Outline

1.1 RDMA Model

What do we need for explicit RDMA Read / Write?

- Exchange Address
- Exchange Memory Key
- Handling Async Operation (send queue / poll completion queue)

1.2 OpenSHMEM Model

- PGAS: Partitioned Global Address Space
  - All process share same memory space (that needs to be shared).
- SPMD (single program, multiple data)
  - The SHMEM processes, called processing elements or **PE**s, all start at the same time and they all run the same program.
- Get/Put Operation
  - `shmem_get` / `shmem_put`
  - `shmem_get_nbi` / `shmem_put_nbi` (Non-blocking)
- Synchrnoization Primitive (similar to Multi-threading Programming)
  - Barrier
  - Wait
  - Fence / Quiet
  - Lock

1.3 Sample

```c
int main(void) {
    shmem_init();
    const int SIZE = 10;
    int local[SIZE] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    int shared[SIZE] = {0};
    // Processing element (PE) 1 is the target.
    int target_pe = 1;
    if (shmem_my_pe == 0){
        // Perform the put operation: copy 'local' array to 'shared' array at target PE.
        shmem_int_put_nbi(shared, local, SIZE, target_pe);
    }
    // Synchronize all processing elements to ensure the put operation completes.
    shmem_barrier_all();
    if (shmem_my_pe() == target_pe) {
        printf("Data received on PE %d:\n", shmem_my_pe());
        for (int i = 0; i < SIZE; i++) {
            printf("%d ", dst[i]);
        }
        printf("\n");
    }
    shmem_finalize();
    return 0;
}
```

Outline

1 OpenShmem 1

  1.1 RDMA Model 2

  1.2 OpenSHMEM Model 3

  1.3 Sample 4

2 Put Operation 6

  2.1 Specification 7

  2.2 Overview 8

  2.3 Entry Point (e.g. `shmem_int_put_nbi`) 9

  2.4 SPML Layer 10

  2.5 UCX Layer 12

  2.6 Get Operation 13

2.1 Specification

Entry Points

- `shmem_#type_put(...)`
- `shmem_#type_put_nbi(...)`
- `shmem_#type_put_signal(...)` (Not Supported with UCX)
- `shmem_#type_put_signal_nbi(...)` (Not Supported with UCX)

Example. `void shmem_double_put(double *target, const double *source, size_t len, int pe)`

2.2 Overview

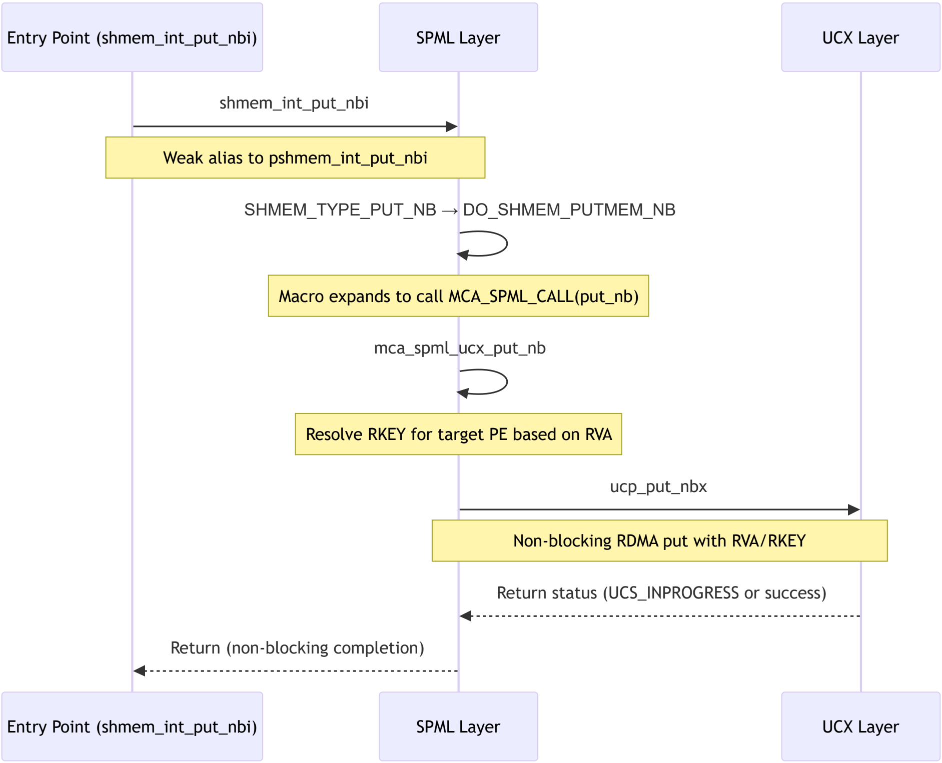
Entry Point (shmem_int_put_nbi)
SPML Layer
UCX Layer
shmem_int_put_nbi
Weak alias to pshmem_int_put_nbi
SHMEM_TYPE_PUT_NB → DO_SHMEM_PUTMEM_NB
Macro expands to call MCA_SPML_CALL(put_nb)
mca_spml_ucx_put_nb
Resolve RKEY for target PE based on RVA
ucp_put_nbx
Non-blocking RDMA put with RVA/RKEY
Return status (UCS_INPROGRESS or success)
Return (non-blocking completion)
Entry Point (shmem_int_put_nbi)
SPML Layer
UCX Layer

Figure 1: OpenSHMEM Illustration

2.3 Entry Point (e.g. shmem_int_put_nbi)

Starting at shmem_put_nb.c, line 117:

```c
#pragma weak shmem_int_put_nbi = pshmem_int_put_nbi
```

This defines a weak symbol alias for the profiling interface. The actual implementation is created from the macro:

```c
SHMEM_TYPE_PUT_NB(_int, int)
```

expands to:

```c
void shmem_put8_nbi(void *target, const void *source, size_t nelems, int pe) {
    DO_SHMEM_PUTMEM_NB(oshmem_ctx_default, target, source, 1, nelems, pe);
    return;
}
```

2.4 SPML Layer

The DO_SHMEM_PUTMEM_NB macro is defined in oshmem_shmem.c:

c

```c
#define DO_SHMEM_PUTMEM_NB(ctx, target, source, element_size, nelems, pe) do { \
        int rc = OSHMEM_SUCCESS;                                    \
        size_t size = 0;                                            \
        ...
        size = nelems * element_size;                               \
        rc = MCA_SPML_CALL(put_nb(                                  \
            ctx,                                                    \
            (void *)target,                                         \
            size,                                                   \
            (void *)source,                                         \
            pe, NULL));                                             \
        RUNTIME_CHECK_RC(rc);                                       \
    } while (0)
```

2.4 SPML Layer

```c
#define MCA_SPML_CALL(a) mca_spml.spml_ ## a
```

Either the default

```c
mca_spml_ucx_t mca_spml_ucx = {
  .super = {
    ...
    .spml_put = mca_spml_ucx_put,
    .spml_put_nb = mca_spml_ucx_put_nb,
    ...
  }
}
```

or if a threadhold for progress is defined

```c
static int spml_ucx_init(void)
{
    ...
    if (mca_spml_ucx.nb_put_progress_thresh) {
        mca_spml_ucx.super.spml_put_nb =
            &mca_spml_ucx_put_nb_wprogress;
    }
    ...
}
```

2.5 UCX Layer

```c
int mca_spml_ucx_put_nb(shmem_ctx_t ctx, void* dst_addr, size_t size, void* src_addr, int dst, void **handle)
{
    void *rva = NULL;
    ucs_status_t status;
    spml_ucx_mkey_t *ucx_mkey = mca_spml_ucx_ctx_mkey_by_va(ctx, dst, dst_addr, &rva, &mca_spml_ucx);
    assert(NULL != ucx_mkey);
    mca_spml_ucx_ctx_t *ucx_ctx = (mca_spml_ucx_ctx_t *)ctx;
    ucs_status_ptr_t status_ptr = ucp_put_nbx(ucx_ctx->ucp_peers[dst].ucp_conn, src_addr, size,
                                  (uint64_t)rva, ucx_mkey->rkey,
                                  &mca_spml_ucx_request_param);
    if (UCS_PTR_IS_PTR(status_ptr)) {
        ucp_request_free(status_ptr);
        status = UCS_INPROGRESS;
    } else {
        status = UCS_PTR_STATUS(status_ptr);
    }
    if (OPAL_LIKELY(status >= 0)) {
        mca_spml_ucx_remote_op_posted(ucx_ctx, dst);
    }
    return ucx_status_to_oshmem_nb(status);
}
```

2.6 Get Operation