GSoC 2018 Ste||ar Group - Blog 1

Project:

All-to-All Communications

Student:

Ahmed Samir

Mentors:

John Biddiscombe and Thomas Heller

Main Goal:

Getting a simple test running with FFLIB library model and working on libfabric. This would be like a base for the libfabric backend of FFLIB. After that the next goal would be to start from this small base to get an implementation of FFLIB collective operations on libfabric backend.

Here is a summary of the last 3 weeks:

There was a simple libfabric ping-pong example developed by OFIWG. So my first part was to get this example working with FFLIB. It was a simple code passing messages like ping- pong between 2 nodes. The example worked on sockets so I could run it successfully on my laptop.

First I worked on dissecting the code into separate parts. Each part was doing a separate job so that I can understand it. It was divided into 3 sections:

The first section was the configuration part. It was about taking arguments from the user to configure how the test should be working. One of the parameters was the connectivity type between the 2 nodes. The user can decide whether it will be connectionless or connection oriented test. Another parameter was the transfer size i.e, How much data would be transferred between the 2 nodes. Another parameter was the number of ping-pong iterations. There were many parameters which the user can use to tweak the test.

The second section was about initializing the test. For example, initializing the connection between the client and the server in the case of connection oriented test. Also allocating the needed resources for the test such as: fabric, domain, endpoints, memory regions, completion queues and other resources that will be used by each node.

The last section was the actual send/receive part. It initializes the data to be sent/received every iteration and posts the data transfer operation.

I understood the implementation of every section and then used the APIs in this test to build my own test with FFLIB. I forked the FFLIB repo and implemented a libfabric FFLIB component which is linked with the libfabric APIs used in the pingpong test. Then I built a send/receive test in FFLIB that worked with this component. There were problems with compiling and linking the files because of some includes that were used in the pingpong file but weren't put in the include directory of libfabric.

At that time, John sent me another libfabric connection test that was implemented by Thomas Heller. It was doing the connection between 2 nodes on sockets and needed only some additional tweaks so that it can be used in a ping-pong test. So I worked on building and understanding this test while John was working on the compilation and linking errors in the other test.

Thomas's test code was very understandable and it was more organized so It didn't take me much time to understand it. I used it to build another test for FFLIB with libfarbic. There were some linking problems because Thomas's test used CPP and FFLIB is a C library but with the help of John everything went well.

Finally I managed to build a FFLIB test working on libfabric. After the build process, there were some logical errors in my code. It took me a lot of time to trace the errors but I managed to fix them.

The modifications I’ve added in my FFLIB fork:

- A libfabric binding file and a libfabric component. The main component files are:

  - The connect_libfabric file. It contains all the implementations of th libfabric calls that are called by the libfabric component. It’s the biggest file because it has all the libfabric logic.
  - The fflibfabric file. It contains the initialization and the finalization APIs.
  - The recv and send files. They contain the posting of the send and receive operations through libfabric.
  - The libfabric progresser file. It contains the methods used by the progresser to track the operation completion. The ffop_libfabric_progresser_progress method is where the receive and send completions queues are polled.

- The send_recv_libfabric test file.

The test goes like this:

- Initializing FFLIB and binding the FFLIB APIs with libfabric backend to be used.
- Creating a progresser thread that tracks the completions of the operations.
- Initializing the libfabric connection between the 2 nodes. Every node allocates the needed resources i.e, fabric, domain, memory region, endpoints, etc… and then establishes a connection with the other node through sockets.
- The sender puts the data into a buffer from the memory region that it has allocated with the endpoint. Then the send operation is posted in libfabric and scheduled in FFLIB.
- The receiver gets a buffer from the memory region allocated with the endpoint and posts a receive operation to reserve the buffer for receiving incoming data. The receive operation is also scheduled in FFLIB.
- Then each node polls for the completion of the operation. The sender polls the send completion queue while the receiver polls for the receive completion queue.
- When the send operation is completed, the sender’s role is now finished. On the other hand, the receiver waits for the receive completion. Then after it receives the message, the message is checked to see if the message content is what is sent by the sender or not.
- Finally the 2 nodes free the resources that they have allocated and exit.

Conclusion:

Now we have a FFLIB test working on libfabric. It's a simple test but it's a basic start for more complex implementations coming. In the next weeks, I will be working on implementing more complex tests invloving communication between multiple nodes and collective operations.