Asymmetric Memory Extension for Openshmem

Latchesar Ionkov
Los Alamos National Laboratory
Los Alamos, NM 87545
lionkov@lanl.gov

Ginger Young
Los Alamos National Laboratory
Los Alamos, NM 87545
gingery@lanl.gov

ABSTRACT

Memory allocation in Openshmem is a global operation that requires in-step calls from all Processing Elements (PEs). Although this approach works with applications that split the work evenly, it prevents using Openshmem in cases where the workload and the memory it uses are allocated dynamically and can change significantly while the application is running. To broaden the cases where Openshmem can be used, we propose an extension – asymmetric memory support. The extension allows PEs to allocate memory independently, and make this memory available for remote access from other PEs.

1. INTRODUCTION

Openshmem [1] is a specification of an API for one-sided communication in a Partitioned Global Address Space (PGAS) environment. It provides a small set of operations that allow Remote Memory Access (RMA) as well as atomics and collectives. Openshmem application consists multiple threads of execution called Processing Elements (PEs), running on multiple nodes of a supercomputer. To access remote memory, the application needs to specify the PE identifier(s) as well as address(es) of memory. Openshmem defines two categories of memory: local memory inaccessible to other PEs and symmetric memory. Symmetric memory can be used by remote PEs to read, write, or perform atomic or collective operations. All global variables defined in an Openshmem program are located in the symmetric memory. In addition, Openshmem provides a function that can dynamically allocate symmetric memory. The function is global, and all PEs need to call it at the same time and with the same parameters. The operation allocates identical memory regions on all PEs, and uses the address of the locally allocated region as an identifier for the remote regions.

Symmetric memory allocation works well for code that splits a problem evenly between all PEs, but won't work well in more heterogeneous environments, where the problems are more asymmetric, or the hardware configuration of the supercomputer's nodes is not homogeneous.

We propose an extension to Openshmem that adds an additional type of memory – asymmetric memory. Asymmetric memory is allocated the same way as local memory is, by using the malloc function or an equivalent. The memory can be *registered* for remote access by other PEs. The PEs can use the tuple (PE, local-address) to access the registered memory.

2. DESIGN

Our main design principle was keeping the extension close to the Openshmem's design philosophy. The return value `mem` of the Openshmem's symmetric memory allocation function (shmalloc) is used for two different purposes. Firstly, it is the address of a local memory region that can be accessed by the local PE directly, without using the Openshmem operations. Secondly, a value between `mem` and `mem+size` can be used to specify a remote memory location as a source or destination of an RMA. We decided to keep the same convention and use the local address of the asymmetric memory region to specify the location for the RMAs. The global nature of the symmetric memory allocation has an important advantage – all PEs know the address to use in order to access remote symmetric memory. For asymmetric memory, the PE that allocated it needs to use separate mechanisms to pass the address to the remote PEs.

Instead of defining an asymmetric memory equivalent of `shmalloc`, we decided to give more freedom to the developers on how to allocate the asymmetric memory regions. The main reason for that was to allow the users to use the `mmap` system call to allow direct access to files from remote PEs. We assume that the asymmetric memory region will be allocated first (by using `malloc`, `new`, or `mmap`) and then registered for remote access.

The ultimate goal is to use the standard Openshmem operations for accessing both symmetric and asymmetric memory. For start, we added separate operations that do RMA access to asymmetric memory, but we expect to remove them and use the standard Openshmem calls.

Our extension defines the following operations:

`init()`	Initializes the asymmetric memory support.

`register(p, sz)`	Registers asymmetric memory region with starting address `p` and size `sz`;
`unregister(p)`	Unregisters asymmetric memory region with starting address `p`;
`put(t, s, n, p)`	Transfers data from local memory `s` to remote asymmetric memory `t`. The remote address `t` needs to be within a registered asymmetric region on remote PE `p`.
`get(t, s, n, p)`	Transfers data from remote asymmetric memory `s` to local memory `t`. The remote address `s` needs to be within a registered asymmetric region on remote PE `p`.

3. IMPLEMENTATION

We based our code on the Openshmem implementation that uses UCCS [2] for low-level communication. In order to perform remote access to another node's memory, the PE needs to know the *remote handle* of the memory. The remote handle is a opaque value of type `uccs_remote_reg_handle_t`, that is provided by UCCS when the `uccs_register_memory` function is called.

The main problem that our implementation needs to solve is how to provide that remote handles to the other PEs, so they can use them for the RMA. We use a region of symmetric memory to build a translation table from local addresses to remote handles. Each PE keeps a copy of every other PEs table in its local memory.

When a PE registers a new region of asymmetric memory, a new entry is added to its table. The entry contains the starting address of the region, its size, and its remote handle. Then the PE can use any mechanism to pass the address of the asymmetric memory to other PEs. When another PE tries to do RMA to an asymmetric memory region, it checks its local copy of the remote PEs table. If it finds an entry that describes the required address, it uses the remote handle from that entry. If no entry is found, the PE uses the standard Openshmem operation to retrieve an up-to-date copy of the table from the remote PEs symmetric memory, and the table is searched again for match. If there is still no match, an error is returned. If a UCCS operation with the cached remote handle fails, we assume that the remote handle might be invalid (for example by unregistering the memory region) and update the local table before retrying the operation.

In order to speed up the address-to-handle lookup, we keep the entries in the table sorted by the starting address of the region. This approach allows us to check a match by using binary search in O($log_2(N)$) comparisons.

4. PERFORMANCE

We evaluated the performance by comparing accessing asymmetric memory to the standard Openshmem operations accessing symmetric memory. Our test runs on 256 nodes, one PE per node, and performs one million RMA operations (approximately half of them puts and half gets) to PE0's memory. We allocate a single symmetric region, and vary the number of asymmetric regions registered to measure the slowdown of the table lookup.

Figure 1 shows the performance results for RMA with block size of 1 KB. There is significant slowdown for large number of registered asymmetric areas.

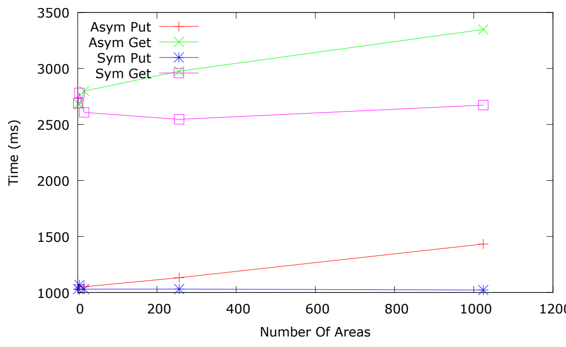

Figure 1: Time to perform 1 million RMA operations

5. CONCLUSION

We have demonstrated an initial implementation of asymmetric memory for Openshmem that would allow for more flexible Openshmem applications, and better utilization of heterogeneous environments. We plan to extend the operations provided to include atomics, and if possible, collectives.

Acknowledgments

This work was performed at the Ultrascale Systems Research Center (USRC) in the High Performance Computing Division at Los Alamos National Laboratory, and is supported by the U.S. Department of Defense. The publication has been assigned the LANL identifier LA-UR-14-27689.

6. REFERENCES

[1] S. W. Poole, O. Hernandez, J. A. Kuehn, G. M. Shipman, A. Curtis, and K. Feind. Openshmem-toward a unified rma model. In *Encyclopedia of Parallel Computing*, pages 1379–1391. Springer, 2011.

[2] P. Shamis, M. Venkata, J. Kuehn, S. Poole, and R. Graham. Universal common communication substrate (uccs) specification. Technical report, Version 0.1. Tech Report ORNL/TM-2012/339, Oak Ridge National Laboratory, ORNL, 2012.