14th ANNUAL WORKSHOP 2018

NON-CONTIGUOUS MEMORY REGISTRATION

Tzahi Oved

[April, 2018]

Mellanox Technologies

AGENDA

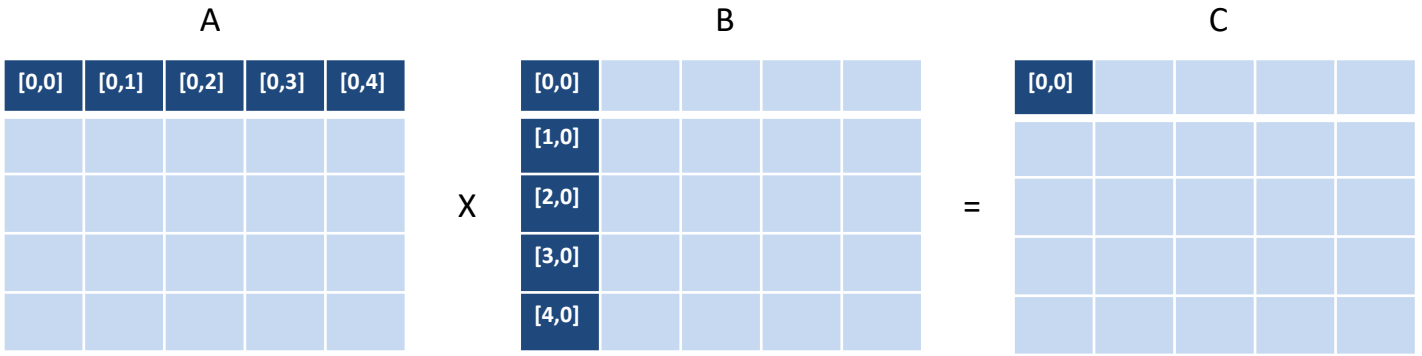
- **Motivation**
- **Address patterns**
- **Verbs object**
- **Verbs API Proposal**
- **Examples and use cases**

GOAL

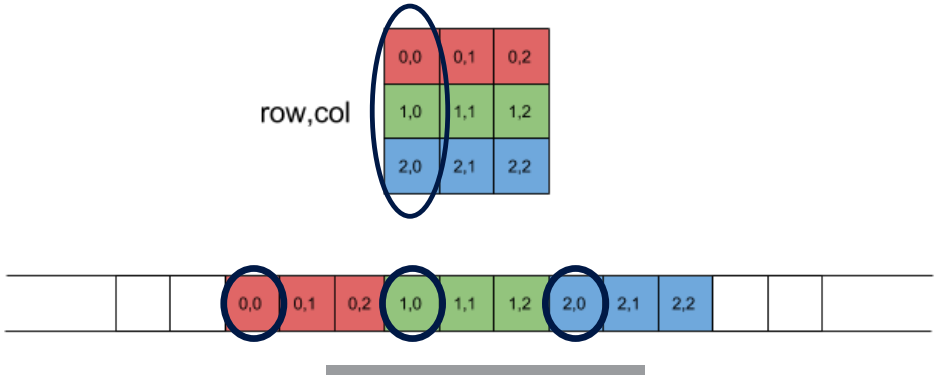
- **ZERO COPY**
- **How can one do zero copy if data is completely scattered?**
- **Apps often reuse a non-contiguous memory layout**
  - HPC – Boundary exchange, Collective operations (e.g. Bruck Alltoall algorithm)
  - Vector Graphics – Interleaved Vertex Data
  - Scientific – Matrix operations (see next slide)
- **RDMA**
  - IBTA defines RDMA operations with single virtual address reference
    - Either Read or Write operation
  - How can we define non contiguous virtual memory with single VA
- **What we need is a compact, reusable memory layout description**
  - Describe the memory layout once
  - Use like a stencil, applied at different base-pointers

EXAMPLE - MATRIX MULTIPLICATION

- **The goal: distribute slices of a matrix for multiplication**
- **The problem: sending a column may not be efficient, because it's not contiguous in memory**
  - In the next example, we need a row from A and a column from B to calculate C[0][0]:



  - Notice that while the row from A is virtually contiguous, the column from B is not:



HOW DO WE SEND A TYPICAL PATTERN TODAY?

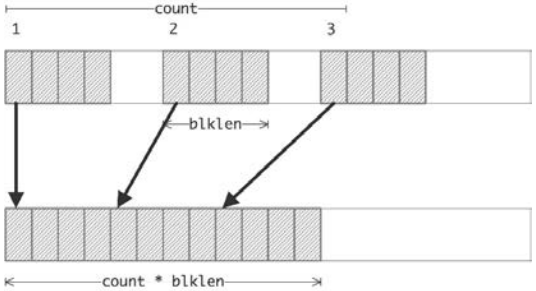
1. List of Scatter-Gather entries

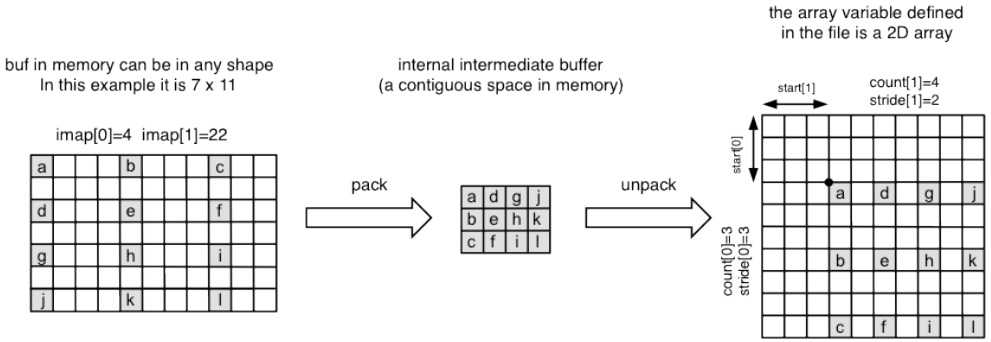
- For example, one column in a Matrix of size N x N requires N entries:
  - Need to construct N s-g entries for the WR
  - s-g list count is limited – may not fit into a single WR


```
/* matrix[ROW#][COL#] */
struct ibv_sge first_column[N];
first_column[0].addr = &matrix[0][0];
first_column[0].length = CELL_SIZE;
first_column[1].addr = &matrix[1][0];
first_column[1].length = CELL_SIZE;
...
first_column[N-1].addr = &matrix[N-1][0];
first_column[N-1].length = CELL_SIZE;
```

2. "User-level packing"

- Data is copied to a contiguous intermediate buffer - No longer zero-copy
- The illustration on the right – from PnetCDF v1.6 manual for ***ncmpi_put_varm()***
- On both sender and receiver sides





SOLUTION OUTLINE

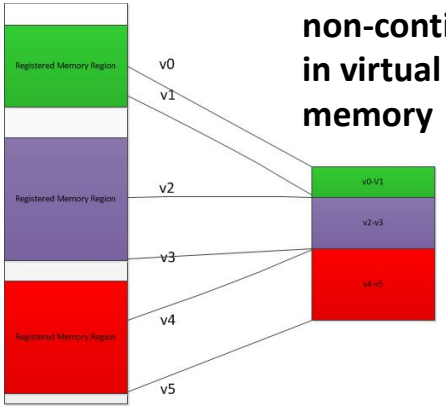
1. **Register the memory layout on the network device**
   - The representation of the layout is stored on the device
   - Either local or remote key is generated
2. **Each transaction may use the pre-registered layout on it's relevant data set - as a "stencil"**
   - Pass only a pointer, length and a handle to the layout
3. **For send, the HCA uses local DMA read to gather the data according to the layout**
4. **For receive, local DMA write will be used to scatter the data according to the layout**
5. **For RDMA, either Read or Write the data according to the layout directly from remote host memory**

TYPICAL ADDRESSED PATTERNS

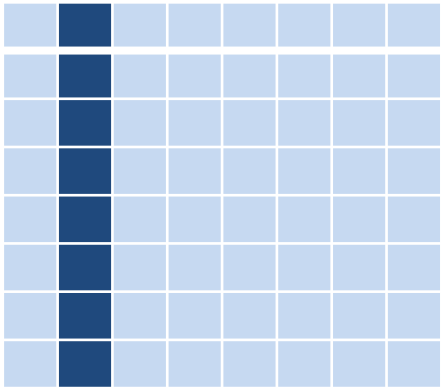
1. Composites

Three memory regions, each referenced by a different memory key

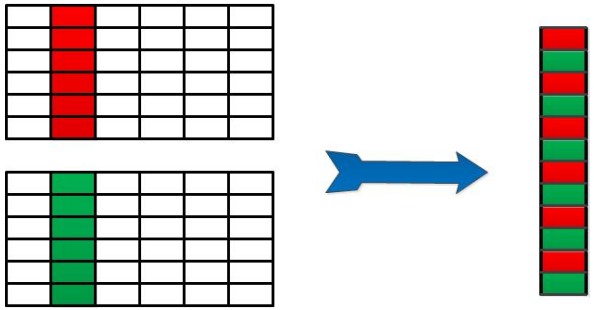
One memory region referenced by one memory key, non-contiguous in virtual memory



2. “Strided” data



3. Interleaved data



VERBS API PROPOSAL

API OUTLINE

1. **Creating an MR object representing memory layouts**
2. **Setting a composite, strided & interleaved memory layout**
3. **API Capabilities**
4. **Completion semantics**
5. **MR modification semantics**
6. **Code samples**

OBJECT CREATION

- **Use Verbs memory region (MR) object – ibv_mr**
- **Indirect ib_mr is created through the following:**
  - addr=NULL
  - ZBVA flag

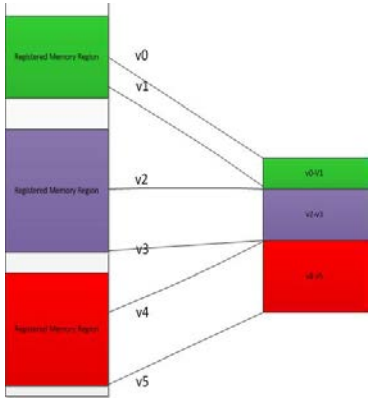
```
mr = ibv_reg_mr(pd, NULL, 0, IBV_ACCESS_ZERO_BASED);
```

- **Currently, only zero-based MRs are used**

NEW VERB – COMPOSITE LAYOUT

include/infiniband/verbs.h:

```
int ibv_mr_set_layout_sg(struct ibv_mr* mr,
                         int flags,
                         int num_sge,
                         struct ibv_sge *sg_list);
```




```
enum ibv_mr_set_layout_flags {
    IBV_MR_SET_LAYOUT_WITH_POST_WR          = (1 << 0),
    IBV_MR_SET_LAYOUT_AVOID_INVALIDATION    = (1 << 1),
    IBV_MR_SET_LAYOUT_FLAGS_SUPPORTED
    =((IBV_MR_SET_LAYOUT_AVOID_INVALIDATION<<1)-1)
};
```

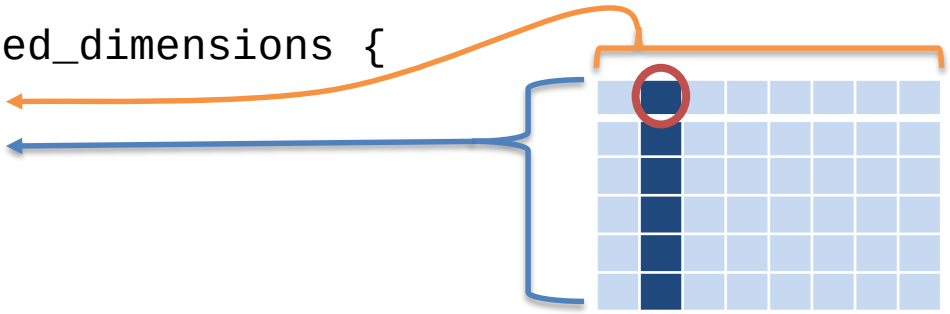
NEW VERB – STRIDED/INTERLEAVED LAYOUT

include/infiniband/verbs.h:

```
int ibv_mr_set_layout_interleaved(struct ibv_mr* mr,
            int flags,
            int num_interleaved, /* =1 for strided,
                >=2 for interleaving buffers from 2 or more sources */
            struct ibv_mr_layout_interleaved *ilv_list);

struct ibv_mr_layout_interleaved {
    struct ibv_sge    first_datum;    /* first single item */
    int               num_repeated;   /* interleaving ratio */
    int               num_dimensions; /* e.g. 3D matrix */
    struct ibv_mr_layout_interleved_dimensions   *dims;
};

struct ibv_mr_layout_interleved_dimensions {
    uint64_t offset_stride;
    uint64_t datum_count;
};
```



NEW CAPABILITIES – NON CONTIGUOUS MEMORY

- **ibv_query_device(struct ibv_device_attr)**

include/infiniband/verbs.h:

```
enum ibv_mr_layout_cap_flags {
     IBV_MR_SET_LAYOUT_SG                                = 1 << 0,
     IBV_MR_SET_LAYOUT_INTERLEAVED                       = 1 << 1,
     IBV_MR_SET_LAYOUT_INTERLEAVED_REPEAT                = 1 << 2,
     IBV_MR_SET_LAYOUT_INTERLEAVED_NONUNIFORM_REPEAT     = 1 << 3,
     IBV_MR_SET_LAYOUT_INTERLEAVED_NONUNIFORM_DATUM_TOTAL = 1 << 4,
};
```

(num_repeated != 1)

(variable repeat)

(variable datum_count)

```
struct ibv_mr_layout_caps {
     uint64_t cap_flags;                  /* mask of the flags above */
     uint32_t max_num_sg;                 /* max. composite buffers per layout */
     uint32_t max_inline_num_sg;          /* max. composite buffers to fit a WR */
     uint32_t max_num_interleaved;        /* max. interleaved sources per layout */
     uint32_t max_inline_num_interleaved; /* max. interleaved source to fir a WR */
     uint32_t max_mr_stride_dimension;    /* max. number of strides/dimensions */
     uint32_t max_mr_nesting_level;       /* max. indirection for nested layouts */
};
```

BIND LAYOUT TO MR

Blocking Flow	Non-blocking Flow

ibv_mr_set_layout_sg(ibv_mr, flags=0)	ibv_mr_set_layout_sg(ibv_mr,
	flags=IBV_MR_SET_LAYOUT_WITH_POST_WR)
	
	struct ibv_send_wr {
	...
	enum ibv_wr_opcode opcode=IBV_WR_BIND_MR
	...
	struct {
	struct ibv_mr *mr;
	} mr_set_layout;
	};
	
	ibv_post_send(opcode=IBV_WR_BIND_MR)

ibv_mr_set_layout_sg() call returns once binding is usable	ibv_mr_set_layout_sg() call returns immediately after recording the layout (stored in the MR) – additional WR has to be posted on some QP
Binding is available on all QPs immediately	Binding is usable for operations on the same QP immediately. Upon completion can be used with other QPs

TYPICAL USAGE FLOW

1. Create non contiguous MR

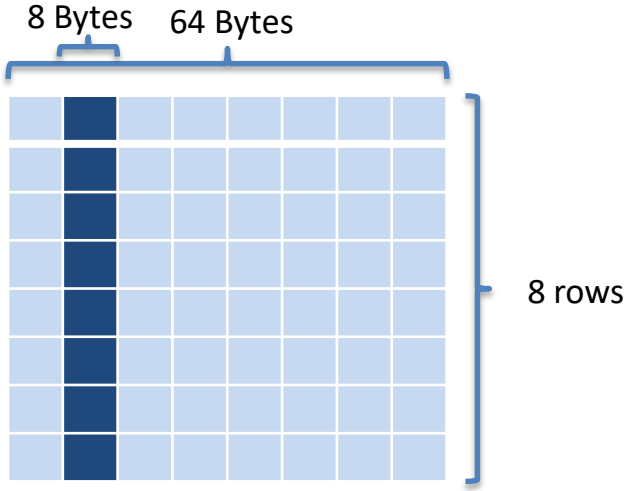
```
struct ibv_mr *nc_mr = ibv_reg_mr(pd, NULL, 0, IBV_ACCESS_ZERO_BASED);
```

2. Create a memory layout

```
layout_interleaved.first_datum.addr       = addr+8;
layout_interleaved.first_datum.length     = 8;
layout_interleaved.first_datum.lkey       = base_mr.lkey; // block addr
layout_interleaved.num_repeated           = 1;
layout_interleaved.num_dimensions         = 1;
layout_interleaved.dims[0].offset_stride = 64;
layout_interleaved.dims[0].datum_count    = 8;
ibv_mr_set_layout_interleaved(nc_mr, 0, 1, &layout_interleaved);//block
```

3. Send a packet

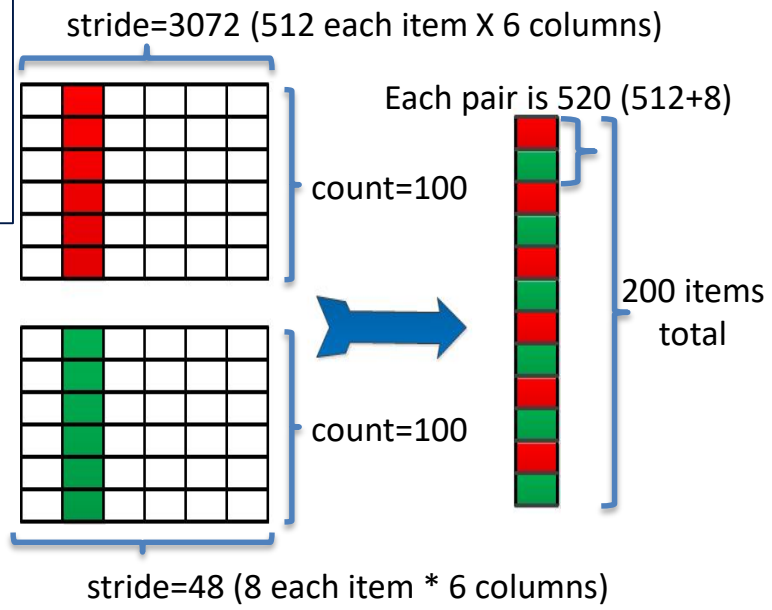
```
wr.sg_list[0].addr = 0;
wr.sg_list[0].length = 8*8;
wr.sg_list[0].lkey = nc_mr->lkey;
ibv_post_send(qp, wr, &bad_wr);
```



INTERLEAVED PATTERN EXAMPLE

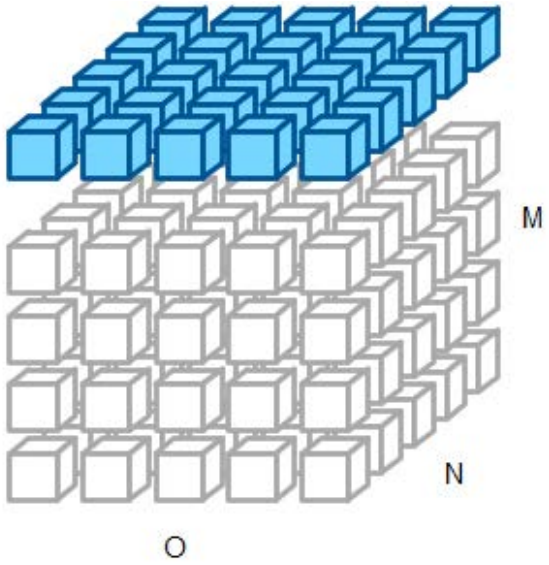
```
layout[0].first_datum.lkey              = mr1->lkey;
layout[0].first_datum.addr              = addr1;
layout[0].first_datum.length            = 512;
layout[0].num_repeated                  = 1;
layout[0].num_dimensions                = 1;
layout[0].dimension[0].offset_stride    = 3072;
layout[0].dimension[0].datum_count      = 100;

layout[1].first_datum.lkey              = mr2->lkey;
layout[1].first_datum.addr              = addr2;
layout[1].first_datum.length            = 8;
layout[1].num_repeated                  = 1;
layout[1].num_dimensions                = 1;
layout[1].dimension[0].offset_stride    = 48;
layout[1].dimension[0].datum_count      = 100;
```

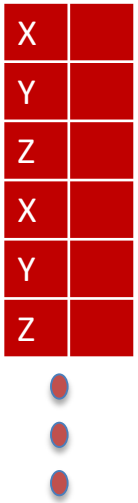


MULTI-DIMENTIONAL PATTERN EXAMPLE

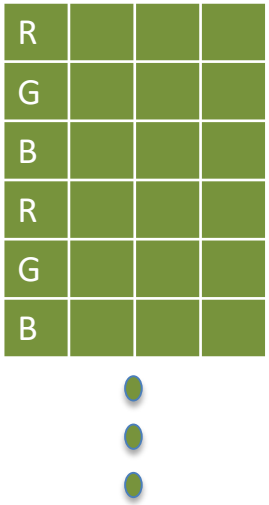
```
layout_entry[0].first_datum.lkey      = mr1;
layout_entry[0].first_datum.addr      = addr1;
layout_entry[0].first_datum.length    = 4;
layout_entry[0].num_repeated          = 1;
layout_entry[0].num_dimensions        = 2;
layout_entry[0].dimension[0].stride   = 4 * M;
layout_entry[0].dimension[0].count    = N;
layout_entry[0].dimension[1].stride   = 4 * M * N;
layout_entry[0].dimension[1].count    = O;
```



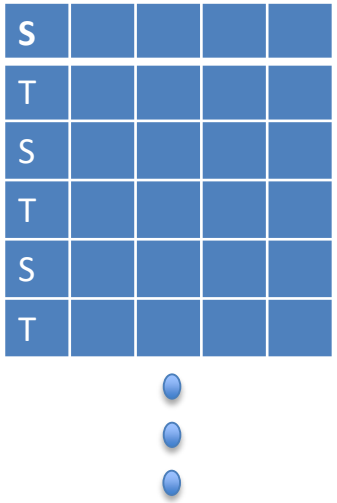
NON-UNIFORM REPETITIONS EXAMPLE



Length = 4
num_repeated= 3
count = 18 (6 cycles X 3 items)
stride = 2

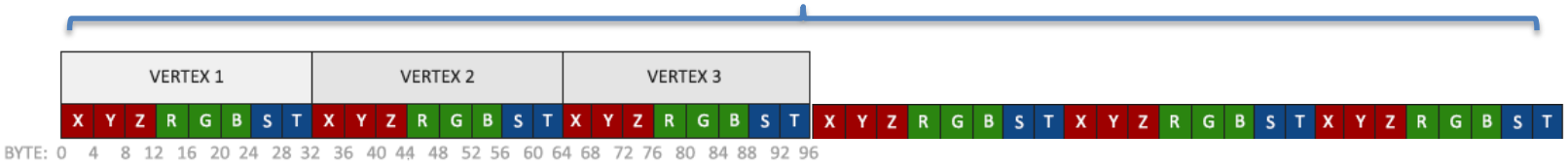


Length = 4
num_repeated= 3
count = 18 (6 cycles X 3 items)
stride = 4



Length = 4
num_repeated= 2
count = 12 (6 cycles X 2 items)
stride = 5

There are 6 cycles in this example



Each cycle takes 3:3:2, according to `num_repeated` of each element
Lengths are 4, cycle is 8*4 bytes

SUMMARY

- **Devices to become aware of application objects memory layout**
- **Such scheme allows the layout re-use as a stencil for all objects with same type**
- **Non contiguous virtual address space can become contiguous for the device**
- **Main benefits**
  - Zero copy – no packing/unpacking
  - Avoid messing with long scatter-gather lists
- **API wise, natural extension of good old ibv_mr**

OPENFABRICS ALLIANCE

14th ANNUAL WORKSHOP 2018

THANK YOU

Tzahi Oved

Mellanox Technologies