Distributed Object Abstraction in HPX

Weile Wei, Maxwell Reeser, Hartmut Kaiser, Adrian Serio, Avah Banerjee, R. Tohid

Louisiana State University

LSU Center for Computation & Technology

STE||AR GROUP

Abstract

In distributed computing programming, lots of large-scale data object will be involved (i.e. matrix multiplication, transposition, etc), which cannot be stored in memory on a single node. A single logical object is often needed to represent and control this large data object across a set of nodes or localities. We implement a C++ based distributed object abstraction using HPX, a C++ Standard Library for Concurrency and Parallelism. This poster will introduce an easy-to-use C++ distributed container with simple use-case, its user-friendly API of handling data transfer between localities, and its background of distributed computing and HPX.

Background

HPX (High Performance ParalleX) is a general purpose C++ runtime system for parallel and distributed applications of any scale. The following infrastructure in HPX provides support for the distributed object:

- Active Global Address Space: AGAS exposes a single uniform address space spanning all localities an application runs on.
- Component: A component is a C++ object which can be accessed remotely.
- Action: An action is a function that can be invoked remotely.

Registration Methods

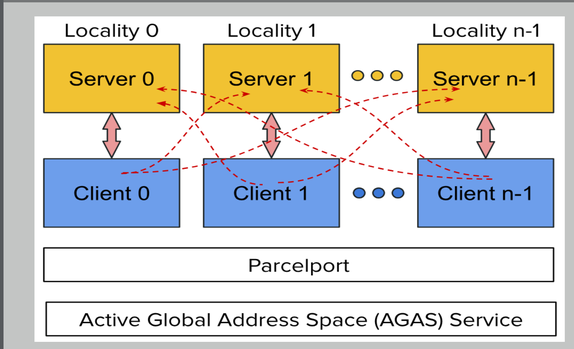

Figure 1:all to all method.

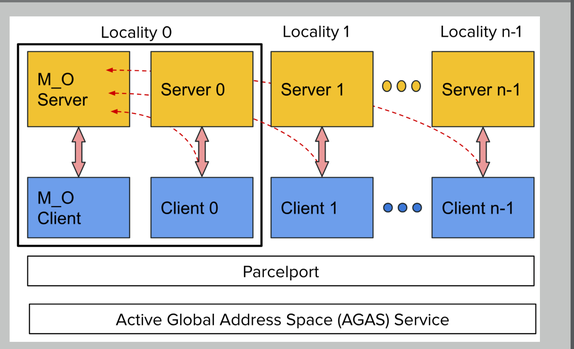

Figure 2:meta object method

- Look-ups happen on an as-needed basis
- Worst case N^2 lookups
- Currently the template's default registration method

- In every case, N lookups must be done, for each locality to find the Meta object
- Slower on startup than All-to-All but a much lower upper-bound on messages sent
- Specified as template parameter

Conclusion

- Provides an easy-to-use distributed container and offers a user-friendly API that hides communication details for user which allows easy transfer of data between localities
- Delivers high reusable code and ensures code portability
- Improves user's programming productivity in high performance computing

Examples

```
template <typename T, construction_type C = all_to_all>
class distributed_object{
Public:

    distributed_object(std::string base, T const& data,
                       std::vector<size_t> sub_localities = all_localities())

    distributed_object(std::string base, T&& data,
                       std::vector<size_t> sub_localities = all_localities())

    T const& operator*() const;
    T& operator*();
    T const* operator->() const;
    T* operator->();
}
```

Figure 3:Distributed Object API

```
void add(distributed_object<int>& local, int& remote) {
       (*local) += remote;
}
//main function
distributed_object<int> dist_int("unique_name", cur_locality);
if (cur_locality == 0)
{
    std::vector<future<void>> results;
    auto range = irange(1, num_localities);
    for_each(seq, begin(range), end(range),
    [&](std::size_t remote_loc)
    {
        future<int> remote_val = dist_int.fetch(remote_loc);
        results.push_back(hpx::dataflow(unwrapped(add), dist_int, f1));
    });
    wait_all(results);
}
```

fetch() is an asynchronous function which returns a future of a copy of the instance of this distributed_object associated with the given locality index.

Figure 4:fetch function

```
// assume if we have more than 3 localities
if (cur_locality == 0 or cur_locality == 1)
{
    std::vector<size_t> participants{0, 1};

    distributed_object<int> dist_int("dist_int"
                                     , cur_locality
                                     , participants);
    // do some computation
}
else {
    // do something else
}
```

The constructor is able to accepts a subset of localities such that workloads are split into constituent parts so relevant distributed_object is only used particular sub localities

Figure 5:sub localities

Acknowledgments

- This material is based upon work supported by the National Science Foundation under Grant No. 1737785. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work is supported by The Defense Technical Information Center under the contract: DTIC Contract FA8075-14-D-0002/0007.

wwei9@lsu.edu