OPENFABRICS ALLIANCE

10TH ANNUAL INTERNATIONAL DEVELOPER WORKSHOP

OFV-WG: Accelerated Verbs

February 24th, 2015

Liran Liss

Agenda

- Backlog grooming for upcoming meetings
- Introduction to Accelerated Verbs
- Design approaches
- Examples
- Discussion

Next Meetings

- 2/24 – Accelerated Verbs
  - Design and approach
  - Might spill over to next week as well…
- 3/3 – Verbs Extensions overview
  - Framework
  - Backward / forward compatibility
  - Vendor specific extensions
- 3/10 – RDMA container support

Introduction

- High-end applications require extremely optimized data access APIs
  - HPC
  - Packet processing

- Goals
  - Provide the best support through Verbs
  - Focus on the fast path
    - Dedicated functions for initiating IO
    - Dedicated functions for polling completions
  - Maintain 100% functional compatibility with existing Verbs

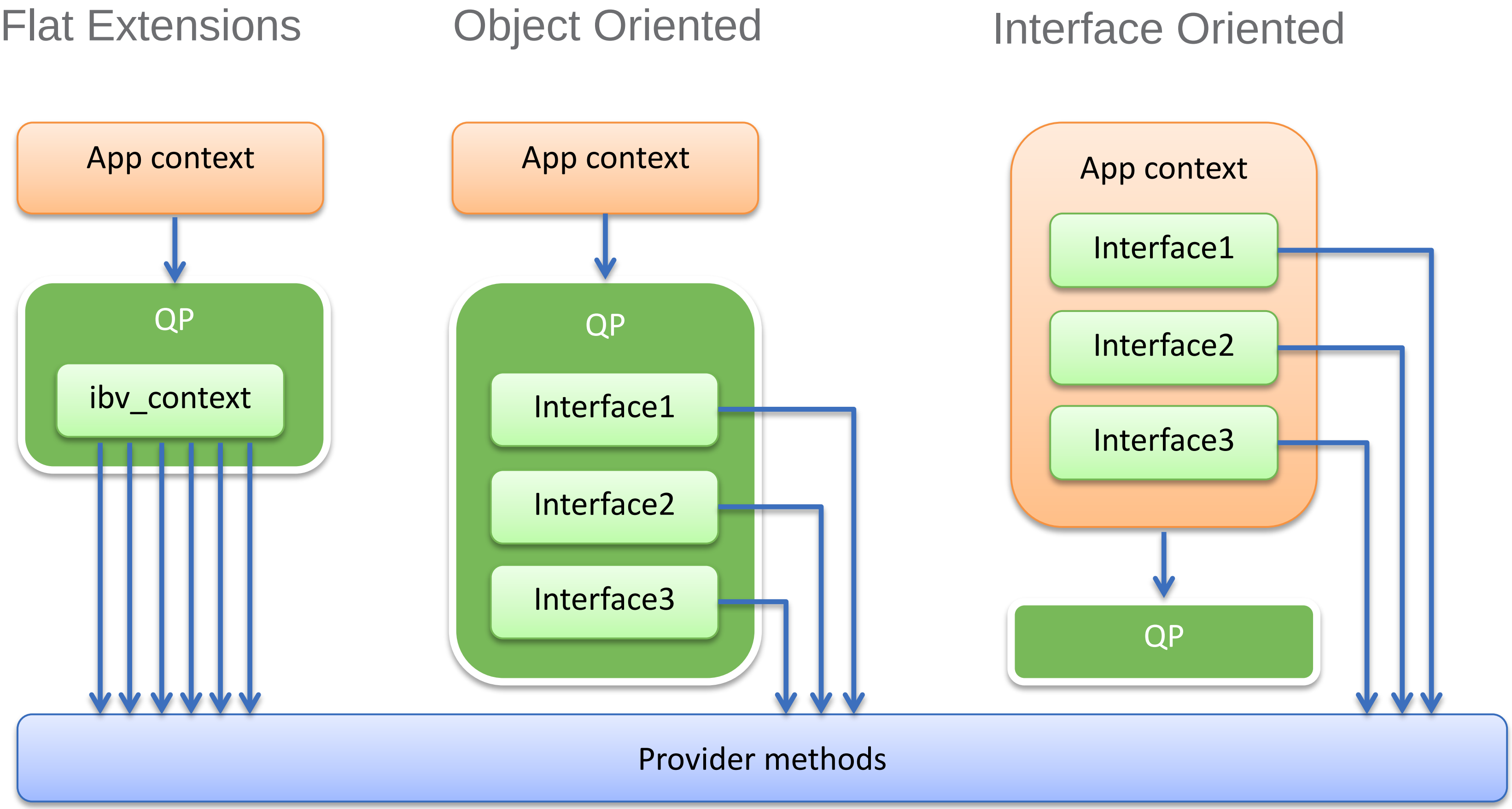
- Non-goals
  - Optimize all Verbs
  - Change the object model or semantics

Approaches

- Verbs extensions
  - Standard approach
  - A “flat” extension of the current Verbs
  - But
    - Incurs access checks overheads
    - Not specialized for certain objects or parameters

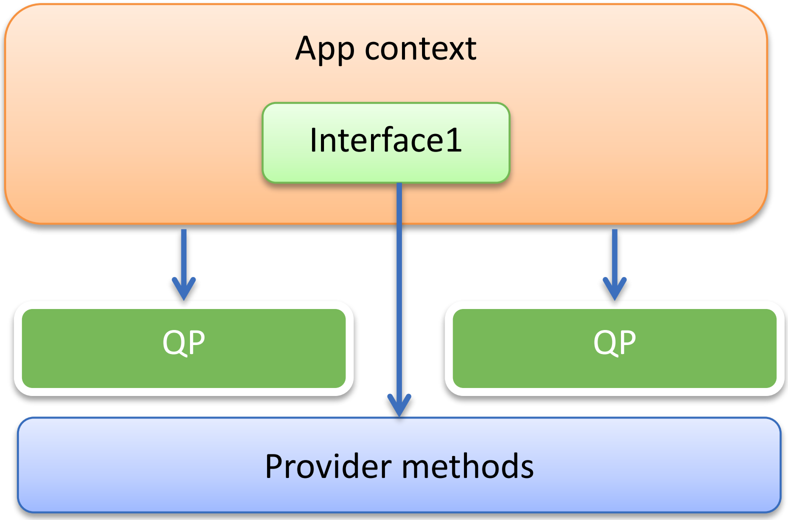
- Obtain accelerated function tables through a parameterized “Query Interface” Verb
  - Allows to obtain different parameterized versions of the same interface for further optimizations
  - All versioning and capability checks done here
    - No overheads during fast path
  - Implemented by direct vendor calls
  - Allows for selective implementation and deprecation of interfaces

Approaches (cont.)

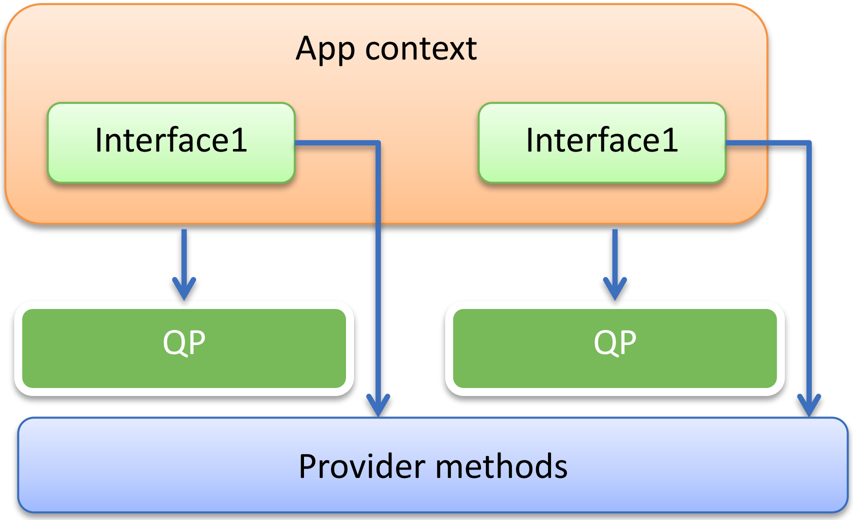


Typed vs. Object-oriented Interfaces

- Typed interfaces
  - Interface requested per type
  - Obtained interface may be applied to all objects that adhere to the same type



- Object-oriented
  - Interface requested per object
  - Obtained interface may only be used for bound object
    - Provider may still choose to return the same interface for multiple objects



Examples

OPENFABRICS ALLIANCE

10TH ANNUAL INTERNATIONAL DEVELOPER WORKSHOP

Query Interface

- Interface attributes
  - Flags: generic and interface specific
  - Interface family ID and version
  - Object to associate with interface

```
struct ibv_exp_query_intf_params {
      uint32_t                    comp_mask;
      uint32_t                    flags;           /* Generic flags */
      enum ibv_intf_family        intf;
      int                         intf_version;    /* Version */

      void                        *obj;            /* QP/CQ/SRQ/etc - depending on family */
      void                        *family_params;  /* Family-sepcific params (optional) */
      uint32_t                    family_flags;    /* Family-specific flags */
};

void *ibv_query_intf(struct ibv_context *context,
                     struct ibv_query_intf_params *params)


int ibv_release_intf(struct ibv_context *context, void *intf);
```

QP RDMA Operations

```
struct ibv_qp_ops_rdma {
      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);
};
```

QP Channel Operations

```
struct ibv_qp_ops_msg {
      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 {
      int poll_formatted(struct ibv_cq *cq, void *buf, size_t len);
      int poll_num();
};
```

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_intf(ctx->context, ... /* QP channel family */);

      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_intf(ctx->context, ... /* Formatted CQ family */);
      ...
}
```

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;
}
```

Discussion

- Accelerated / parameterized Verbs approach
  - Relation to flat Verbs
    - Always a strict subset
  - Type/object oriented
- Interface parameters
  - Interface ID
    - String
      - “experimental.raw_eth”
      - “vendor.mlx.raw_eth”
      - “raw_eth”
    - GUID
    - Enum
  - Versioning
  - Generic flags
    - Check strictness
- Different Query Interface per Object type?
- Deprecating interfaces

Thank You

OPENFABRICS ALLIANCE

10TH ANNUAL INTERNATIONAL DEVELOPER WORKSHOP