Title

Efficient Active Message RMA in GASNet Using a Target-Side Reassembly Protocol (Extended Abstract)

Authors

Hargrove, P

Bonachea, Dan

Publication Date

2019-11-17

DOI

10.25344/S4PC7M

Peer reviewed

Efficient Active Message RMA in GASNet Using a Target-Side Reassembly Protocol

Extended Abstract

Paul H. Hargrove and Dan Bonachea

Computational Research Division, Lawrence Berkeley National Laboratory
Berkeley, CA 94720, USA

gasnet-staff@lbl.gov

***Abstract*—GASNet is a portable, open-source, high-performance communication library designed to efficiently support the networking requirements of PGAS runtime systems and other alternative models on future exascale machines. This paper investigates strategies for efficient implementation of GASNet's "AM Long" API that couples an RMA (Remote Memory Access) transfer with an Active Message (AM) delivery.**

We discuss several network-level protocols for AM Long and propose a new target-side reassembly protocol. We present a microbenchmark evaluation on the Cray XC Aries network hardware. The target-side reassembly protocol on this network improves AM Long end-to-end latency by up to 33%, and the effective bandwidth by up to 49%, while also enabling asynchronous source completion that drastically reduces injection overheads.

The improved AM Long implementation for Aries is available in GASNet-EX release v2019.9.0 and later.

Keywords—Active Messages, RMA, GASNet, PGAS, HPC, Networking, Supercomputing

I. INTRODUCTION

Active Messages (AM) [8] is a communication paradigm often used to implement higher-level communication protocols in HPC networking. In essence, AM is a restricted form of Remote Procedure Call (RPC), where an initiating process sends a point-to-point message to a target process. Upon arrival the AM triggers execution of a piece of handler code, described in the message, which incorporates the payload into the ongoing computation at the target and possibly generates a response message. AM protocols usually impose semantic restrictions which are chosen to improve efficiency of implementation, while still providing the generality needed to build higher-level protocols.

One popular and portable implementation of Active Messages is provided as part the GASNet (Global Address Space Networking) communication layer [4][9]. GASNet has a proven track record of enabling high performance across many interconnects and supporting a wide range of applications and high-level programming abstractions. Notable GASNet clients include: Berkeley UPC [7][11], UPC++ [2], the Legion Programming System [3] and Cray Chapel [6]. Under funding from the Exascale Computing Project (ECP) we are designing and implementing GASNet-EX [5], a second-generation GASNet API, which is focused on exascale requirements and incorporates over 15 years of lessons-learned. GASNet-EX includes a backwards-compatibility layer that allows it to transparently support clients using the legacy GASNet interfaces.

GASNet's AM design is based on Berkeley Active Messages [10] and exposes a strict point-to-point request/reply semantic, with no other communication permitted in handlers. AM's may carry a limited payload that is either delivered into an anonymous buffer (which is recycled upon handler exit), or into a client-owned location specified by the initiator. GASNet refers to the latter operation as *AM Long*, and it is semantically analogous to an RMA Put operation that runs a handler after delivery of the payload. The initiating process is notified only of *local completion*, the point in time when the payload's source buffer is safe to modify. Separate AM Long operations may be reordered in the network, but the implementation must ensure each payload is delivered before executing the corresponding handler. In HPC networks, AM Longs of sufficient size are usually most efficiently implemented using a zero-copy RDMA (Remote Direct Memory Access) for the payload transfer and a separate message for the AM envelope (which carries handler arguments and metadata).

This paper investigates implementation of the GASNet AM Long semantics on the Cray Aries network in Cray XC systems [1]. The uGNI communication layer on Cray Aries provides RDMA Put semantics with completion notification at the initiator and target, and even allows the RMA to deliver a few bits of out-of-band data. Aries notably lacks point-to-point ordered delivery semantics in the highest-performance mode that uses dispersive routing to maximize bandwidth. Consequently, one cannot rely upon wire-level delivery ordering to ensure the RDMA transfer is complete before the envelope message arrival triggers handler execution.

We discuss several AM Long protocol options, and evaluate two implementations: (1) a protocol that stalls the initiator for global completion of the RMA Put before injection of the AM envelope, and (2) a target-side reassembly protocol that pipelines injection of the RMA Put and AM envelope, using matching logic at the target to enforce correct ordering. These protocols generalize to implementation of AM Long semantics on other RDMA-based HPC networks with similar properties.

II. Protocol Options

Relevant measures of efficiency in implementing AM Long include not only standard benchmarks such as "ping-pong latency" and "flood bandwidth", but also metrics like the overhead (CPU time spent within the library) of message injection and the sensitivity to attentiveness. A good algorithm should spend as little time in message injection as practical without deferring necessary work to later library calls to make progress (the frequency of which we called *attentiveness*). Since time spent stalled at injection, or between multiple phases of progress, is time not spent communicating, improving these measures correlates well with improvement to the latency and bandwidth benchmarks. Additionally, prompt signaling of local completion at the initiator is desired to allow the client to reuse the source buffer. With these metrics in mind, this section describes four different algorithms.

On ordered networks, implementation of AM Long can be as simple as injection of independent transfers for the two elements; with the assurance that the arrival of the second is both necessary and sufficient to allow the target to run the appropriate AM handler. This approach has low overhead, is insensitive to initiator attentiveness, and can provide timely signaling of local completion. However, while uGNI supports an in-order delivery mode, its use disables use of the multiple paths inherent in the Aries network. While it is a good choice on some networks, we do not consider this approach appropriate on Aries due to the associated reduction in performance (bandwidth and latency) of large payload transfers.

While Aries does not provide desirable means for ordering transfers, it is possible for the AM Long implementation to enforce the necessary semantics by using a class of protocols we call *initiator chaining*, which has synchronous and asynchronous variants. The synchronous variant initiates the payload RMA transfer, and then stalls to await global completion (the point in time when the RMA transfer is known to be fully committed to the remote memory system). Only once the initiator knows the payload transfer is complete does it send the envelope. Prior to the work described here, GASNet's aries-conduit (through release v2019.6.0) used synchronous initiator chaining, also referred to as *put-sync-send*. The asynchronous variant of initiator chaining returns after injecting the payload, and enqueues the envelope to be sent by the progress engine at some point after global completion of the RMA. These variants share the property that the transfer of payload and envelope cannot be pipelined. While the asynchronous variant does permit overlap of *other* communication, it is a poor candidate for overlap of computation, which may reduce the attentiveness of the injector and thus delay injection of the envelope.

A notable feature of AM Long is that the initiator knows the addresses for both the source and destination of the payload transfer. While other algorithms use this to perform an RMA Put from the initiator, one can alternatively send both addresses in the envelope and initiate an RMA Get from the target. In such a *rendezvous get* algorithm, the initiator overhead is minimal (injecting only an envelope). However, the need for the target to receive the envelope and initiate a Get for the payload significantly increases the latency of isolated messages (as in a ping-pong benchmark) and leads to a delay in signaling of source buffer completion at the initiator. These delays equal a network round trip time in the best case. In the worst case, inattentive targets not only lengthen these delays, but can also consume resources at the injector, leading to head-of-line blocking. For these reasons, we do not consider rendezvous get to be a desirable protocol where alternatives are available.

A fourth protocol and the subject of this work is what we refer to as *target-side reassembly*. This algorithm has the initiator inject the envelope and payload transfers back-to-back without concern for what order they arrive at the target. At the target, the payload and envelope are matched (as described below) and the AM handler runs only when both elements have arrived. Relative to initiator chaining, one expects back-to-back injection to reduce latency and increase bandwidth. The overhead of injection is less than the synchronous variant of initiator chaining, and comparable to that of the asynchronous variant (including the payload and envelope injections separated in time). Use of an appropriate RMA Put protocol for the payload allows use of multiple paths through the network and provides for timely signaling of local completion at the initiator.

The key to the viability of target-side reassembly for AM Long is the ability for the target to match the envelope and payload to ensure the AM handler is run only after both elements are present. It is generally safe to assume that the mechanism chosen for transfer of the envelope can carry any additional metadata required for matching (although in Aries we leverage an existing header field). The potentially more difficult concern is conveyance of matching metadata within the RMA payload transfer. At a minimum, the target must receive some notification (as through a uGNI completion queue) that the payload has arrived, and it must be sufficient to identify the source rank. To eliminate idle time in the protocol, additional bits of metadata are desirable in the target notification to distinguish multiple AM Longs arriving pipelined from the same initiator (potentially out of order). It is the challenge of providing this additional metadata through the Cray uGNI API that has deferred implementation until now.

III. Target-side Reassembly Over Cray uGNI

As mentioned above, the key challenge faced in this work was to transfer a few bits of additional metadata together with an RMA payload transfer. There are two candidate mechanisms in the uGNI API. One is put-with-sync-flag in which a single API call transfers two payloads to distinct addresses with an in-order guarantee, but with several restrictions. The most significant restriction is a limit on payloads to 1MB or less, which we considered unacceptable. The other option is the 32-bit field known as the instance ID, which is a part of the target-side completion queue event. Normally this field simply contains the source rank. However, the initiator is permitted to dynamically modify the contents of this field. Since support for 2^{32} processes is unrealistic for multiple reasons, we have dedicated some of

the upper bits to carry a "nonce" used for the reassembly. An encoding scheme (beyond the scope of this extended abstract) is chosen for the nonce that ensures the matching envelope and payload events are uniquely distinguishable from other unrelated events that may concurrently be present in the same completion queue.

At the target, the progress engine processes completion queue events for the arrivals of AM Long payloads and envelopes. A simple chained-bucket hash table is used to match the second arrival to the first, and the AM handler executes only when a match is found (and the entry removed from the hash table). This matching occurs within the scope of an existing mutex-protected critical section, and thus did not require any additional synchronization for thread-safety.

The nonce value must be unique per source from at least the time the AM is injected into the network until the matching is complete. Realistically, the lifetime extends until the (possibly implicit) AM Reply from target to initiator arrives, allowing the initiator to recycle the nonce value. The scheme for buffer management in aries-conduit already had an identifier meeting the requirements for use as a nonce, including lifetime and fitting in the available bits. Therefore, no additional logic was required to manage the nonce, no additional space was required in the envelope, and no additional communication was required to retire a nonce value.

IV. Performance Evaluation

This section reports briefly on the improvement in performance obtained by replacing the put-sync-send protocol with the target-side reassembly protocol in the GASNet implementation of AM Long for Aries. Three metrics are presented: bandwidth, latency, and overhead. Measurements were taken on NERSC's Cori Phase II, a Cray XC with 1.4GHz Xeon Phi CPUs. The relatively slow CPUs of this platform accentuate the cost of added matching logic at the target, meaning that other XC systems with more powerful CPUs may exhibit even larger improvements. The reader is referred to the Reproducibility Appendix for additional information regarding the system and experimental methodology.

Figure 1 shows the bandwidth of the two protocols as measured by a round-trip ping-pong (Request + Reply) test, as well as their ratio (dotted blue series). The largest improvement is a 49% increase at a 4KiB payload size. Measurements of other patterns including one-way and bidirectional flood patterns are qualitatively similar.

Inverting the bandwidth data for the ping-pong pattern in Figure 1 yields a measure of roundtrip AM Long latency, shown in Figure 2. With a 4KiB payload, the latency is reduced from 13.0us for put-sync-send to just 8.7us for target-side reassembly (a 33% reduction). As the payload size increases as high as 1MiB the latency reduction varies in the range 3.7us to 6.1us.

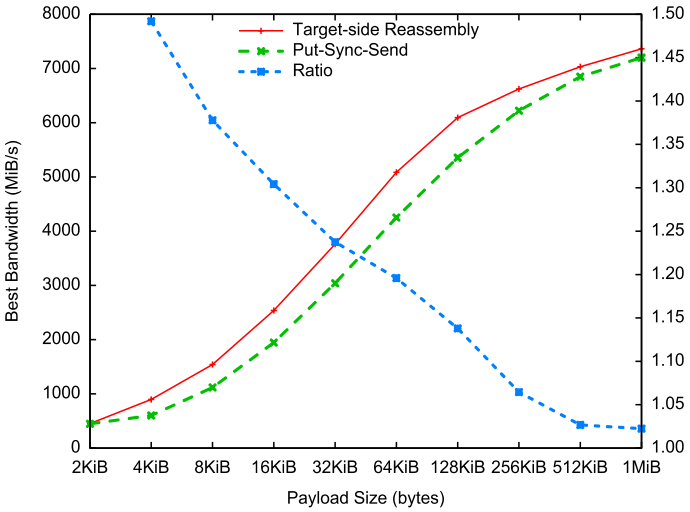

Figure 1. AM Long Ping-Pong Bandwidth Comparison

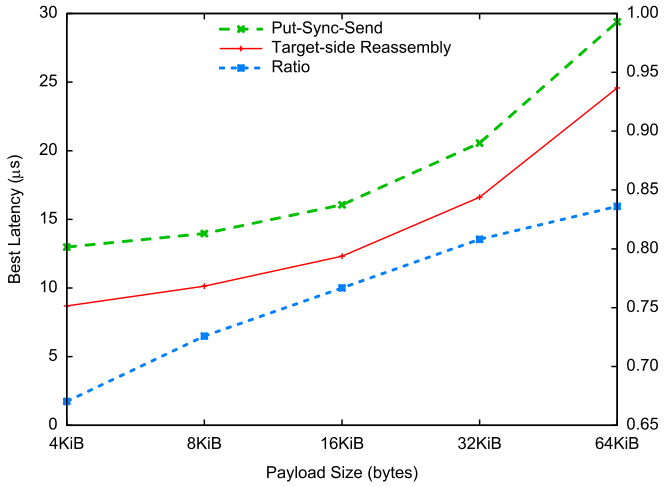

Figure 2. AM Long Roundtrip Latency Comparison

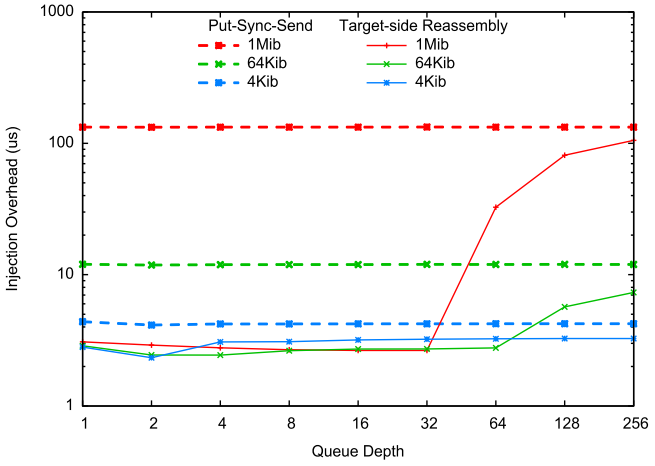

Figure 3. AM Long Injection Overhead Comparison

Because put-sync-send stalls AM Long injection until global RMA completion, it did not provide for asynchronous signaling of local completion. However, target-side reassembly is well suited to implementation of asynchronous local completion. Figure 3 shows the injection overhead of the two protocols, at three payload sizes, as the queue depth (number of semantically overlapped operations) is varied for the case of implicit-handle signaling of local completion. The stall for global RMA completion dominates the overhead of put-sync-send, and thus all three corresponding series (solid lines) are essentially flat. However, target-side reassembly (dashed lines) shows greatly improved (nearly constant) injection overhead until encountering flow control back-pressure, at which point the overhead rises relative to shallower queue depths, while remaining uniformly better than the put-sync-send protocol for the same payload size.

V. Conclusions

We've discussed several network-level protocols for AM Long and proposed a new target-side reassembly protocol that allows pipelining of an RMA payload transfer with the AM envelope message on unordered network stacks, using CPU-based matching logic at the target to enforce AM Long's payload delivery semantics. The protocol does not rely upon properties unique to Cray Aries or the uGNI API, and should generalize to other networks that allow RMA Put to deliver a few bits of out-of-band data to the target, via a completion queue, completion call-back, or similar.

Microbenchmark evaluation on the Cray XC Aries network hardware demonstrates the target-side reassembly protocol on this network improves AM Long end-to-end latency relative to put-sync-send by up to 33%, and the effective bandwidth by up to 49%, while also enabling asynchronous source completion that drastically reduces injection overheads.

The improved target-side reassembly protocol is deployed as the default AM Long implementation for aries-conduit in GASNet-EX release v2019.9.0 and later. Future work may investigate deploying this AM Long protocol for additional networks sharing similar properties.

Acknowledgments

This research was funded in part by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] Alverson, B., Froese, E., Kaplan, L., Roweth, D., "Cray XC Series Network". Cray White Paper WP-Aries01-1112, 2012. cray.com/sites/default/files/resources/CrayXCNetwork.pdf

[2] Bachan, J., Baden, S. B., Hofmeyr, S., Jacquelin, M., Kamil, A., Bonachea, D., Hargrove, P. H., Ahmed, H., "UPC++: A High-Performance Communication Framework for Asynchronous Computation". 33rd IEEE International Parallel & Distributed Processing Symposium (IPDPS), 2019. doi:10.25344/S4V88H

[3] Bauer, M., Treichler, S., Slaughter, E., Aiken, A., "Legion: Expressing Locality and Independence with Logical Regions". Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC), 2012. doi:10.1109/SC.2012.71

[4] Bonachea, D., Hargrove, P.H., "GASNet Specification, v1.8.1". Lawrence Berkeley National Laboratory Technical Report (LBNL-2001064), 2017. doi:10.2172/1398512

[5] Bonachea D., Hargrove, P.H., "GASNet-EX: A High-Performance, Portable Communication Library for Exascale". Lawrence Berkeley National Laboratory Technical Report (LBNL-2001174). Languages and Compilers for Parallel Computing (LCPC), 2018. doi:10.25344/S4QP4W

[6] Callahan, D., Chamberlain, B.L., Zima, H.P., "The Cascade High Productivity Language". International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS), 2004. doi:10.1109/HIPS.2004.10002

[7] Chen, W., Bonachea, D., Duell, J., Husband, P., Iancu, C., Yelick, K., "A Performance Analysis of the Berkeley UPC Compiler". Proceedings of the 17th International Conference on Supercomputing (ICS), 2003. doi:10.1145/782814.782825

[8] von Eicken, T., Culler, D.E., Goldstein, S.C., Schauser, K.E., "Active Messages: a Mechanism for Integrated Communication and Computation". Proceedings of the 19th International Symposium on Computer Architecture (ISCA), 1992. doi:10.1145/139669.140382

[9] GASNet Web Page https://gasnet.lbl.gov

[10] Mainwaring A, Culler, D., "Active Message Applications Programming Interface and Communication Subsystem Organization". U.C. Berkeley EECS Technical Report UCB/CSD-96-918, 1996. doi:10.25344/S48C7W

[11] UPC Consortium, "UPC Language and Library Specifications, v1.3". Lawrence Berkeley National Laboratory Technical Report (LBNL-6623E), 2013. doi:10.2172/1134233

REPRODUCIBILITY APPENDIX

A. Abstract

This Appendix describes the methodology for the PAW-ATM19 paper: *Efficient Active Message RMA in GASNet Using a Target-Side Reassembly Protocol*

B. Test Platform

Experimental results were collected on the following Cray XC40 system, in which nodes are connected via Cray Aries network hardware:

NERSC Cori-II
Node: 68-core 1.4 GHz Intel Xeon Phi 7250, 96 GB DDR4 (quad-cache mode)
Compiler: Intel C Compiler, v18.0.1.163
System software: Cray PrgEnv-intel/6.0.4, Cray PE/2.5.15
Batch system: SLURM (srun)

All communication is performed using the GASNet-EX aries-conduit, with a pre-release version of 2019.9.0. The put-sync-send series use the AM Long protocol which was active through release version 2019.6.0, and the target-side reassembly series use the AM Long protocol which is the new default behavior starting in release version 2019.9.0.

C. Methodology for Figure 1: AM Long Ping-Pong Bandwidth Comparison

These results were gathered using the `testam` microbenchmark included in the GASNet distribution. The exact command used for data collection was:

```
srun -n2 -N2 testam 50000 1048576 G
```

These arguments request 50,000 iterations with payload sizes up to 1MiB, and restricts the runs to sub-test "G: AMLong ping-pong roundtrip ReqRep". All runs are point-to-point, using a single process on each of two nodes connected by the network fabric. These runs report average latency and bandwidth from a ping-pong test that consists of one rank sending an AM Long Request of a given size and the recipient sending an AM Long Reply of the same size. The first rank waits to receive the Reply before sending its next Request. This ping-pong exchange is repeated 50,000 times. The test reports the bandwidth at each size as the sum of the payload size of all the Requests and Replies, divided by the elapsed time to complete all of the iterations of that size.

The ping-pong test is performed at the sizes 0 and 1MiB, and all the powers-of-two in between. Since payloads less than the 4KiB data point are transferred using a "packed Long" protocol not described in this paper, the Figure omits data below 2KiB where the two series are identical. The 2KiB data point is included to confirm the lack of performance difference at that point.

Because the Cray Aries network shares resources among distinct jobs, the bandwidth values reported by `testam` are too noisy to make a meaningful comparison between just two runs. Two executables, built from sources differing in the implementation of AM Long, were run in alternating sequence, 22 times each. Each data point shown in Figure 1 is a maximum over 22 runs.

D. Methodology for Figure 2: AM Long Roundtrip Latency Comparison

Figure 2 is an alternate visualization of the same data from Figure 1. Each data point is the quotient of the payload size and the corresponding bandwidth from a data point in Figure 1. Because the results are from a round-trip ping-pong test, the resulting value represents the average round-trip latency time for that payload size. The range of payload sizes shown on the x-axis has been restricted, in order to highlight the most interesting behavior.

E. Methodology for Figure 3: AM Long Injection Overhead Comparison

These results were gathered using the `testqueue` microbenchmark included in the GASNet distribution. The exact command used for data collection was:

```
srun -n2 -N2 testqueue -l -i 1000 256
```

These arguments request 1,000 iterations, a maximum queue depth of 256, and restrict the runs to the sub-test for AM Long (`-l`) with implicit-handle (`-i`) signaling of local completion. All runs are point-to-point, using a single process on each of two nodes connected by the network fabric. These runs report "injection overhead", defined as the time required to initiate communication without waiting for its completion. For a given queue depth (`d`) and payload size (`n`), the test measures the time for the first rank to make `d` consecutive calls to `gex_AM_RequestLong0()` with `n`-byte payload, and `GEX_EVENT_GROUP` to request implicit-handle signaling of local completion. As this is a test of injection overhead, all communication is completed outside the timed region; the initiator waits for local completion, the target waits to have received `d` AM Requests, and then both ranks participate in a barrier. This alternation of timed injection and network quiescence is repeated multiple times (1,000 for the arguments used) and the average time for the injection portion is reported.

The test runs the procedure described in the previous paragraph for queue depth from 1 to 256 and payload sizes from 0 to 2MiB, using the powers-of-two in the respective ranges. The data in Figure 3 reports the data from runs of two executables, built from sources differing in the implementation of AM Long, run back-to-back in the same job. Because injection time is relatively insensitive to interference in the network, results from single runs are used.

Figure 3 reports data for three payload sizes: 4KiB, 64KiB and 1MiB. The data for payload sizes between these three is qualitatively similar: monotonically non-decreasing as either size or queue depth increases. Since payloads less than 4KiB are transferred using a "packed Long" protocol not described in this paper, the data at such sizes is not presented.