# Report

## 1 architecture & usage

### Intro.

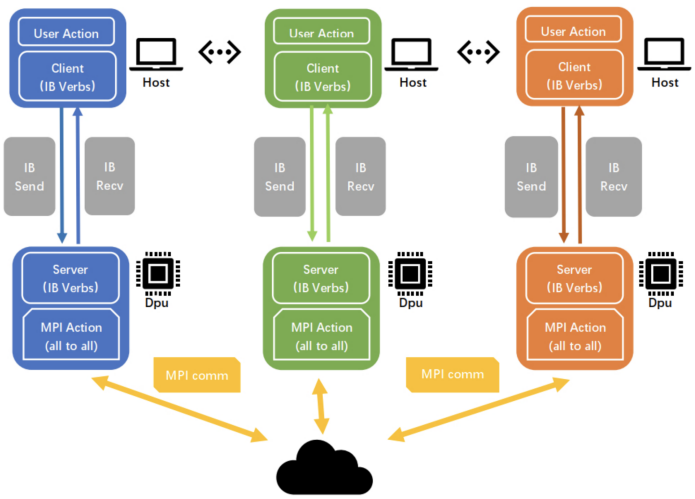
We provide a framework to do MPI_Ialltoall & MPI_Alltoall. The user only needs to include some head files to run their application. The Interfaces of our framework are almost the same as MPI provided.  We list all the interfaces we have been implemented below:

```
int my_mpi_init();                                          // init operation
int my_mpi_clean();                                        // clean opertaion
int my_mpi_test(uint32_t task_id,int * flag);                        // check c
int my_mpi_test_for_ready(uint32_t ** task_ids,uint32_t *read_lenth);
                                                          // check  tasks ready
int my_mpi_ialltoall(const void *sendbuf, int sendcount,     //  alltoall (u
            MPI_Datatype sendtype, void *recvbuf, int recvcount,
            MPI_Datatype recvtype, uint32_t task_id,uint32_t send_lenth);
int my_mpi_alltoall(const void *sendbuf, int sendcount,         // alltoall
            MPI_Datatype sendtype, void *recvbuf, int recvcount,
            MPI_Datatype recvtype, uint32_t task_id,uint32_t send_lenth);
```

We put the main effort into alltoall operations and support all kinds of MPI_Datatype. Instead of using MPI_request, we use a uin32 to represent the task_id for simplicity.

### usage

The design architecture is showed in the below figure.  All processes from CPU hosts are created by using mpi_run. They are logically connected. Before all these processes started, we need to run server processes on DPU hosts. Each server process on DPU servers one client process on CPU.



At here we use a demo to show how to use our framework. We start four server processes on two DPU hosts use the command:

**mpirun –n 4 –-hostfile host_file –x LD_LIBRARY_PATH –x PATH server**

we already provide the Executable file server on our code packet. By default, we do not need to change the code in it (Except the user wants to change some parameters). At the client end, the user needs to include our head files and place all other head/sources files in a right place to use our API.

Here is the simple code to demonstrate how to use the ”my_mpi_alltoall” function.

```
// init the our my_mpi envirment
my_mpi_init();
/*
create data, in this example.
process 1:  0 1 2  3  4  5  6  7
process 2:  8 9 10 11 12 13 14 15
process 3:  16 17 18 19 20 21 22 23
process 4:  24 25 26 27 28 29 30 31
*/
int my_values[8];
for(int i = 0; i < 8; i++)
{
    my_values[i] = my_rank * 8 + i;
}

// create a recv buffer
int my_values2[8];
memset(my_values2,0 ,8 * sizeof(int));

// alltoall until it finish the work
if(my_mpi_alltoall(my_values,2,MPI_INT,my_values2,2,MPI_INT,1,32)
{
    printf("some thing badly,happended\n");
}
/*
data becomes:
process 1:  0 1 8 9 16 17 24 25
process 2:  2 3 10 11 18 19 26 27
process 3:  4 5 12 13 20 21 28 29
process 4:  6 7 14 15 22 23 30 31
*/

//my_mpi_clean
my_mpi_clean();
```

We also use mpirun to start four client processes on CPU hosts.

**mpirun –n 4 –-hostfile host_file ./client**

This is only a simple example to show the basic usage. More details about other APIs can be found in our sample code.

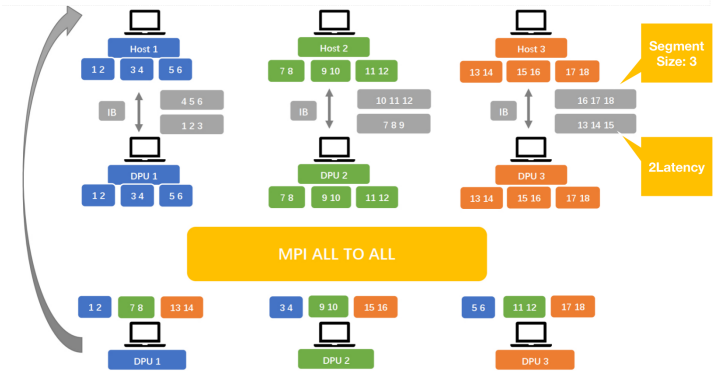
# 2 Merits

## use inflight tasks limitation to avoid memory burden on DPU

Like MPI, we also provide 'my_mpi_ialltoall' to do the task in an unblocking way. Considering the limited memory on DPU hosts, we will limit the inflight tasks user can send. When the number of outstanding tasks exceeds a threshold, we will notify the user by the return value of 'my_mpi_ialltoall'. Of course, the threshold is configurable, there are other parameters that we can tune to gain better performance. We can find in the head file 'server_rc.h' and 'client_rc.h' .

## segment shuffle

We support large messages for alltoall operations without worrying about the memory limitation in DPU.




The figure above shows how we deal with a large message. The main idea behind this is that if the message is too big, we split it into small messages. Then we send the small messages to the DPU host. But, when we implement our design, we can not naively cut the big message into several parts, we need to reorder the data as shown in the figure.

More detail can be found in "my_mpi_op.c". Of course, users do not need to worry about their message is too big. We did all these checks and split stuff for them. There is a reference code example in our code packets.

## Connecting API

We use IB verbs to create and maintain the communication between DPU hosts and CPU hosts. To make our framework easier to develop, we privide another framework used for fast network connection. Our my_mpi_xxx API is developed based on the connection APIs. We show the interfaces below:

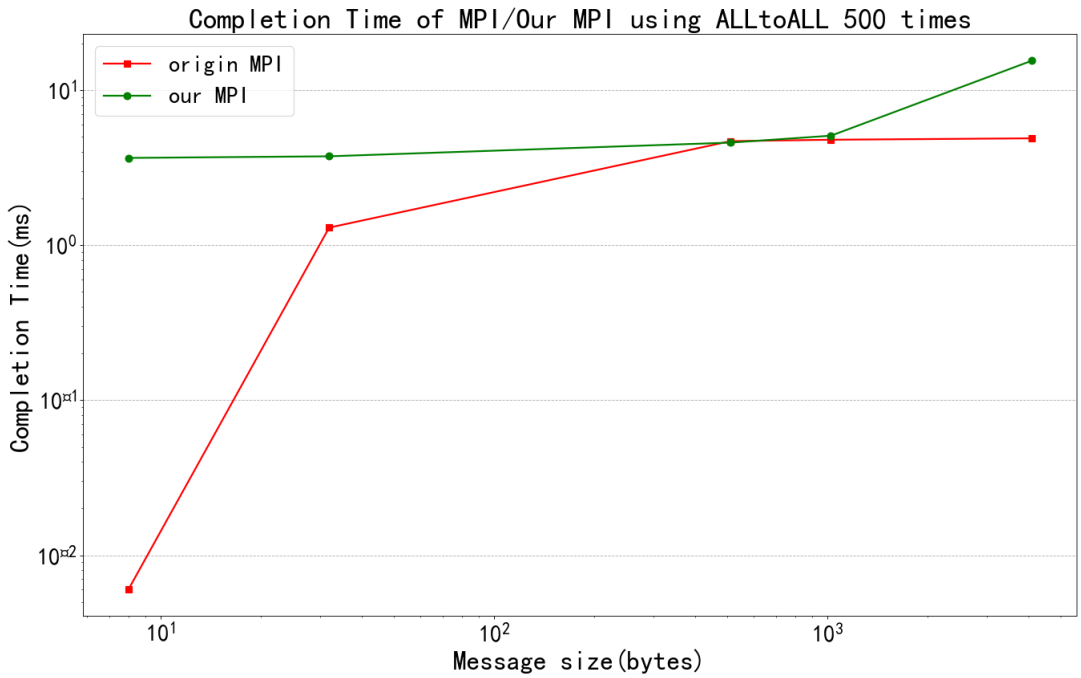
```
int dpu_client_init(char * ip,unsigned int port);
int dpu_client_recv(char **message, uint32_t * get_lenth);
int dpu_client_send(char *message, uint32_t send_size);
int dpu_client_send_free(char *message, uint32_t send_size);
int dpu_client_try_recv(char ** message, uint32_t * get_lenth);
int dpu_client_clean();
int dpu_server_init(unsigned int port);
```

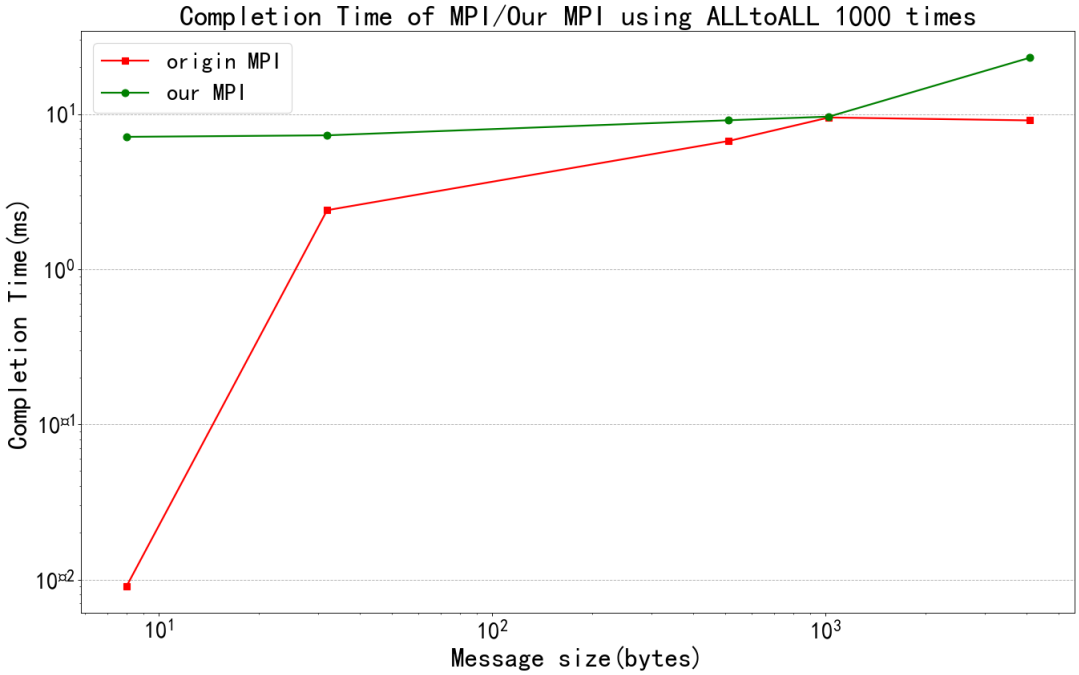
```
int dpu_server_recv(char **message, uint32_t * get_lenth);
int dpu_server_try_recv(char **message, uint32_t * get_lenth);
int dpu_server_send(char *message, uint32_t send_size);
int dpu_server_send_free(char *message, uint32_t send_size);
int dpu_server_clean();
void* receive_array[MAX_RECEIVE_WR];
```

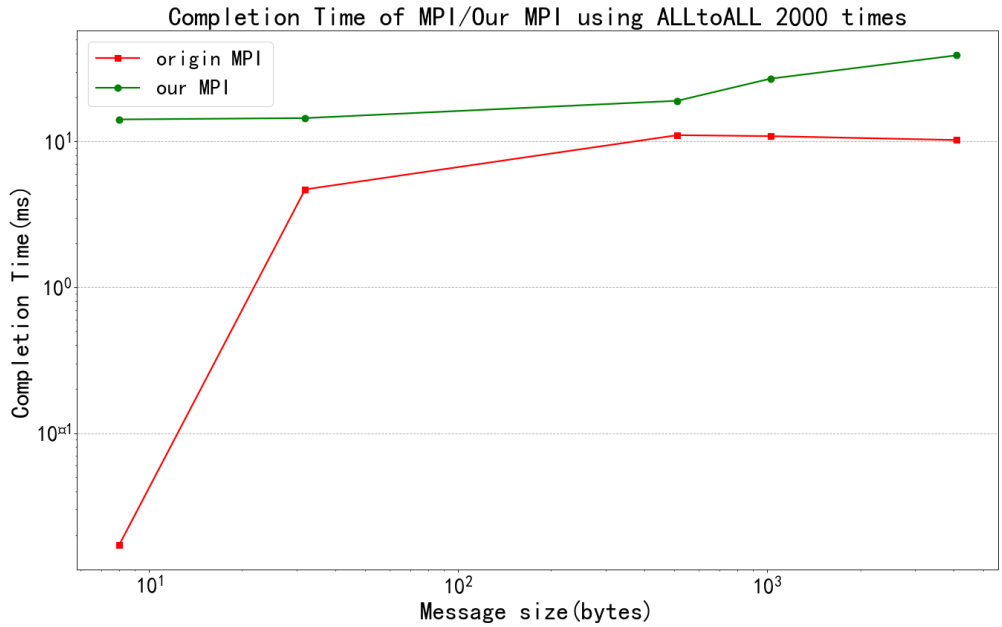
Actually, it takes most of our time to finish this connection framework. It is easy to use as TCP but as efficient as Verbs. Also, users do not need to worry about memory registration, etc, we manage all the trivial things by our seld. We argue that with this connection API, we can offload almost all the MPI_XX operations in DPU hosts easily.

# 3 Experiments

We conduct an experiment on testing the finish time of the procedure between our design and MPI_ialltoall. We set different numbers of iterations for both us and the origin MPI. And, each figure shows the program finish time with different task sizes. As we can see, the finish time between these two different methods is close. Of course, we have a lower performance due to the propagation of our data. BUT!!!!!!!!! we free up the CPU from doing the all-to-all operation.




Completion Time of MPI/Our MPI using ALLtoALL 1000 times
origin MPI
our MPI
Completion Time(ms)
Message size(bytes)


Completion Time of MPI/Our MPI using ALLtoALL 2000 times
origin MPI
our MPI
Completion Time(ms)
Message size(bytes)