Title

GASNet-EX: A High-Performance, Portable Communication Library for Exascale

ISBN

978-3-030-34627-0

Authors

Bonachea, Dan

Hargrove, P

Publication Date

2019-11-13

DOI

10.25344/S4QP4W

Peer reviewed

GASNet-EX: A High-Performance, Portable Communication Library for Exascale

Dan Bonachea* Paul H. Hargrove†

{DOBonachea,PHHargrove}@lbl.gov
Computational Research Division,
Lawrence Berkeley National Laboratory
Berkeley, CA 94720, USA

October 1, 2018

Abstract

Partitioned Global Address Space (PGAS) models, typified by languages such as Unified Parallel C (UPC) and Co-Array Fortran, expose one-sided communication as a key building block for High Performance Computing (HPC) applications. Architectural trends in supercomputing make such programming models increasingly attractive, and newer, more sophisticated models such as UPC++, Legion and Chapel that rely upon similar communication paradigms are gaining popularity.

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 in future exascale machines. The library is an evolution of the popular GASNet communication system, building upon over 15 years of lessons learned. We describe and evaluate several features and enhancements that have been introduced to address the needs of modern client systems. Microbenchmark results demonstrate the RMA performance of GASNet-EX is competitive with several MPI-3 implementations on current HPC systems.

Keywords: HPC, PGAS, RMA, Active Messages, Middleware.

1 Introduction

1.1 Background on GASNet-1

The GASNet project began in 2002 [14] as an effort to provide a common, open-source HPC communication API tailored for use as a compilation target by Partitioned Global Address Space (PGAS) languages, notably including UPC [75], Titanium [38], and Co-Array Fortran [65]. Communication behavior in these models is often characterized by one-sided, remote-memory-access (RMA) communication (i.e., Puts and Gets operating on physically distributed memory), and sensitivity to the latency and overheads of fine-grained communication. The initial GASNet API (hereafter referred to as GASNet-1) offers two primary modes of communication: (1) a one-sided RMA interface that exposes the RDMA capabilities of network hardware, enabling their use to directly implement PGAS Put/Get operations on user data structures, and (2) a streamlined Active Message (AM) [30] interface to provide extensibility and efficient management of the client's parallel runtime system.

Design goals for the GASNet communication system included: network-independence (insulating long-lived clients from low-level hardware details and changes), language-independence (leaving details of the

*ORCID: 0000-0002-0724-9349
†ORCID: 0000-0001-6691-5287

Workshop on Languages and Compilers for Parallel Computing (LCPC'18)
The final authenticated version is available online at doi:10.1007/978-3-030-34627-0_11

PGAS system such as global pointer representation and allocation strategy to the client), robust multi-threading support (efficiently allowing a variety of client threading models on multi-core architectures), and widespread portability. The GASNet development effort has focused on providing a high-performance, production-quality communication layer tailored for the needs of PGAS systems.

The GASNet API [17] has become the *de-facto* communication standard targeted by portable PGAS system implementations developed by many institutions. Current and historical GASNet clients include: LBNL UPC++ [3, 4, 79], Berkeley UPC [22], GCC/UPC [46], Clang UPC [45], Cray Chapel [19], Stanford Legion [6], Titanium [78], Rice Co-Array Fortran [26], OpenUH Co-Array Fortran [29], OpenCoarrays in GCC Fortran [32], OpenSHMEM Reference implementation [70], Omni XcalableMP [57], and several miscellaneous projects [10, 18, 20, 27, 51, 52, 71]. Some of these clients implement models that fall outside the traditional PGAS definition, showing that the applicability of GASNet exceeds the original goals. The services provided and the match to modern hardware capabilities make GASNet an excellent communication substrate for implementing a wide variety of models.

GASNet uses the term "conduit" to refer to any complete implementation of the GASNet API which targets a specific network device or lower-level networking layer. GASNet conduits have been written that target a variety of past and current vendor-proprietary or hardware-specific networking interfaces, including: OpenFabrics Verbs/VAPI for InfiniBand [37, 42], Mellanox MXM for InfiniBand [53], Cray GNI for Gemini and Aries fabrics [1, 36, 41], Intel PSM2 for Omni-Path [9, 44], IBM PAMI for BlueGene/Q (and others) [49], IBM DCMF for BlueGene/P [50, 63], IBM LAPI for SP Colony/Federation [40], Cray Portals for XT3/XT4 [16], SHMEM for the Cray X1 [8] and SGI Altix [28], Quadrics elan3/elan4 for QsNetI/II [69], Myricom GM for Myrinet [7, 11], and Dolphin SISCI [73]. There are also GASNet conduits implemented over portable network APIs that enable deployment on early systems or those lacking HPC networking hardware. These include: udp-conduit (for any network with a TCP/IP stack, such as Ethernet) mpi-conduit (for any system providing MPI 1.1 [55] or newer), ofi-conduit (targeting the portable libfabric API [35]), portals4-conduit (for Sandia Portals 4 [5]), and smp-conduit (for single-node systems, such as laptops).

Most of the conduits described above were authored by members of our group, but several conduits have been contributed by a relevant vendor or external group. Additionally, some projects have developed forks of GASNet (for instance to target non-public network APIs), including MVAPICH2-X [47] and others [48, 77]. GASNet's implementation is designed so that authors of new conduits only need to port a minimal core (consisting of a few job management routines and the AM interfaces) to achieve full functionality. We provide reference implementations of all other interfaces, which can be incrementally replaced with higher-performing native versions. The GASNet implementation is written in standard C and is very portable across architectures and operating systems. Over the years, it has been ported to a diverse range of systems, encompassing over 10 compiler families, 15 operating systems and dozens of architectures – see [33] for details.

1.2 Philosophy of GASNet-EX improvements

GASNet-EX is the next generation of the GASNet-1 communication system, continuing our commitment to provide portable, high-performance, production-quality, open-source software. The GASNet-EX upgrade is being done over the next several years as part of the U.S. Department of Energy's Exascale Computing Program (ECP). The GASNet interfaces are being redesigned to accommodate the emerging needs of exascale supercomputing, providing communication services to a variety of programming models on current and future HPC architectures. This work builds on fifteen years of lessons learned with GASNet-1, and is informed and motivated by the evolving needs of distributed runtime systems.

The end of Moore's Law scaling for serial processor performance has led to increasing levels of on-die parallelism, lighter-weight cores, and deeper on-node memory hierarchies; these trends are expected to continue in future HPC architectures. We expect future runtime systems and applications will migrate away from bulk-synchronous parallel algorithms and increasingly adopt approaches with looser inter-node synchronization, using aggressively asynchronous communication such as in UPC++ [3] or dynamic tasking features available in systems such as Chapel [19], Legion [6] and X10 [21]. This motivates a communication system interface that enables the client to adapt to the dynamic behavior of the system, for example adjusting the communication schedule on-the-fly based on network backpressure. There is also motivation to improve the efficiency of memory buffer behavior in the communication system, for example providing finer-grained

control over buffer lifetime and exposing mechanisms to reduce in-memory payload copying. Modern HPC networks often include hardware support for offloading various communication-related tasks from the host processor, such as packing/unpacking of non-contiguous communication buffers, performing atomic memory updates initiated by remote peers, and orchestrating collective communications (e.g., reductions and barriers). Interfaces are being added in GASNet-EX that allow clients to express these high-level patterns in ways that can take advantage of such hardware support where available. Finally, there is a need for improved system abstractions to enable interoperability in hybrid programs, allow finer-grained or thread-level partitioning of communication work, and ensure all parts of the communication system scale efficiently to millions of ranks.

2 Design of GASNet-EX

2.1 Overview of Improvements

The GASNet interface is being redesigned and extended in a number of ways to meet the needs of exascale runtime systems. Most of the functionality and abstractions from GASNet-1 are still present, but have been generalized in several ways. (The GASNet-EX distribution notably includes a backwards-compatibility layer to enable incremental migration of current GASNet-1 client software to GASNet-EX). In GASNet-1, initialization was monolithic and assumed a single client/endpoint/segment per process. In GASNet-EX, initialization becomes more incremental and includes an object model where the Client, registered memory Segments and communication Endpoints are all managed separately and explicitly. This design has already enabled several interface improvements, and enables clients to naturally express more complicated use cases. For example, AM handler registration is now per-Endpoint and can be performed incrementally, improving client modularity. The Endpoint abstraction allows for multiple isolated communication contexts to coexist within a process, for example enabling GASNet-EX Active Messages to target specific threads within a remote process. GASNet-EX adds APIs to scalably query and manage hierarchical process layouts and memory Segments residing in inter-process shared memory.

Here is the signature for a representative non-blocking RMA Put operation in GASNet to demonstrate some of the changes:

```
gasnet_handle_t    /* GASNet-1 */
gasnet_put_nb(gasnet_node_t node, void *dest_addr,
              void *src_addr, size_t nbytes);

gex_Event_t        /* GASNet-EX */
gex_RMA_PutNB(gex_TM_t tm, gex_Rank_t rank,
              gex_Addr_t dest_addr,
              void *src_addr, size_t nbytes,
              gex_Event_t *lc_opt, gex_Flags_t flags);
```

In both cases the contiguous source payload is indicated by a base address and size, but everything else has changed. In GASNet-1, the destination process of every point-to-point operation was indicated using an integer node id. In GASNet-EX a destination Endpoint is named via a team (TM) and rank id pair – improving client composability, and enabling Endpoints to be dynamically added to the system for various purposes. The team argument names not only an ordered set of Endpoints, but also the local representative Endpoint and its containing Client. This object hierarchy can be traversed by client code, which can query various attributes and even set client-owned context attributes.

In the GASNet-1 API, the remote target for the RMA Put is specified using a virtual address. The GASNet-EX API still allows this, but additionally enables offset-based addressing into a memory Segment bound to the destination Endpoint – potentially improving scalability of client metadata, and enabling future work in binding of memory Segments to non-DRAM device memory. GASNet-EX adds a flags argument to most functions for extensibility, allowing the semantics and performance characteristics of many calls to be modified by passing appropriate flags (e.g., passing assertions about the argument values that can obviate the need for more expensive dynamic checking).

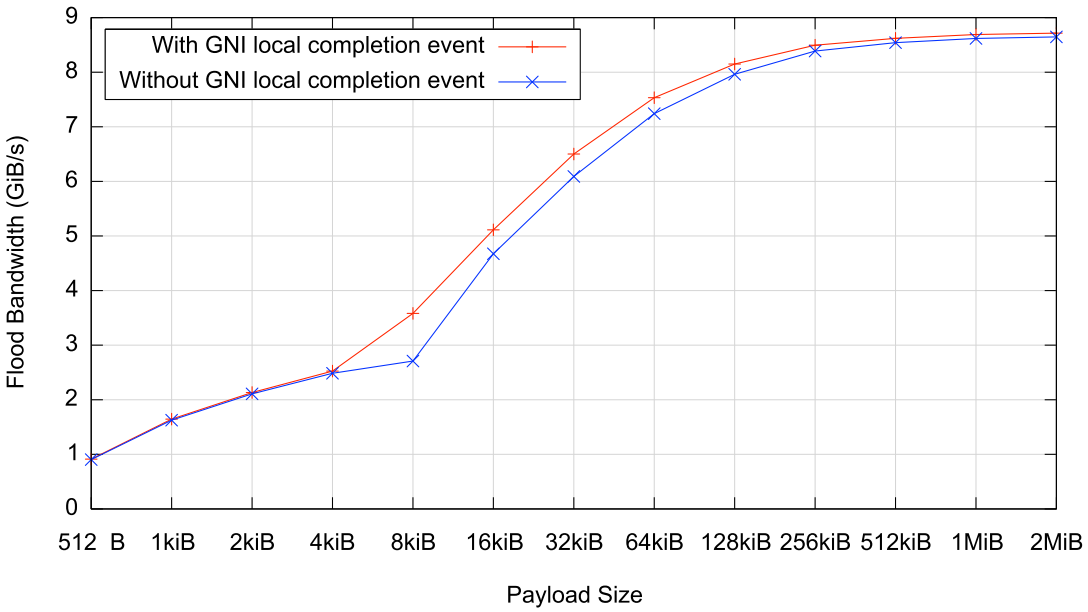

Figure 1: Non-bulk Put flood bandwidth on Cray Aries with and without use of a local completion event at the GNI level.

GASNet-1 non-blocking operations return a monolithic handle used for later synchronization. This concept has been generalized to GASNet-EX Events, which can have sub-Events representing intermediate steps that occur before completion of an entire operation, enabling clients to explicitly respond to such state changes. For example, the RMA Put has an argument for specifying the local completion behavior of the source memory (i.e. the two options in GASNet-1 being stall upon injection or delay until remote completion). GASNet-EX allows the operation to generate a sub-Event, so the client initiating the Put can independently track both local and remote completion of the same operation.

All figures in the remainder of Sec. 2 are reproduced (with permission) from our technical report [36] and show the performance of GASNet-EX aries-conduit on Cray XC40 systems (Cray Aries network) – see the report for full methodological details (omitted from this paper due to space constraints).

2.2 Local Completion Control

As mentioned above, GASNet-EX adds sub-Events into the generalization of the GASNet-1 handle abstraction, and "local completion" is one of the uses for this new concept. With the new option to independently track local completion, a client can free or reuse a source buffer as soon as it is safe to do so without the cost of blocking for local completion in the injection call (the only mechanism available in GASNet-1 for separating local and remote completion). This provides an increase in time available for overlap of communication with computation, or with additional communication. To evaluate the effects of this enhancement, we measured the bandwidth achieved by a microbenchmark where the client issues a series of non-blocking Puts but requests each injection to stall for local completion before return. Fig. 1 shows this benchmark can be improved by as much as 32% through the separation of local completion from operation completion.

2.3 Immediate-mode Communication Injection

Both GASNet APIs permit "non-blocking" communication injection operations to block temporarily (stall) when resources are not readily available to initiate the requested communication. This backpressure behavior arises fundamentally from a design principle prohibiting unbounded buffering within our GASNet implementation. However, a client of GASNet-EX can use the new flags argument to request "immediate mode" injection, wherein an operation that determines it would stall will instead be cancelled and return a

distinguishing value. This enables 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 effect of stalling can be especially evident in communications using AMs destined to a peer which is not actively entering the GASNet library (an "inattentive" peer). Our investigation found that exposing backpressure in the Active Message APIs can reduce running time by as much as 97% on a synthetic benchmark simulating communication with inattentive peers. This is illustrated in Fig. 2 which shows the reduction in overall communication time obtained by using immediate-mode AM injection to dynamically adjust the communication schedule in response to backpressure, as compared to three static schedules that stall on backpressure.

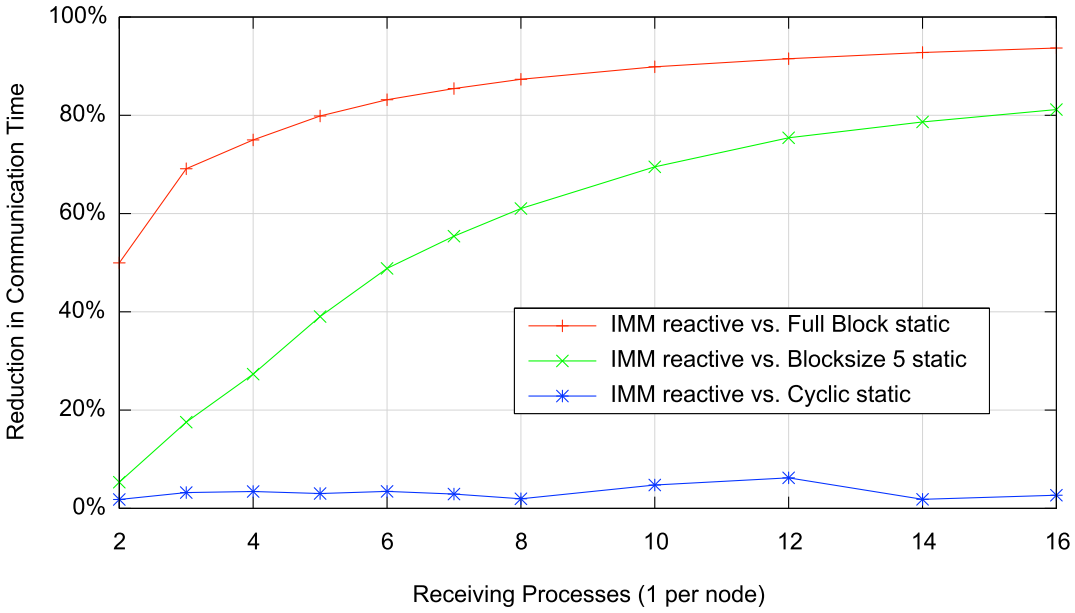

Figure 2: Reduced communication delays using immediate-mode Active Messages.

2.4 Active Message Improvements

Inclusion of AMs in GASNet-1 provides extensibility and efficient management of the client's parallel runtime, for instance UPC shared-heap management or locks. More recent GASNet clients such as Cray Chapel [19] and Stanford Legion [6] make heavy use of GASNet AMs for moving computation to data. GASNet-EX introduces several improvements to the AM interfaces, primarily related to efficient use of memory and reduced in-memory copies. GASNet-EX AM calls provide for immediate-mode injection and local-completion control, as described previously. These are notable improvements over GASNet-1, where AM injection calls unconditionally block until the message is guaranteed to enter the network, and return only after local completion of the payload. While GASNet-1 has APIs to query the maximum AM payload for distinct classes of message, GASNet-EX refines the precision of these queries; this enables the client to, for instance, take advantage of space otherwise occupied by unused arguments, or to send significantly larger payloads when the destination is reachable through shared memory.

In addition to these incremental improvements, GASNet-EX adds an entirely new family of AM interfaces known as "Negotiated-Payload" AMs (NP-AM). The new NP-AM feature utilizes a split-phase send that, among other new capabilities, allows GASNet-EX to provide a network-level buffer into which the client directly writes its outgoing payload. In clients that construct payloads dynamically (for instance combining a header with data from a higher layer) this eliminates an in-memory copy often required to concatenate the

AM header and to move the payload into memory registered with the network. Our measurements show that this use of NP-AM to reduce memory copies in the critical path improves measured bandwidth on an AM ping-pong benchmark by as much as 14% relative to the traditional "Fixed-Payload" AMs in GASNet-1, as illustrated by the upper (red) series in Fig. 3.

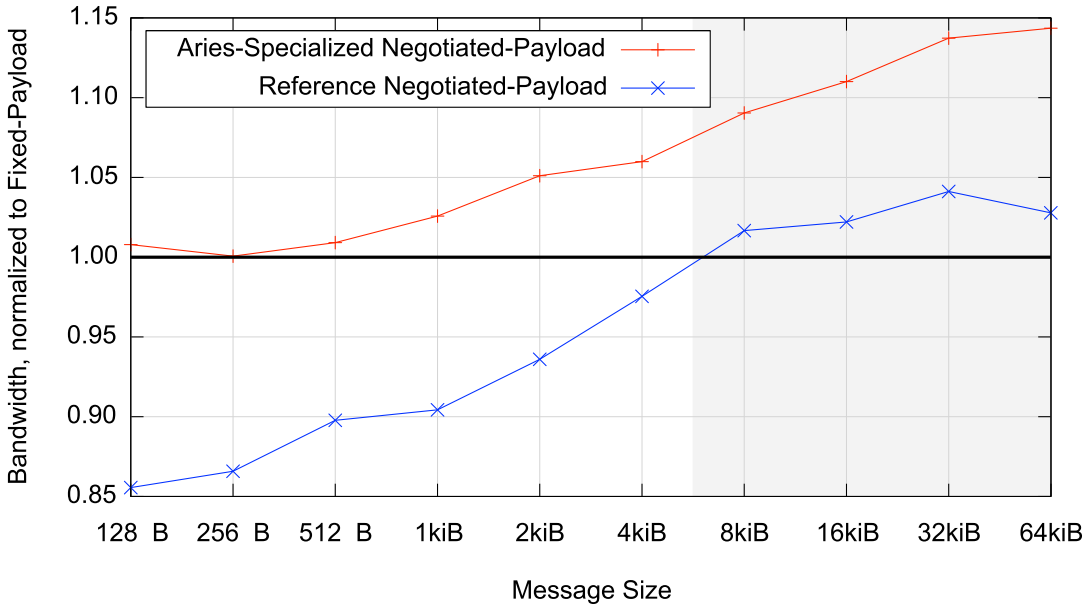

Figure 3: NP-AM speedup of ping-pong test with dynamically generated payload.

2.5 Remote Atomics

Remote atomics are a new feature in GASNet-EX, providing non-blocking interfaces to perform a rich set of operations atomically on several data types in distributed memory. The semantic design for GASNet-EX remote atomics is derived from that used in UPC 1.3 [75], where operations are performed with respect to an "atomic domain". An atomic domain is constructed (outside the critical path) by specifying a data type and a set of atomic operations, and is later used to initiate any of the given operations on data of the given type. Use of atomic domains allows for selection of the fastest-available implementation that can *correctly* provide the set of atomic operations needed concurrently by the application. This is important because in general one cannot mix atomics offloaded to a NIC with others implemented using the host CPU concurrently to the same target location, due to coherency problems on many modern systems. Atomic domains address this by selecting NIC offload implementations if and only if the entire set of operations given at domain creation can be coherently offloaded, and a CPU-based implementation otherwise. Optional flags to atomic domain construction can guide algorithm selection in application-specific ways, for example to favor the performance of accesses across the network, or trade it off for improved performance of updates from shared-memory peers.

Our measurements show there is significant advantage to offloading of atomic operations to the network hardware support provided by Cray Aries, as compared to a network-independent reference implementation, such as one a client author could write using AMs. The latency of a 64-bit fetch-and-add was reduced by 70% on a point-to-point test, and a hot-spot test was shown to scale robustly as illustrated in Fig. 4. Future work to offload atomic operations to InfiniBand network hardware is expected to yield qualitatively similar results.

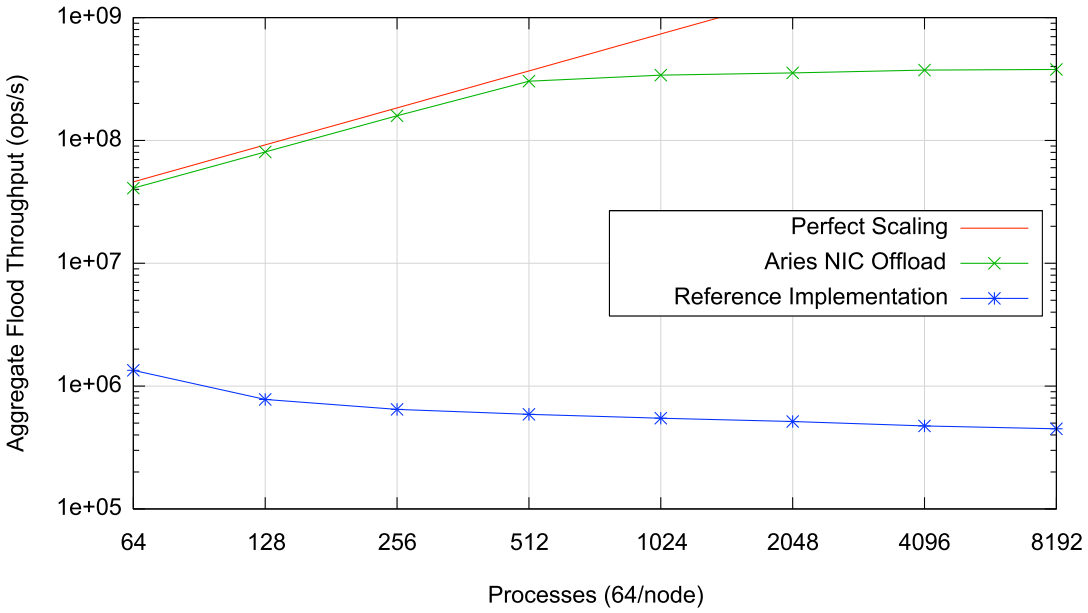

Figure 4: Scaling of a remote atomics hot-spot test on the Cray Aries network.

2.6 Non-contiguous RMA

A set of extensions to GASNet-1 were proposed in [12] to support non-contiguous RMA operations. These types of operations may be generated by optimizations performed by UPC and CAF compilers, or can be used by application authors or distributed array libraries to express transfer of multidimensional array sections. The extensions are jointly referred to as "VIS" and include "Vector", "Indexed" and "Strided" APIs for Put and Get, differing in the generality (and thus size) of the metadata used to describe the source and destination regions. The Strided design for multidimensional array sections was influenced by that of ARMCI [62]. While GASNet-1 fully implemented the VIS extensions, they were never included in the formal specification. GASNet-EX incorporates the VIS APIs (no longer considered extensions) with updates to express local-completion control and immediate-mode injection. The implementation of VIS in GASNet-EX leverages new EX features (most notably the Active Message enhancements) to improve performance relative to GASNet-1. Fig. 5 demonstrates the bandwidth improvement of a microbenchmark measuring a representative 3-d Strided Put operation, implemented inside GASNet-EX using traditional AM or NP-AM, relative to the bandwidth achieved by the GASNet-1 VIS implementation.

2.7 Collective Communication

As illustrated in Sec. 2.1, point-to-point communication in GASNet-EX uses a `(team, rank)` pair to identify the peer, whereas GASNet-1 took only a rank. In addition to this role in point-to-point communication, teams name the participants in collective communications operations. As with VIS, collectives were implemented in GASNet-1 [64] but never appeared in a formal specification. GASNet-EX adds specification and implementation of collective operations, with key improvements over the APIs implemented in GASNet-1. GASNet-EX collectives are always non-blocking, using the same Event type as all other asynchronous operations, whereas GASNet-1 has a distinct type and APIs for tracking completion of collectives. The use of the general Event infrastructure enables local-completion control for collectives. Finally, the GASNet-EX reduction operation includes type information, lacking from GASNet-1, which is critical to enabling network hardware offload.

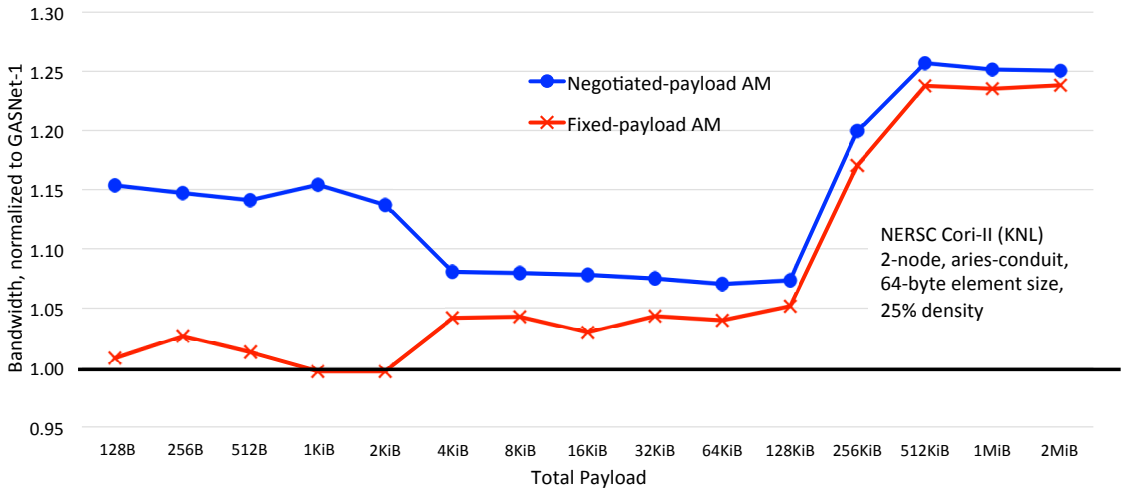

Figure 5: Improved Strided Put performance, relative to GASNet-1.

2.8 Design Improvements for Scalability

One of the primary motivations behind the redesign of GASNet is to improve scalability of both the implementation and client-facing APIs—a necessary step towards achieving exascale performance on upcoming systems, which are expected to reach millions of cores. Several GASNet-1 APIs were designed without sufficient allowance for such extreme-scale systems, and GASNet-EX replaces these with more scalable alternatives. For example, the GASNet-1 function to query segment information writes to a client-allocated array with entries for *every* process in the job, imposing a non-scalable requirement on both the client and library. GASNet-EX instead provides a query to retrieve information about a *selected* peer, consuming only a small constant amount of memory and enabling implementations that discover peer information on-demand at scale. There are also new scalable queries for processes to discover information regarding co-located peers within a hierarchical system.

GASNet-EX API extensions to use offset-based addressing in RMA calls (Sec. 2.1) will enable client runtimes supporting symmetric heap features to eliminate non-scalable base address tables. Immediate-mode injection (Sec. 2.3) improves support for asynchrony, strengthening latency tolerance and enabling dynamic adjustment to congestion and load imbalance that become more prevalent at scale. The redesigned GASNet-EX teams interface (Sec. 2.7) includes scalable rank translation queries designed to keep non-scalable tables out of client data structures. Enhancements enabling NIC hardware offload of collectives (Sec. 2.7) and remote atomics (Sec. 2.5) are expected to become increasingly important at extreme scale.

3 RMA Microbenchmarks

Both the specification and implementation of GASNet-EX are still evolving. However, as described in the previous section, the new features are already capable of delivering measurable benefits for use cases of interest. These new features have *not* come at the cost of GASNet-1's core competencies in RMA and AM. This section presents microbenchmarks measuring the RMA performance of GASNet-EX on four systems, demonstrating that it remains competitive with MPI-3 RMA. Application-level benchmarks would introduce overheads specific to the client runtime, and are outside the scope of this paper.

Our measurements attempt to reproduce the experience of a non-expert end-user. On three vendor-integrated systems using environment modules, we have used the default modules with only one exception to be described below. On a commodity InfiniBand cluster we have used the compiler pre-installed as `/usr/bin/gcc`. When building software (including GASNet-EX and all microbenchmarks) we followed the instructions without the application of any expert knowledge. No configuration settings, environment vari-

ables, or similar means were used to improve the performance of GASNet-EX or MPI[1]. We benchmarked GASNet-EX version `2018.9.0` using two tests selected from those provided with the source code distribution. For MPI-3 benchmarking we have selected the publicly available Intel MPI Benchmarks [43] (IMB), version `v2018.1`.

3.1 Description of the Systems

The first two systems are the partitions of the Cray XC40 [23, 31] system at NERSC [61], known as "Cori". Both use a Cray Aries [1] network, but they have distinct node types: "Cori-I" [59] nodes each have two Intel Xeon E5-2698v3 16-core "Haswell" processors and "Cori-II" [60] nodes each have a single Intel Xeon Phi 7250 "KNL" processor with 272 hardware threads. All tests on the Cori systems were compiled with the default programming environment modules: `PrgEnv-intel/6.0.4`, `intel/18.0.1.163` and `cray-mpich/7.7.0`. The only non-default modules used were for CPU-specific optimization: following NERSC's user documentation, compilation of code to execute on Cori-I and Cori-II used the `craype-haswell` and `craype-mic-knl` environment modules, respectively.

The "Gomez" system at JLSE [2] is a commodity InfiniBand cluster. Each node has two Intel Xeon E7-8867v3 "Haswell-EX" CPUs and is connected to a 100Gb/s EDR InfiniBand network by a Mellanox "ConnectX-4" Host Channel Adapter (HCA). All tests on this system were compiled with the system-default GNU compilers, version `4.8.5 20150623 (Red Hat 4.8.5-16)`. MPI tests used MVAPICH2 [58], version `2.3`.

The "Summitdev" [67] system at OLCF [66] consists of IBM S822LC [76] nodes, each with two 10-core POWER8 CPUs and connected to a 100Gb/s EDR InfiniBand network by two Mellanox "ConnectX-4" HCAs, each with affinity to a single socket. Software compiled on this system used the default IBM XL compilers, version `V13.1.6`. MPI tests use the default IBM Spectrum MPI, version `10.2.0.0-20180110`.

3.2 RMA Flood Bandwidth Benchmark

A "flood bandwidth" benchmark measures achievable bandwidth at a given transfer size by initiating a large number of non-blocking transfers and waiting for them all to fully complete. The reported metric is the total volume of data transferred, divided by the total elapsed time. We report uni-directional (one initiator to one target) flood bandwidths, where the passive target waits in a barrier.

For GASNet-EX we used the `testlarge` microbenchmark to measure performance of the `gex_RMA_PutNBI` and `gex_RMA_GetNBI` functions, synchronized with a final `gex_NBI_Wait`. We measured flood bandwidth of the `MPI_Put` and `MPI_Get` functions using the "Aggregate" timings from, respectively, the `Unidir_put` and `Unidir_get` tests from the `IMB-RMA` suite (these tests measure the time to issue RMA and synchronize using `MPI_Win_flush`, within a passive-target access epoch established by a `MPI_Win_lock(SHARED)` call outside the timed region – see [43] for further details). The `testlarge` benchmark reports bandwidths in units of "MiB/s" (2^{20} bytes per second), whereas `IMB-RMA` uses "MB/s" (10^6 bytes per second). Both have been converted to "GiB/s" (2^{30} bytes per second) for the plots which follow. To allow comparison between RMA and message passing, the plots which follow also report uni-directional bandwidth of `MPI_Isend`/`MPI_Irecv`, from the "Aggregate" timings of the `Uniband` test from the `IMB-MPI1` suite.

All tests ran between two compute nodes, using a single process per node. Data was collected from 16 distinct batch jobs, each running one instance of each GASNet-EX and MPI test back-to-back. Each data point plotted reports the maximum achieved bandwidth for that benchmark and transfer size. For RMA tests we used 100,000 iterations on the Aries systems, and 10,000 on the EDR InfiniBand systems. For the message passing test, we used 5,000 and 500 iterations, respectively.

In Fig. 6, "×" markers denote GASNet-EX RMA, "○" markers denote MPI-3 RMA, and "+" markers denote MPI message passing. RMA Put results are distinguished by the use of solid lines in shades of blue, while RMA Get results use dot-dashed lines in shades of red. Dashed green lines are message-passing results.

On three of the four systems, the bandwidth of GASNet-EX Put and Get are seen to rise rapidly to saturation, at payloads as small as 4 or 8KiB. All GASNet-EX saturation bandwidths are at least comparable

[1] On Summitdev we set one environment variable to restrict the MPI implementation to a single rail of the dual-rail network, to provide a meaningful comparison to GASNet-EX. We recommend this configuration because use of a single rail per process can yield significant latency improvements.

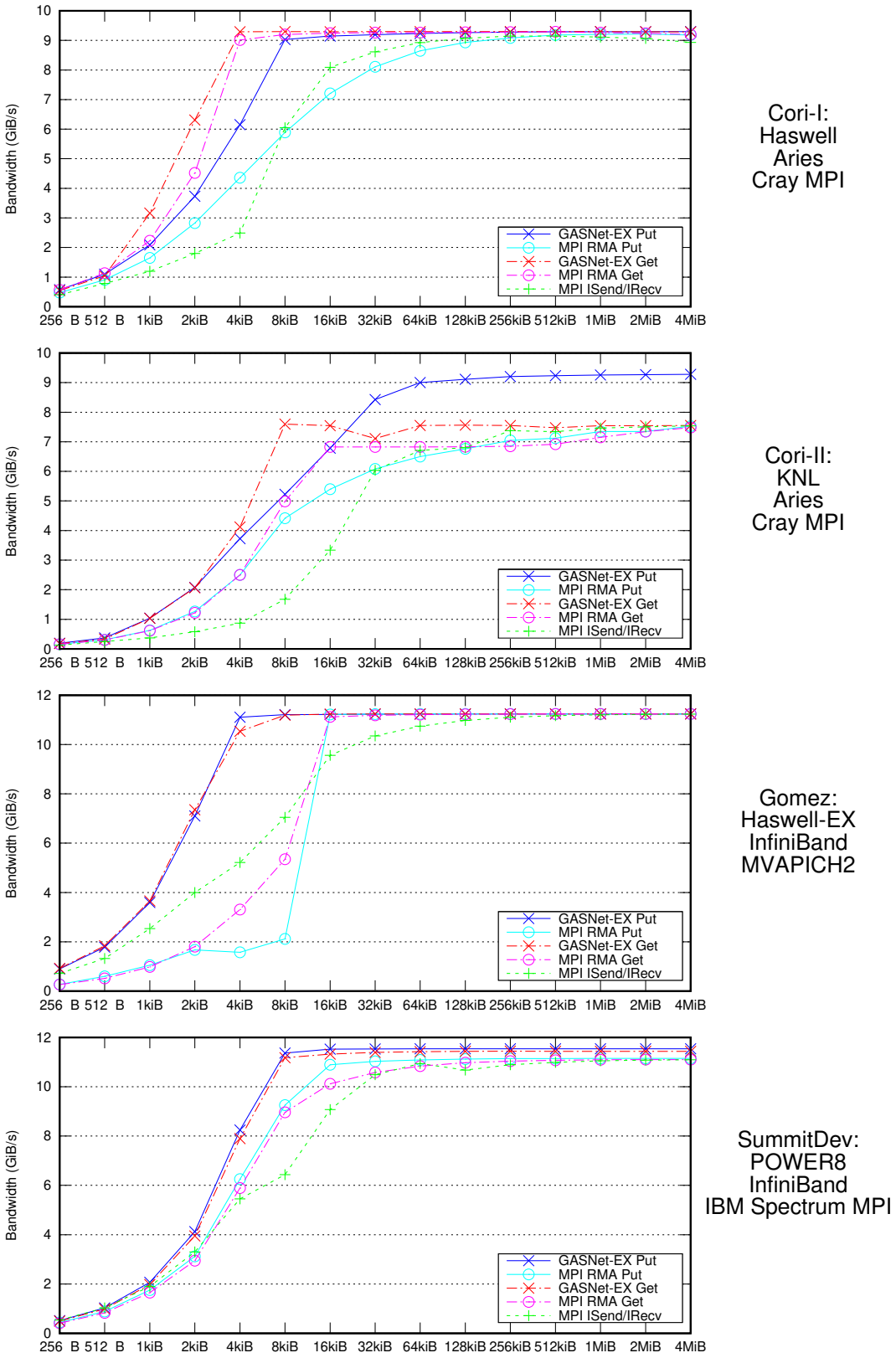

Figure 6: Uni-directional flood bandwidth versus transfer size.

Table 1: Round-trip latency of 8-Byte RMA accesses.

System	8-Byte RMA Put Latency			8-Byte RMA Get Latency		
	GASNet-EX	MPI-3 RMA	Ratio	GASNet-EX	MPI-3 RMA	Ratio
Cori-I	1.07 μs	1.20 μs	0.89	1.43 μs	1.57 μs	0.91
Cori-II	2.15 μs	3.42 μs	0.63	2.60 μs	4.06 μs	0.64
Gomez	1.41 μs	1.51 μs	0.94	1.82 μs	1.91 μs	0.95
Summitdev	1.61 μs	8.10 μs	0.20	2.10 μs	8.13 μs	0.26

to their MPI-3 RMA analogue. The KNL-based Cori-II system shows behavior different from the other three, and this is particularly unexpected because Cori-I and Cori-II use the same network and software versions. Doerfler et al. [25] identified the cause of the two maximum bandwidth plateaus on this system as an issue with PCIe bus latency.

3.3 RMA Latency Benchmark

We next report on the round-trip latency of GASNet-EX and MPI-3 RMA operations. These benchmarks report the mean time to fully complete a single RMA Put or Get operation, computed by timing a long sequence of blocking operations. For GASNet-EX we measured the `gex_RMA_PutBlocking` and `gex_RMA_GetBlocking` functions using the `testsmall` microbenchmark. For MPI-3 benchmarking we report the "Non-aggregate" timings from the `Unidir_put` and `Unidir_get` tests from the `IMB-RMA` suite, which are semantically equivalent to the GASNet-EX test. These are the same tests used to measure flood bandwidth, but differ by executing a sequence of `MPI_Put` (or `MPI_Get`) calls alternating with calls to `MPI_Win_flush`, whereas the "Aggregate" timings used for bandwidth have only a *single* `MPI_Win_flush` at the end.

Data was collected from the same 16 batch jobs described for the flood bandwidth benchmark. Our results are summarized in Tbl. 1, which reports the minimum latency achieved by each benchmark for the cases of blocking RMA Put and Get with 8-byte payloads. Each row includes the ratio of the corresponding GASNet-EX and MPI-3 results, which is also representative of timings over power-of-two sizes from 4 bytes to 1024 bytes (not shown). In all cases measured, GASNet-EX demonstrated comparable or improved latency relative to MPI-3 RMA.

4 Related Work

The GASNet library provides communication services for a wide variety of runtime clients, as discussed in Sec. 1.1. The communication requirements of these clients can be broadly summarized as including portable, high-performance RMA for one-sided data motion, and Active Messages that trigger remote code execution as a building block for higher-level distributed protocols. This section describes competing middleware efforts that provide related facilities.

At the time the GASNet project began, the most notable related effort was the RMA extensions introduced in the MPI-2 specification [54]. The widespread availability and investment in MPI implementations over the years makes MPI a politically attractive communication substrate. However as explained in [15], a number of fundamental semantic defects in the MPI-2 RMA specification made it unsuitable for practical use as communication middleware for PGAS runtimes – justifying the investment in approaches tailored to the needs of PGAS clients, such as GASNet and ARMCI [62]. Most of these defects were subsequently addressed fourteen years later in the MPI-3 specification [56]. Most importantly, the introduction of the (optional) `MPI_WIN_UNIFIED` memory model and dynamic MPI RMA windows made it feasible to use passive-target MPI RMA to satisfy the basic RMA communication needs of some PGAS runtimes. Sec. 3 demonstrates the performance of GASNet-EX RMA is competitive with that of MPI RMA in several widely used MPI-3 implementations. A number of efforts are underway to improve the behavior and performance of MPI-3 RMA implementations, for example [34, 39]. During the six month interval that we performed data collection in preparation for this paper, we've observed noticeable improvement in the performance of all three MPI-RMA implementations measured. There are also efforts underway to implement some PGAS systems

using MPI-3 RMA [80]. MPI-3 offers a wide variety of features that are absent from GASNet-EX, which focuses on providing AM and RMA services for parallel runtimes.

However the current MPI-3 API still lacks several features that are important to GASNet-EX's clients – most notable amongst these is Active Messages, which are critical to systems such as Berkeley UPC, UPC++, Legion, Chapel, and others. Prior work [13] has demonstrated that emulating active message functionality over MPI's message-passing interfaces is possible, but the performance may be prohibitively expensive relative to native implementations. ComEx [24] implements the Global Arrays PGAS library using MPI message passing and hidden progress processes operating on MPI shared memory segments (deliberately avoiding the MPI-RMA interface), however they admit their approach is insufficient for providing clients with Active Message functionality.

GASNet-EX and MPI-3 RMA differ in other details of relevance to PGAS clients. For example, GASNet-EX offers fine-grained control over the synchronization of RMA operations, whereas MPI-3 RMA notably lacks the ability to independently synchronize remote completion of concurrent Put and Accumulate operations targeting the same remote window. Atomic domains in GASNet-EX enable aggressive use of offload hardware for remote atomics, even when concurrently mixing different atomic update operations to the same target location, whereas MPI's accumulate semantics disallow this. The GASNet-EX immediate-mode injection feature introduced in Sec. 2 has no direct analogue in MPI RMA.

There are two relatively recent industry-driven efforts to provide portable, open-source HPC networking middleware to sit below parallel runtimes, similar to GASNet-EX. OpenFabrics libfabric [35, 68] portably provides one-sided RMA and remote atomic operations suitable for implementing PGAS-style RMA, in addition to tag matching and messaging queues suitable for implementing message-passing APIs. There are implementations of GASNet, OpenSHMEM, MPI-3 and other models over libfabric, which in turn offers providers that run across a variety of high-performance network fabrics. Unified Communications X (UCX) [72, 74] is a similar, independent effort to provide a portable network abstraction layer for authors of HPC middleware such as MPI and PGAS runtimes.

5 Conclusions

This paper describes GASNet-EX, a portable, open-source, high-performance, next-generation communication library designed to efficiently support the networking requirements of distributed runtime systems in future exascale machines. We presented several extensions and enhancements that GASNet-EX adds to the traditional GASNet APIs, including: independent local-completion control, immediate-mode communication injection, Active Message improvements, and remote atomic operations, as well as improved support for non-contiguous RMA, teams and collective communication. Initial evaluations of the new features and enhancements are positive, showing a potential for improved communication efficiency by reducing buffer memory size and lifetime, eliminating injection stalls, and streamlining several GASNet interfaces to maximize scalability. Finally, we presented microbenchmark results demonstrating the RMA performance of GASNet-EX is competitive with several MPI-3 implementations on modern HPC-relevant systems.

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.

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

1. Alverson, B., Froese, E., Kaplan, L., Roweth, D.: Cray XC Series Network. White Paper WP-Aries01-1112, Cray Inc. (November 2012), https://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf
2. Argonne National Laboratory: Joint Laboratory for System Evaluation. https://www.jlse.anl.gov
3. Bachan, J., Baden, S.B., Bonachea, D., Hargrove, P.H., Hofmeyr, S., Jacquelin, M., Kamil, A., van Straalen, B.: UPC++ Specification, v1.0 Draft 8. Tech. Rep. LBNL-2001179, Lawrence Berkeley National Laboratory (September 2018). doi:10.25344/S45P4X
4. 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. In: Proceedings of the Second Annual PGAS Applications Workshop. pp. 7:1–7:4. PAW17, ACM, New York, NY, USA (2017). doi:10.1145/3144779.3169108
5. Barrett, B.W., Brightwell, R., Hemmert, S., Pedretti, K., Wheeler, K., Underwood, K., Riesen, R., Maccabe, A.B., , Hudson, T.: The Portals 4.0 Network Programming Interface. Technical Report SAND2012-10087, Sandia National Laboratories (November 2012). doi:10.2172/1088065
6. Bauer, M., Treichler, S., Slaughter, E., Aiken, A.: Legion: expressing locality and independence with logical regions. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. SC '12 (2012). doi:10.1109/SC.2012.71
7. Bell, C., Bonachea, D.: A new DMA registration strategy for pinning-based high performance networks. In: Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS) (2003). doi:10.1109/IPDPS.2003.1213363
8. Bell, C., Chen, W., Bonachea, D., Yelick, K.: Evaluating Support for Global Address Space Languages on the Cray X1. In: 19th Annual International Conference on Supercomputing (ICS) (June 2004). doi:10.1145/1006209.1006236
9. Birrittella, M.S., Debbage, M., Huggahalli, R., Kunz, J., Lovett, T., Rimmer, T., Underwood, K.D., Zak, R.C.: Intel Omni-path Architecture: Enabling Scalable, High Performance Fabrics. In: IEEE 23rd Annual Symposium on High-Performance Interconnects. pp. 1–9 (Aug 2015). doi:10.1109/HOTI.2015.22
10. Bocchino, R.L., Adve, V.S., Chamberlain, B.L.: Software transactional memory for large scale clusters. In: Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'08). pp. 247–258. ACM, New York, NY, USA (2008). doi:10.1145/1345206.1345242
11. Boden, N.J., Cohen, D., Felderman, R.E., Kulawik, A.E., Seitz, C.L., Seizovic, J.N., Su, W.K.: Myrinet: A Gigabit-per-Second Local Area Network. IEEE Micro **15**(1), 29–36 (1995). doi:10.1109/40.342015
12. Bonachea, D.: Proposal for extending the UPC memory copy library functions and supporting extensions to GASNet, v2.0. Tech. Rep. LBNL-56495-v2.0, Lawrence Berkeley National Laboratory (March 2007). doi:10.2172/920052
13. Bonachea, D.: AMMPI home page, https://gasnet.lbl.gov/ammpi
14. Bonachea, D.: GASNet specification, v1.1. Tech. Rep. UCB/CSD-02-1207, University of California, Berkeley (October 2002). doi:10.25344/S4MW28
15. Bonachea, D., Duell, J.: Problems with using MPI 1.1 and 2.0 as compilation targets for parallel language implementations. International Journal of High Performance Computing and Networking **1**(1-3), 91–99 (2004). doi:10.1504/IJHPCN.2004.007569
16. Bonachea, D., Hargrove, P., Welcome, M., Yelick, K.: Porting GASNet to Portals: Partitioned Global Address Space (PGAS) Language Support for the Cray XT. In: Cray Users Group (2009). doi:10.25344/S4RP46
17. Bonachea, D., Hargrove, P.H.: GASNet specification, v1.8.1. Tech. Rep. LBNL-2001064, Lawrence Berkeley National Laboratory (August 2017). doi:10.2172/1398512
18. Buntinas, D., Mercier, G., Gropp, W.: Design and evaluation of Nemesis, a scalable, low-latency, message-passing communication subsystem. In: Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06). vol. 1, pp. 521–530 (May 2006). doi:10.1109/CCGRID.2006.31
19. 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) pp. 52–60 (2004). doi:10.1109/HIPS.2004.10002
20. Chan, C., Wang, B., Bachan, J., Macfarlane, J.: Mobiliti: Scalable Transportation Simulation Using High-Performance Parallel Computing. In: IEEE International Conference on Intelligent Transportation Systems (ITSC). pp. 634–641 (November 2018). doi:10.1109/ITSC.2018.8569397
21. Charles, P., Grothoff, C., Saraswat, V., Donawa, C., Kielstra, A., Ebcioglu, K., von Praun, C., Sarkar, V.: X10: an object-oriented approach to non-uniform cluster computing. In: Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications. (OOPSLA'05) (2005). doi:10.1145/1103845.1094852

22. Chen, W., Bonachea, D., Duell, J., Husband, P., Iancu, C., Yelick, K.: A Performance Analysis of the Berkeley UPC Compiler. In: Proceedings of the 17th International Conference on Supercomputing (ICS) (June 2003). doi:10.1145/782814.782825
23. Cray, Inc.: Cray XC Series. https://www.cray.com/sites/default/files/Cray-XC-Series-Brochure.pdf, accessed 2018-07-17
24. Daily, J., Vishnu, A., Palmer, B., van Dam, H., Kerbyson, D.: On the Suitability of MPI as a PGAS Runtime. In: 21st International Conference on High Performance Computing (HiPC) (Dec 2014). doi:10.1109/HiPC.2014.7116712
25. Doerfler, D., Austin, B., Cook, B., Deslippe, J., Kandalla, K., Mendygral, P.: Evaluating the networking characteristics of the Cray XC-40 Intel Knights Landing-based Cori supercomputer at NERSC. Concurrency and Computation: Practice and Experience **30**(1), e4297 (September 2017). doi:10.1002/cpe.4297
26. Dotsenko, Y., Coarfa, C., Mellor-Crummey, J.: A Multi-platform Co-Array Fortran Compiler. In: Proc. 13th International Conference on Parallel Architecture and Compilation Techniques (PACT) (2004). doi:10.1109/PACT.2004.1342539
27. Driscoll, M.: PyGAS, https://mbdriscoll.github.io/pygas
28. Dunigan, T.H., Vetter, J.S., Worley, P.H.: Performance evaluation of the SGI Altix 3700. In: International Conference on Parallel Processing (ICPP'05). pp. 231–240 (June 2005). doi:10.1109/ICPP.2005.61
29. Eachempati, D., Jun, H.J., Chapman, B.: An Open-source Compiler and Runtime Implementation for Coarray Fortran. In: Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Models. pp. 13:1–13:8. (PGAS'10), ACM (2010). doi:10.1145/2020373.2020386
30. von Eicken, T., Culler, D.E., Goldstein, S.C., Schauser, K.E.: Active Messages: a Mechanism for Integrated Communication and Computation. In: Proceedings of the 19th International Symposium on Computer Architecture. pp. 256–266. Gold Coast, Australia (May 1992). doi:10.1145/139669.140382
31. Faanes, G., Bataineh, A., Roweth, D., Court, T., Froese, E., Alverson, B., Johnson, T., Kopnick, J., Higgins, M., Reinhard, J.: Cray Cascade: A Scalable HPC System Based on a Dragonfly Network. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. pp. 103:1–103:9. SC '12, IEEE Computer Society Press, Los Alamitos, CA, USA (2012). doi:10.1109/SC.2012.39
32. Fanfarillo, A., Burnus, T., Cardellini, V., Filippone, S., Nagle, D., Rouson, D.: OpenCoarrays: Open-source Transport Layers Supporting Coarray Fortran Compilers. In: Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models. pp. 4:1–4:11. PGAS '14, ACM, New York, NY, USA (2014). doi:10.1145/2676870.2676876
33. GASNet website: https://gasnet.lbl.gov
34. Gerstenberger, R., Besta, M., Hoefler, T.: Enabling Highly-scalable Remote Memory Access Programming with MPI-3 One Sided. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. pp. 53:1–53:12. (SC'13), ACM, New York, NY, USA (2013). doi:10.1145/2503210.2503286
35. Grun, P., Hefty, S., Sur, S., Goodell, D., Russell, R.D., Pritchard, H., Squyres, J.M.: A Brief Introduction to the OpenFabrics Interfaces - A New Network API for Maximizing High Performance Application Efficiency. In: IEEE 23rd Annual Symposium on High-Performance Interconnects. pp. 34–39 (Aug 2015). doi:10.1109/HOTI.2015.19
36. Hargrove, P.H., Bonachea, D.: GASNet-EX performance improvements due to specialization for the Cray Aries network. Tech. Rep. LBNL-2001134, Lawrence Berkeley National Laboratory (March 2018). doi:10.2172/1430690
37. Hargrove, P.H., Bonachea, D., Bell, C.: Experiences Implementing Partitioned Global Address Space (PGAS) Languages on InfiniBand. In: OpenFabrics Alliance International Workshop (April 2008), https://downloads.openfabrics.org/Media/Sonoma2008/Sonoma_2008_Wed_PGAS%20over%20IB.pdf
38. Hilfinger, P., Bonachea, D., Datta, K., Gay, D., Graham, S., Kamil, A., Liblit, B., Pike, G., Su, J., Yelick, K.: Titanium language reference manual. Tech Report UCB/EECS-2005-15.1, University of California, Berkeley (November 2001). doi:10.25344/S4H59R
39. Hjelm, N.: An Evaluation of the One-Sided Performance in Open MPI. In: Proceedings of the 23rd European MPI Users' Group Meeting. pp. 184–187. EuroMPI 2016, ACM, New York, NY, USA (2016). doi:10.1145/2966884.2966890
40. IBM: LAPI programming guide (2003), IBM Technical report SA22-7936-00
41. Ibrahim, K.Z., Yelick, K.: On the Conditions for Efficient Interoperability with Threads: An Experience with PGAS Languages Using Cray Communication Domains. In: Proceedings of the 28th ACM International Conference on Supercomputing. pp. 23–32. ICS '14, ACM (2014). doi:10.1145/2597652.2597657
42. InfiniBand Trade Association home page, https://infinibandta.org
43. Intel Corporation: Introducing Intel®MPI Benchmarks. https://software.intel.com/en-us/articles/intel-mpi-benchmarks, accessed 2018-07-17
44. Intel Corporation: Performance Scaled Messaging 2 (PSM2) Programmer's Guide (April 2017), order No.: H76473-6.0
45. Intrepid Technology, Inc.: Clang UPC Compiler, https://clangupc.github.io

46. Intrepid Technology, Inc.: GCC/UPC Compiler, https://gccupc.org
47. Jose, J., Hamidouche, K., Zhang, J., Venkatesh, A., Panda, D.K.: Optimizing Collective Communication in UPC. In: IEEE International Parallel Distributed Processing Symposium Workshops. pp. 361–370 (May 2014). doi:10.1109/IPDPSW.2014.49
48. Krasnov, A., Schultz, A., Wawrzynek, J., Gibeling, G., Droz, P.Y.: RAMP Blue: A Message-Passing Manycore System In FPGAs. In: Proceedings of International Conference on Field Programmable Logic and Applications. pp. 54–61 (August 2007). doi:10.1109/FPL.2007.4380625
49. Kumar, S., Mamidala, A.R., Faraj, D.A., Smith, B., Blocksome, M., Cernohous, B., Miller, D., Parker, J., Ratterman, J., Heidelberger, P., Chen, D., Steinmacher-Burrow, B.: PAMI: A Parallel Active Message Interface for the Blue Gene/Q Supercomputer. In: 2012 IEEE 26th International Parallel and Distributed Processing Symposium. pp. 763–773 (May 2012). doi:10.1109/IPDPS.2012.73
50. Kumar, S., Dozsa, G., Almasi, G., Chen, D., Giampapa, M.E., Heidelberger, P., Blocksome, M., Faraj, A., Parker, J., Ratterman, J., Smith, B., Archer, C.: The Deep Computing Messaging Framework: Generalized Scalable Message Passing on the Blue Gene/P Supercomputer. In: 22nd Annual International Conference on Supercomputing (ICS) (June 2008). doi:10.1145/1375527.1375544
51. Matsumiya, R., Endo, T.: Scalable RMA-based Communication Library Featuring Node-local NVMs. In: Proceedings of the IEEE High Performance Extreme Computing Conference (HPEC'18). pp. 1–7 (2018). doi:10.25344/S4F594
52. Mattson, T.G., Cledat, R., Cavé, V., Sarkar, V., Budimlic, Z., Chatterjee, S., Fryman, J., Ganev, I., Knauerhase, R., Lee, M., Meister, B., Nickerson, B., Pepperling, N., Seshasayee, B., Tasirlar, S., Teller, J., Vrvilo, N.: The Open Community Runtime: A runtime system for extreme scale computing. In: IEEE High Performance Extreme Computing Conference (HPEC). pp. 1–7 (Sept 2016). doi:10.1109/HPEC.2016.7761580
53. Mellanox Technologies Inc.: MellanoX Messaging Library User Manual, Rev 2.1 (2014), document Number: 4113
54. MPI Forum: MPI–2: a message-passing interface standard. International Journal of High Performance Computing Applications **12**, 1–299 (1998), https://www.mpi-forum.org/docs/mpi-2.0/mpi-20.ps
55. MPI Forum: MPI: A message-passing interface standard, v1.1. Technical report, University of Tennessee, Knoxville (June 12, 1995), https://www.mpi-forum.org/docs/mpi-1.1/mpi-11.ps
56. MPI Forum: MPI: A message-passing interface standard, version 3.0. Technical report, University of Tennessee, Knoxville (September 21, 2012), https://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf
57. Murai, H., Nakao, M., Iwashita, H., Sato, M.: Preliminary Performance Evaluation of Coarray-based Implementation of Fiber Miniapp Suite Using XcalableMP PGAS Language. In: Proceedings of the Second Annual PGAS Applications Workshop. pp. 1:1–1:7. PAW17, ACM (2017). doi:10.1145/3144779.3144780
58. MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE. http://mvapich.cse.ohio-state.edu
59. NERSC: Cori Haswell Nodes. doi:10.25344/S4859K
60. NERSC: Cori Intel Xeon Phi (KNL) Nodes. doi:10.25344/S4D012
61. NERSC: National Energy Research Scientific Computing Center. https://www.nersc.gov
62. Nieplocha, J., Carpenter, B.: ARMCI: A portable remote memory copy library for distributed array libraries and compiler run-time systems. In: Proceedings of the IPPS/SPDP'99 Workshops. pp. 533–546. Springer-Verlag, Berlin, Heidelberg (1999). doi:10.1007/BFb0097937
63. Nishtala, R., Hargrove, P.H., Bonachea, D.O., Yelick, K.A.: Scaling Communication-Intensive Applications on BlueGene/P Using One-Sided Communication and Overlap . In: Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS) (2009). doi:10.1109/IPDPS.2009.5161076
64. Nishtala, R., Zheng, Y., Hargrove, P., Yelick, K.A.: Tuning collective communication for Partitioned Global Address Space programming models. Parallel Computing **37**(9), 576–591 (2011). doi:10.1016/j.parco.2011.05.006
65. Numrich, R.W., Reid, J.: Co-array Fortran for parallel programming. ACM SIGPLAN Fortran Forum **17**(2), 1–31 (1998). doi:10.1145/289918.289920
66. Oak Ridge Leadership Computing Facility. https://www.olcf.ornl.gov
67. Summitdev. https://www.olcf.ornl.gov/tag/summitdev/, accessed 2018-07-17
68. OpenFabrics libfabric home page, https://ofiwg.github.io/libfabric/
69. Petrini, F., chun Feng, W., Hoisie, A., Coll, S., Frachtenberg, E.: The Quadrics network (QsNet): high-performance clustering technology. In: HOT 9 Interconnects. Symposium on High Performance Interconnects. pp. 125–130 (2001). doi:10.1109/HIS.2001.946704
70. Pophale, S., Nanjegowda, R., Curtis, T., Chapman, B., Jin, H., Poole, S., Kuehn, J.: OpenSHMEM Performance and Potential: A NPB Experimental Study. In: Proceedings of the 6th Conference on Partitioned Global Address Space Programming Models (PGAS'12). (2012), https://www.osti.gov/biblio/1055092
71. Shah, V.B.: An Interactive System for Combinatorial Scientific Computing with an Emphasis on Programmer Productivity. Ph.D. thesis, University of California at Santa Barbara, Santa Barbara, CA, USA (2007)
72. Shamis, P., Venkata, M.G., Lopez, M.G., Baker, M.B., Hernandez, O., Itigin, Y., Dubman, M., Shainer, G., Graham, R.L., Liss, L., Shahar, Y., Potluri, S., Rossetti, D., Becker, D., Poole, D., Lamb, C., Kumar, S., Stunkel,

C., Bosilca, G., Bouteiller, A.: UCX: An Open Source Framework for HPC Network APIs and Beyond. In: IEEE 23rd Annual Symposium on High-Performance Interconnects. pp. 40–43 (Aug 2015). doi:10.1109/HOTI.2015.13
73. Su, H., Gordon, B., Oral, S., George, A.: SCI Networking for Shared-Memory Computing in UPC: Blueprints of the GASNet SCI Conduit. In: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks. pp. 718–725. LCN '04, IEEE Computer Society, Washington, DC, USA (2004). doi:10.1109/LCN.2004.107
74. UCX: Unified Communication X home page, https://openucx.org/
75. UPC Consortium: UPC Language and Library Specifications, v1.3. Tech. Rep. LBNL-6623E, Lawrence Berkeley National Laboratory (Nov 2013). doi:10.2172/1134233
76. Vetter, S., Caldeira, A., Kahle, M.E., Saverimuthu, G., Vearner, K.C.: IBM Power System S822LC Technical Overview and Introduction (December 2015), IBM Form #REDP-5283-00
77. Willenberg, R., Chow, P.: A Heterogeneous GASNet Implementation for FPGA-accelerated Computing. In: Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models. pp. 2:1–2:9. PGAS '14, ACM, New York, NY, USA (2014). doi:10.1145/2676870.2676885
78. Yelick, K., Hilfinger, P., Graham, S., Bonachea, D., Su, J., Kamil, A., Datta, K., Colella, P., Wen, T.: Parallel Languages and Compilers: Perspective from the Titanium Experience. International Journal of High Performance Computing Applications **21**(3), 266–290 (2007). doi:10.1177/1094342007078449
79. Zheng, Y., Kamil, A., Driscoll, M.B., Shan, H., Yelick, K.: UPC++: A PGAS extension for C++. In: IEEE 28th International Parallel and Distributed Processing Symposium. pp. 1105–1114 (May 2014). doi:10.1109/IPDPS.2014.115
80. Zhou, H., Mhedheb, Y., Idrees, K., Glass, C.W., Gracia, J., Fürlinger, K.: DART-MPI: An MPI-based Implementation of a PGAS Runtime System. In: Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models. pp. 3:1–3:11. PGAS '14 (2014). doi:10.1145/2676870.2676875