Programming with OpenFabrics Verbs and LibFabrics

Follow us on

https://twitter.com/mvapich

Dhabaleswar K. (DK) Panda
The Ohio State University
E-mail: panda@cse.ohio-state.edu
http://www.cse.ohio-state.edu/~panda

Hari Subramoni
The Ohio State University
E-mail: subramon@cse.ohio-state.edu
http://www.cse.ohio-state.edu/~subramon
Software Convergence with OpenFabrics

- Open source organization (formerly OpenIB)
  - www.openfabrics.org

- Incorporates both IB, RoCE, and iWARP in a unified manner
  - Support for Linux and Windows

- Users can download the entire stack and run
  - Latest stable release is OFED 4.17-1
    - New naming convention to get aligned with Linux Kernel Development
    - OFED 5.3 is under development
Programming with OpenFabrics

Sample Steps

1. Create QPs (endpoints)
2. Register memory for sending and receiving
3. Send
   – Channel semantics
     • Post receive
     • Post send
   – RDMA semantics
Verbs: Post Send

- Prepare and post send descriptor (channel semantics)

```c
struct ibv_send_wr *bad_wr;
struct ibv_send_wr sr;
struct ibv_sge sg_entry;

sr.next = NULL;
sr.opcode = IBV_WR_SEND;
sr.wr_id = 0;
sr.num_sge = 1;
if (len < max_inline_size) {
    sr.send_flags = IBV_SEND_SIGNALED | IBV_SEND_INLINE;
} else {
    sr.send_flags = IBV_SEND_SIGNALED;
}
sr.sg_list = &(sg_entry);
sg_entry.addr = (uintptr_t) buf;
sg_entry.length = len;
sg_entry.lkey = mr_handle->lkey;

ret = ibv_post_send(qp, &sr, &bad_wr);
```
Verbs: Post RDMA Write

- Prepare and post RDMA write (memory semantics)

```c
struct ibv_send_wr *bad_wr; struct ibv_send_wr sr;
struct ibv_sge sg_entry;

sr.next = NULL;
sr.opcode = IBV_WR_RDMA_WRITE; /* set type to RDMA Write */
sr.wr_id = 0;
sr.num_sge = 1;
sr.send_flags = IBV_SEND_SIGNALED;
sr.wr.rdma.remote_addr = remote_addr; /* remote virtual addr. */
sr.wr.rdma.rkey = rkey; /* from remote node */
sr.sg_list = &sg_entry;
sg_entry.addr = buf; /* local buffer */
sg_entry.length = len;
sg_entry.lkey = mr_handle->lkey;

ret = ibv_post_send(qp, &sr, &bad_wr);
```
Libfabrics Connection Model

Server Process
- fi_fabrics: Open fabrics
- fi_passive_ep: Open Passive EP
- fi_eq_open: Open Event Q
- fi_pep_bind: Bind Passive EP
- fi_listen: Listen for Incoming Connections
- fi_eq_sread: New Event Detected on EQ
- Validate New Event == FI_CONNREQ

OFI Provider
- Sockets/Verbs/PSM
- GigE/IB/TrueScale

HCA
- GigE/IB/TrueScale

Client Process
- fi_fabrics: Open fabrics
- fi_domain: Open domain
- fi_eq_open: Open Event Q
- fi_ep_open: Register Mem
- fi_mr_reg: Open Endpoint
- fi_endpoint: Open Comp Q
- fi_cq_open: Bind EP to CQ
- fi_ep_bind: fi_connect
- Connect to Remote EP
- fi_connect: fi_fabrics
Libfabrics Connection Model (Cont.)

Server Process  OFI Provider  HCA  OFI Provider  Client Process

fi_ep_open  Open EndPoint  Sockets/Verbs/PSM  GigE/IB/TrueScale  Sockets/Verbs/PSM  Open fabrics
fi_cq_open  Open Event Q
fi_ep_bind  Bind EP to CQ
fi_accept  Accept Connection
fi_eq_sread  Validate New Event == FI_CONNECTED
fi_recv  Post Recv
fi_cq_read / fi_cq_sread  Poll / Wait for Data
fi_send  Post Send
fi_shutdown  Shutdown Channel
fi_close *  Close all open resources

New Event Detected on EQ
Validate New Event == FI_CONNECTED
Post Send
Poll / Wait for Data
Recv Completion
Shutdown Channel
Close all open resources
Post Recv
Scalable EndPoints Vs Shared TX/RX Context

- Similar to socket / QP
- Simple / Easy to use
- Share HW resources
- # EP >> HW resources
- Use more HW resources
- Higher performance per EP

Libfabrics: Fabric, Domain and Endpoint creation

• Open Fabric, Domain and EP

```c
struct fi_info *info, *hints;
struct fid_fabric *fabric;
struct fid_domain *dom;
struct fid_ep *ep;

hints = fi_allocinfo();
/* Obtain fabric information */
rc = fi_getinfo(VERSION, node, service, flags, hints, &info);
/* Free fabric information */
fi_freeinfo(hints);
/* Open fabric */
rc = fi_fabric(info->fabric_attr, &fabric, NULL);
/* Open domain */
rc = fi_domain(fabric, entry.info, &dom, NULL);
/* Open End point */
rc = fi_endpoint(dom, entry.info, &ep, NULL);
```
Libfabrics: Memory Registration

• Open Fabric / Domain and create EQ, EP to end nodes
  – Connection establishment is abstracted out using connection management APIs (fi_cm) – fi_listen, fi_connect, fi_accept
  – Fabric provider can implement them with connection managers (rdma_cm or ibcm) or directly through verbs with out-of-band communication

• Register memory

```c
int fi_mr_reg(struct fid_domain *domain, const void *buf, size_t len, uint64_t access,
             uint64_t offset, uint64_t requested_key, uint64_t flags, struct fid_mr **mr, void *
             context);

rc = fi_mr_reg(domain, buffer, size, FI_SEND | FI_RECV,
               0, 0, 0, &mr, NULL);
rc = fi_mr_reg(domain, buffer, size,
               FI_REMOTE_READ | FI_REMOTE_WRITE, 0,
               user_key, 0, &mr, NULL);
```

Permissions can be set as needed
Libfabrics: Post Receive (Channel Semantics)

• Prepare and post receive request

```c
ssize_t fi_recv(struct fid_ep *ep, void *buf, size_t len,
                void *desc, fi_addr_t src_addr,
                void *context);

- For connected EPs
size_t fi_recvmsg(struct fid_ep *ep,
                   const struct fi_msg *msg, uint64_t flags);

- For connected and un-connected EPs

struct fid_ep    *ep;
struct fid_mr   *mr;

/* Post recv request */
rc = fi_recv(ep, buf, size, fi_mr_desc(mr), 0,
             (void *)(uintptr_t)RECV_WCID);
```
Libfabrics: Post Send (Channel Semantics)

- Prepare and post send descriptor

```c
ssize_t fi_send(struct fid_ep *ep, void *buf, size_t len,
    void *desc, fi_addr_t dest_addr, void *context);

- For connected EPs

ssize_t fi_sendmsg(struct fid_ep *ep, const struct fi_msg *msg,
    uint64_t flags);

- For connected and un-connected EPs

ssize_t fi_inject(struct fid_ep *ep, void *buf, size_t len,
    fi_addr_t dest_addr);

- Buffer available for re-use as soon as function returns
- No completion event generated for send

struct fid_ep       *ep;
struct fid_mr       *mr;
static fi_addr_t   remote_fi_addr;

rc = fi_send(ep, buf, size, fi_mr_desc(mr), 0,
    (void *)(uintptr_t)SEND_WCID);
rc = fi_inject(ep, buf, size, remote_fi_addr);
```
Libfabrics: Post Remote Read (Memory Semantics)

- Prepare and post receive request

```c
ssize_t fi_read(struct fid_ep *ep, void *buf, size_t len,
                 void *desc, fi_addr_t src_addr, uint64_t addr,
                 uint64_t key, void *context);
- For connected EPs

ssize_t fi_readmsg(struct fid_ep *ep,
                    const struct fi_msg_rma *msg,
                    uint64_t flags);
- For connected and un-connected EPs

struct fid_ep       *ep;
struct fid_mr       *mr;
struct fi_context   fi_ctx_read;
/* Post remote read request */
ret = fi_read(ep, buf, size, fi_mr_desc(mr), local_addr,
             remote_addr, remote_key, &fi_ctx_read);
```
Libfabrics: Post Remote Write (Memory Semantics)

- Prepare and post send descriptor

```c
ssize_t fi_write(struct fid_ep *ep, const void *buf, size_t len,
                  void *desc, fi_addr_t dest_addr, uint64_t addr,
                  uint64_t key, void *context);
- For connected EPs

ssize_t fi_writemsg(struct fid_ep *ep,
                     const struct fi_msg_rma *msg, uint64_t flags);
- For connected and un-connected EPs

ssize_t fi_inject_write(struct fid_ep *ep, const void *buf,
                         size_t len, fi_addr_t dest_addr,
                         uint64_t addr, uint64_t key);
- Buffer available for re-use as soon as function returns
- No completion event generated for send

ssize_t fi_writedata(struct fid_ep *ep, const void *buf,
                      size_t len, void *desc, uint64_t data,
                      fi_addr_t dest_addr, uint64_t addr,
                      uint64_t key, void *context);
- Similar to fi_write
- Allows for the sending of remote CQ data
```
Thank You!

panda@cse.ohio-state.edu; subramon@cse.ohio-state.edu

Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

The High-Performance MPI/PGAS Project
http://mvapich.cse.ohio-state.edu/

The High-Performance Deep Learning Project
http://hidl.cse.ohio-state.edu/