…a modularity-enabled MPI library.

MPI/NAM Integration

Documentation

- Proposal for accessing the NAM via MPI
- API Prototype Implementation
- Usage Example on the DEEP-EST SDV

API Prototype Implementation

For evaluating the proposed semantics and API extensions, we have already developed a shared-memory-based prototype implementation where the persistent NAM is (more or less) ?emulated? by persistent shared-memory (with *deep_mem_kind=deep_mem_persistent*).

Advice to users

Please note that this prototype is not intended to actually emulate the NAM but shall rather offer a possibility for the later users and programmers to evaluate the proposed semantics from the MPI application?s point of view. Therefore, the focus here is not put on the question of how remote memory is managed at its location (currently by MPI processes running local to the memory later by the NAM manager or the NAM itself), but on the question of how process-foreign memory regions can be exposed locally. That means that (at least currently) for accessing a persistent RMA window, it has to be made sure that there is at least one MPI process running locally to each of the window?s memory regions.

Extensions to MPI

The API proposal strives to stick to the current MPI standard as close as possible and to avoid the addition of new API functions and other symbols. However, in order to make the usage of the prototype a little bit more convenient for the user, we have added at least a small set of new symbols (denoted with MPIX) that may be used by the applications.

```
extern int MPIX_WIN_DISP_UNITS;
#define MPIX_WIN_FLAVOR_INTERCOMM        (MPI_WIN_FLAVOR_CREATE +        \
                                         MPI_WIN_FLAVOR_ALLOCATE +        \
                                         MPI_WIN_FLAVOR_DYNAMIC +        \
                                         MPI_WIN_FLAVOR_SHARED + 0)
#define MPIX_WIN_FLAVOR_INTERCOMM_SHARED (MPI_WIN_FLAVOR_CREATE +        \
                                         MPI_WIN_FLAVOR_ALLOCATE +        \
                                         MPI_WIN_FLAVOR_DYNAMIC +        \
                                         MPI_WIN_FLAVOR_SHARED + 1)
```

Code Example for a ?Hello World? workflow

The following two C codes should demonstrate how it shall become possible to pass intermediate data between two subsequent steps of a workflow (Step 1: hello / Step 2: world) via the persistent memory of the NAM (currently emulated by persistent shared-memory):

```
/** hello.c **/

    /* Create persistent MPI RMA window: */
       MPI_Info_create(&win_info);
       MPI_Info_set(win_info, "deep_mem_kind", "deep_mem_persistent");
       MPI_Win_allocate(sizeof(char) * HELLO_STR_LEN, sizeof(char), win_info, MPI_COMM_WORLD,
                                         &win_base, &win);

       /* Put some content into the local region of the window: */
       if(argc > 1) {
               snprintf(win_base, HELLO_STR_LEN, "Hello World from rank %d! %s", world_rank, argv[1]);
       } else {
               snprintf(win_base, HELLO_STR_LEN, "Hello World from rank %d!", world_rank);
       }
       MPI_Win_fence(0, win);
```

```
        /* Retrieve port name of window: */
        MPI_Info_free(&win_info);
        MPI_Win_get_info(win, &win_info);
        MPI_Info_get(win_info, "deep_win_port_name", INFO_VALUE_LEN, info_value, &flag);

        if(flag) {
                strcpy(port_name, info_value);
                if(world_rank == root) printf("(%d) The Window's port name is: %s\n", world_rank, port_name);
        } else {
                if(world_rank == root) printf("(%d) No port name found!\n", world_rank);
        }
```

```
/** world.c **/

        /* Check for port name: (to be passed as a command line argument) */
        if(argc == 1) {
                if(world_rank == root) printf("[%d] No port name found!\n", world_rank);
                goto finalize;
        } else {
                strcpy(port_name, argv[1]);
                if(world_rank == root) printf("[%d] The Window's port name is: %s\n", world_rank, port_name);
        }

        /* Try to connect to the persistent window: */
        MPI_Info_create(&win_info);
        MPI_Info_set(win_info, "deep_win_connect", "true");
        MPI_Comm_set_errhandler(MPI_COMM_WORLD, MPI_ERRORS_RETURN);
        errcode = MPI_Comm_connect(port_name, win_info, root, MPI_COMM_WORLD, &inter_comm);
        printf("[%d] Connection to persistent memory region established!\n", world_rank);

         /* Retrieve the number of remote regions: (= former number of ranks) */
        MPI_Comm_remote_size(inter_comm, &remote_size);
        if(world_rank == root) printf("[%d] Number of remote regions: %d\n", world_rank, remote_size);

        /* Create window object for accessing the remote regions: */
        MPI_Win_create_dynamic(MPI_INFO_NULL, inter_comm, &win);
        MPI_Win_get_attr(win, MPI_WIN_CREATE_FLAVOR, &create_flavor, &flag);
        assert(*create_flavor == MPIX_WIN_FLAVOR_INTERCOMM);
        MPI_Win_fence(0, win);

        /* Check the accessibility and the content of the remote regions: */
        for(i=0; i<remote_size; i++) {
                char hello_string[HELLO_STR_LEN];
                MPI_Get(hello_string, HELLO_STR_LEN, MPI_CHAR, i, 0, HELLO_STR_LEN, MPI_CHAR, win);
                MPI_Win_fence(0, win);
                printf("[%d] Get from %d: %s\n", world_rank, i, hello_string);
        }
```

Usage Example on the DEEP-EST SDV

On the DEEP-EST SDV, there is already a special version of ParaStation MPI installed that features all the introduced API extensions. It is accessible via the module system:

```
> module load parastation/5.2.1-1-mt-wp6
```

When allocating a session with N nodes, one can run an MPI session (let?s say with n processes distributed across the N nodes) where each of the processes is contributing its local and persistent memory region to an MPI window:

```
> salloc --partition=sdv --nodes=4 --time=01:00:00
salloc: Granted job allocation 2514
> srun -n4 -N4 ./hello 'Have fun!'
(0) Running on deeper-sdv13
(1) Running on deeper-sdv14
(2) Running on deeper-sdv15
(3) Running on deeper-sdv16
(0) The Window's port name is: shmid:347897856:92010569
(0) Calling finalize...
(1) Calling finalize...
(2) Calling finalize...
(3) Calling finalize...
(0) Calling finalize...
(0) Finalize done!
(1) Finalize done!
(2) Finalize done!
(3) Finalize done!
```

Afterwards, on all the nodes involved (and later on the NAM) one persistent memory region has been created by each of the MPI processes. The ?port name? for accessing the persistent window again is in this example:

```
shmid:347897856:92010569
```

By means of this port name (here to be passes as a command line argument), all the processes of a subsequent MPI session can access the persistent window provided that there is again at least one MPI processes running locally to each of the persistent but distributed regions:

```
> srun -n4 -N4 ./world shmid:347897856:92010569
[0] Running on deeper-sdv13
[1] Running on deeper-sdv14
[2] Running on deeper-sdv15
[3] Running on deeper-sdv16
[0] The Window's port name is: shmid:347897856:92010569
[1] Connection to persistent memory region established!
[3] Connection to persistent memory region established!
[0] Connection to persistent memory region established!
[2] Connection to persistent memory region established!
[0] Number of remote regions: 4
[0] Get from 0: Hello World from rank 0! Have fun!
[1] Get from 0: Hello World from rank 0! Have fun!
[2] Get from 0: Hello World from rank 0! Have fun!
[3] Get from 0: Hello World from rank 0! Have fun!
[0] Get from 1: Hello World from rank 1! Have fun!
[1] Get from 1: Hello World from rank 1! Have fun!
[2] Get from 1: Hello World from rank 1! Have fun!
[3] Get from 1: Hello World from rank 1! Have fun!
[0] Get from 2: Hello World from rank 2! Have fun!
[1] Get from 2: Hello World from rank 2! Have fun!
[2] Get from 2: Hello World from rank 2! Have fun!
[3] Get from 2: Hello World from rank 2! Have fun!
[0] Get from 3: Hello World from rank 3! Have fun!
[1] Get from 3: Hello World from rank 3! Have fun!
[2] Get from 3: Hello World from rank 3! Have fun!
[3] Get from 3: Hello World from rank 3! Have fun!
[0] Calling finalize...
[1] Calling finalize...
[2] Calling finalize...
[3] Calling finalize...
[0] Finalize done!
[1] Finalize done!
[2] Finalize done!
```

```
[3] Finalize done!
```

Advice to users

Pleases note that if not all persistent memory regions are covered by the subsequent session, the ?connection establishment? to the remote RMA window fails:

```
> srun -n4 -N2 ./world shmid: 347897856:92010569
[3] Running on deeper-sdv14
[1] Running on deeper-sdv13
[2] Running on deeper-sdv14
[0] Running on deeper-sdv13
[0] The Window's port name is: shmid:347930624:92010569
[0] ERROR: Could not connect to persistent memory region!
application called MPI_Abort(MPI_COMM_WORLD, -1)
?
```

Cleaning up of persistent memory regions

If the connection to a persistent memory region succeeds, the window and all of its memory regions will eventually be removed by the MPI_Win_free call of the subsequent MPI session (here by world.c) at least if not deep_mem_persistent is passes again as an Info argument. However, if a connection attempt fails, the persistent memory regions still persist.

For explicitly cleaning up those artefacts, one can use a simple batch script:

```
#!/bin/bash
keys=`ipcs | grep 777 | cut -d' ' -f2`
for key in $keys ; do
       ipcrm -m $key
done
```

Advice to administrators

Obviously, a good idea would be the integration of e such an automated cleaning-up procedure as a default into the epilogue scripts for the jobs.