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

Paul H. Hargrove and Dan Bonachea

gasnet.lbl.gov
doi.org/10.25344/S4PC7M

Parallel Applications Workshop, Alternatives to MPI+X (PAW-ATM)
Held in conjunction with SC19
November 17, 2019
Overview

• Introduce GASNet-EX
• Introduce Active Messages
• Discuss protocols for implementing Active Messages
• Measure a new “target-side reassembly” protocol for Active Messages on the Cray XC’s Aries network
GASNet-1: Overview

• Started in 2002 to provide a portable network communication runtime for three PGAS languages:
  - UPC, CAF and Titanium

• Primary features:
  - Non-blocking RMA (one-sided Put and Get)
  - Active Messages (simplification of Berkeley AM-2)

• Motivated by semantic issues in (then current) MPI-2.0
  - Dan Bonachea, Jason Duell, "Problems with using MPI 1.1 and 2.0 as compilation targets for parallel language implementations", IJHPCN 2004. doi.org/10.25344/S4JP4B
GASNet: Adoption and Portability

• Client runtimes
  - UPC++
  - Berkeley UPC
  - GCC/UPC
  - Clang UPC
  - Chapel
  - Legion
  - Titanium
  - Rice Co-Array Fortran
  - OpenUH Co-Array Fortran
  - OpenCoarrays in GCC Fortran
  - OpenSHMEM reference impl.
  - Omni XcalableMP
  - At least 7 others known to us

• Network conduits
  - OpenFabrics Verbs (InfiniBand)
  - Mellanox MXM and VAPI (InfiniBand)
  - Cray uGNI (Gemini and Aries)
  - Intel PSM2 (OmniPath)
  - UDP (any TCP/IP network)
  - MPI 1.1 or newer
  - IBM PAMI (BG/Q and others)
  - IBM DCMF (BG/P)
  - IBM LAPI (Colony and Federation)
  - Cray Portals3 (Seastar)
  - OFI/libfabric
  - Sandia Portals4
  - SHMEM (Cray X1 and SGI Altix)
  - Quadric elan3/4 (QsNet I/II)
  - Myricom GM (Myrinet)
  - Dolphin SISCI
  - Shared memory (no network)

• Supported platforms
  - Over 10 compiler families, 15 operating systems and dozens of architectures

* These lists and counts include both current and past support
GASNet-EX: Overview

• GASNet-EX is the next generation of GASNet
  - Addressing needs of newer programming models such as UPC++, Legion and Chapel
  - Incorporating over 15 years of lessons learned
  - Provides backward compatibility for GASNet-1 clients

• Motivating goals include
  - Support more client asynchrony
  - Enable more client adaptation
  - Decrease memory footprint
  - Improve threading support
  - Support offload to network h/w
  - Support multi-client applications
  - Support for device memory
GASNet-EX: Status

• GASNet-EX is still evolving
  - Not every new feature has been implemented yet
  - Most have - with benefits shown in prior work

• Four prominent clients actively adopting GASNet-EX
  - UPC++ and Berkeley UPC Runtime require GASNet-EX
  - Chapel embeds GASNet-EX
  - Legion has started work to use EX-specific features
GASNet-EX: New Features Include

- Local completion control
  - Improved control over buffer lifetime to increase overlap
- Immediate-mode injection
  - Avoid stalls in low-resource conditions
- Negotiated-payload Active Messages
  - Construct messages in GASNet’s buffers to avoid memcpy()
- Remote atomic operations
  - Utilize offload capabilities in modern network interfaces
- Subset teams
- Numerous small API additions and improvements

For details see Languages and Compilers for Parallel Computing (LCPC’18). doi.org/10.25344/S4QP4W
Status: GASNet-EX RMA Performance Versus MPI

- Three different MPI implementations
- Two distinct network hardware types
- On four systems the performance of GASNet-EX matches or exceeds that of MPI RMA and message-passing:
  - 8-byte Put latency 6% to 55% better
  - 8-byte Get latency 5% to 45% better
  - Better flood bandwidth efficiency, typically saturating at \( \frac{1}{2} \) or \( \frac{1}{4} \) the transfer size

8-Byte RMA Operation Latency (one-at-a-time)

GASNet-EX results from v2018.9.0 and v2019.6.0. MPI results from Intel MPI Benchmarks v2018.1.

For more details see Languages and Compilers for Parallel Computing (LCPC’18).
doi.org/10.25344/S4QP4W

More recent results on Summit here replace the paper’s results from the older Summitdev.
Active Messages (AM)

- AM is a restricted form of remote procedure call
  - Executes code (handler) on a remote node
  - Request handler may only send back an optional Reply
  - No other communication is permitted in AM handlers

- Request and Reply APIs take
  - Integer “handler index” (which table entry to run)
  - Zero or more 32-bit integer arguments
  - Optional bulk data payload

- These arguments and payload are provided to handler
AM Payloads / Problem Statement

• Three “categories” depending on presence and handling of the optional payload
  - Short: No payload
  - Medium: Payload buffered by implementation
  - Long: Payload delivered to client-specified address
    AM delivery coupled with RMA payload Put

• AM Long presents implementation challenge: to both...
  - Leverage RDMA h/w for most efficient payload transfer
  - Ensure the payload is in-place before handler runs
AM Long Protocol Tuning

Evaluating alternatives on the Cray XC's Aries network

- Latency between initiator and target
- Overheads (CPU use on initiator and target)
- Bandwidth between initiator and target
- Sensitivity to attentiveness
  - Is more than one library entry needed to complete?
- Timely signaling of local completion
  - Allows initiator to reuse or free payload source memory

“L”, “o” and “g” of LogP model
Ordered Networks (including selectively ordered)

- Simple if available, but seldom “free”
- Aries provides only at cost of defeating multi-pathing
AM Long Protocols 2

Initiator Chaining

- Simplest but also poor by most of our metrics
- Synchronous variant (aka “put-sync-send”) ties up injector until Put is complete [aries used this previously]
- Asynchronous variant relies on attentiveness
AM Long Protocols 3

Rendezvous Get

- Initiator adds source address to the message envelope
- Target uses an RDMA Get for the payload
- Adds a round-trip latency
- Also delays notification of local completion
AM Long Protocols 4

Target-Side Reassembly

• Subject of this presentation
• At very high level:
  - Payload and envelope injected into unordered network
  - Logic at target is tolerant of any reordering

Pipelined injection

Initiator node

Target node

Time

RDMA Put

AM envelope

Reassembly logic matches xfers, triggers AM handler only after arrival of both
Target-side Reassembly

- Both envelope and payload sent without delay
  - Can leverage multiple paths of networks like Aries
  - No attentiveness problem on either end
  - No network round-trips, and thus no stalls
  - No delays in signaling of local completion
  - However, additional network-specific metadata is required to allow target to match them

![Diagram](image.png)

Reassembly logic matches xfers
Target-side Reassembly On Aries

• Target needs a “nonce” to match AM envelope + payload
• Fairly simple to add a field to AM envelope
  - But we didn’t actually need to in this case
    • An existing buffer management field “fits the bill”
• Not always simple to deliver a nonce with an RDMA Put
  - Other network APIs have “Put with immediate data”
  - uGNI API for Aries has a 32-bit source identifier
    • Under software control, allowing us to steal some bits
Performance Results

• Measurements on NERSC’s Cori Phase II
  - Cray XC30
  - 1.4GHz Xeon Phi CPUs

• First results: AM Long ping-pong latency
  - At most one message in flight at any time:
    • Node 0 sends Request of given size
    • Node 1 issues Reply of same size
  - Report average time to complete many iterations
AM Long Ping-Pong Latency

- Figure shows the range 4KiB – 64KiB
- At 4KiB
  - Old: 13.0µs
  - New: 8.7µs
  - Ratio: 33% faster
- 4KiB – 1MiB range: reductions between 3.7µs and 6.1µs
Measuring One-way Injection Overhead

• Where put-sync-send must block for payload RDMA, target-side reassembly can return immediately
  - Reduced injection overhead ➔ more overlap
• Report the average time for many repetitions of
  start timer
  for (int i=0; i < d; ++i)
    gex_AM_RequestLong(..., size, ...);
  end timer
  drain network
• Report for various values of queue depth $d$ and $size$
Reduced Injection Overhead

- Time to inject \( d \) AMs of a given size
  - \( d \): on the x-axis
  - size: color & symbol
  - old/new: dashed/solid
- Average injection time is up to 50x faster
- Remains faster when flow-control sets in
Conclusions

- Presented several algorithms for AM Long on Aries
- Identified “target-side reassembly” as most promising
- Overcame challenge of coupling nonce with payload
- Presented microbenchmarks showing improvement
- We believe this algorithm is a good choice for other networks with similar properties
- New implementation released in GASNet-EX 2019.9.0
THANK YOU

gasnet.lbl.gov

doi.org/10.25344/S4PC7M
Acknowledgements

• This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy’s Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation’s exascale computing imperative.

• 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.