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 in 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
  local_map[key] = fut;
  return fut.then(
    [val, sz](const global_ptr<char> &loc) -> future<>{
      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

Challenges:
- 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**

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 corresponds to the fastest of these two variants.

* For more details see IPDPS’19. https://doi.org/10.25344/S4V88H