the same memory region, which allows fast data storing and avoids data moving operations at the remote site after a multirail RDMA write operation.

Due to the event-driven mechanism and data transfer operation mechanism utilized in uDAPL (see section 2), the timed section would include a phase of waiting for notification events. Moreover, to explore the peak bandwidth

4. The source code of these benchmarks are available at <http://ccnuma.anu.edu.au/dsm/multirail>.

of using uDAPL over multirail InfiniBand networks, we RDMA-transferred a chunk of data multiple times. The completion event suppress flag is utilized to avoid multiple RDMA DTO completion events. The benchmarks only waits for the events from the notification messages sending after all RDMA DTOs. The details of implementation for the different benchmarks will be described and discussed in the rest of this section.

### 4.2. Single-Rail Benchmark

A single-rail bandwidth benchmark was implemented to measure the base line bandwidth. In the benchmark, a single IA and a single set of leaf objects (EP, PZ, LMR/RMR) are created at both the server (local) and the client (remote) sides. A single PSP object is created at the server side for connection establishment.

As showed in 2, the benchmark contains a major loop which is looping for different data sizes (N) from 0 bytes to 2MB. Within the loop, the server side RDMA writes data 20 times followed by a message sent to the remote side. Then the server issues a receive operation to receive an acknowledgment back from the client, and a event-wait to wait for the completion event of this receive operation. The capture of the event indicates the RDMA-written data is visible at the client side.

Server (local) Side:

```

Loop for data size (N) from 0 bytes to 2 Mbytes

    Start Timer (t1)

    Loop from 0 to 20
        dat_ep_post_rdma_write() with suppress flag
    End Loop

    dat_ep_post_send() with suppress flag

    dat_ep_post_recv() matching client post send
    dat_evd_wait() wait for event of above recv
        completion

    Stop Timer (t2)
    Bandwidth_N = 20*N/(t2-t1)
End Loop

```

Client (remote) Side:

```

Loop for data size (N) from 0 bytes to 2 Mbytes

    dat_ep_post_recv() with default flag
    dat_evd_wait() wait for event for above recv
        completion

    dat_ep_post_send() with suppress flag to indicate
        server RDMA data is visible on
        the client side.

End Loop

```

Figure 2. Single-rail bandwidth benchmark

A timer has been inserted on the server side. If the time spent on all above operations on server side is  $\Delta T$ , then we

have the bandwidth for RDMA write of data of length  $N$  as  $20N/\Delta T$  on a single-rail InfiniBand network.

### 4.3. Multirail Benchmark

The basic idea to implement the multirail benchmark is to split the data to be RDMA transferred into two even parts, with the RDMA sending one part through one rail and another part through another rail simultaneously.

According to our previous discussion in section 4.1, different IAs need to be opened for different DAT InfiniBand devices. For each IA, we need to create an EP, a PSP, a PZ and a LMR/RMR for it. However, uDAPL supports that the LMR/RMR for different IAs can be registered to the same memory region, in which the send/recv buffer has been allocated, as shown in figure 3.

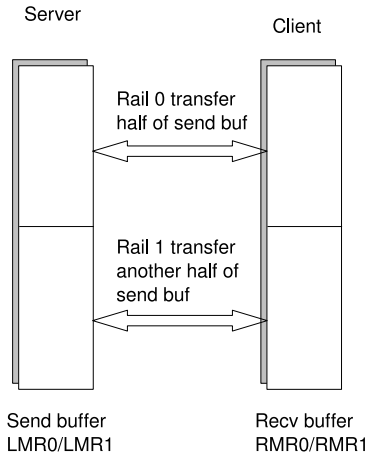


Figure 3. Multirail communication memory access pattern.

The multirail benchmarks have been implemented with two different approaches. The first approach utilizes a single process at both the server and the client sides to handle the communication through both rails, while the second approach utilizes one thread per rail, and explores the full parallelism of data sending through different rails.

**4.3.1. non-threaded.** The implementation for the non-threaded multirail bandwidth benchmark is extended based on the single-rail benchmark, as shown in figure 4.

The communications on both rails are through a loop over different rail IDs to achieve the simultaneous data transfer. At the server side, for each data size, the benchmark is designed to send data through both rails multiple times “simultaneously”, followed by a notification sent to the client side. Then an acknowledgment is received from the client to indicate that the data is visible at the remote memory. At the last, a completion event is captured “simultaneously” on each rail. The timer is set at the server side as for the

```

Server (local) Side:
  Loop for data size (N) from 0 bytes to 2 Mbytes
    Start Timer (t1)
    Loop from 0 to 20
      Loop for different rails from 0 to 1
        rdma_write() N/2 data with suppress flag
      End Loop
    End Loop

    Loop for different rails from 0 to 1
      dat_ep_post_send() with suppress flag
    End Loop

    Loop for different rails from 0 to 1
      dat_ep_post_recv() matching client post send
      dat_evd_wait() wait for event of above recv
        completion
    End Loop

    Stop Timer (t2)
    Bandwidth_N = 20*N/(t2-t1)
  End Loop
-----
Client (remote) Side:
  Loop for data size (N) from 0 bytes to 2 Mbytes
    Loop for different rails from 0 to 1
      dat_ep_post_recv() with default flag
      dat_evd_wait() wait for completion event for
        recv
    End Loop

    Loop for different rails from 0 to 1
      dat_ep_post_send() with suppress flag to indicate
        server RDMA data is visible on
        the client side.
    End Loop
  End Loop

```

Figure 4. Non-threaded multirail bandwidth benchmark

single-rail design, which includes both rail RDMA writes, message send/recv and event wait.

The term “simultaneously” does not mean the data transfer through both rail happened at exactly the same time. The RDMA function calls are actually serialized; however, the real data transfer operations are maximally overlapped.

Although the overlapping of communication on different rails boost the benchmark performance, the waiting for DTO completion events is still fully serialized and can not be overlapped. This is the major overhead for the non-threaded approach. This problem will be solved by utilizing the threaded approach as discussed in next section.

**4.3.2. threaded.** To avoid the overhead introduced by the non-threaded approach, it is extended with OpenMP threads to fully parallelize the communication and event waiting on the different rails. The details are shown in figure 5.

For each data size, the main program thread will fork into 2 working threads and start the parallel region. At the server side, one working thread RDMA writes (with a suppress event flag) the top half of the data via rail0 multiple times, and another working thread RDMA writes (with a suppress

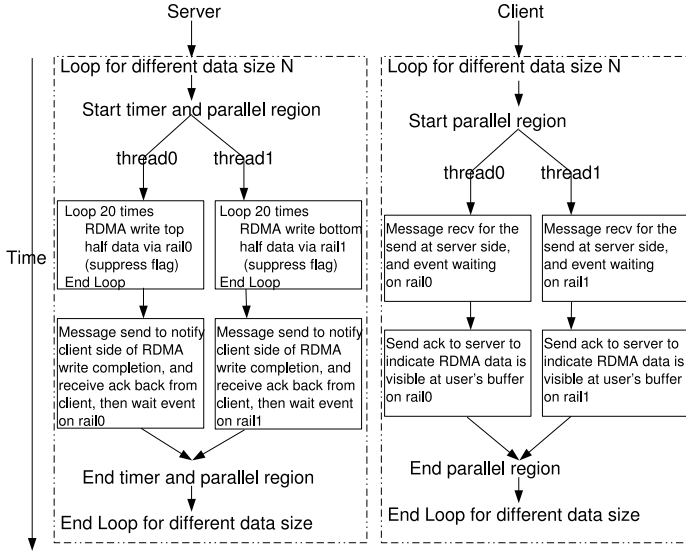


Figure 5. Threaded multirail benchmark design.

event flag) the bottom half of the data via rail1 multiple times as well. Then each thread sends a message to notify the client that all RDMA writes are completed, and waits for an acknowledgment from the client to indicate that the data is visible at remote user’s buffer. Once the both sides captured the recv completion event, the RDMA writes are guaranteed to be completed at both sides. The parallel region terminates and the 2 working threads are joined.

The timer is started before the parallel region starts and stopped after the parallel region is stopped.

## 5. Performance Evaluation

In this section, we present performance results of our uDAPL non-threaded and threaded benchmarks on an InfiniBand cluster with different multirail configurations. Both the uni-directional and the bi-directional bandwidth benchmarks are employed. The bi-directional benchmarks required trivial extensions based on the uni-directional benchmarks. The latency results were obtained by running the uni-directional benchmark with short message sizes. These benchmarks will be evaluated on the multirail network configurations (a) and (b), as described in section 3.

### 5.1. Experimental Setup

Our experimental InfiniBand cluster consists of four Sun Ultra24 workstations, which contain one Intel Core 2 Quad Q6600 CPU with 4GB DDR2 memory on each node, and are connected using Mellanox ConnectX MHGH28-XTC dual-port HCAs.

Each Sun Ultra24 workstation contains two x16 PCI-e Gen2 slots, which supports up to 4GB/s bandwidth for

one x8 PCI-e 2.0 connectors. Additionally, the Sun Ultra24 workstation supplies two DDR2 memory channels, connected to the two x16 PCI-e Gen2 slots separately.

The Mellanox ConnectX MHGH28-XTC dual-port HCA has a host bus speed at PCIe 1.1 speed, which is 2.0GB/s. The InfiniBand speed of each port in MHGH28-XTC HCA is 4x DDR InfiniBand, which is 2.0GB/s (peak).

Based on equations (1) and (2), the theoretical peak uni-bandwidth on the multi-HCA configured InfiniBand cluster is 4GB/s, and the theoretical peak uni-bandwidth on the multi-port configured InfiniBand cluster is 2.0GB/s. By contrast, the peak uni-directional bandwidth for the single-rail network is limited by the InfiniBand speed (or HCA PCI speed), which is 2.0GB/s.

Moreover, for the threaded benchmark, two different threads has been set affinity to the core 0 and 3 respectively on each node.

### 5.2. Latency

As we have mentioned in the beginning of this section, the uni-bandwidth benchmarks are utilized to collect the short message (from 8 Bytes to 4KB) latency data on different network configurations. Figure 6 shows the RDMA write latency ( $\mu s$ ) for the non-threaded and the threaded benchmarks on three different network configurations: single-rail, multi-port and multi-HCA. However, the latency on the single-rail network is only measured with the non-threaded benchmark.

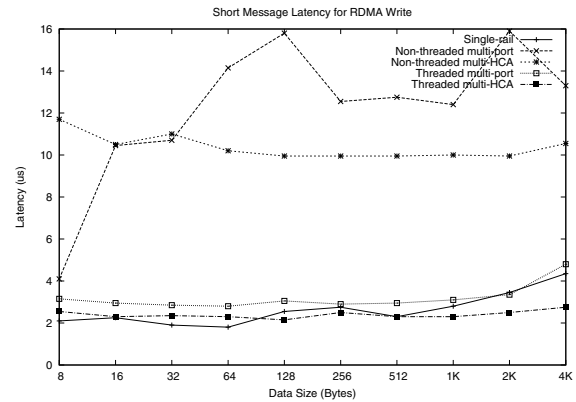


Figure 6. RDMA write latency comparison.

In figure 6, the latencies for the threaded benchmark on the multi-HCA and the multi-port configurations are quite comparable with the single-rail latency, in the range of  $2\mu s$  to  $4\mu s$ . The latencies for the multi-port threaded benchmark are slightly higher than single-rail latencies, which may due to hardware contention on the single HCA processor. On the contrary, when the message size is smaller than 128 bytes, the latencies for the multi-HCA threaded benchmark are a little higher ( $\sim 0.5\mu s$ ) than the single-rail latency. When the message size is larger than 128 bytes, the latencies of the

multi-HCA threaded benchmark are less than the latency of the single-rail case. The threaded benchmark with the multi-HCA configuration achieved  $2.7\mu s$  for a 4KB message, which is  $\sim 0.62\%$  of the single-rail latency.

The non-threaded benchmark shows significant higher latencies ( $>10\mu s$ ) for both the multi-port and the multi-HCA configurations. This is due to the overhead brought by serializing RDMA function calls, and event waiting. Hardware contention affects are more marked for the non-threaded approach.

In terms of performance improvement, the threaded approach starts showing distinct improvement when the data size larger than 256 bytes on the multi-HCA configuration and 1KB on the multi-port configuration. The non-threaded approach does not show any improvement on the small messages with both configurations.

### 5.3. Uni-directional Bandwidth

Figures 7 and 8 show the uni-directional bandwidth comparison between the non-threaded, the threaded and the single-rail bandwidth on the multi-port and the multi-HCA configurations. The bandwidth results are measured from 8 bytes to 2MB. On the single-rail InfiniBand network, the uni-directional bandwidth peaks at  $\sim 1350\text{MB/s}$ .

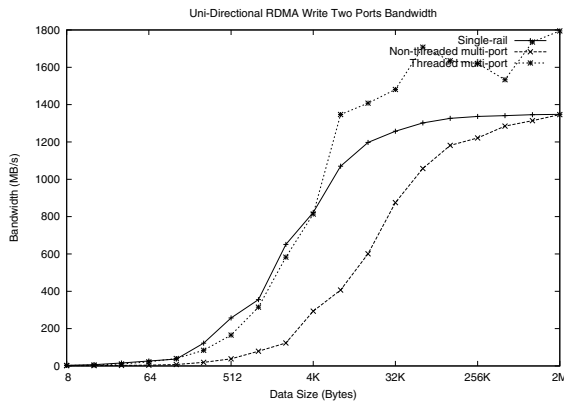


Figure 7. Uni-directional multi-port bandwidth.

As the theoretical bandwidth of the multi-port configuration is  $2.0\text{GB/s}$ , which is the same as the single-rail bandwidth limit. The non-threaded multirail benchmark shows around the same peak bandwidth on the multi-port configured InfiniBand network as on the single-rail, see figure 7. However, when the data size is less than 2MB, the non-threaded multirail benchmark shows lower bandwidth than the single-rail. This is due to hardware and software contention with a single thread to service two different network interface communications.

On the contrary, the threaded benchmark with the multi-port configuration starts achieving the bandwidth improvement when message size is larger than 4KB, and shows the

peak bandwidth at  $\sim 1800\text{MB/s}$  between 32KB and 2MB. An  $\sim 33\%$  improvement on bandwidth is achieved.

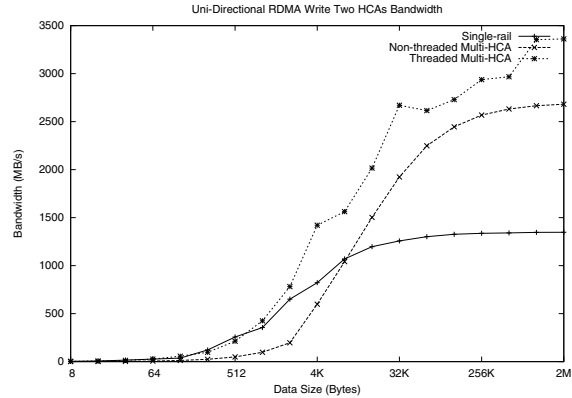


Figure 8. Uni-directional multi-HCA bandwidth.

In figure 8, on the multi-HCA configured InfiniBand cluster, the non-threaded approach starts achieving a bandwidth improvement from 8KB, and the threaded approach starts achieving bandwidth improvement from 512 bytes. Moreover, the non-threaded approach reaches the peak uni-directional bandwidth at  $\sim 2600\text{MB/s}$  and the threaded approach reaches the peak uni-directional bandwidth at  $\sim 3350\text{MB/s}$ . The non-threaded approach improves the uni-directional bandwidth by  $\sim 90\%$ , and surprisingly an improvement of  $\sim 148\%$  is achieved with the threaded approach.

By comparing two different multirail configurations, the non-threaded approach achieves  $\sim 90\%$  higher bandwidth on the multi-HCA network than on the multi-port network, and the threaded approach achieves  $\sim 89\%$  higher bandwidth on the multi-HCA network than on the multi-port network.

### 5.4. Bi-directional Bandwidth

Figures 9 and 10 show the bi-directional bandwidth comparison between the non-threaded and the threaded approach on the multi-port and the multi-HCA configurations. Similar to the uni-directional bandwidth tests, the bandwidth is measured between 8 bytes and 2MB data sizes. The bi-directional bandwidth on single-rail InfiniBand cluster reaches the peak at  $\sim 2350\text{MB/s}$  with 2MB data size.

A similar trend has been observed in figure 9 for the bi-directional multi-port bandwidth as the uni-directional multi-port bandwidth results shown in figure 7. The non-threaded approach does not show any improvement; however, the threaded approach shows around 34% improvement for the peak bi-directional bandwidth at  $3150\text{MB/s}$  reached at 2MB data size.

In figure 10, both the non-threaded and the threaded approaches achieve bandwidth improvement on the multi-HCA configuration when the data size is larger than 32KB

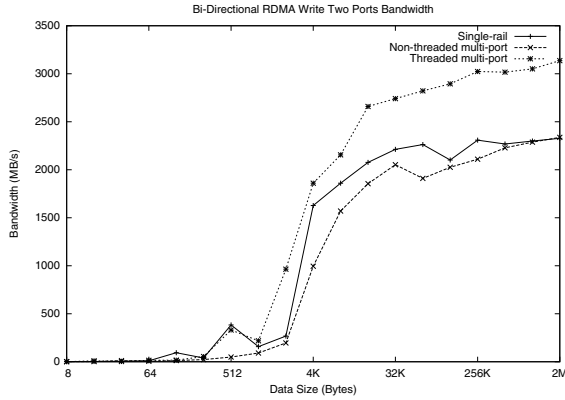


Figure 9. Bi-directional multi-port bandwidth.

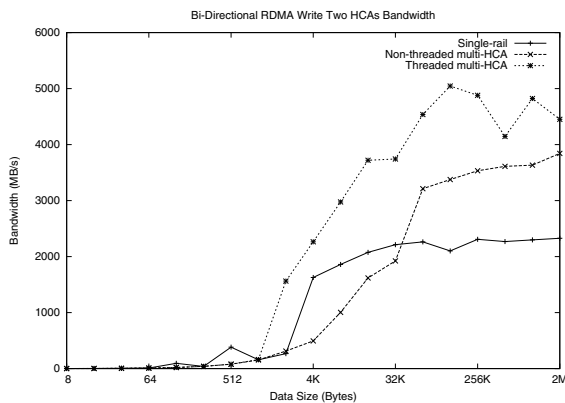


Figure 10. Bi-directional multi-HCA bandwidth.

and 1KB respectively. The non-threaded approach achieved the peak bi-directional bandwidth at  $\sim 3950\text{MB/s}$  and  $\sim 68\%$  improvement when the data size is larger than 32KB. On the contrast, the threaded approach achieved the peak bi-directional bandwidth at  $\sim 5045\text{MB/s}$  and an improvement  $\sim 114\%$  at a data size of 128KB. Then the bandwidth decreased to  $4450\text{MB/s}$  for 2MB message with some fluctuation.

According to figures 9 and 10, around 68% more bi-directional bandwidth was obtained on the multi-HCA network by the non-threaded approach compared to the multi-port network, and around 60% more bi-directional bandwidth was achieved by the threaded approach for the multi-HCA configuration as well.

### 5.5. Elapsed Time Breakdown

To understand the different behaviors of the non-threaded and the threaded approach, we break the elapsed time for running the uni-directional benchmarks on the multi-HCA configured cluster down into two different parts: the time for RDMA write function return, and the time for busy waiting

on DTO completion events. Here, the elapsed time is the total time spent on transferring a fixed sized message 20 times with notifications. Two different data sizes, 512 bytes and 4KB, are used for the following demonstration.

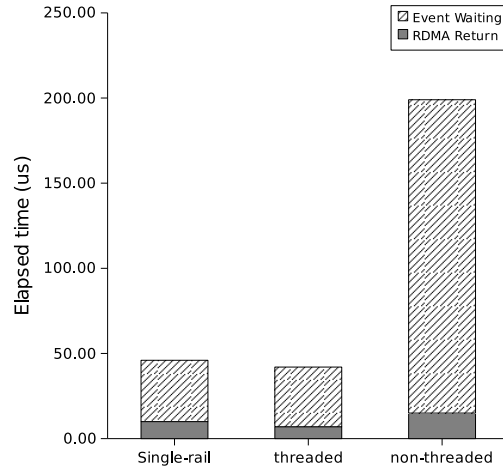


Figure 11. Benchmarks elapsed time breakdown for 512bytes message.

Figure 11 shows the breakdown for the elapsed time on transferring a 512 bytes message 20 times. The time spends on RDMA function return is  $10\mu\text{s}$  and  $7\mu\text{s}$  for the single-rail and the threaded approach respectively, which are compatible. The non-threaded approach spends  $\sim 15\mu\text{s}$  on RDMA function return, which is due to the serialized RDMA function calling procedure utilized in non-threaded benchmark design.

$36\mu\text{s}$  and  $35\mu\text{s}$  are spent on waiting event for the single-rail and the threaded approach respectively. For any message smaller than 512 bytes, the threaded approach does not show much improvement on performance, due to the overheads on joining threads and ending for parallel region. Moreover,  $184\mu\text{s}$  is spent on waiting for events for the non-threaded approach, which is around four times more than the single-rail. This is because that the event waiting on different rails is fully serialized in non-threaded design, although some portion of data transferring is overlapped.

Figure 12 shows the breakdown for the elapsed time on transferring a 4KB message 20 times. The threaded approach spends  $4\mu\text{s}$  less on waiting for RDMA return than the single-rail ( $9\mu\text{s}$ ), while the non-threaded approach spends  $14\mu\text{s}$  on waiting RDMA return. This is quite similar to the 512 bytes case.

However, for the 4KB message, the single-rail test spends  $78\mu\text{s}$  on waiting for DTO completion events. As data transfer is fully parallelized, the threaded approach spends  $45\mu\text{s}$  on event waiting. On the contrast, the non-threaded approach spends  $197\mu\text{s}$  on event waiting, which is more than twice that of the single-rail, and this ratio is much less than the

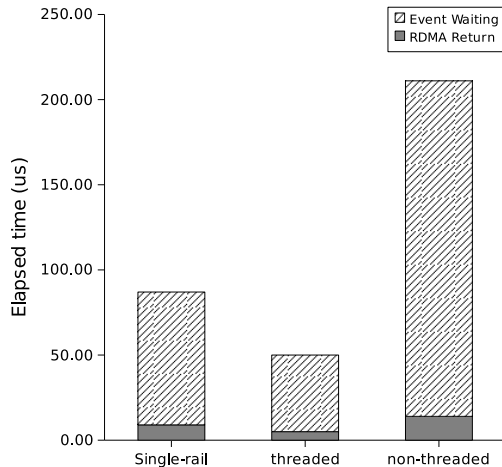


Figure 12. Benchmarks elapsed time breakdown for 4KB message.

512 bytes case. This is because the larger data size allows more communication overlap.

## 6. Conclusions and Future Work

In this paper, we have developed the non-threaded and the threaded approaches with uDAPL to explore the bandwidth improvement on multirail networks. The results of the non-threaded approach are quite similar with MVAPICH2 InfiniBand Verb multirail implementation on the multi-HCA configurations. As some different InfiniBand HCAs were used, MVAPICH2 achieved some improvement on the non-threaded approach with the multi-port configuration as well [8], [10]. However, in our setup, utilizing a portable communication library does not prevent the performance improvement over multirail networks.

In summary, an  $\sim 33\%$  and  $\sim 148\%$  improvement for the large message uni-directional bandwidth is achieved by the threaded approach on the multi-port and the multi-HCA configured cluster respectively. No improvement and 90% improvement for the large message uni-directional bandwidth is achieved by the non-threaded approach on the multi-port and the multi-HCA configured cluster respectively. A similar pattern of improvement has been achieved for the bi-directional bandwidth tests for both the threaded and the non-threaded approaches.

Since the threaded approach fully parallelizes the data transfer on the different rails, it shows significantly better improvement than the non-threaded approach. As utilizing the multiple HCAs increases the theoretical peak system bandwidth, both the threaded and the non-threaded approaches achieved the better improvement than on the multiple port configured network. Therefore, the best way to achieve the bandwidth improvement is to use the threaded approach over

a multi-HCA configured network.

In the short future, we are planning to apply both non-threaded and threaded uDAPL approaches to Cluster OpenMP and MVAPICH2.

## Acknowledgment

This work is part funded by Australian Research Council Grant LP0669726 and Intel Corporation, and was supported by the National Computing Infrastructure at the ANU.

## References

- [1] J. Lentini, V. Pham, S. Sears, and R. Smith, "Implementation and analysis of the user direct access programming library," 2003. [Online]. Available: <http://sourceforge.net/project/dapl>
- [2] MVAPICH(2) Project, 2009. [Online]. Available: <http://mvapich.cse.ohio-state.edu/>
- [3] L. Chai, R. Noronha, and D. Panda, "MPI over uDAPL: Can high performance and portability exist across architectures?" in *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, 2006, pp. 19–26.
- [4] Intel, 2009. [Online]. Available: <http://www.intel.com/>
- [5] Open MPI, 2009. [Online]. Available: <http://www.openmpi.org/>
- [6] J. P. Hoeflinger, "Extending openmp to clusters," *White Paper, Intel Corporation*, 2006.
- [7] Top500 List, 2009. [Online]. Available: <http://www.top500.org>
- [8] J. Liu, A. Vishnu, and D. K. Panda, "Building multirail InfiniBand clusters: MPI-level design and performance evaluation," in *In SuperComputing*. IEEE Computer Society, 2004, p. 33.
- [9] S. Coll, E. Frachtenberg, F. Petrini, A. Hoisie, and L. Gurvits, "Using multirail networks in high-performance clusters," in *In Proceedings of the 2001 IEEE International Conference on Cluster Computing*. IEEE Computer Society, 2001, pp. 15–24.
- [10] A. Vishnu, G. Santhanaraman, W. Huang, H.-W. Jin, and D. K. Panda, "Supporting MPI-2 one sided communication on multi-rail infiniband clusters: Design challenges and performance benefits," in *High Performance Computing - HiPC, 12th International Conference, Goa, India, December 18-21, Proceedings*, 2005, pp. 137–147.
- [11] "uDAPL: User direct access programming library," DAT Collaborative, 2008. [Online]. Available: <http://www.datcollaborative.org>
- [12] "ConnectX Adapter Card Product Brief," Mellanox, 2009. [Online]. Available: <http://www.mellanox.com/>