OPENFABRICS ALLIANCE

10TH ANNUAL INTERNATIONAL DEVELOPER WORKSHOP

Addressing Application Concerns with OFI Verbs

Application Concerns

- Optimized data path
- Scalability
- Enhanced memory management
- Additional functionality

Optimized Data Path

- Requirements
  - Dedicated functions for initiating IO
  - Dedicated functions for polling completions

- Proposed approach
  - Query fast path interfaces
    - All versioning and capability checks done here
  - Use interfaces using direct vendor calls
  - Release interface when done

RDMA Operations

```
struct ibv_qp_ops_rdma {
#define IBV_INTF_QP_OPS_RDMA_V1          ...
      uint64_t interface;
      int read(struct ibv_qp *qp, uint64_t addr, uint32_t length, uint32_t lkey,
               uint64_t remote_addr, uint32_t rkey, uint64_t wr_id);
      int write(struct ibv_qp *qp, uint64_t addr, uint32_t length, uint32_t lkey,
                uint64_t remote_addr, uint32_t rkey, uint64_t wr_id);
      int write_imm(struct ibv_qp *qp, uint64_t addr, uint32_t length, uint32_t lkey,
                    uint32_t imm_data, uint64_t remote_addr, uint32_t rkey, uint64_t wr_id);
      int write_inline(struct ibv_qp *qp, uint64_t addr, uint32_t length,
                       uint64_t remote_addr, uint32_t rkey, uint64_t wr_id);
};



enum ibv_ops_flags {
      IBV_THREAD_UNSAFE = (1 << 0)
};

void *ibv_query_qp_interface(struct ibv_qp *qp, uint64_t interface, uint64_t ops_flags);
void ibv_release_qp_interface(struct ibv_qp *qp, void *ops);
```

Channel Operations

```
struct ibv_qp_ops_msg {
#define IBV_INTF_QP_OPS_MSG_V1          ...
      uint64_t interface;
      int recv(struct ibv_qp *qp, uint64_t addr, uint32_t length, uint32_t lkey, uint64_t wr_id);
      int recv_repost(struct ibv_qp *qp, int num); /* repost 'num' last completed WRs */
      int send(struct ibv_qp *qp, uint64_t addr, uint32_t length, uint32_t lkey, uint64_t wr_id);
      int send_imm(struct ibv_qp *qp, uint64_t addr, uint32_t length, uint32_t lkey,
                   uint32_t imm_data, uint64_t wr_id);
      int send_inline(struct ibv_qp *qp, uint64_t addr, uint32_t length, uint64_t wr_id);
      int sendto(struct ibv_qp *qp, uint64_t addr, uint32_t length, uint32_t lkey,
                 struct ibv_ah *ah, uint32_t remote_qpn, uint32_t remote_qkey, uint64_t wr_id);
      int sendto_imm(struct ibv_qp *qp, uint64_t addr, uint32_t length, uint32_t lkey,
                     uint32_t imm_data, struct ibv_ah *ah, uint32_t remote_qpn,
                     uint32_t remote_qkey, uint64_t wr_id);
      int sendto_inline(struct ibv_qp *qp, uint64_t addr, uint32_t length,
                        struct ibv_ah *ah, uint32_t remote_qpn, uint32_t remote_qkey,
                        uint64_t wr_id);
};
```

Completion Operations

```
struct ibv_cq_attr {
        struct {
                uint16_t cq_count;
                uint16_t cq_period;
        } moderation;
        uint64_t format_mask; /* see below */
};

enum ibv_cq_attr_mask {
        IBV_CQ_MODERATION               = (1 << 0),
        IBV_CQ_FORMAT_MASK              = (1 << 1)
};

int ibv_modify_cq(struct ibv_cq *cq,
              struct ibv_cq_attr *cq_attr,
              int cq_attr_mask);

/* Used for successful completions only; on error use poll_cq() */
struct ibv_cq_formatted_ops {
#define IBV_INTF_CQ_OPS_FORMATTED_V1          ...
      uint64_t interface;
      int poll_formatted(struct ibv_cq *cq, void *buf, size_t len);
      int poll_num();
};

void *ibv_query_cq_interface(struct ibv_qp *cq, uint64_t interface, uint64_t flags);
int ibv_release_cq_interface(struct ibv_qp *cq, void *ops);
```

Completion Operations (cont.)

```
struct ibv_wc_base {
      uint64_t          wr_id;
      uint32_t          byte_len;
       uint32_t         wc_flags;
};

struct ibv_wc_imm {
      uint32_t          imm_data;
};

struct ibv_wc_dest {
      uint32_t          qp_num;
};

struct ibv_wc_source {
      uint32_t          src_qp;
      uint16_t          slid;
};

struct ibv_wc_path {
      uint8_t           sl;
      uint8_t           dlid_path_bits;
};

struct ibv_wc_ts {
      uint64_t          timestamp;
};
```

```
enum ibv_wc_format {
      IBV_WC_BASE             = (1 << 0),
      IBV_WC_IMM              = (1 << 1),
      IBV_WC_DEST             = (1 << 2),
      IBV_WC_SOURCE           = (1 << 3),
      IBV_WC_PATH             = (1 << 4),
      IBV_WC_TS               = (1 << 5)
};
```

Application Code

```
struct context {
      struct ibv_qp *qp;
      struct ibv_qp_ops_msg *msg;

      struct ibv_cq *cq;
      struct ibv_cq_formatted_ops *formatted;
};

int create_ctx(struct context *ctx)
{
      struct ibv_cq_attr cq_attr;
      ...
      ctx->qp = ibv_create_qp(...);
      ctx->msg = ibv_query_qp_interface(ctx->qp, IBV_INTF_QP_OPS_MSG_V1, IBV_THREAD_UNSAFE);

      ctx->cq = ibv_create_cq(...);
      cq_attr.format_mask = IBV_WC_BASE | IBV_WC_TS;
      ibv_modify_cq(cq, cq_attr, IBV_CQ_FORMAT_MASK);

      ctx->formatted = ibv_query_cq_interface(ctx->cq, IBV_INTF_CQ_OPS_FORMATTED_V1,
                                              IBV_THREAD_UNSAFE);
      ...
}
```

Application Code (cont.)

```
/* Send an inline message and take timestamp without taking any locks */
int send_message(struct context *ctx)
{
      char my_msg[] = "blah";
      struct {
            struct ibv_wc_base base;
            struct ibv_wc_ts ts;
      } my_comp;
      int ret;

      ctx->msg->send_inline(ctx->qp, my_msg, sizeof my_msg, SEND_WR_ID);

      while ( !(ret = ctx->formatted->poll_formatted(ctx->cq, my_comp, sizeof(my_comp))) );
      if (ret < 0) {
            /* Poll for error and bail out */
            ...
      }
      printf("wr_id:%d timestamp:%lld\n", my_comp.base.wr_id, my_comp.ts.timestamp);
      return ret;
}
```

Scalability

- Constant memory footprint
  - Addressed by Reliable Datagram transports
    - Dynamic Connected (DC) transport
    - Reliable Datagram (RD) transport
- Asynchronous address resolution
  - Supported by RDMACM for Address Handles
  - Additional abstractions such as Address Vectors could be added

Enhanced Memory Support

- Avoiding local registrations
  - Addressed by Implicit On-Demand-Paging (ODP)
- Register arbitrary address ranges
  - Not necessarily backed by physical memory at registration time (FI_DYNAMIC_MR)
  - Addressed by Explicit ODP
- User-mode Memory Registration (UMR)
  - Allows indirect MRs
  - Asynchronous kernel-bypass operation
  - Arbitrary virtual IO address
  - Scatter/gather within a single MR or across MRs
    - Optimized for strided access
- Application-controlled R_Key
  - Add as an extended ibv_reg_mr() Verb

Additional Functionality

- Manual progress
  - Could be supported by polling CQs associated with QPs that require manual progress
- Relaxed ordering
  - Add additional QP creation flags
- Counters
  - Count completions by optimized CQ polling
  - Flow counters could be added as new objects
- Vendor-specific extensions
  - Add a query_interface() extended Verb

Thank You

OPENFABRICS ALLIANCE

10TH ANNUAL INTERNATIONAL DEVELOPER WORKSHOP