**OFA Face-to-face Hillsboro, OR August 19, 2014**

**SHMEM (one-sided) working group**

Goals for Illustrative Examples

1) setup/initialization

2) data transfer (put/get) with bulk (fence/quiet) synchronization

3) bundled put operations (counting puts)

4) put with notification

5) put/get/amo types - blocking / nonblocking implicit / nonblocking explicit

6) atomic operations aka AMOs

Beyond SHMEM (or not common SHMEM)

1) General active messages

2) Noncontiguous transfers (strided, vector, subarray, etc.)

3) integer / floating-point accumulate (atomics or active messages?)

Endpoint type - for a simple pseudo code (and for scalability) FID\_RDM is best

* endpoint ops FI\_INJECT (inline in ib speak),
* use fi\_setopt/fi\_getopt to query set FI\_OPT\_MAX\_INJECTED\_SEND
* FI\_REMOTE\_COMPLETE
* must have FI\_WRITE\_COHERENT
* Open question: How do we support transports like Mellanox DCT ?

Completion notification - use EQ (not counter for now, may use fid\_ctr for bundled ops)

* use FI\_EVENT to control EQ generation (for blocking and non-blocking explicit)
* use fi\_sync for shmem\_quiet/shmem\_fence/MPI\_Win\_flush(\_local) and non-blocking implicit. Note: fl\_sync does not map well on shmem\_fence().
* can use the inject option for small puts/put style AMOs

For hints in fi\_getinfo we need these for ep\_cap

* ep\_cap = FI\_RMA | FI\_ATOMICS | FI\_INJECT
* may use FI\_MSG for internal control messages
* op\_flags = FI\_REMOTE\_COMPLETE
* domain\_cap = FI\_WRITE\_COHERENT
* would also like FI\_USER\_MR\_KEY | FI\_DYNAMIC\_MR
* type = FI\_RDM (for this simple example)
* addr\_format = FI\_ADDR\_INDEX
* endpoint attributes
* inject size (get it) we don’t want this to be big, only 8 bytes or so
* check the max\_order\_xxx\_size to see if non-zero
* msg\_order (don’t need order for our example)

Do need a way to get SHMEM thread hot. How do we do this?  Multiple endpoints? Or multiple domains?

/\*

 \* lawyer blurb goes here

\*/

struct fi\_info\_shmem, \*fi\_info\_out=NULL;

struct fid\_fabric \*fid\_fabric\_shmem;

struct fid\_domain \*fid\_domain\_shmem;

struct fid\_ep \*fid\_ep\_shmem;

struct fid\_eq \*fid\_eq\_shmem;

struct fi\_eq\_attr eq\_attr\_shmem;

void start\_of\_shmem\_init(void)

{

/\* see all the info stuff above to see how we initialize \*/

fi\_getinfo(“fast-local-rdma-dev-ip-addr”, NULL,0UL,&fi\_info\_shmem,&fi\_info\_out);

assert (fi\_info\_out != NULL);

/\* assume we only got one fi\_info\_out back, i.e. don’t care about multiple rails right now\*/

fi\_fabric(fi\_info\_shmem->fabric\_name, 0, &fid\_fabric\_shmem, (void \*)our\_shmem\_context);

fi\_fdomain(fid\_fabric\_shmem, fi\_info\_shmem, &fid\_domain\_shmem, NULL, (void \*)our\_shmem\_context);

/\* we skip binding EQ to the domain \*/

/\* open the end point \*/

fi\_endpoint(fid\_domain\_shmem, fi\_info\_shmem, &fid\_ep\_shmem, (void \*)our\_shmem\_context);

memset(&eq\_attr\_shmem,0,sizeof(eq\_attr\_shmem));

eq\_attr\_shmem.domain =  FI\_EQ\_DOMAIN\_COMP;

eq\_attr\_shmem.mask = FI\_EQ\_ATTR\_MASK\_V1;

eq\_attr\_shmem.format = FI\_EQ\_FORMAT\_CONTEXT;

/\* open the eq \*/

fi\_eq\_open(fid\_domain\_shmem, &eq\_attr\_shmem, &fid\_eq\_shmem, (void \*)our\_shmem\_context);

/\* bind the eq to the ep, we only want entries back when we ask for them \*/

fi\_ep\_bind(fid\_ep\_shmem, fid\_eq\_shmem, FI\_EVENT);

/\* now set up the adress vector \*/

/\* use out of band mechanism to exchange ip addrs for our node \*/