NAME

mlx5dv_wr_set_dc_addr − Attach a DC info to the last work request

SYNOPSIS

```
#include <infiniband/mlx5dv.h>

static inline void mlx5dv_wr_set_dc_addr(struct mlx5dv_qp_ex *mqp,
                                         struct ibv_ah *ah,
                                         uint32_t remote_dctn,
                                         uint64_t remote_dc_key);

struct mlx5dv_mr_interleaved {
    uint64_t        addr;
    uint32_t        bytes_count;
    uint32_t        bytes_skip;
    uint32_t        lkey;
};

static inline void mlx5dv_wr_mr_interleaved(struct mlx5dv_qp_ex *mqp,
                        struct mlx5dv_mkey *mkey,
                        uint32_t access_flags, /* use enum ibv_access_flags
                        uint32_t repeat_count,
                        uint16_t num_interleaved,
                        struct mlx5dv_mr_interleaved *data);

static inline void mlx5dv_wr_mr_list(struct mlx5dv_qp_ex *mqp,
                      struct mlx5dv_mkey *mkey,
                      uint32_t access_flags, /* use enum ibv_access_flags *
                      uint16_t num_sges,
                      struct ibv_sge *sge);
```

DESCRIPTION

The MLX5DV work request APIs (mlx5dv_wr_*) is an extension for IBV work request API (ibv_wr_*) with mlx5 specific features for send work request. This may be used together with or without ibv_wr_* calls.

USAGE

To use these APIs a QP must be created using mlx5dv_create_qp() with *send_ops_flags* of struct ibv_qp_init_attr_ex set.

If the QP does not support all the requested work request types then QP creation will fail.

The mlx5dv_qp_ex is extracted from the IBV_QP by ibv_qp_to_qp_ex() and mlx5dv_qp_ex_from_ibv_qp_ex(). This should be used to apply the mlx5 specific features on the posted WR.

A work request creation requires to use the ibv_qp_ex as described in the man for ibv_wr_post and mlx5dv_qp with its available builders and setters.

QP Specific builders

RC QPs

mlx5dv_wr_mr_interleaved()

registers an interleaved memory layout by using an indirect mkey and some interleaved data. The layout of the memory pointed by the mkey after its registration will be the *data* representation for the *num_interleaved* entries. This single layout representation is repeated by *repeat_count*.

The *data* as described by struct mlx5dv_mr_interleaved will hold real data defined by *bytes_count* and then a padding of *bytes_skip*. Post a successful registration, RDMA operations can use this *mkey*. The hardware will scatter the data according to the pattern. The *mkey* should be used in a zero−based mode. The *addr* field in its *ibv_sge* is an offset in the total data.

Current implementation requires the IBV_SEND_INLINE option to be on in *ibv_qp_ex−>wr_flags* field. To be able to have more than 3 *num_interleaved* entries, the QP should be created with a larger WQE size that may fit it. This should be done using the *max_inline_data* attribute of *struct ibv_qp_cap* upon its creation.

As one entry will be consumed for strided header, the *mkey* should be created with one more entry than the required *num_interleaved.*

In case *ibv_qp_ex−>wr_flags* turns on IBV_SEND_SIGNALED, the reported WC opcode will be MLX5DV_WC_UMR. Unregister the *mkey* to enable another pattern registration should be done via ibv_post_send with IBV_WR_LOCAL_INV opcode.

mlx5dv_wr_mr_list()

registers a memory layout based on list of ibv_sge. The layout of the memory pointed by the *mkey* after its registration will be based on the list of *sge* counted by *num_sges*. Post a successful registration RDMA operations can use this *mkey*, the hardware will scatter the data according to the pattern. The *mkey* should be used in a zero−based mode, the *addr* field in its *ibv_sge* is an offset in the total data.

Current implementation requires the IBV_SEND_INLINE option to be on in *ibv_qp_ex−>wr_flags* field. To be able to have more than 4 *num_sge* entries, the QP should be created with a larger WQE size that may fit it. This should be done using the *max_inline_data* attribute of *struct ibv_qp_cap* upon its creation.

In case *ibv_qp_ex−>wr_flags* turns on IBV_SEND_SIGNALED, the reported WC opcode will be MLX5DV_WC_UMR. Unregister the *mkey* to enable other pattern registration should be done via ibv_post_send with IBV_WR_LOCAL_INV opcode.

QP Specific setters

DCI **QPs**

mlx5dv_wr_set_dc_addr() must be called to set the DCI WR properties. The destination address of the work is specified by *ah*, the remote DCT number is specified by *remote_dctn* and the DC key is specified by *remote_dc_key*. This setter is available when the QP transport is DCI and send_ops_flags in struct ibv_qp_init_attr_ex is set. The available builders and setters for DCI QP are the same as RC QP.

EXAMPLE

```
/* create DC QP type and specify the required send opcodes */
attr_ex.qp_type = IBV_QPT_DRIVER;
attr_ex.comp_mask |= IBV_QP_INIT_ATTR_SEND_OPS_FLAGS;
attr_ex.send_ops_flags |= IBV_QP_EX_WITH_RDMA_WRITE;

attr_dv.comp_mask |= MLX5DV_QP_INIT_ATTR_MASK_DC;
attr_dv.dc_init_attr.dc_type = MLX5DV_DCTYPE_DCI;

ibv_qp *qp = mlx5dv_create_qp(ctx, attr_ex, attr_dv);
ibv_qp_ex *qpx = ibv_qp_to_qp_ex(qp);
mlx5dv_qp_ex *mqpx = mlx5dv_qp_ex_from_ibv_qp_ex(qpx);

ibv_wr_start(qpx);

/* Use ibv_qp_ex object to set WR generic attributes */
qpx->wr_id = my_wr_id_1;
qpx->wr_flags = IBV_SEND_SIGNALED;
ibv_wr_rdma_write(qpx, rkey, remote_addr_1);
ibv_wr_set_sge(qpx, lkey, local_addr_1, length_1);
```

```
        /* Use mlx5 DC setter using mlx5dv_qp_ex object */
        mlx5dv_wr_set_wr_dc_addr(mqpx, ah, remote_dctn, remote_dc_key);

        ret = ibv_wr_complete(qpx);
```

SEE ALSO

ibv_post_send(3), **ibv_create_qp_ex(3)**, **ibv_wr_post(3)**.

AUTHOR

Guy Levi <guyle@mellanox.com>