GASNet-EX Performance Improvements Due to Specialization for the Cray Aries Network

Paul H. Hargrove and Dan Bonachea

Computational Research Division,

Lawrence Berkeley National Laboratory

Berkeley, CA 94720, USA

gasnet-staff@lbl.gov

Abstract—**GASNet-EX 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 reports on the improvements in performance observed on Cray XC-series systems due to enhancements made to the GASNet-EX software. These enhancements, known as "specializations", primarily consist of replacing network-independent implementations of several recently-added features with implementations tailored to the Cray Aries network. Performance gains from specialization include (1) Negotiated-Payload Active Messages improve bandwidth of a ping-pong test by up to 14%, (2) Immediate Operations reduce running time of a synthetic benchmark by up to 93%, (3) non-bulk RMA Put bandwidth is increased by up to 32%, (4) Remote Atomic performance is 70% faster than the reference on a point-to-point test and allows a hot-spot test to scale robustly, and (5) non-contiguous RMA interfaces see up to 8.6x speedups for an intra-node benchmark and 26% for inter-node. These improvements are all available in GASNet-EX version 2018.3.0 and later.**

Keywords—Active Messages, RMA, Remote Atomics, PGAS, HPC, Networking, Supercomputing

I. INTRODUCTION

The GASNet (Global Address Space Networking) communication layer [3] 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], LBNL UPC++ [1], Stanford Legion [2] and Cray Chapel [6]. Under funding from the Exascale Computing Project (ECP) we are designing and implementing GASNet-EX [4], a second-generation GASNet API, which is focused on exascale requirements and incorporates over 15 years of lessons-learned.[1]

---

[1] GASNet-EX includes a backwards-compatibility layer that allows it to transparently support clients that are still using the legacy GASNet interfaces.

Throughout this project, the GASNet-EX developers are providing quarterly software releases to make new features and performance improvements available to client developers. This paper will use the GASNet-EX release numbers, such as "2017.9.0", to refer to the software releases made in the corresponding calendar months (*e.g.*, 2017.9.0 denotes the release in September 2017).

The 2017.6.0 release of GASNet-EX introduced three new features, known as "New Active Message Interfaces", "Immediate Operations" and "Local Completion". The 2017.12.0 release introduced a new "Remote Atomic" feature. These two releases contain network-independent "reference" implementations of these features, which provide implementations in terms of the pre-existing functionality available in GASNet-EX on all networks. While these reference implementations are correct and functionally complete, in general they cannot provide the best performance on every network. We use the term "specialization" to describe the process of providing network-*specific* implementations of a given feature to obtain improvements on a given target platform, such as higher speeds or lower resource utilization. The main focus of specialization in this paper is GASNet-EX's support for the Cray Aries network used in Cray XC-series systems, and the term "aries-conduit" is used to denote the code implementing this support.

This paper describes the performance improvements observed due to specialization in aries-conduit of the four features listed above, specializations which were delivered in the 2017.12.0 and 2018.3.0 releases of GASNet-EX. Additionally, we report on two additional specializations, not specific to aries-conduit, related to the implementation of "Expanded VIS Interfaces", a feature introduced in the 2017.12.0 release. The remainder of this paper consists of sections that each report on a given specialization and the observed performance improvement it yields. To keep these sections focused, a separate Appendix provides more detailed descriptions of the HPC systems and testing methodologies used. Because each client runtime introduces distinct overheads, performance is measured using GASNet-level microbenchmarks. Application-level measurements are beyond the scope of this paper.
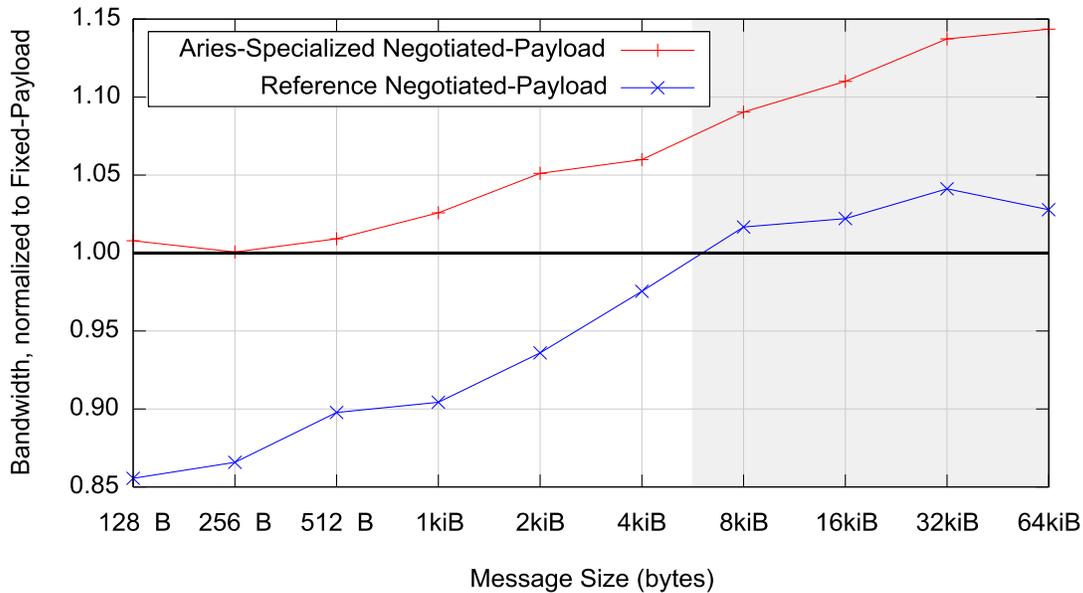
Figure 1. Speedup of AM Medium Ping-Pong with Dynamically Generated Payload

II. NEGOTIATED-PAYLOAD ACTIVE MESSAGES

One portion of the "New Active Messages Interfaces" introduced in the 2017.6.0 release was "Negotiated-Payload AMs" (NP-AM). This is a set of split-phase interfaces that complement the traditional AM interfaces (now dubbed "Fixed-Payload" or "FP-AM" to distinguish them) for Medium and Long Active Messages. While the Fixed-Payload AMs take an address and length of the caller's buffer, the Negotiated-Payload interfaces take an *optional* address and a *range* of lengths in a "Prepare" call. The result of the Prepare call provides a maximum length. If no address was passed by the caller it also provides a buffer allocated by GASNet-EX. This enables the pattern in Listing 1, in which the client allows GASNet-EX to allocate the buffer and then assembles its payload into the GASNet-allocated buffer. The communication is injected by the second phase "Commit" call in which the client provides an actual length, handler index and handler arguments.

For many networks, the use of a GASNet-allocated NP-AM buffer has the potential to eliminate a memcpy() from the critical path in the case that the client did not already have its payload instantiated in a contiguous buffer. Another usage case for NP-AM is for "streaming" of large transfers through multiple Medium or Long AMs. In this case, the client passes their total length to the Prepare call and the implementation *may* respond with a length larger than the

normal maximum for a Fixed-Payload AM, if the current resource allocation state permits. This mode of operation can result in improved performance by streaming large transfers through a smaller number of larger messages, but requires the client to be adaptable to the sizes sent.

The reference implementation of NP-AM was present in the 2017.12.0 release of GASNet-EX, and used by all conduits. The 2018.3.0 release contains the specialization of NP-AM for Medium Request and Reply on aries-conduit, as well as a significantly rewritten reference implementation to ease specialization for additional conduits in the future. This rewrite also improves the performance of NP-AM through shared-memory and loopback.

While one of the goals of NP-AM is to eliminate a memcpy() from the critical path for certain usage cases, the reference implementation does not achieve this because it lacks access to the network-specific logic which manages the buffers used for sending AMs. The Aries specialization consisted of splitting the implementation of Medium AM injection into an internal split-phase version, and structuring both outward-facing interfaces (FP-AM and NP-AM) in terms of this new internal interface.

The results in Figure 1 demonstrate both the weakness of the reference implementation of NP-AM, and the strength of the Aries-specialized implementation. The data illustrates the performance of a simple ping-pong test using AM

```
// NP-AM pattern used to avoid memcpy() via payload assembly into a GASNet-owned buffer:
gex_AM_SrcDesc_t sd = gex_AM_PrepareRequestMedium(team, peer, NULL, len, len, NULL, flags, 2);
assemble_payload(gex_AM_SrcDescAddr(sd), len); // writes to a GASNet-owned buffer
gex_AM_CommitRequestMedium2(sd, handler_index, len, arg0, arg1);

// Equivalent via FP-AM, in which most conduits must memcpy() client_buf to internal buffers:
assemble_payload(client_buf, len);  // writes to a client-owned buffer
gex_AM_RequestMedium2(tm, peer, handler_index, client_buf, len, GEX_EVENT_NOW, flags, arg0, arg1);
```

Listing 1. Comparison of calling conventions for Active Message injection
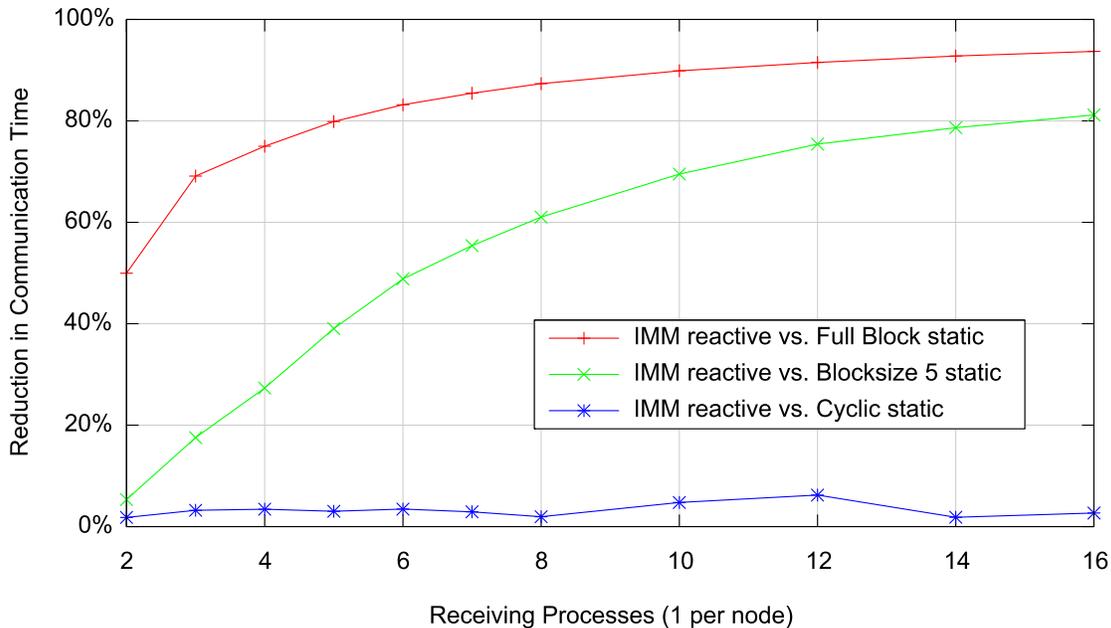
Figure 2. Reduced Communication Delays Using Immediate Active Messages

Medium with both implementations of NP-AM, leveraging the GASNet-allocated buffer described above. This data is normalized to the performance of the same test using FP-AM. The data up to and including 4KiB messages show that while the reference implementation uniformly under-performs FP-NP, the Aries-specialized NP-AM uniformly meets or exceeds the performance of FP-AM (by up to about 6%).

The data for message sizes above 4KiB (with a shaded background) show a non-default configuration of GASNet-EX in which the maximum size of a Medium payload has been increased to 64KiB. The data is presented here to show that the advantage of NP-AM continues to grow with the payload size (to over 14% at 64KiB). This will be directly relevant when NP-AM Long is specialized, and when future work on scalable buffer management in aries-conduit increases the default for the maximum Medium payload.

III. IMMEDIATE OPERATIONS

The "Immediate Operations" feature, introduced in the 2017.6.0 release of GASNet-EX, allows (but does not require) operations to return a distinguishing value during attempts at communication injection that encounter backpressure, such as due to flow control or any temporary lack of necessary resources. This allows clients of GASNet-EX to avoid stalling on injection (for example, due to head-of-line blocking) especially in cases such as work-stealing where the client may be able to pursue alternate useful actions. This behavior is optional, and clients may request it by passing an IMMEDIATE flag during communication injection. Since the nature of the feature is to permit a behavior without requiring it, the reference implementation of Immediate Operations is to trivially ignore the flag.

A complete (non-trivial) implementation of Immediate Operations in aries-conduit involved changes to several RMA and AM paths to allow them to "unwind" after encountering a transient resource shortage. Unwinding from *partial* resource acquisition required more significant changes to the AM code path than were applied to the RMA paths, which use only a single Cray GNI post descriptor resource. All contiguous point-to-point communication operations in the 2018.3.0 release of aries-conduit honor the IMMEDIATE flag, returning immediately if sufficient resources are not available to begin the operation.

The data in Figure 2 show results from a benchmark that mimics a simple client that optionally uses Immediate Operations to avoid head-of-line blocking in an AM-based communication. In the absence of Immediate Operations, the client cannot know if a given AM Request injection call will complete quickly, versus stalling due to backpressure. Those calls that stall due to backpressure may consume a significant amount of time before returning and permitting the client to proceed to issuing the next call. However, by using Immediate Operations such calls can be made to "fail quickly", allowing the client to dynamically respond to the resource congestion along that path in a client-specific manner; for example rescheduling the operation for later retry or electing to attempt communication with a *different*, less-congested peer (as one might do when implementing a work-stealing task scheduler).

The figure shows the reduction in communication times for the variant of the benchmark using Immediate Operations, relative to the variant without. Both variants eventually complete the same communication operations, but not in the same order. Use of Immediate Operations allows a static communication schedule to be replaced by a dynamic (reactive) schedule.
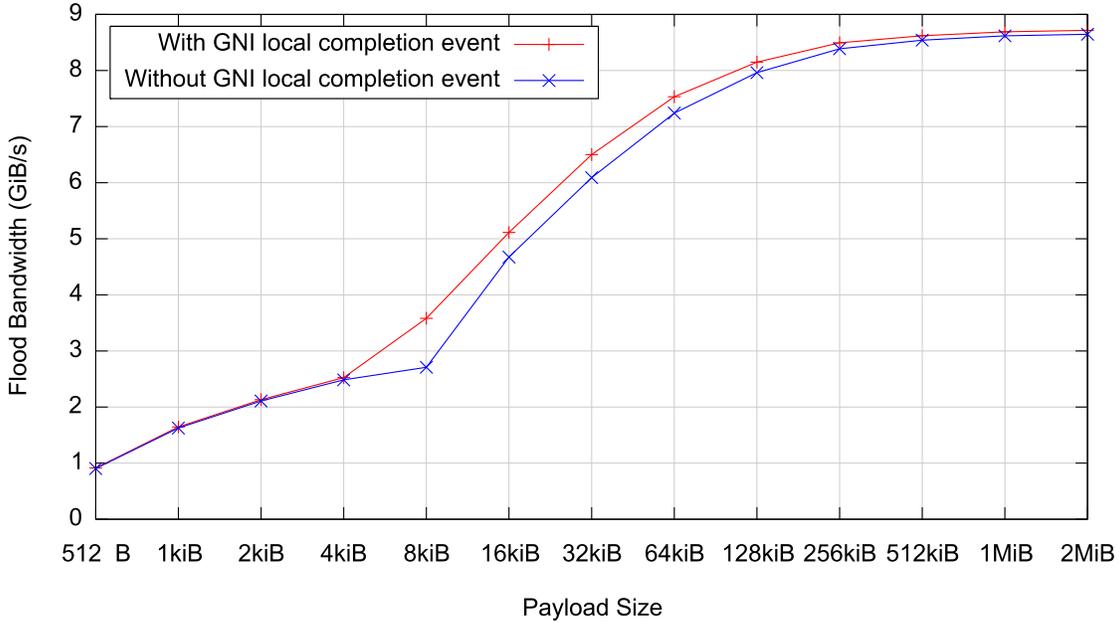
Figure 3. Non-bulk Put flood bandwidth on Cray Aries with and without local completion at the GNI level.

In the "Full Block" series the static schedule requires rank 0 to send the entire volume of messages to each of the other ranks before it may send to another, and the use of Immediate Operations to avoid the backpressure that results provides sizable reductions in the total time to complete the communication (by up to 93%). The "Blocksize=5" series requires that rank 0 send five messages to each rank before it may send to another, though this repeats until the same volume of messages has been sent. In this series the amount of backpressure encountered is less and the advantage due to Immediate Operations is also less, though still significant (up to 81%). Finally the "Cyclic" series shows the results for a static schedule that has rank 0 communicate "round robin" with the other ranks. This is intuitively the optimal static schedule for this test, since it maximizes the time between sends to any given rank. However, even in this case there appears to be a small advantage (averaging slightly over 3%) to the use of Immediate Operations. The probability that any given communication operation will encounter backpressure is quite low for this schedule, and the majority of the advantage seen for use of Immediate Operations in the Cyclic case is likely due to a marginally lower cost for successful AM Request operations due to the lack of polling to recover resources (since failure due to their lack is permitted).

IV. LOCAL COMPLETION

In GASNet we use the term "local completion" to denote when a client-provided buffer to a communication injection operation (such as the source of an RMA Put) can safely be overwritten or freed by the client. In GASNet-1 [3], non-blocking Put operations are available in "bulk" or "non-bulk" variants. A call to initiate a non-bulk Put delays returning until after local completion. Initiation of a bulk Put returns before ensuring local completion, without any means to

separate it from remote completion of the entire Put operation. GASNet-EX retains those two options, but adds the ability to test (or wait) for local completion *between* the return from initiation and the synchronization on remote completion. The goal of this specialization was to implement the GASNet-EX local completion semantics for RMA Puts as efficiently as possible using the facilities of the Aries network exposed by Cray GNI.

In GASNet-1, aries-conduit already provided distinct implementations of "bulk" and "non-bulk" RMA Puts. Therefore, the focus of this specialization was to expose the GASNet-EX event that clients can use to test (or wait) for local completion. Initially, GASNet-EX aries-conduit utilized the low-performance reference implementation approach of blocking (as in a non-bulk Put) for local completion when the corresponding event was requested. The basis of this specialization work for local completion was the ability to independently request GNI-level completion queue events for local and global completion, thereby achieving independent GASNet-EX-level events. In the process of exposing the GASNet-EX local completion event, we discovered an opportunity to significantly improve the performance of the non-bulk Puts by using the same GNI-level facilities.

Figure 3 illustrates the performance improvement obtained by applying this approach to the GASNet-EX equivalent to GASNet-1's non-bulk Puts. The new implementation matches or exceeds the performance of the previous implementation, providing a bandwidth improvement of up to 32% (at 8kiB payload size), illustrating the non-optimal behavior of the previous implementation (which did not utilize the GNI-level local completion event).
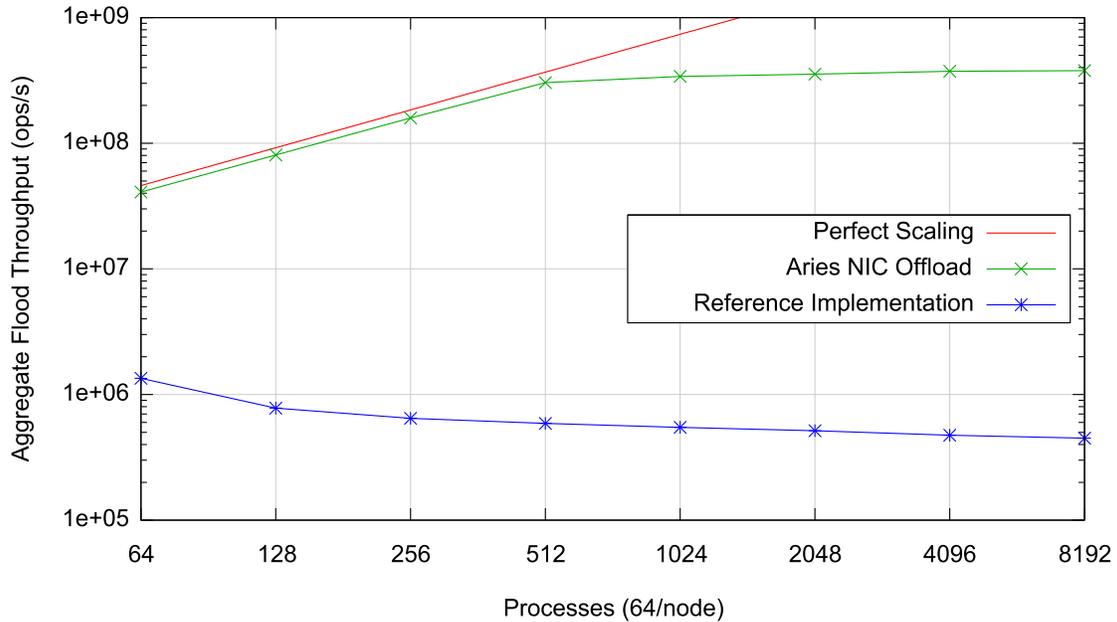
Figure 4. Weak Scaling of 64-bit Unsigned Integer FADD Hot-Spot Test

V. REMOTE ATOMICS

Remote Atomics were introduced as a new feature in the 2017.12.0 release of GASNet-EX, and provide interfaces to perform a rich set of operations atomically on several data types in distributed memory. The GASNet-EX design for Remote Atomics is derived from that used in UPC [8], in that operations are performed with respect to an "atomic domain" that is created with one data type and a set of atomic operations and then used to initiate those operations on data of the given type. This design allows for runtime selection of the fastest-available implementation that can correctly provide the set of atomic operations needed by the application. This is important because in general one cannot mix atomics that are offloaded to a NIC with others implemented using the CPU, as this would suffer from coherency problems on many modern systems. The atomic domains approach addresses this issue by selecting NIC offload implementations if and only if the *entire* application-specified set of operations can be offloaded, and a CPU-based implementation otherwise. Unlike the UPC atomics, GASNet-EX includes only non-blocking interfaces for atomics (amongst other differences).

The 2017.12.0 release of GASNet-EX included a complete reference implementation of this subsystem based on Active Messages (AM), and using CPU-based atomic instructions to perform the memory accesses (utilizing our GASNet-Tools library, which implements all the necessary

local-memory CPU atomics on an extremely wide variety of architectures and compilers). Additionally, that release contained an initial specialization for the atomic operation capabilities of the Cray Aries NIC, completed in the subsequent 2018.3.0 release. This specialization consists of logic to check at the time an atomic domain is created whether or not the requested data type and operations set are supported by the Aries NIC, and the code to use the Cray GNI functions which initiate and complete Aries-offloaded atomic operations.

In Table 1 we show that Aries specialization of Remote Atomics delivers at least a 1.7x improvement on a simple point-to-point test of the atomic fetch-and-add (FADD) operation, relative to the network-independent reference implementation over AM.

In addition to the 1.7x advantage on a point-to-point test, the Aries-specialized atomics show greatly improved scalability in a many-to-one atomics "hot-spot" test. Figure 4 shows results of such a benchmark in which all 64 cores on one or more compute nodes simultaneously perform 64-bit unsigned integer FADD operations on a single location (located on rank 0). The figure shows the aggregate FADD throughput as a function of the number of processes. The data shows that as the process count increases, the aggregate performance of the AM-based reference implementation actually drops (due to overheads of message reception dominating). Meanwhile the performance of the Aries-

Table 1. Remote Atomics Speedup Due to Aries Specialization

	FADD Latency			FADD Throughput		
Data Type	AM Reference	Aries Specialized	Ratio	AM Reference	Aries Specialized	Ratio
32-bit unsigned integer	4.9 us	2.9 us	1.7	429 kop/s	745 kop/s	1.7
64-bit unsigned integer	4.9 us	2.8 us	1.7	424 kop/s	742 kop/s	1.8

specialized version rises steadily as the node count increases from 1 to 8 (64 to 512 processes), and continues to rise gradually from that point to the highest concurrency measured (128 nodes = 8192 processes). For comparison, the "Perfect Scaling" line (in red at the upper-left of the figure) shows the throughput of a single-process run scaled by the process count.

VI. VECTOR-INDEX-STRIDED (VIS)

The term Vector-Indexed-Strided (VIS) refers to the three forms of metadata used to describe the payload in a non-contiguous GASNet transfer. The interfaces for non-contiguous data transfer are therefore collectively known as "VIS Interfaces". The GASNet-1 specification [3] lacks official interfaces for non-contiguous data, which prior to the EX work existed only as an unofficial proposed extension [5]. The 2017.12.0 release of GASNet-EX delivered an untuned, network-independent implementation of Expanded VIS Interfaces, making them an official part of GASNet-EX and expanding upon their functionality in several ways.

The expansion involved changes to the function arguments and corresponding updates to the constraints on these arguments. The most significant outcome of these interface changes is the new ability to express Strided N-dimensional rectangular transfers that transpose or reflect elements across coordinate axes. The updated interfaces also support the same new capabilities that GASNet-EX has added to the contiguous Remote Memory Access (RMA) interfaces (including teams and immediate operations, among others).

The 2018.3.0 release implements the new VIS capabilities: most notably Strided transposes and reflections, and teams and immediate support for all variants. The VIS implementation for each of the three categories includes several low-level transfer mechanisms, and the mechanism used to satisfy a given operation is selected based on the operation parameters – most notably, size of the contiguous segments and locality of the peer memory.

The work delivered in the 2018.3.0 release entailed a complete rewrite of the Strided implementation to support transposes and other non-translational inputs. The new Strided metadata format allows expression of translational and transposing copies between arbitrary rectangular sections of densely stored N-dimensional arrays. However even within this restricted set of inputs there are many possible metadata inputs that express an equivalent data transfer, and the format chosen by the user (e.g., to most naturally match the data structures in their application), is not always the most efficient format to use for actually executing the transfer. For example, the user may specify a 3-d strided copy with transfer parameters such that the accesses in linear memory are equivalent to a 1-d strided copy, where the latter representation would lead to more efficient packing code. The GASNet-1 Strided implementation contained some ad-hoc optimizations to transform the input metadata in very limited ways before executing the transfer. The GASNet-EX extensions to the Strided metadata format relaxed the GASNet-1 Strided linearity requirements, further increasing the degrees of freedom for expressing equivalent Strided transfers.

The rewritten Strided implementation in the 2018.3.0 GASNet-EX release includes a general metadata stride optimizer that applies several sophisticated optimizations to dynamically rewrite the input Strided metadata into a format more amenable to efficient execution. The optimizations performed include:

- Null Dimension Removal – dimensions with an extent of 1 can trivially be removed

- Stride Inversion – an optimization that ensures all strides for one end of the transfer are non-negative

- Dimensional Sort – sorting of dimensions that ensures the strides for one end of the transfer are in non-decreasing order

- Dimensional Folding – an optimization to remove trivial dimensions, by folding them together and/or into the element size.

The final optimization (Dimensional Folding) is the most important for performance of Strided (un)packing code, because it amounts to a run-time application of a loop transformation optimization; it reduces the nesting depth of the loop nests used to traverse the elements in the strided section, by unrolling inner loops over contiguous elements and merging amenable adjacent loops in the nesting structure. The earlier optimization passes mostly serve to normalize the metadata into a form most amenable to Dimensional Folding. The Stride Inversion and Dimensional Sort optimizations both favor normalization of the peer end of the transfer, increasing the linearity and contiguity size of the (potentially remote) segments in order to optimize for the use of initiator-driven RDMA-based mechanisms and favor access locality of (un)packing loops executing in AM handlers at the passive peer.

A. Shared-Memory Bypass for VIS

The motivation for the work described in this paper was to specialize the GASNet-EX implementation for the Cray XC series of supercomputers. These supercomputers include a variety of multi-core/many-core processor configurations – for example each node of Cori-I has two 16-core 2-way hyper-threaded Intel Haswell processors (for a total of 64 hyper-threads per node), whereas each node of Cori-II has a 68-core Intel Xeon Phi processor with 4 hardware threads per core (for a total of 272 hardware threads per node). All configurations of these systems feature a large number of cores/threads sharing a single cache-coherent physical memory domain and Aries ASIC. Consequently, the performance of intra-node GASNet operations (those between processes co-located on a physical node) can be very important, especially for applications that closely map their locality of access to match the hierarchical system configuration. GASNet has dedicated support to implement such intra-node operations with minimal overhead by using shared-memory-bypass mechanisms to avoid the I/O bus crossings involved with activating the network hardware. Mechanisms employed vary by target system, and include
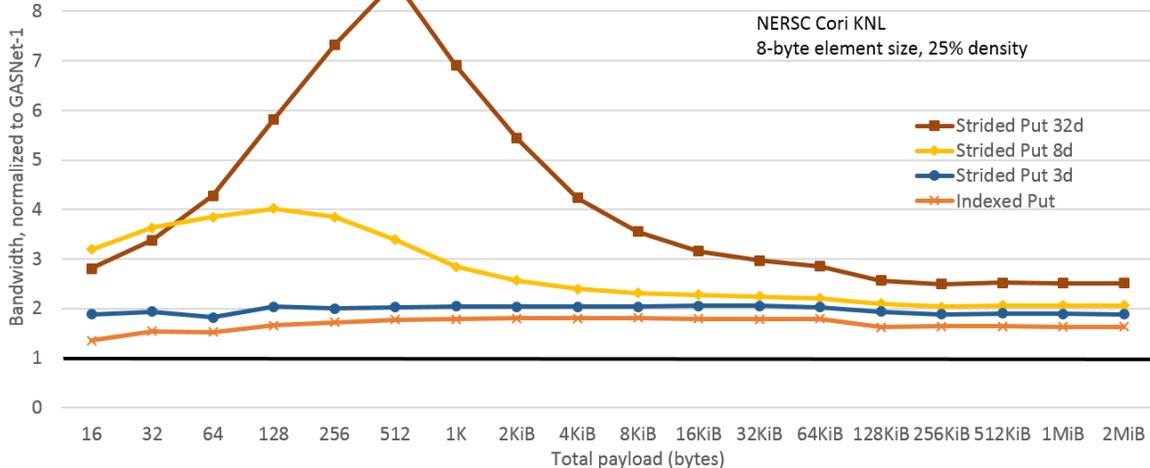
Figure 5. Speedups due to Shared-Memory Bypass Improvements

POSIX and SystemV shared memory. In the case of the Cray XC-series, XPMEM is used.

As part of this work, the shared-memory-bypass mechanism used to satisfy intra-node VIS operations was updated to use more efficient internal interfaces for inter-process address translation. Figure 5 demonstrates the bandwidth performance speedup achieved in the 2018.3.0 release of GASNet-EX for a range of intra-node VIS operations, relative to the bandwidth of the same operations using the GASNet-1 VIS implementation. The intra-node Indexed Put bandwidth for this non-contiguous access pattern improved by an average of 66.7%, due to the use of these more efficient internal interfaces for shared-memory bypass.

The other series in Figure 5 show the bandwidth improvement obtained for equivalent Strided Put operations, ranging in dimensionality of the input metadata from 3d to 32d. In addition to using more efficient shared-memory bypass, these operations show further improvement due to the new stride optimizer. The Strided metadata for these operations are all amenable to Dimensional Folding down to a single dimension of actual striding. The stride optimizer added in GASNet-EX achieves this optimal folding resulting in the use of a data transfer loop with a single level of nesting for all three series, whereas the GASNet-1 implementation of the same operation uses deeply nested loops, adding overhead and progressively degrading transfer performance for increasing input dimensionality. This results in an average improvement of 1.9x, 2.6x and 3.9x for each of 3d, 8d, and 32d (respectively), with a peak improvement of 8.6x at 32d. These improvements generalize beyond aries-conduit, improving the performance of the VIS implementation for all conduits on multi-core systems.

B. Negotiated-Payload AM for VIS

The VIS implementation in the 2018.3.0 GASNet-EX release also updated the Active-Message-based mechanisms (used to service many inter-node non-contiguous network transfers) to optionally use the new Negotiated-Payload AM

interfaces added in GASNet-EX. The main purpose of this upgrade was to leverage the NP-AM GASNet-allocated buffer capability, allowing VIS operations to pack payload data directly into the outgoing network buffer (for the aries-conduit specialized version of NP-AM), thus eliding the payload copy costs paid by the FP-AM version of this mechanism. Additionally, the mechanism was upgraded to use the `gex_AM_Max{Request,Reply}Medium()` queries added in GASNet-EX that allow fitting up to 64 bytes of additional payload into each AM, potentially reducing the total message count used to implement some operations.

Finally, the GASNet-EX AM-based inter-node mechanism for Strided operations additionally benefits from the new stride optimizer – reducing both the pack/unpack costs at each process, and furthermore often reducing the amount of descriptor metadata sent with each Active Message (thanks to Dimensional Folding).

The results in Figure 6 demonstrate the bandwidth speedup of inter-node Strided Puts in GASNet-EX using the FP-AM- and NP-AM-based mechanisms, relative to the bandwidth of the same operation using the GASNet-1 Strided implementation. The GASNet-EX FP-AM mechanism shows an average speedup of 7.3% and peak speedup of 23.8% relative to the GASNet-1 implementation (also using FP-AM). This improvement is due to the stride optimizer and increased network packet occupancy achieved by the new Strided implementation. The NP-AM mechanism shows an additional average speedup of 6.2%, which is due entirely to the removal of the `memcpy()` operation in AM Request injection enabled by the Aries-specialized implementation of NP-AM.

VII. CONCLUSIONS

Past releases of GASNet-EX have introduced several new features listed in the introduction to this paper. Each of these has a network-independent "reference implementation" that is correct for all networks but is not expected to be optimal for most networks. This paper documents our work
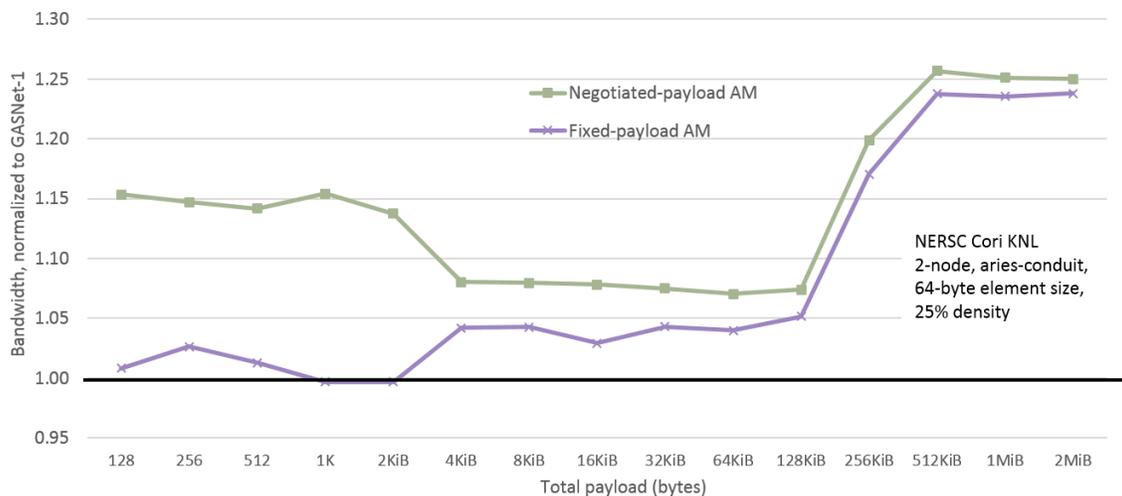
Figure 6. Strided Put Speedups for Fixed-Payload and Negotiated-Payload AMs

in specializing the implementations of several of these features for the Cray Aries network, as released in GASNet-EX version 2018.3.0. This paper has (a) described these specialization efforts and (b) presented performance results highlighting the benefits of these specializations, as measured on NERSC's Cori and ALCF's Theta systems. The results repeatedly show that the specialized implementations improve performance relative to earlier (reference or GASNet-1) implementations, validating the designs of these features and justifying the effort of specialization.

REFERENCES

[1] Bachan, J., Bonachea, D., Hargrove, P.H., Hofmeyr, S., Jacquelin, M., Kamil, A., van Straalen, B., Baden, S.B., "The UPC++ PGAS Library for Exascale Computing", Proceedings of the Second Annual PGAS Applications Workshop (PAW). 2017. https://doi.org/10.1145/3144779.3169108

[2] 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. https://doi.org/10.1109/SC.2012.71

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

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

[5] Bonachea D., "Proposal for Extending the UPC Memory Copy Library Functions and Supporting Extensions to GASNet, v2.0", Lawrence Berkeley National Laboratory Technical Report (LBNL-56495 v2.0). Mar 2007. https://doi.org/10.2172/920052

[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. https://doi.org/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. 2003. https://doi.org/10.1145/782814.782825

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

GASNet-EX Performance Improvements Due to Specialization for the Cray Aries Network

Reproducibility Appendix

Paul H. Hargrove and Dan Bonachea

Computational Research Division,

Lawrence Berkeley National Laboratory

Berkeley, CA 94720, USA

`gasnet-staff@lbl.gov`

I.   TEST PLATFORMS

Experimental results were collected on the following Cray XC40 systems, in which nodes are connected via Cray Aries network hardware, and communication is performed using the GASNet-EX aries-conduit:

NERSC Cori-I
Node: dual 16-core 2.3 GHz Intel Haswell, 128 GB
Compiler: Intel C Compiler, v18.0.0.128
System software: Cray PrgEnv-intel/6.0.4, Cray PE/2.5.12
Batch system: SLURM (srun)

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/2.5.12
Batch system: SLURM (srun)

ALCF Theta
Node: 64-core 1.3 GHz Intel Xeon Phi 7230, 192 GB of DDR4 (quad-cache mode)
Compiler: Intel C Compiler, v18.0.0.128
System software: Cray PrgEnv-intel/6.0.4, Cray PE/2.5.13
Batch system: ALPS (aprun)

II.   METHODOLOGY FOR FIGURE 1 (NEGOTIATED-PAYLOAD ACTIVE MESSAGES)

These results were gathered using the `testam` microbenchmark that is included in the GASNet-EX distribution, run on ALCF's Theta. The exact command used for data collection was:

```
aprun -n2 -N1 testam [-fp|-np-gb] -src-generate 250000 65472 B
```

where either `-fp` or `-np-gb` was used as described below.

All runs are point-to-point, using a single process on each of two nodes connected by the network fabric. The "B" option restricts the test to reporting the performance of a ping-pong test using AM Mediums. This test consists of one rank sending an AM Medium Request of a given size and the recipient sending an AM Medium Reply of the same size. The first rank waits to receive the Reply before sending its next Request. This Request-Reply ping-pong exchange is repeated 250,000 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. With these parameters, this test is performed at the sizes 0 and 65472, and all the powers-of-two in between.

The `-fp` option causes `testam` to use the calls `gex_AM_RequestMedium()` and `gex_AM_ReplyMedium()` (the default behavior of `testam`). These Fixed-Payload interfaces accept the address and length of a buffer allocated by the caller. The `-np-gb` option causes the test to instead call the Negotiated-Payload interfaces `gex_AM_PrepareRequestMedium()` and `gex_AM_CommitRequestMedium()`, as well as the corresponding `Reply` calls, with arguments which request that the Prepare call allocate the buffer to be sent by the Commit call.

The `-src-generate` option to `testam` causes the test to dynamically generate the payload to be sent in each and every Request or Reply, filling it with consecutive integers generated on-the-fly (not, for instance, by `memcpy()` from a prepared location). This simulates a client that does not have the payload fully assembled in memory at the start of a communication. In the Fixed-Payload case the data is generated into the caller-owned buffer, while in the Negotiated-Payload case it is generated into the buffer received from the Prepare call. On aries-conduit the use of the Negotiated-Payload interface eliminates a copy from caller-owned memory to GASNet-owned buffers.

Runs were performed with two copies of `testam`, built from aries-conduit sources with and without the specialized Negotiated-Payload support. The sources without the specialization use the reference implementation. The data plotted in each series shows a value collected using one of these two executables with the `-np`-`gb` flag, normalized by a value collected with the `-fp` flag. These three values are, in turn, each the median of 19 runs. Use of median was chosen over mean due to lower sensitivity to the low-performing outliers that are common on this platform.

All of the data used for normalization (run with the `-fp` flag) was collected using the executable containing aries-specialized support for Negotiated-Payload. However, the two executables show no measurable difference for the Fixed-Payload tests.

III.   METHODOLOGY FOR FIGURE 2 (IMMEDIATE OPERATIONS)

These results were gathered using the `testimm` microbenchmark that is included in the GASNet-EX distribution, run on ALCF's Theta. The exact command used for data collection was:

```
aprun -nN -N1 testimm -m -b B M
```

where N denotes the job size (1 + receiving processes), B denotes a block size, and M denotes a message count. The B and M parameters are more fully defined below.

With these parameters, the benchmark performs two variants of a communication pattern using Active Message Medium Requests. In both variants, the rank 0 process sends a series of AM Mediums (of length 4032 bytes with these parameters) to the remaining processes in the job (the "receiving processes"), followed by a barrier. For the duration of the communication, receiving processes are alternating between sleeps of length 500us and calls to test for completion of the non-blocking barrier. Testing of the barrier also progresses reception of arriving AMs, and without such progress the test would not terminate. The presence of sleeps between calls making AM progress simulates applications that alternate periods of computation with communication.

In the first variant the communication is performed with a static schedule, without the use of the IMMEDIATE flag. Letting N denote the job size, the schedule can be expressed using the following pseudo-code:

```
for (int k = 0; k < M; k += B) {
        for (int j = 1; j < N; ++j) {
                for (int i = 0; i < B; ++i) {
                        gex_AM_RequestMedium(… rank=j …);
                }
        }
}
```

In other words, the rank 0 process sends groups of B consecutive messages to each receiving process in sequence, and this is repeated until a total of M message have been sent to each receiving process. Let T_{static} denote the time to complete this schedule, including the subsequent barrier.

The second variant of the communications schedule is a dynamic one using the IMMEDIATE flag to avoid stalling due to backpressure (which will result due to the sleeps by receiving processes). The triply nested loop now describes a "nominal" schedule that would execute if no injection failures occur due to the IMMEDIATE flag. However, if a `gex_AM_RequestMedium()` call *does* fail, this adaptive variant calls `gasnet_AMPoll()` once, to attempt recovery of resources, before retrying the same operation. If the operation fails a second time, the communication advances to the next receiving process even though the group of B messages has not been completed. The skipped messages are not omitted, but are instead deferred. Additional logic ensures that the equivalent of the outer loop continues until all deferred communication is completed.

Letting $T_{dynamic}$ denote the time to complete the dynamic communication schedule, the test reports the reduction in communications time as a percentage of the static time:

```
100% * (Tstatic - Tdynamic) / Tstatic
```

2

Experiments were run with values 1, 5 and *M* for the block size *B,* where the value *M* effectively eliminates the outer loop in the static schedule. It should be noted that buffering at the AM sender is sufficient to hold at most three messages of the size used in this test before an inattentive receiver would result in backpressure.

Runs were performed for several job sizes to collect the data shown in the figure, always placing one process per compute node. As the job size varied, the message count parameter *M* was strong-scaled, taking on the value 20,000 divided by the number of receiving processes. Each data point is a median of 5 runs with the same parameters. Use of median was chosen over mean due to lower sensitivity to the low-performing outliers that occur in approximately one out of every ten runs.

IV. METHODOLOGY FOR FIGURE 3 (LOCAL COMPLETION)

These results were gathered using the `testsmall` RMA microbenchmark that is included in the GASNet-EX distribution, run on NERSC's Cori-I. The exact command used for data collection was:

```
srun –n2 –N2 testsmall –m –p –out 50000 2097152 F
```

All runs are point-to-point, using a single process on each of two nodes connected by the network fabric. The selected parameters measure flood bandwidth for non-bulk explicit-handle non-blocking RMA Put operations (corresponding to `gasnet_put_nb()` in GASNet-1 and `gex_RMA_PutNB(…GEX_EVENT_NOW…)` in GASNet-EX), where the initiator-side payload does not reside in the GASNet-registered segment. GASNet's non-bulk Put semantics delay return from the injection operation until the client can freely overwrite the source buffer for the Put operation without affecting the result. This stresses the throughput of the conduit's Local Completion facility for RMA Puts. The payload size per Put operation was varied from 1 byte to 2 MB. At each payload size, 50,000 such non-blocking Puts were injected back-to-back and then synchronized, and the flood bandwidth computed as the quotient of the total payload volume and the elapsed time.

Runs were performed with two copies of `testsmall`, built from sources before and after the modifications to utilize the GNI-level local completion event.

V. METHODOLOGY FOR TABLE 1 (REMOTE ATOMICS)

These results were gathered using the `testfaddperf` Remote Atomic microbenchmark included in the GASNet-EX distribution, run on NERSC's Cori-II. The exact command used for data collection was:

```
aprun –n2 –N1 testfaddperf 100000
```

All runs are point-to-point, using a single process on each of two nodes connected by the network fabric. The test reports the performance of an atomic fetch-and-add operation for each supported data type in terms of two metrics: latency and throughput. The latency metric is measured by injection and synchronization of a single atomic operation, repeated 100,000 times. The reported latency is the quotient of the elapsed time for all iterations and the iteration count. Injecting 100,000 atomic operations back-to-back and then synchronizing them all yields the throughput metric as the quotient of the operation count and the elapsed time.

Runs were performed with two copies of `testfaddperf`, built with and without Aries-specialized Remote Atomics enabled at compile time (they are enabled by default). The copy without the specialized atomics additionally lacked the shared-memory optimization present in the GASNet-EX release. This lack significantly reduces the reported performance of the reference implementation for a single node (64 processes), but by doing so renders it comparable to the rest of the reference implementation results.

VI. METHODOLOGY FOR FIGURE 4 (REMOTE ATOMICS)

These results were gathered using the `testfaddperf` Remote Atomic microbenchmark included in the GASNet-EX distribution, run on ALCF's Theta. The exact command used for data collection was:

```
aprun –nN –N64 testfaddperf –S 250000 D
```

where *N* denotes the job size in processes.

These parameters request a single-target (hot-spot) test in which all processes issue 250,000 64-bit unsigned integer atomic fetch-and-add operations back-to-back, targeting a single location on rank 0, followed by a barrier. All processes, including rank 0 and the other 63 processes on its compute node, are active in issuing atomic operations. The test reports the throughput of each process as the number of operations divided by the time from the start of the first operation to completion of the barrier. The experiment varied the number of compute nodes used from 1 to 128 (by powers-of-two) and ran 64 processes per node. This was a weak-scaling experiment in which the 250,000 operations per process was kept fixed. Each data point on this figure reports the aggregate throughput of all processes, and is the mean of 13 runs at each job size.

Runs were performed with two copies of `testfaddperf`, built with and without Aries-specialized Remote Atomics enabled at compile time (they are enabled by default).

VII. METHODOLOGY FOR FIGURE 5 (VECTOR-INDEXED-STRIDED (VIS) SHARED-MEMORY BYPASS)

These results were gathered using the `testvisperf` VIS microbenchmark included in the GASNet-EX distribution, run on NERSC's Cori-II. An older version of the same microbenchmark was included in GASNet-1, and improvements made to the microbenchmark in this quarter were back-ported to the GASNet-1 version; the GASNet-1 version used in these experiments is available in the 1.32.0 release of GASNet-1.

Both versions of the test execute equivalent data movement operations; the only difference is the VIS call signatures have changed slightly in GASNet-EX. The exact command used for data collection in this figure was:

```
srun -N 1 -n 2 -c 1 --cpu_bind=cores testvisperf -sl D 0.5 CE
```

where "CE" selects the Indexed Put and Strided Put tests, and D denotes the dimensionality for the strided metadata. All runs in this test used two processes on a single node, communicating through shared-memory bypass - using the Cray XPMEM variant of GASNet's PSHM support.

The selected parameters measure flood bandwidth performance for non-contiguous Puts using two of the three VIS interface metadata types: Indexed format (list of addresses with one fixed element size) and Strided format (N-dimensional array section descriptor).

All operations included in the measurement used non-contiguous payloads residing in the GASNet-registered segment on each node. Total payload size per operation varied in powers of two from 16 bytes to 2MB. The data reported in this figure uses a fixed 8-byte element size (the size of the contiguously stored segments), where those elements were distributed uniformly and non-contiguously in linear memory at 25% density (i.e. a repeating pattern of 8 bytes of payload followed by 24 bytes of skip). At each transfer configuration, a number of non-blocking operations were injected back-to-back and then synchronized, and the flood bandwidth computed as the quotient of the total payload volume and the elapsed time. The number of operations injected was dynamically scaled for each data point to achieve an elapsed time of at least 0.5 sec for each measurement. At each payload size, the operation specified in each series performs the same actual underlying payload data motion (only the metadata used to specify the transfer differs), thus providing more directly comparable results. In the case of the Strided operations, the metadata generator distributes the extent factors across dimensions, minimizing the use of null dimensions as permitted by the element count.

The bandwidth for each transfer configuration was measured using two executables – one using the GASNet-1 implementation of VIS (with default settings) and the second using the VIS implementation of the GASNet-EX 2018.3.0 release (with default settings). The figure shows the speedup ratio delivered at each transfer configuration by the GASNet-EX implementation, relative to the GASNet-1 implementation.

VIII. METHODOLOGY FOR FIGURE 6 (VECTOR-INDEXED-STRIDED (VIS) NEGOTIATED-PAYLOAD AM)

These results were gathered using the `testvisperf` VIS microbenchmark (see previous section), run on NERSC's Cori-II. The exact command used for data collection in this figure was:

```
srun -N 2 -n 2 -c 1 --cpu_bind=cores testvisperf -sl 3 0.5 E
```

which selects a 3-dimensional Strided Put test. All runs are point-to-point, using a single process on each of two nodes connected by the network fabric. The selected parameters measure flood bandwidth performance for non-contiguous Strided Puts using a 3-dimensional array section descriptor.

All operations included in the measurement used non-contiguous payloads residing in the GASNet-registered segment on each node. Total payload size per operation varied in powers-of-two from 128 bytes to 2MB. The data reported in this figure uses a fixed 64-byte element size (the size of the contiguously stored segments), where those elements were distributed uniformly and non-contiguously in linear memory at 25% density (i.e. a repeating pattern of 64 bytes of payload followed by 192 bytes of skip). At each transfer configuration, a number of non-blocking Strided operations were injected back-to-back and then synchronized, and the flood bandwidth computed as the quotient of the total payload volume and the elapsed time. The number of operations injected was dynamically scaled for each data point to achieve an elapsed time of at least 0.5 sec for each measurement.

The bandwidth for each transfer configuration was measured using three executables – one using the GASNet-1 implementation of VIS (with default settings), the second and third using the VIS implementation of the current GASNet-EX release, compiled to implement the operations being measured using either Fixed-Payload AMs or Negotiated-Payload AMs (the default setting). The figure shows the speedup ratio delivered at each transfer configuration by the GASNet-EX implementations, relative to the GASNet-1 implementation.