

OpenSHMEM

2025-06-19

Outline

1	OpenShmem	1
1.1	RDMA Model	2
1.2	OpenSHMEM Model	3
1.3	Sample	4
2	Put Operation	5
2.1	Specification	6
2.2	Overview	7
2.3	Entry Point (e.g. shmem_int_put_nbi)	8
2.4	SPML Layer	9
2.5	UCX Layer	11
2.6	Get Operation	12

1.1 RDMA Model

What do we need for explicit RDMA Read / Write?

- Setup Queue Pair (QP) / Connection
- Register Memory Region (MR)
- 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).
 - `shmalloc` / `shfree` (Shared Memory Allocation / Free)
- SPMD (single program, multiple data)
 - The SHMEM processes, called processing elements or **PEs**, 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)
- Synchronoization Primitive (similar to Multi-threading Programming)
 - `Barrier ()`
 - `Wait`
 - `Fence` / `Quiet`
 - `Lock`

1.3 Sample

c

```
1 int main(void) {
2     shmem_init();
3     const int SIZE = 10;
4     int local[SIZE] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
5     int shared[SIZE] = {0};
6     int target_pe = 1; // Processing element (PE) 1 is the target.
7     if (shmem_my_pe == 0){
8         // Perform the put operation: copy 'local' array to 'shared' array at target PE.
9         shmem_int_put_nbi(shared, local, SIZE, target_pe);
10    }
11    shmem_barrier_all(); // Synchronize all processing elements to ensure the put operation completes.
12    if (shmem_my_pe() == target_pe) {
13        printf("Data received on PE %d:\n", shmem_my_pe());
14        for (int i = 0; i < SIZE; i++) {
15            printf("%d ", dst[i]);
16        }
17        printf("\n");
18    }
19    shmem_finalize();
20    return 0;
21 }
```

Outline

1	OpenShmem	1
1.1	RDMA Model	2
1.2	OpenSHMEM Model	3
1.3	Sample	4
2	Put Operation	5
2.1	Specification	6
2.2	Overview	7
2.3	Entry Point (e.g. shmem_int_put_nbi)	8
2.4	SPML Layer	9
2.5	UCX Layer	11
2.6	Get Operation	12

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

RVA: Remote Virtual Address

Rkey: Remote Key

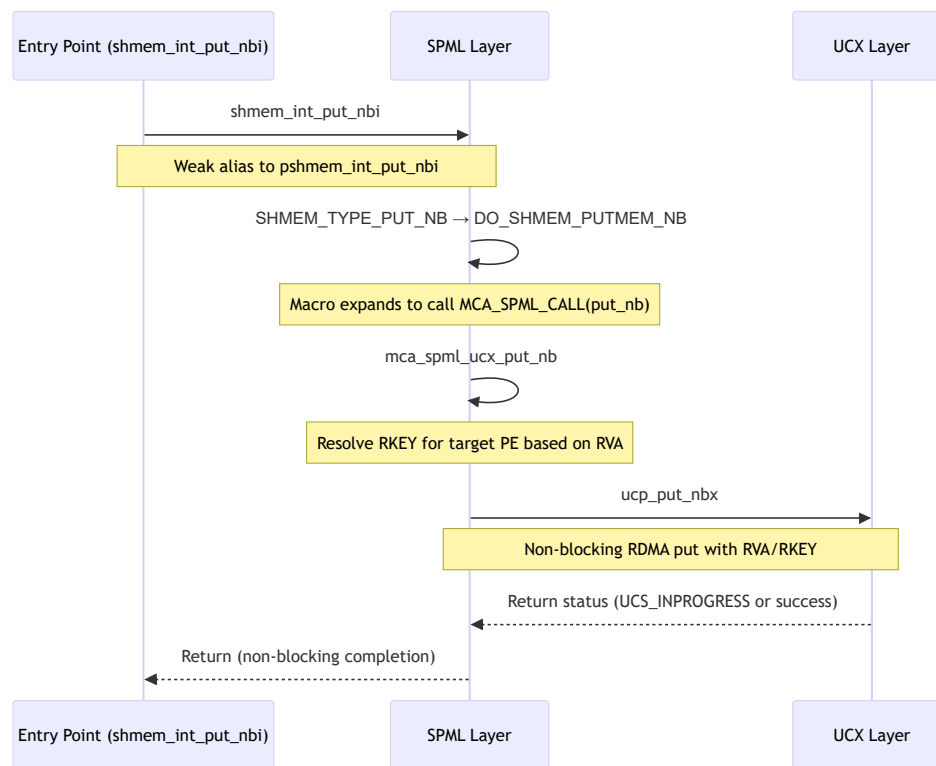


Figure 1: OpenSHMEM Illustration

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

Starting at `shmem_put_nb.c`, line 117:

```
1 #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:

```
1 SHMEM_TYPE_PUT_NB(_int, int)
```

expands to:

```
1 void shmem_put8_nbi(void *target, const void *source, size_t nelems, int pe) {  
2     DO_SHMEM_PUTMEM_NB(oshmem_ctx_default, target, source, 1, nelems, pe);  
3     return;  
4 }
```

2.4 SPML Layer

The DO_SHMEM_PUTMEM_NB macro is defined in oshmem_shmem.c:

```
1 #define DO_SHMEM_PUTMEM_NB(ctx, target, source, element_size, nelems, pe) do { \
2     int rc = OSHMEM_SUCCESS; \
3     size_t size = 0; \
4     ... \
5     size = nelems * element_size; \
6     rc = MCA_SPML_CALL(put_nb( \
7         ctx, \
8         (void *)target, \
9         size, \
10        (void *)source, \
11        pe, NULL)); \
12     RUNTIME_CHECK_RC(rc); \
13 } while (0)
```

c

2.4 SPML Layer

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

Either the default

```
1 mca_spml_ucx_t mca_spml_ucx = {
2   .super = {
3     ...
4   .spml_put = mca_spml_ucx_put,
5   .spml_put_nb = mca_spml_ucx_put_nb,
6   ...
7   }
8 }
```

or if a threshold for progress is defined

```
1 static int spml_ucx_init(void)
2 {
3   ...
4   if (mca_spml_ucx.nb_put_progress_thresh) {
5       mca_spml_ucx.super.spml_put_nb =
6       &mca_spml_ucx_put_nb_wprogress;
7   }
8   ...
9 }
```

2.5 UCX Layer

c

```
1 int mca_spml_ucx_put_nb(shmem_ctx_t ctx, void* dst_addr, size_t size, void* src_addr, int dst, void **handle)
2 {
3     void *rva = NULL;
4     ucs_status_t status;
5     spml_ucx_mkey_t *ucx_mkey = mca_spml_ucx_ctx_mkey_by_va(ctx, dst, dst_addr, &rva, &mca_spml_ucx);
6     assert(NULL != ucx_mkey);
7     mca_spml_ucx_ctx_t *ucx_ctx = (mca_spml_ucx_ctx_t *)ctx;
8     ucs_status_ptr_t status_ptr = ucp_put_nbx(ucx_ctx->ucp_peers[dst].ucp_conn, src_addr, size,
9        (uint64_t)rva, ucx_mkey->rkey,
10        &mca_spml_ucx_request_param);
11     if (UCS_PTR_IS_PTR(status_ptr)) {
12         ucp_request_free(status_ptr);
13         status = UCS_INPROGRESS;
14     } else {
15         status = UCS_PTR_STATUS(status_ptr);
16     }
17     if (OPAL_LIKELY(status >= 0)) {
18         mca_spml_ucx_remote_op_posted(ucx_ctx, dst);
19     }
20     return ucx_status_to_oshmem_nb(status);
21 }
```

-
-
-

-
-
-
-
-
-
-

2.6 Get Operation