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 a PE to launch tasks on a remote PE. These tasks may be used for performing computation or data transfers on the remote PE.

1 Introduction and Related Work

There exists multiple communication libraries that support active messages. Examples include GASNet [1] and IBM's LAPI [5] and PAMI [4]. These interfaces provided by these libraries are very low level and makes in challenging to be incorporated in HPC applications.

1.1 Active Messages in MPI

Implementations There have been multiple different approaches of adding active messaging support to MPI. These implementations have either been built directly on top of MPI or as additional service threads that run in parallel to the actual MPI processes. Examples of the former include AM++ [6] and AMMPI [2]. Recent work by Zhao et al. describes an approach of using an additional internal thread to support asynchonous progress of active messages within MPICH [7]. This implementation is integrated to the progress engine that supports one-sided communication in MPICH.

Proposed Interface MPI already has an *Accumulate* function that allows users to execute pre-defined computations on a remote MPI rank. Zhao et al. extend this function to allow remote execution of user-defined functions. The user-defined function accepts pointers to the input and output buffers. When called, the function can obtain the input data from the input buffer and is responsible for writing the result of the computation back to the output buffer. The active message interfaces also contain specific interfaces to initiate an a message (am_send), *collectively* register and deregister user defined function calls. An alternative function model that has been proposed is to allow registration of a single user-defined function. This function - called the head handler - acts as a gateway to all other local functions. The rationale behind this is to enable this single ”middle-man” function on the remote process to efficiently handle incoming data buffers of multiple data types and stride counts. Some of the additional design considerations for active messages in MPI have been discussed by Hoefler et al. [3].

2 Terminology

3 Proposed API

3.1 Registraction 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 the challenges that arise in introducing the concept of Active Messages in all architectures that support OpenSHMEM. There are some important lessons to be learned from the GASNet community.

References

1. Bonachea, D.: GASNet specification, v1.1, University of California, Berkely, Tech. Rep. CSD-0201207 (October 2002)
2. Bonachea, D.: AMMPI: Active Messages over MPI - Quick Overview, `http://www.cs.berkeley.edu/{~}bonachea/ammpi/`
3. Hoefler, T., Willcock, J.: Active Messages for MPI (2009)
4. Kumar, S., Mamidala, A.R., Faraj, D.A., Smith, B., Blocksome, M., Cernohous, B., Miller, D., Parker, J., Ratterman, J., Heidelberger, P., Chen, D., Steinmacher-Burrow, B.: Pami: A parallel active message interface for the blue gene/q supercomputer. In: Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International. pp. 763–773 (May 2012)
5. Shah, G., Nieplocha, J., Mirza, J., Kim, C., Harrison, R., Govindaraju, R.K., Gildea, K., DiNicola, P., Bender, C.: Performance and experience with lapi-a new high-performance communication library for the ibm rs/6000 sp. In: Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998. pp. 260–266 (Mar 1998)
6. Willcock, J.J., Ave, S.W., Edmonds, N.G., Lumsdaine, A.: AM ++ : A Generalized Active Message Framework (2010)
7. Zhao, X., Buntinas, D., Zounmevo, J., Dinan, J., Goodell, D., Balaji, P., Thakur, R., Afsahi, A., Gropp, W.: Toward asynchronous and MPI-interoperable active messages. Proceedings - 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2013 pp. 87–94 (2013)