Active Messages for OpenSHMEM

Siddhartha Jana

Department of Computer Science,
University of Houston,
Houston, Texas
sidjana@cs.uh.edu

Abstract. This document introduces the concept of Active Messages within OpenSHMEM. The Active Message interface provides a medium for the executing PE to launch tasks on a remote PE. These tasks may be used for performing computation or dara transfers on the remote PE.

1 Introduction and Related Work

There exists multiple communication libraries that support active messages. Examples include GASNet, IBM's LAPI and DCMF, Myrinet's MX, POOMA/CHEETAH as well as application layers like PBGL. There have also been proposals of introducing this concept into MPI.

2 Terminology

3 Proposed API

3.1 Registration of Active Message Handlers

shmemx_am_attach Enables the calling PE to register a handler with the OpenSHMEM implementation. This is a collective operation. The prototype of the handler differs slightly based on the type of communication model being used.

void shmemx_am_attach (int handler_id, shmemx_am_handler function_handler)

- *handler_id* The integer Id used to identify an AM handler
- *function_handler* A pointer to the function that holds the body of the AM handler. The signature of the function handler is:
  - For 1-sided Active Messages:
    *void function_name (void *buf, size_t nbytes, int req_pe)*
  - For 2-sided Active Messages:
    *void function_name (void *buf, size_t nbytes, int req_pe, shmemx_am_token_t token)*
  - Where,
    - **buf* The pointer to the user buffer being transferred to the remote PE *req_pe*.
    - *nbytes* Size of the user buffer *buf*
    - *req_pe* Id of the remote PE that will execute the handler

shmemx_am_detach A call to this function removes the mapping between the handler id and the function. This is a collective operation. Once detached, it is illegal for a PE to initiate an active message with the same function handler id unless it explicitly maps the id again using shmemx_am_attach.

void shmemx_am_detach (int handler_id);

- *handler_id* This is the handler id of the function that needs to be deregistered with the underlying implementation.

3.2 Initiating Active Messages

shmemx_am_launch This function is used to launch an active message on a remote PE. There is no guarantee of completion of execution of the handler on function return. The source buffer can be reused on return.

void shmemx_am_launch (int dest, int handler_id, void* source_addr, size_t nbytes)

- *dest* Id of the remote PE that will execute the AM handler
- *handler_id* Id of the handler that is executed at the remote PE.
- *source_addr* Start address of the user buffer that is passed to the AM handler
- *nbytes* Size of the user buffer *source_addr*

shmemx_am_request In a two-sided request-reply communication model, this function is used to launch a request AM handler on the remote PE. Typically, the request handler in turn initiates a reply AM handler on the current PE.

void shmemx_am_request(int dest, int handler_id, void* source_addr, size_t nbytes)

- *dest* Id of the remote PE that will execute the AM handler
- *handler_id* Id of the handler that is executed at the remote PE.
- *source_addr* Start address of the user buffer that is passed to the AM handler
- *nbytes* Size of the user buffer *source_addr*

shmemx_am_reply In a two-sided request-reply communication model, this function is used by the request AM handler to launch a reply AM handler at the remote PE that had originally initiated the executing handler on the current PE.

void shmemx_am_reply(int handler_id, void* source_addr, size_t nbytes, shmemx_am_token_t temp_token)

- *handler_id* Id of the handler that is executed at the remote PE.
- *source_addr* Start address of the user buffer that is passed to the AM handler
- *nbytes* Size of the user buffer *source_addr*
- temp_token This is the token passed to the executing request handler. This token is passed as is to the reply handler.

3.3 Completion of Active Messages

shmemx_am_quiet This function waits till the completion of all active messages initiated by the current PE.

```
void shmemx_am_quiet()
```

4 Impact on existing OpenSHMEM implementation

5 Possible Extensions

6 Microbenchmarks

7 Conclusion

8 Acknowledgments

The author is to be solely blamed for this report. This report is not exhaustive and does NOT take into account all the challenges that arise in introducing the concept of Active Messages in all the architectures that support OpenSHMEM.