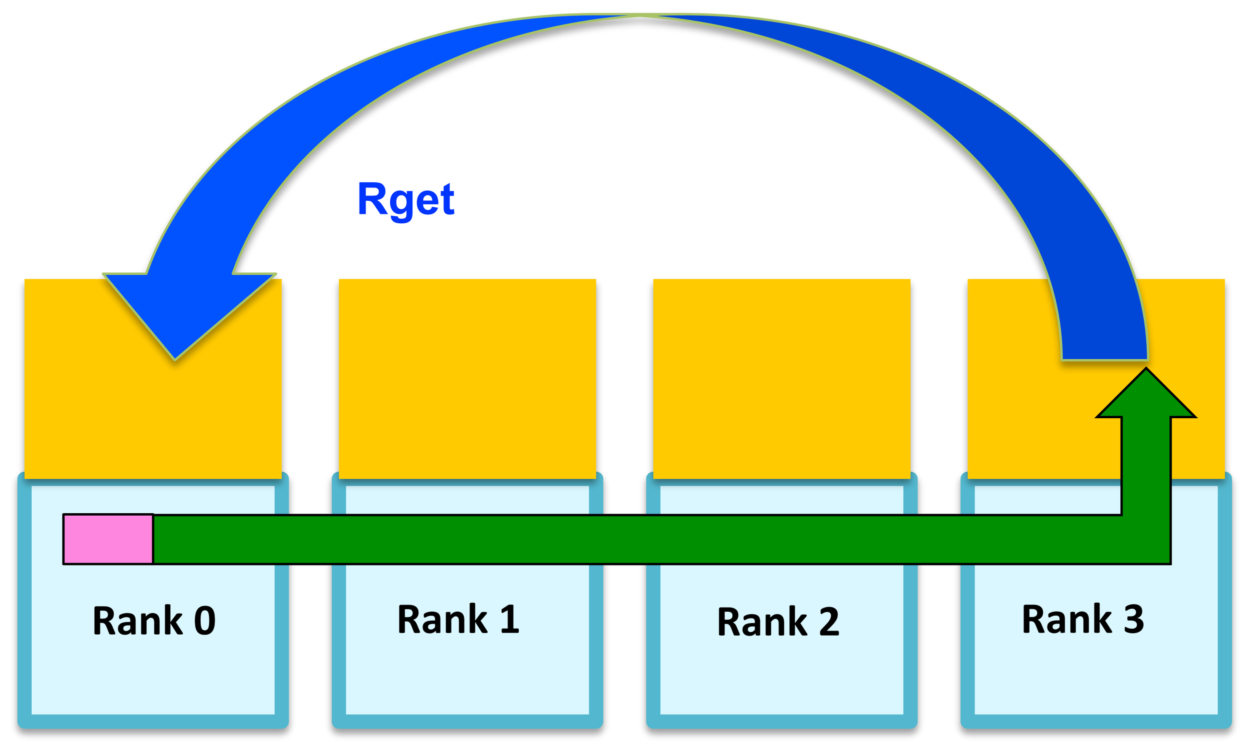
# Pagoda: Lightweight Communications and Global Address Space Support for Exascale Applications - UPC++

Scott B. Baden (PI), Paul H. Hargrove (co-PI),
Hadia Ahmed, John Bachan, Dan Bonachea, Steve Hofmeyr, Mathias Jacquelin, Amir Kamil, Brian van Straalen

## UPC++ at Lawrence Berkeley National Lab (http://upcxx.lbl.gov)




Private address spaces

- UPC++ is a C++11 PGAS library
  - Lightweight, asynchronous, one-sided communication (RMA)
  - Asynchronous remote procedure call (RPC)
  - Data transfers may be non-contiguous
  - Futures manage asynchrony, enable communication overlap
  - Collectives, teams, remote atomic updates
  - Provides building blocks to construct irregular data structures
- Easy on-ramp and integration
  - Enables incremental development
  - Selectively replace performance-critical sections with UPC++
  - Interoperable with MPI, OpenMP, CUDA, etc.
- Latest software release: September 2018
  - Runs on systems from laptops to supercomputers

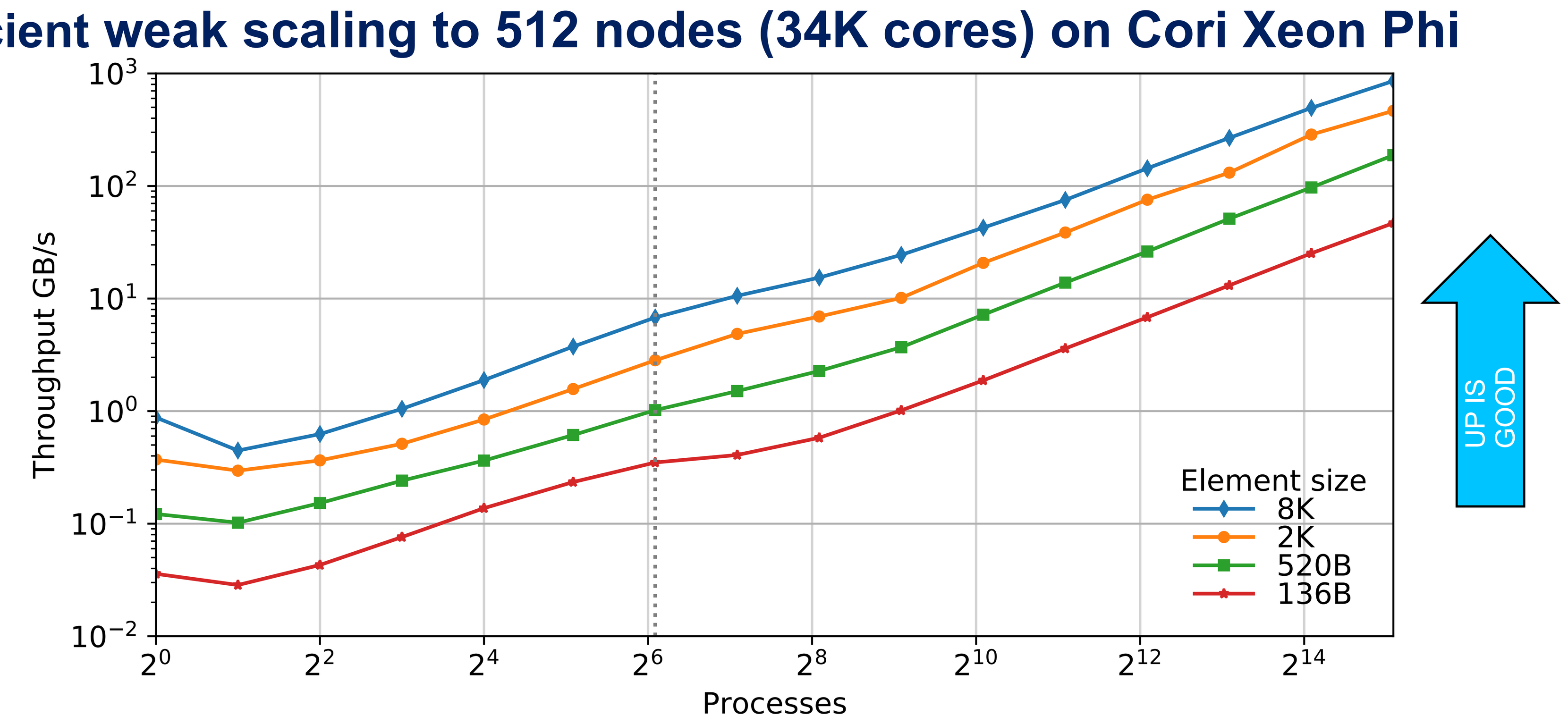
## Case 1: Easy Distributed Hash-Table via Function Shipping and Futures

- **Distributed hash-table design is based on function shipping**
  - RPC inserts the key metadata at the target
  - Once the RPC completes, an attached callback issues a one-sided RMA Put (rput) to store the value data

```
// C++ global variables correspond to rank-local state
std::unordered_map<uint64_t, global_ptr<char>> local_map;
// insert a key-value pair and return a future
future<> dht_insert(uint64_t key, char *val, size_t sz) {
 future<global_ptr<char>> fut =
     rpc(key % rank_n(),             // RPC obtains location for the data
            [key,sz]() -> global_ptr<char> {     // lambda invoked by RPC
              global_ptr<char> gptr = new_array<char>(sz);
              local_map[key] = gptr;               // insert in local map
              return gptr;
            });
 return fut.then(              // callback executes when RPC completes
       [val,sz](global_ptr<char> loc) -> future<> {
           return rput(val, loc, sz); });    // RMA Put the value payload
}
```

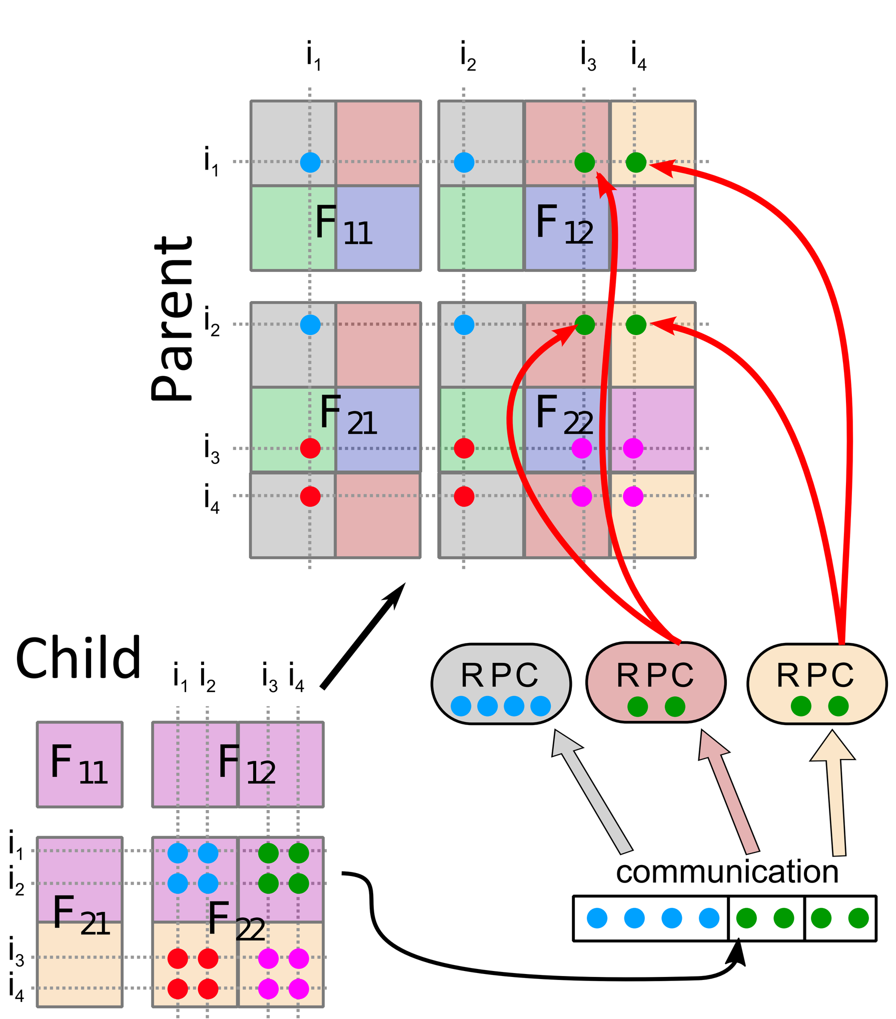
- **Benefits:**
  - Use of **RPC** simplifies distributed data-structure design
    - Argument passing, remote queue management and progress engine are factored out of the application code
  - Asynchronous execution enables overlap

**Efficient weak scaling to 512 nodes (34K cores) on Cori Xeon Phi**




## Case 2: Asynchronous Sparse Matrix Solvers

- **A time consuming operation in multifrontal sparse solvers:**
  - ***Extend-add:*** update a distributed sparse matrix, scattering the packed data source
- **Challenge:**
  - This operation has low computational intensity and exhibits irregular communication patterns
- **Solution**:
  - UPC++ **function shipping** via RPC enables efficient communication and asynchrony, increasing overlap and improving performance of ***Extend-add***

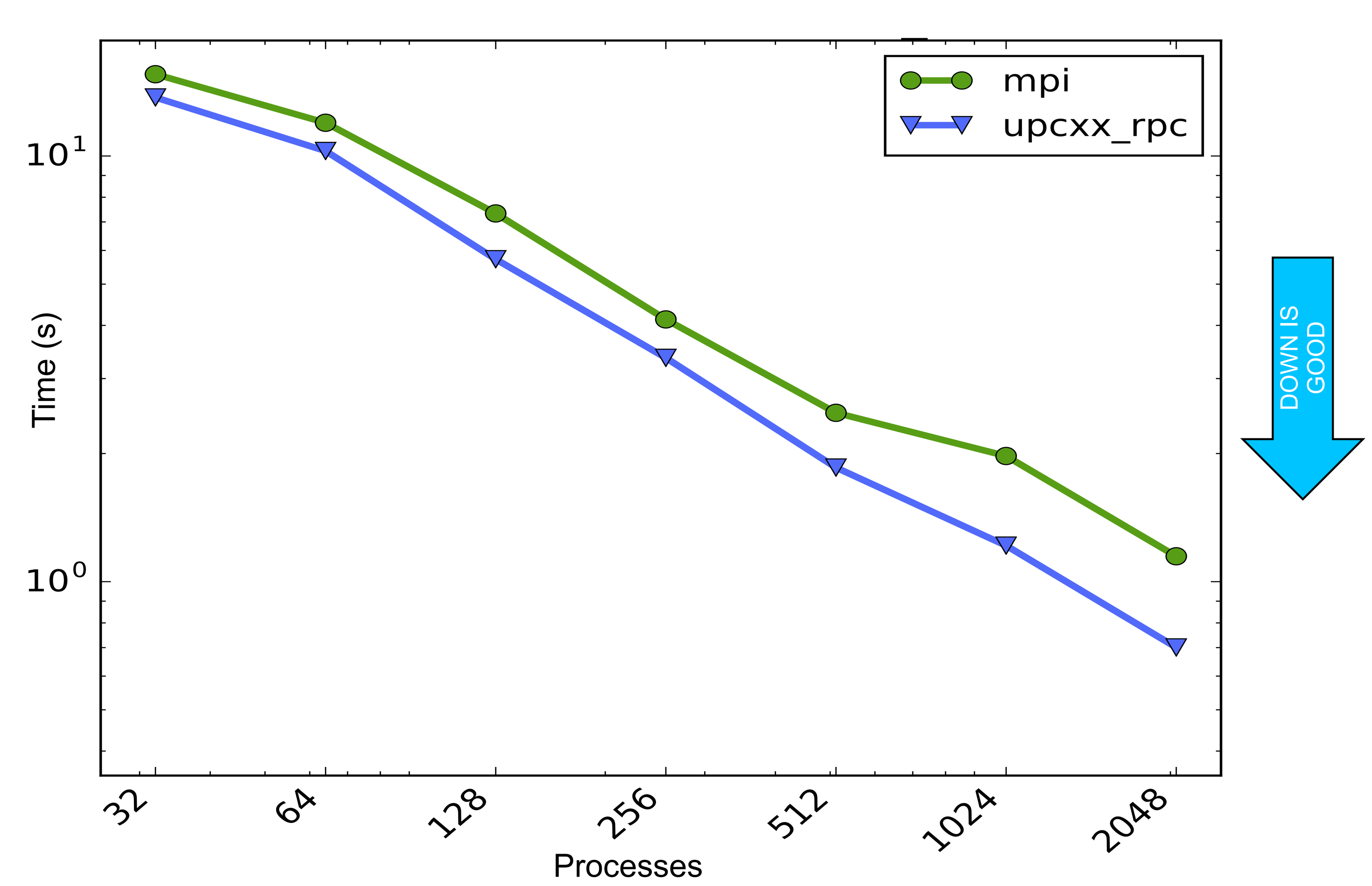



**Overview of Extend-add**

- Updates are shown for the left child only.
- Colored squares depict the distribution of parent and child matrices.
- Dots in the lower left child matrix depict the data to be sent and accumulated in the parent.
- The communication step initiated by one process in the left child is depicted in the lower right corner.
- RPCs communicate the data to the parent, which carries out the accumulation. Data linearization is handled by UPC++ views.

- **Impact:**
  - UPC++ enhances overlap in ***Extend-add***, yielding up to a 1.63x speedup over MPI collective and 3.11x over MPI message-passing implementations. The green line in the figure below corresponds to the fastest of these two variants.

**Strong scaling comparison of the UPC++ implementation of Extend-add using RPC and an MPI variant for the audikw_1 matrix on NERSC Cori Xeon Phi (using 64 cores/node)**




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.