UPC++ and GASNet-EX: PGAS Support for Exascale Applications and Runtimes (Extended Poster Abstract)

Scott B. Baden, Paul H. Hargrove, Hadia Ahmed, John Bachan, Dan Bonachea,
Steven Hofmeyr, Mathias Jacquelin, Amir Kamil, Brian van Straalen
pagoda@lbl.gov
Lawrence Berkeley National Laboratory, Berkeley CA

INTRODUCTION

Lawrence Berkeley National Lab is developing a programming system to support HPC application development using the Partitioned Global Address Space (PGAS) model. This work consists of two major components: UPC++ (a C++ template library) and GASNet-EX (a portable, high-performance, global-address-space communication library).

1 UPC++

UPC++ [2, 3, 26] is a C++ library that supports Partitioned Global Address Space (PGAS) programming. An early predecessor of UPC++ was released in 2012 [28]. This poster presents a new incarnation of UPC++ with a very different design, which is tailored to meet the needs of exascale applications that require PGAS support. There are three main principles behind the redesign of UPC++. First, wherever possible, *operations are asynchronous* by default, to allow the overlap of computation and communication, and to encourage programmers to avoid global synchronization. Second, all *data motion is explicit*, to encourage programmers to consider the costs of communication. Third, UPC++ encourages the use of *scalable data-structures*, and avoids non-scalable library features. All of these principles are intended to provide a programming model that can scale efficiently to potentially millions of processors.

Like other PGAS models, UPC++ supports physically distributed global memory, which can be referenced via special global pointers. Using global pointers, processes can copy data between their local memory and remote global memory using one-sided Remote Memory Access (RMA) operations. Unlike pointer-to-shared in UPC [27], UPC++ global pointers cannot be directly dereferenced, as this would violate our principle of making all communication explicit. In addition to RMA, UPC++ also supports Remote Procedure Calls (RPCs), whereby the caller can induce a remote process to invoke a user function, including any arguments and generating an optional result for return to the sender.

All operations that involve communication are non-blocking and are managed through an API that includes *futures* and *promises*. A future is the interface through which the status of the operation can be queried and the results retrieved, so it represents the consumer side of a non-blocking operation. Each asynchronous operation has an associated promise object, which is created either explicitly by the user or implicitly by the runtime when the non-blocking operation is invoked. A promise represents the producer side of the operation, and it is through the promise that the results of the operation are supplied and its dependencies fulfilled. A user can pass a promise to a communication operation, which registers a dependency with the promise and subsequently fulfills the dependency when the operation completes. The same promise can be passed to multiple communication operations, so that a user can be notified when all such operations have completed.

Futures allow the construction of elaborate dependence-driven graphs of asynchronously executed operations. A user can *chain* a *callback* to a future via the `.then()` method, and the callback will be invoked on the values encapsulated by the future when they are available. The callback can be a function or a lambda, and it can initiate asynchronous operations of its own. The `.then()` method itself produces a new future, which can have further callbacks chained onto it. Multiple futures can be *conjoined* to produce a single future that represents readiness of all the input futures and their resulting values.

UPC++ has several other powerful features that can enhance programmer productivity and improve performance. For example, UPC++ supports remote atomic operations, which are particularly useful in managing distributed lock-free data structures. UPC++ also supports non-contiguous RMA transfers (vector, indexed and strided), enabling programmers to conveniently express more complex patterns of data movement.

The current version of UPC++ utilizes the GASNet-EX communication library [7] to deliver a low-overhead, high-performance runtime. We present the results for two application motifs that exemplify the benefits of UPC++, and more generally PGAS programming. The first motif is a distributed hash table, which relies on irregular, fine-grained communication, where latency is a limiting factor. The second motif is a direct linear solver for sparse symmetric matrices; it also benefits from the features of UPC++, such as RPCs and RMA.

Fig. 1 demonstrates a strong scaling experiment on NERSC Edison [23] comparing the symPACK solver written with UPC++ [4], against two state-of-the-art direct linear solvers for sparse symmetric matrices: MUMPS 5.1.2 [1] and PaSTiX 5.2.3 [17]. SymPACK makes aggressive use of the RPC and RMA capabilities of UPC++ to enable a pull-based scheduling strategy. Thus these results report the net result of replacing message passing (as implemented by MPI) with RPC and RMA (as implemented by UPC++). As can be seen, symPACK consistently displays lower execution times than the other solvers up to 768 cores (32 nodes), demonstrating the low overhead of UPC++ and the efficiency of the one-sided pull strategy it enables. The same trend is shown by strong scaling experiments conducted on the Extend-Add operation, a building block to multifrontal sparse solvers. A UPC++ implementation using RPCs is consistently faster than two MPI-based implementations on NERSC Cori [22].

SC18, November 11–16, 2018, Dallas, TX, USA

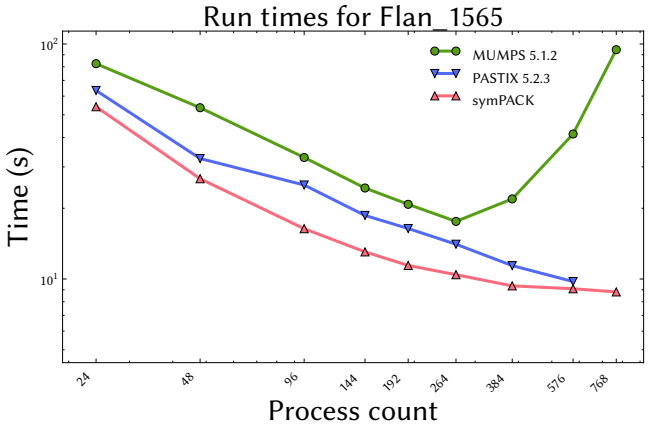

Figure 1: SymPACK/UPC++ vs. competing solvers

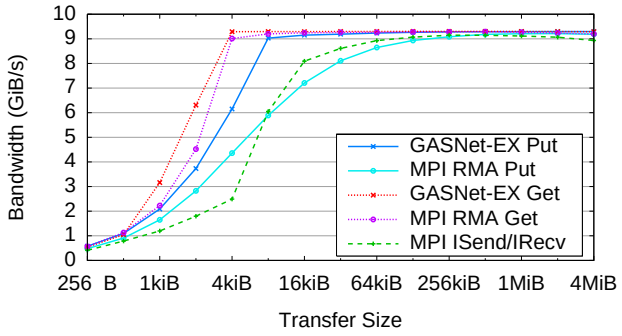

Figure 2: Flood Bandwidth on Cray XC40 (Aries network)

2 GASNET-EX

GASNet-EX [7] is a lightweight communications middleware layer designed to support exascale clients, and is implemented over the native APIs of many networks, including all of those in use at the HPC centers of the U. S. Department of Energy's Office of Science [10]. It features one-sided communication via Remote Memory Access (RMA), remote procedure calls via Active Messages (AMs), remote atomic operations, and non-blocking collectives.

GASNet-EX is an evolution of GASNet-1 [6, 15] and includes a backwards-compatibility layer to enable incremental migration of current GASNet-1 client software. Compared to GASNet-1, GASNet-EX provides enhancements needed for modern asynchronous PGAS models including adjusted interfaces for improved scalability, reduced CPU and memory overheads, and improved many-core support [16]. GASNet-1 has many important clients, including: Stanford's Legion programming system [5], Cray's Chapel language [8], the OpenSHMEM reference implementation [25], the Omni Xcalable Compiler [20], and many UPC [9, 18, 19] and CAF/Fortran08 [11, 12, 14] compilers. To this collection, GASNet-EX adds `UPC++` [26], and use of GASNet-EX is being explored by the PaRSEC [24] and ExaBiome [13] projects. Some of these clients are informing the direction of GASNet-EX development: features critical to `UPC++` are being co-designed, and the GASNet-EX design is influenced by input from the Stanford Legion and Cray Chapel teams, who are migrating from GASNet-1 to GASNet-EX.

Some API enhancements made in GASNet-EX include: endpoint naming using `(team,rank)` (for improved composability), "immediate mode" injection (to avoid stalls due to backpressure), explicit handling of local-completion (for improved buffer lifetime), "Negotiated-Payload" AM (to reduce buffer copying between layers), non-contiguous point-to-point RMA APIs, atomic operations in distributed memory (implemented using NIC offload where available), and non-blocking collectives. Future work enabled by GASNet-EX API changes include: multiple endpoints and segments (to improve multi-threading support and to support access to device memory), and offset-based addressing (for device memory support and improved support of symmetric-heap clients).

The listing below illustrates some of the changes in the API since GASNet-1 with the prototype for an RMA put. New types include `gex_Event_t`, which is a generalization of the GASNet-1 `gasnet_handle_t`; `gex_TM_t`, which denotes a team; `gex_Rank_t`, the index of a rank within the team; and `gex_Addr_t`, which enables virtual-address and offset-based addressing via the same interface. Also shown is the `lc_opt` argument, which introduces explicit control over local completion, generalizing the bulk/non-bulk interfaces of GASNet-1. Adding a new parameter of type `gex_Flags_t` provides extensibility and control over various aspects of an operation (e.g., selecting optional behaviors such as offset-based addressing, or passing assertions about the argument values that can obviate the need for more expensive dynamic checking).

```
gex_Event_t
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);
```

We report the results of RMA flood bandwidth microbenchmarks on several systems. Fig. 2 shows results for one system: Cori Phase-I [21], a Haswell-based Cray XC40 at NERSC. This figure is representative of the results on all systems measured in that bandwidth achieved with GASNet-EX RMA saturates faster than MPI-3 RMA, and reaches the same peak values (or higher).

CONCLUSIONS

GASNet-EX leverages hardware support to portably and efficiently implement Active Messages and Remote Memory Access (RMA). `UPC++` provides higher-level productivity abstractions appropriate for PGAS programming such as: one-sided communication (RMA), remote procedure call, locality-aware APIs for user-defined distributed objects, and robust support for asynchronous execution to hide latency. Both libraries are portable beyond HPC centers and support development on systems ranging from laptops to supercomputers. Together, these libraries enable agile, lightweight communication such as arises in irregular applications, libraries and frameworks running on exascale systems.

ACKNOWLEDGEMENTS

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.

REFERENCES

[1] P. Amestoy, I. Duff, J.-Y. L'Excellent, and J. Koster. 2001. A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling. *SIAM J. Matrix Anal. Appl.* 23 (2001), 15–41. https://doi.org/10.1137/S0895479899358194

[2] John Bachan, Scott B. Baden, Dan Bonachea, Paul H. Hargrove, Steven Hofmeyr, Mathias Jacquelin, Amir Kamil, and Brian van Straalen. 2018. *UPC++ Programmer's Guide, v1.0-2018.9.0.* Technical Report LBNL-2001180. Lawrence Berkeley National Laboratory. https://doi.org/10.25344/S49G6V

[3] John Bachan, Scott B. Baden, Dan Bonachea, Paul H. Hargrove, Steven Hofmeyr, Mathias Jacquelin, Amir Kamil, and Brian van Straalen. 2018. *UPC++ Specification, v1.0 Draft 8.* Technical Report LBNL-2001179. Lawrence Berkeley National Laboratory. https://doi.org/10.25344/S45P4X

[4] John Bachan, Dan Bonachea, Paul H. Hargrove, Steve Hofmeyr, Mathias Jacquelin, Amir Kamil, Brian van Straalen, and Scott B. Baden. 2017. The UPC++ PGAS Library for Exascale Computing. In *Proceedings of the Second Annual PGAS Applications Workshop (PAW17).* ACM, New York, NY, USA, Article 7, 4 pages. https://doi.org/10.1145/3144779.3169108

[5] Michael Bauer, Sean Treichler, Elliott Slaughter, and Alex Aiken. 2012. Legion: expressing locality and independence with logical regions. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '12).* https://doi.org/10.1109/SC.2012.71

[6] Dan Bonachea and Paul H. Hargrove. 2017. *GASNet Specification, v1.8.1.* Technical Report LBNL-2001064. Lawrence Berkeley National Laboratory. https://doi.org/10.2172/1398512

[7] Dan Bonachea and Paul H. Hargrove. 2018. *GASNet-EX: A High-Performance, Portable Communication Library for Exascale.* Technical Report LBNL-2001174. Lawrence Berkeley National Laboratory. https://doi.org/10.25344/S4QP4W To appear: Languages and Compilers for Parallel Computing (LCPC'18).

[8] David Callahan, Bradford L. Chamberlain, and Hans P. Zima. 2004. The Cascade High Productivity Language. *International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS)* (2004), 52–60. https://doi.org/10.1109/HIPS.2004.10002

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

[10] DOE Advanced Scientific Computing Research (ASCR). Facilities. https://science.energy.gov/ascr/facilities.

[11] Y. Dotsenko, C. Coarfa, and J. Mellor-Crummey. 2004. A Multi-platform Co-Array Fortran Compiler. In *Proc. 13th International Conference on Parallel Architecture and Compilation Techniques (PACT).* https://doi.org/10.1109/PACT.2004.1342539

[12] Deepak Eachempati, Hyoung Joon Jun, and Barbara Chapman. 2010. An Open-source Compiler and Runtime Implementation for Coarray Fortran. In *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Models ((PGAS'10)).* ACM, Article 13, 8 pages. https://doi.org/10.1145/2020373.2020386

[13] Exabiome. Exascale Solutions for Microbiome Analysis. https://sites.google.com/lbl.gov/exabiome.

[14] Alessandro Fanfarillo, Tobias Burnus, Valeria Cardellini, Salvatore Filippone, Dan Nagle, and Damian Rouson. 2014. OpenCoarrays: Open-source Transport Layers Supporting Coarray Fortran Compilers. In *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models (PGAS '14).* ACM, New York, NY, USA, Article 4, 11 pages. https://doi.org/10.1145/2676870.2676876

[15] GASNet. home page. http://gasnet.lbl.gov.

[16] Paul H. Hargrove and Dan Bonachea. 2018. *GASNet-EX Performance Improvements Due to Specialization for the Cray Aries Network.* Technical Report LBNL-2001134. Lawrence Berkeley National Laboratory. https://doi.org/10.2172/1430690

[17] Pascal Hénon, Pierre Ramet, and Jean Roman. 2002. PASTIX: a high-performance parallel direct solver for sparse symmetric positive definite systems. *Parallel Comput.* 28, 2 (2002), 301–321. https://doi.org/10.1016/S0167-8191(01)00141-7

[18] Intrepid Technology, Inc. Clang UPC Compiler. http://clangupc.github.io.

[19] Intrepid Technology, Inc. GCC/UPC Compiler. http://www.gccupc.org.

[20] Hitoshi Murai, Masahiro Nakao, Hidetoshi Iwashita, and Mitsuhisa Sato. 2017. Preliminary Performance Evaluation of Coarray-based Implementation of Fiber Miniapp Suite Using XcalableMP PGAS Language. In *Proceedings of the Second Annual PGAS Applications Workshop (PAW17).* ACM, Article 1, 7 pages. https://doi.org/10.1145/3144779.3144780

[21] National Energy Research Scientific Computing Center (NERSC). Cori Haswell Nodes. http://www.nersc.gov/users/computational-systems/cori/configuration/cori-phase-i/.

[22] National Energy Research Scientific Computing Center (NERSC). Cori Intel Xeon Phi (KNL) Nodes. http://www.nersc.gov/users/computational-systems/cori/configuration/cori-intel-xeon-phi-nodes/.

[23] National Energy Research Scientific Computing Center (NERSC). Edison. http://www.nersc.gov/users/computational-systems/edison.

[24] PaRSEC. Parallel Runtime Scheduling and Execution Controller. http://icl.cs.utk.edu/parsec.

[25] Swaroop Pophale, Ramachandra Nanjegowda, Tony Curtis, Barbara Chapman, Haoqiang Jin, Stephen Poole, and Jeffery Kuehn. 2012. OpenSHMEM Performance and Potential: A NPB Experimental Study. In *Proceedings of the 6th Conference on Partitioned Global Address Space Programming Models (PGAS'12).* https://www.osti.gov/biblio/1055092

[26] UPC++. home page. http://upcxx.lbl.gov.

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

[28] Y. Zheng, A. Kamil, M. B. Driscoll, H. Shan, and K. Yelick. 2014. UPC++: A PGAS Extension for C++. In *IEEE 28th International Parallel and Distributed Processing Symposium.* 1105–1114. https://doi.org/10.1109/IPDPS.2014.115