# GASNet-EX Memory Kinds: Support for Device Memory in PGAS Programming Models


Paul H. Hargrove, Dan Bonachea, Colin A. MacLean, Daniel Waters

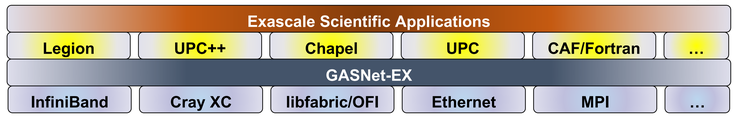
Applied Mathematics and Computational Research Division, Lawrence Berkeley National Laboratory

https://gasnet.lbl.gov

pagoda@lbl.gov

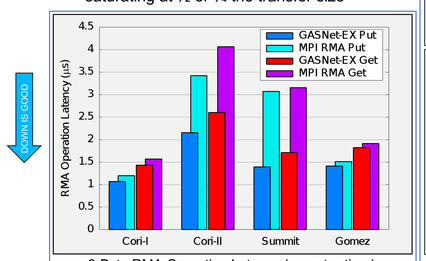

## GASNet-EX Background

- GASNet-EX is communications middleware to support exascale clients
  - Widely adopted for implementation of Partitioned Global Address Space (PGAS) programming models
  - One-sided communication – Remote Memory Access (RMA)
  - Active Messages (AMs) - remote procedure call
  - Implemented over native APIs of all networks of interest to DOE
- GASNet-EX is an evolution of GASNet-1 for exascale
  - Retains GASNet-1's wide portability (laptops to supercomputers)
  - Focus remains on one-sided RMA and Active Messages
  - Reduces CPU and memory overheads
  - Improves support for aggressive multi-threading
  - Adds support for device memory (GPUs)

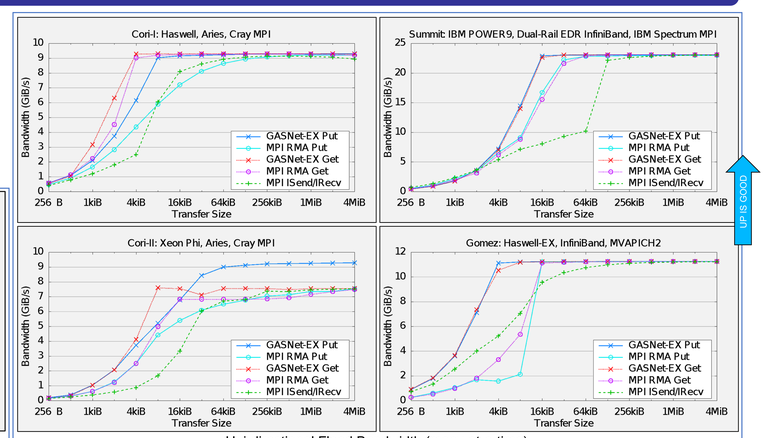



## GASNet-EX Host Memory RMA Performance versus MPI RMA and Isend/Irecv

- 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 ½ or ¼ the transfer size




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




Uni-directional Flood Bandwidth (many-at-a-time)

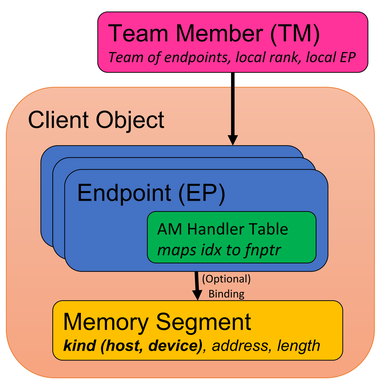
GASNet-EX results from v2018.9.0
MPI results from Intel MPI Benchmarks v2018.1

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

## GASNet-EX Memory Kinds

- Performance for host memory RMA (see panel above) is largely due GASNet-EX's ability to efficiently utilize Remote Direct Memory Access (RDMA) capabilities in modern network hardware
  - No remote CPU involvement required for network adapter to access application memory
  - Simple and efficient implementation due to good semantic fit to the PGAS model
- The goal: extend GASNet-EX RMA to memory in devices such as GPUs, in addition to host memory
  - Challenge 1: RDMA capability for GPU memory is needed for best performance
    - Solved by the hardware vendors via GPUDirect RDMA (see upper-right panel)
  - Challenge 2: Retaining efficiency in the GASNet-EX implementation
    - This is addressed by our design of a new "memory kinds" abstraction in GASNet-EX
  - Challenge 3: Design for extensibility to other byte- or block-addressable "memories"
    - Our design encompasses such possible kinds as FPGA and storage
- Our solution: extend the concept of the "remote-access segment"
  - The arguments to every RMA operation specify a local and remote communications endpoint
  - To be valid for RMA operations, an endpoint must have an associated "bound segment"
  - NEW:
    - A "kind" can be created for each device: e.g. "CUDA device 0", "HIP device 2"
    - Instead of just a single implicit segment, multiple explicit segments are supported
    - Creation of a segment names the kind, along with a size and optional address
- This design means every RMA operation knows the device(s) involved from the existing arguments
  - Device addresses identified *without* dynamic queries or interposing on memory management APIs
- Good match to UPC++ and Legion programming models which already track host vs. device memory

### High-level Object Model




## GASNet-EX Support for GPUDirect RDMA (GDR)

- Memory kinds concept expresses use of device memory in RMA endpoint arguments
  - Implementation can easily identify appropriate transfer mechanism, including hardware-assisted technologies such as GPUDirect RDMA (GDR)
- GASNex-EX supports GDR in recent releases
  - Removes host CPU and memory bottlenecks from one-sided transfers to/from GPU memory, achieving true zero-copy (see diagram at right →)
  - Currently supports Nvidia and AMD GPUs; and Mellanox network adapters
  - Other accelerators and networks are the subject of future work, including
    - Intel GPUs; HPE Slingshot-11 network

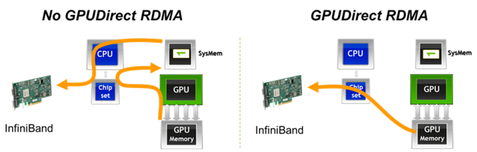
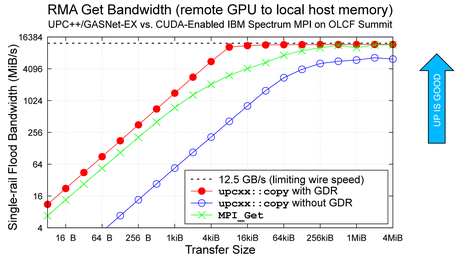



Image source: https://developer.nvidia.com/blog/introduction-cuda-aware-mpi/

## UPC++ Microbenchmark Results with GPU Memory

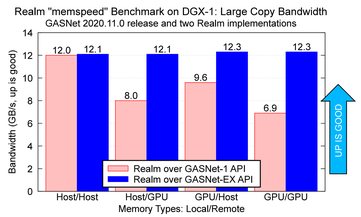
- UPC++ is our C++ productivity layer over GASNet-EX
  - Has its own analogous memory kinds concepts
  - Uses `upcxx::copy()` to express RMA between arbitrary shared memory
- Measurements of flood bandwidth of `upcxx::copy()` on OLCF's Summit
- Difference between UPC++ releases with and without GASNet-EX memory kinds shows benefit of GASNet-EX's new support for GDR
  - No longer staging through host memory (true zero-copy)
  - Large transfers: 2x better bandwidth
  - Small transfers: up to 30x better bandwidth
- Get operations to/from GPU memory now perform comparably to host memory
- Comparisons to MPI-3 RMA in CUDA-enabled IBM Spectrum MPI show UPC++ saturating to peak bandwidth at smaller transfer sizes

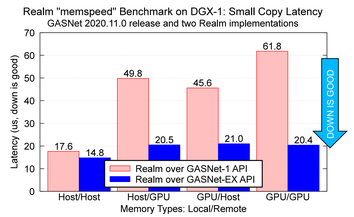



UPC++ results were collected using the version of the `cuda_benchmark` test that appears in the 2020.11.0 release.
MPI results are from `osu_get_bw` test in a CUDA-enabled build of OSU Micro-Benchmarks 5.6.3.
All tests were run between two nodes of OLCF Summit, over its EDR InfiniBand network.

## Legion Microbenchmark Results with GPU Memory

- Legion characterizes itself as "a data-centric programming model for writing high-performance applications for distributed heterogeneous architectures".
- Legion's Realm runtime provides communication services
  - Originally implemented over GASNet-1
  - Still works using legacy API support in current GASNet-EX
- Realm recently introduced a new GASNet-EX backend which embraces EX-specific capabilities
  - Leverages immediate, NPAM, and LC handles for AM
  - Leverages memory kinds for RMA, gaining GDR support
  - Multi-endpoint for RMA to/from additional host memory regions
- Figures at right show some performance benefits of using GDR
  - Large GPU transfers: bandwidth now matches host memory
  - Small GPU transfers: 2.2x to 3.0x latency improvement



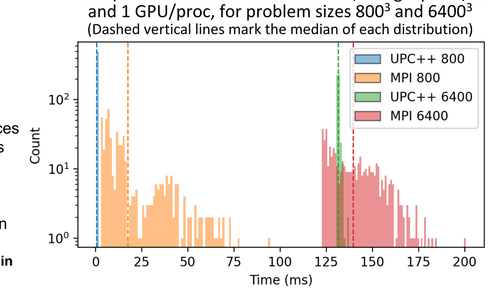





The authors gratefully acknowledge Sean Treichler of the Legion development team for collecting the data we present above.


## UPC++ Application Kernel Performance

- Kokkos[1] is a popular on-node programming model providing C++ abstractions for use of multi-core and GPU-accelerated nodes with a single version of the application source code
- We converted a Kokkos tutorial example[2] from use of MPI message passing to UPC++ RMA
  - Use of UPC++ memory kinds for device memory management
  - Use of `upcxx::copy` for one-sided RMA data movement and remote notification
- Despite no changes to the computation, we observed unexpected performance differences
  - UPC++ RMA delivers better strong scaling (not pictured here) than MPI message passing
  - Even when performance is comparable, MPI version's performance has much larger variance
- This figure shows the relative performance and variance for two problem sizes and fixed resources
  - Histograms show the frequency of execution times, using a sliding window over two time steps to fairly account for MPI Recv and Send matching (this *reduces* MPI's measured variance)
  - UPC++ runs show lower (faster) median time step latency than the corresponding MPI runs
  - UPC++ runs show narrower (lower-variance) distributions than the corresponding MPI runs
- Future work planned to try to understand and eliminate MPI variance, and repeat the comparison
- More complete information will appear in a workshop talk on Friday morning:
  D. Waters, C. A. MacLean, D. Bonachea, P. H. Hargrove. "**Demonstrating UPC++/Kokkos Interoperability in a Heat Conduction Simulation (Extended Abstract)**", In *2021 IEEE/ACM Parallel Applications Workshop, Alternatives To MPI+X (PAW-ATM)*, St. Louis, MO, Nov 2021. 5 pages. https://doi.org/10.25344/S4630V




UPC++ results use UPC++ v2021.3.0 w/ GPUDirect RDMA
MPI results use CUDA-enabled IBM Spectrum MPI v10.3.1.2


[1] https://kokkos.org
[2] https://go.lbl.gov/paw21-kokkos-mpi-heat-conduction



This research was supported 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.
We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.



© 2021, Lawrence Berkeley National Laboratory


https://doi.org/10.25344/S4P306