Development and Extension of Atomic Memory Operations in OpenSHMEM

Pavel Shamis, Manjunath Gorentla Venkata, Stephen W. Poole
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831-6173
{shamisp,manjugv,spoole}@ornl.gov

Swaroop Pophale, Michael Dubman, Richard Graham, Dror Goldenberg, Gilad Shainer
Mellanox Technologies
Sunnyvale, California 94085
{swaroop, miked, richardg, gdror, shainer}@mellanox.com

ABSTRACT

A distinguishing characteristic of OpenSHMEM compared to other PGAS programming model implementations is its support for atomic memory operations (AMOs). It provides a rich set of AMO interfaces supporting 32-bit and 64-bit datatypes. On most modern networks, network-implemented AMOs are known to outperform software-implemented AMOs. So, for achieving high-performance, an OpenSHMEM implementation should try to offload AMOs to the underlying network hardware when possible. Nevertheless, the challenge arises when (a) underlying hardware does not support full set of atomic operations, (b) more that one device is used, and (c) heterogeneous systems with multiple types of devices are involved. In this paper, we analyze the challenges and discuss potential solutions to address these challenges.

Categories and Subject Descriptors

C.2.6 [**COMPUTER-COMMUNICATION NETWORKS**]: Internetworking—*Standards*

1. INTRODUCTION

OpenSHMEM [1] is a de-facto standard for SHMEM libraries that defines the library API, its behavior, and functionality. While the concept of SHMEM was introduced before the Partitioned Global Address Space (PGAS) model was formally introduced, the SHMEM memory model fits the PGAS programming model. Each processing element (PE) manages a partition of the symmetric memory heap used for symmetric data object allocations. Memory allocations within the symmetric heap can be accessed through a rich set of remote memory access (RMA) operations including AMOs. The unique characteristic of AMOs is the fact that the target memory is updated atomically with respect to other AMOs. In other words, at any given moment, only a single PE may access (read or write) the target memory through the AMOs. It worth noting that atomicity of memory updates is not guaranteed with respect to any other OpenSHMEM operations.

The OpenSHMEM AMOs include two major classes of operations: the fetch (blocking) operations that return the original value and the non-fetch operations. The fetch operations include fetch-and-add (FADD), fetch-and-increment (FINC), swap (SWAP), and compare-and-swap (CSWAP). All the above operations atomically update the remote symmetric data object and fetch the original value stored in the specified symmetric data object. The non-fetch operations include add (ADD) and increment (INC). The non-fetch operations atomically update the remote symmetric data object without fetching the original value. All operations defined for 32 and 64 bit integer datatypes.

Hardware components like CPU, PCIe, and network devices such as InfiniBand HCA, Cray's Gemini/Aries ASIC etc. typically provide some AMOs. But in the current High-Performance Computing (HPC) systems leveraging and coordination of these operations between hardware components is a non-trivial task. In this paper we discuss certain challenges faced by OpenSHMEM library implementers while providing performance driven AMO implementations.

The rest of the paper is organized as follows: Section 2 discusses the challenge associated with implementation of the AMOs. Section 3 proposes solutions that aim to address the challenges and Section 4 evaluates performance of one of the proposed solutions. Section 5 summarize the paper and concludes the discussion.

2. THE CHALLENGE

In the section we discuss in detail each one of the challenges outlined in the abstract.

2.1 Underlying hardware does not support full set of operations.

If the underlying hardware does not support all the AMOs defined by the OpenSHMEM specification, the implementation has to fallback to a common mechanism that handles all AMOs. The common mechanism may provide implementations for all AMOs and ignore the operations that are supported in hardware or implement some logic that synchronizes AMOs between different types of hardware and

software. While the last option might be the most desirable, to the best of our knowledge, most OpenSHMEM implementations fallback to the software based implementation of AMOs for all operations because some operations are not possible in hardware. As a result, a single unsupported AMO in hardware may result in a fallback to subpar software AMO implementation for all operations supported by OpenSHMEM. In most cases, software implementations introduce additional overheads, which discourages the usage of AMOs by application developers. Therefore, the OpenSHMEM community should be cautious when adding new AMOs to the API, since it may negatively affect the usability of already supported operations.

2.2 More than one device is used

When more than a single hardware device of the same type is used for node connectivity in an HPC system, the devices have to coordinate the AMO execution. While in theory, it is possible to implement such coordination, most popular interconnects do not support it. As a result, OpenSHMEM implementations have to either re-route all AMOs through a single device, or, similar to the case described in 2.1, fallback to a subpar software implementation.

2.3 Heterogeneous systems with multiple types of devices

This challenge is most relevant for most of today's systems since majority of the machines provide different communication path for inter-node and intra-node communication. Intra-node communication is typically implemented through shared memory that by-passes the inter-node communication path. As a result, OpenSHMEM implementations choose to fallback to common AMO mechanism, which is implemented in software. Alternatively, similar to challenge described in 2.2, the implementation may select one communication path over another. For example, some OpenSHMEM implementations redirect all AMOs through InfiniBand device and bypass shared memory optimizations.

The common denominator for the above challenges is the fact that OpenSHMEM implementation have to switch to a common AMO mechanism. This typically result in a software based implementation of AMOs and slower performance. In the following section 3 we discuss potential solutions that aim to mitigate the issue.

3. POTENTIAL SOLUTIONS

The OpenSHMEM library implementation can go through hardware atomic path, or choose to employ the software implementation of the atomic. The main challenge is to decide when to fallback to the suboptimal software implementation. In this section we discuss possible approaches that could be used.

3.1 Hints by the Programmer

With new innovations in hardware, OpenSHMEM library developers may be able to develop faster AMOs by offloading certain functionality to the hardware rather than relying on suboptimal software solutions. A programmer could declare ahead of time which AMOs the application uses. As we know, InfiniBand specification does not define the SWAP operation nor support 32bit AMOs. If the application notifies the OpenSHMEM library that the unsupported operations are not used by the application, the library may switch to offloaded AMOs, otherwise software based implementation is used. This could be incorporated with a slight change to the OpenSHMEM library initialization. The proposed solution aims to solve the challenge 2.1, and it helps mitigate the effect of challenges 2.2 and 2.3.

The drawback of this approach is that the application developer has to review the application and identify all AMOs used. Moreover, such approach does not guarantee that all AMOs will be offloaded to hardware, even so, the OpenSHMEM library will have more information in order to make optimal selection of an AMO implementation. This approach requires only minimal modifications from an application perspective and provides opportunities for different OpenSHMEM library optimization.

3.2 Hints by the OpenSHMEM library implementation

Another approach that might be helpful is if the specific OpenSHMEM library implementation exposes information about the quality or nature of the AMOs (or any other operations) implementation available. For instance, if the implementation does not provide high-performance SWAP operation, the application programmer may choose to replace it with some other equivalent operation. Similar to the approach discussed before, the solution aims to solve challenge 2.1 and alleviates the effect of challenges 2.2 and 2.3.

This approach simplifies the OpenSHMEM library development and potentially enables efficient utilization of underlaying hardware capabilities. With this approach the burden of AMO management is upon the application developers, who now have to understand the technical details of the underlaying OpenSHMEM implementation. This may not be a portable solution either.

3.3 Manage a different memory store for each variation of AMOs

The central implication of all the challenges is that the atomicity cannot be guaranteed when the AMOs are provided by different implementations (providers). i.e. a complete set of AMOs are provided by different network devices, by a combination of CPU and network hardware, or a combination of hardware and software. This approach ensures atomicity by using a different memory store for each of the providers.

This approach assigns a different symmetric heap for each atomic provider. Where a complete set of operations are provided by a combination of software and hardware implementation, the OpenSHMEM library will allocate a separate memory regions for hardware AMOs, and a separate memory regions for software AMOs. Similarly, for the challenge 2.3 where the PEs are connected using different networks, the library manages a separate memory region for each different type of the network.

This approach ensures that AMO performance is not degraded when there are multiple AMO providers. For example, in an OpenSHMEM job, where PEs are connected through a shared-memory and InfiniBand networks, if a par-

tial set of OpenSHMEM AMOs are faster with CPU implementation and another partial set of OpenSHMEM AMO are faster on InfiniBand, this approach allows the application to use them both. In contrast to the above solutions, the proposed approach solves challenges 2.1, 2.2, and 2.3.

The primary drawback of this approach is that it delegates the complexity of AMO's management to an application developer. The developer is responsible to enquire about different groups of AMOs, allocate and manage AMOs over a particular region of memory.

3.4 Using existing hardware AMOs as building blocks

Another alternative for software based collectives could be an implementation that leverages already existing AMOs to implement operations that are not directly supported by hardware. For example, OpenSHMEM-UCCS [2][3], which is a high-performance implementation of the OpenSHMEM specification, implements the SWAP operations and 32 bit AMOs for Mellanox InfiniBand and Cray Gemini interconnects through 64bit CSWAP operation that is supported in hardware.

The initiator of the operation atomically reads the remote value, based on an operation updates the value, and executes compare-and-swap (CSWAP) using the originally fetched value and the updated value. If the operation fetches a value that is identical to the one that was fetched in the first step, the update is considered successful. Otherwise, the update fails and the algorithm repeats previous steps. For the rest of the paper we refer to the algorithm as the *hardware-software* algorithm. Using the above algorithm the OpenSHMEM-UCCS implementation addresses the challenge described in 2.1.

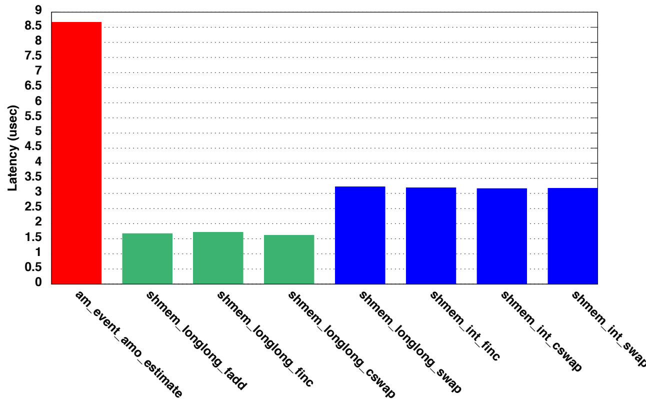

Figure 1: Evaluation of OpenSHMEM-UCCS AMOs

4. EVALUATION

The Figure 1 shows a comparison between blocking (fetch based) AMOs supported in hardware, AMOs implement with the above algorithm, and software based AMOs. The evaluation was conducted on a HP ProLiant DL380p system located at the Oak Ridge National Laboratory's Extreme Scale System Center. The system consists of two compute nodes, each with two Intel Xeon E5-2650 CPUs, for a total of 16 CPU cores and 32 threads. Compute nodes are interconnected with Mellanox ConnectX-3 VPI HCA connected back-to-back (no switch). The system runs CentOS release 6.5 with MLNX-OFED-2.2-1.5.5 and OpenSHMEM-UCCS v0.3.

The performance of AMOs was measured using OSU OpenSHMEM Atomic Test v4.4. It is worth noting that this test does not evaluate AMOs performance under contention, but reports the base-line (most optimized) performance for each operation; this is aligned with the goal of our investigation. In this evaluation we focused on blocking (fetched based) AMOs, since this type of operations are affected most by the quality of the AMO implementation and measure the full round trip communication latency. The green bars represent AMOs implemented in hardware (64bit/longlong FADD, FINC, CSWAP), the blue bars represent AMOs implemented within OpenSHMEM-UCCS using the *hardware-software* algorithm (64bit/longlong SWAP and 32bit/int FADD, FINC, CSWAP, SWAP), and the red bar represents a simulation software based AMO using ib_send_lat benchmark. For the simulation we calculated a round trip latency, where an initiator sends an active message (InfiniBand send work request) with a AMO operation descriptor and the receiver handles the active message receive event though a thread (we assuming one-sided implementation of atomics) followed by the reply message with a fetched value. The initiator receives the reply using busy-loop pull on a completion queue. Since the simulation is realized using InfiniBand VERBS interface, it represents close-to-hardware (optimal) performance for this type of AMO implementation. In context of the paper we do not consider AMO implementations that require explicit progress from the application level. These type of implementations are useless from the application's perspective.

As seen in Figure 1, the *hardware-software* AMOs (blue bars) are about two times slower than the hardware based AMOs (green bars). This is expected, since for the best case scenario (no contention) the *hardware-software* algorithm issues two AMO requests (READ and CSWAP), where hardware based implementation issue a single request. Moreover, as expected, *hardware-software* operation has no side-effect on an AMO implemented with hardware. The software based implementation is about x5.3 times slower compared to hardware AMOs, and x2.7 slower compared to *hardware-software* AMOs. The relative poor performance of software based AMOs is a result of overhead introduced by asynchronous event handling on the destination PE.

5. CONCLUSION

The primary motivation of the paper is to highlight the challenges associated with implementing high performance AMOs. These problems will be amplified with introduction of new AMOs in the OpenSHMEM specification. Introduction of new AMOs may have side-effects and negatively affect the performance of already existing AMOs in OpenSHMEM. Since the gap between the software requirement and hardware capabilities is always expected to be present, OpenSHMEM community has to provide a mechanism to enable coexistence of software and hardware based AMOs. In the paper we proposed a few potential solutions that aim, in some measure, to bridge the gap between the software requirement and current hardware capabilities. In addition, the OpenSHMEM community must closely collab-

orate with hardware vendors to ensure that new operations are supported at the hardware level and the disparity between software requirements and hardware AMOs support does not increase.

Acknowledgment

This work is supported by the United States Department of Defense and used resources of the Extreme Scale Systems Center located at the Oak Ridge National Laboratory.

6. REFERENCES

[1] OpenSHMEM specification.

[2] P. Shamis, M. G. Venkata, J. A. Kuehn, S. W. Poole, and R. L. Graham. Universal Common Communication Substrate (UCCS) Specification. Version 0.1. Tech Report ORNL/TM-2012/339, Oak Ridge National Laboratory (ORNL), 2012.

[3] P. Shamis, M. G. Venkata, S. W. Poole, A. Welch, and T. Curtis. Designing a high performance openshmem implementation using universal common communication substrate as a communication middleware. In *OpenSHMEM and Related Technologies. Experiences, Implementations, and Tools - First Workshop, OpenSHMEM 2014, Annapolis, MD, USA, March 4-6, 2014. Proceedings*, pages 1–13, 2014.