RDMA Programming

based on the [libibverbs](https://www.openfabrics.org/downloads/verbs/) 1.2.0

**Install libibverbs**

Download libibverbs and enter the root dir of libibverbs.

$ ./configure --prefix=/path/to/yours

Then add the path **/.../libibverbs-1.2.0/lib** to **LD\_LIBRARY\_PATH** for library linking

Call **ibv\_devinfo -v** from every machine’s terminal in order to check the current settings of ibverb.

General API introduction of libibverbs:

<http://www.rdmamojo.com/2012/05/18/libibverbs/>

<https://github.com/jcxue/RDMA-Tutorial/wiki>

**Queue Pair**

To draw an analogy from everyday mail service, queue pair (QP) defines the address of the communication endpoints, or equivalently, sockets in traditional socket based programming. Each communication endpoint needs to create a QP in order to talk to each other.

There are three different types of QP: RC (Reliable Connection), UC (Unreliable Connection) and UD (Unreliable Datagram). A more recent optimization from Mellanox introduces DCT (Dynamically Connected Transport) to solve the QP scalability problem.

If QP is connected (RC or UC), each QP can only talk to ONE other QP. Otherwise, if QP is created as UD or DCT, the QP is able to talk to ANY other QPs. A more detailed discussion on how to choose the type of QP can be found later in #Choice of QP types.

**Verbs**

In RDMA based programming, verb is a term that defines the types of communication operations. There are two different communication primitives: channel semantics (send/receive) and memory semantics (read/write). If we only consider how data is delivered to the other end, channel semantics involves both communication endpoints: the receiver needs to pre-post receives and the sender posts sends; while memory semantics only involves one side of the communication endpoint: the sender can write the data directly to the receiver's memory region, or the receiver can read from the target's memory region without notifying the target.

Generally speaking, memory semantics has less overhead compared to channel semantics and thus has higher raw performance; On the other hand, channel semantics involves less programming effort.

**Section 1: RDMA Initialization**

We start off with a simple example only involves two nodes: a server and a client. Messages are ping-ponged between the server and client, while the client initiates the sending of the messages. This simple example is used to demonstrate the common process of setting up connection using IB. (In this tutorial, we use RDMA and IB interchangeably since IB, short for Infiniband, is one of the most popular hardware implementation of RDMA technology). In this first example, we use QP with RC type.

Step 1: Get IB context struct ibv\_context

First, we need to get IB device list by calling **ibv\_get\_device\_list**; Sometimes **ibv\_get\_device\_name** will also be used to specify the IB device name, but more generally, we often use the first IB device. Then we can get IB context by calling [**ibv\_open\_device**](http://www.rdmamojo.com/2012/06/29/ibv_open_device/). Next, check the **atomicity level** **(also called atomic\_cap, see ibv\_query\_device)** for rdma operations by calling [**ibv\_query\_device**](http://www.rdmamojo.com/2012/07/13/ibv_query_device/). Finally, we are done with device list and free it by calling [**ibv\_free\_device\_list**](http://www.rdmamojo.com/2012/06/07/ibv_free_device_list/).

Step 2: Query port properties & Allocate IB protection domain struct ibv\_pd

IB port attribute can be obtained by calling [**ibv\_query\_port**](http://www.rdmamojo.com/2012/07/21/ibv_query_port/) and it returns the attributes of a port of an RDMA device context. The field lid from port attribute will later be used to connect QPs. Protection domain will later be used to register IB memory region and create Queue Pairs. Protection domain is allocated by calling: [**ibv\_alloc\_pd**](http://www.rdmamojo.com/2012/08/24/ibv_alloc_pd/).

Step 3: Register IB memory region struct ibv\_mr ***(IMPORTANT)***

IB memory region is used to store messages exchanged among nodes. Older version of hardware requires these memory regions to be pinned (thus, cannot be swapped out by OS); however, newer hardware that supports on demand paging relaxes such restriction. Memory region is managed by the programmer. The programmer needs to make sure the old message that has not been processed, does not get overwritten by the newly incoming message. Memory region is registered by calling [**ibv\_reg\_mr**](http://www.rdmamojo.com/2012/09/07/ibv_reg_mr/). While the argument **ibv\_access\_flags (see ibv\_reg\_mr)** controls the desired memory access attributes by the RDMA device. It is either 0 or the bitwise OR of one or more of the following flags: IBV\_ACCESS\_LOCAL\_WRITE, IBV\_ACCESS\_REMOTE\_WRITE, IBV\_ACCESS\_REMOTE\_READ, IBV\_ACCESS\_REMOTE\_ATOMIC.

Step 4: Create Completion Queue struct ibv\_cq

Completion Queue (CQ) is commonly used along with channel semantics (send/receive). It provides a way to notify the programmer that the operation is complete. Completion queue can be created using [**ibv\_create\_cq**](http://www.rdmamojo.com/2012/11/03/ibv_create_cq/)

Step 5: initialize the attributes of Queue Pair

In particular, we need to specify the **qp\_type (ibv\_qp\_init\_attr.** **qp\_type, see ibv\_create\_qp in the following step)** during this step. Three Transport Service Types of QP are provided, IBV\_QPT\_RC, IBV\_QPT\_UC and IBV\_QPT\_UD respectively.

Step 6: Create Queue Pair struct ibv\_qp

Queue Pair (QP) can be created by calling [**ibv\_create\_qp**](http://www.rdmamojo.com/2012/12/21/ibv_create_qp/)

Step 7: Connect QP ***(IMPORTANT)***

Now QPs are created, they need to know whom they are talking to. IB needs to use out-of-band communication to connect QP. Commonly, socket and zmq are often used to exchange QP related information (i.e. addr, rkey, qp\_num, lid ...). In the case of channel semantics (send/recv), only two pieces of information need to be exchanged in order to connect QP: lid from struct ibv\_port\_attr and qp\_num from struct ibv\_qp. **After lid and qp\_num has been received, qp\_state can be changed from *INIT* to *RTR* (ready to receive) and then to *RTS* (ready to send)** by calling[**ibv\_modify\_qp**](http://www.rdmamojo.com/2013/01/12/ibv_modify_qp/)

***Now we suppose there are 2 different processes between 2 different machines (with 2 unique infiniband NICs) named “reader”& “writer”respectively.***

**Section 2: RDMA Send/Recv**

Step1: reader pre-posts a receiving opeation ([**ibv\_post\_recv**](http://www.rdmamojo.com/2013/02/02/ibv_post_recv/))

Step2: after reader pre-posts the receiving, writer calls rdma\_send ([**ibv\_post\_send**](http://www.rdmamojo.com/2013/01/26/ibv_post_send/)**, wr.opcode= IBV\_WR\_SEND**) which should be followed by a poll\_completion ([**ibv\_poll\_cq**](http://www.rdmamojo.com/2013/02/15/ibv_poll_cq/))

Step3: reader calls poll\_completion ([**ibv\_poll\_cq**](http://www.rdmamojo.com/2013/02/15/ibv_poll_cq/)) and the new coming message will be stored at the beginning address of the reader’s rdma buffer if reader’s poll\_completion executes successfully.

**Section 3: RDMA Write**

Step1: copy the new message to the beginning address of rdma buffer at write’s side.

Step2: writer calls rdma\_write ([**ibv\_post\_send**](http://www.rdmamojo.com/2013/01/26/ibv_post_send/)**, wr.opcode= IBV\_WR\_RDMA\_WRITE**) which should be followed by a poll\_completion ([**ibv\_poll\_cq**](http://www.rdmamojo.com/2013/02/15/ibv_poll_cq/))

Step3: reader calls busy\_read until the incoming rdma message is written completely.

**Section 4: RDMA Read**

Step1: copy the new message to the beginning address of rdma buffer at write’s side.

Step2: reader calls rdma\_read ([**ibv\_post\_send**](http://www.rdmamojo.com/2013/01/26/ibv_post_send/)**, wr.opcode= IBV\_WR\_RDMA\_READ**) which should be followed by a poll\_completion ([**ibv\_poll\_cq**](http://www.rdmamojo.com/2013/02/15/ibv_poll_cq/)) and then the incoming rdma message will be placed at the beginning address of rdma buffer at reader’s side.

**Section 5: Release RDMA resource**

Some IBV calling should be invoked here in order to release the RDMA resource, including:

**ibv\_dereg\_mr**

**ibv\_dealloc\_pd**

**ibv\_close\_device**

**free(the buffer of rdma)**

**ibv\_destroy\_qp**

**ibv\_destroy\_cq**

**(ibv\_destroy\_srq, if SRQ (shared receive queue) is created)**