HCL: Distributing Parallel Data Structures in Extreme Scales

Cluster’20: IEEE Cluster 2020

Hariharan Devarajan, Anthony Kougkas, and Xian-He Sun
hdevarajan@hawk.iit.edu
HCL: Hermes Container Library

- HCL is a C++ template library that offers
  - STL-like distributed data structures.
  - Remote Callbacks on data structure operations.
  - A PGAS (partition global address space) programming system
    - No custom (pre-)compiler
    - Extends data structures with persistent Memory or SSD.

- PGAS Terminology – ShMem Analogy

<table>
<thead>
<tr>
<th>Shared</th>
<th>Key Value</th>
<th>Key Value</th>
<th>hcl::unordered_map&lt;int, int&gt; a()</th>
<th>Key Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private</td>
<td>int a;</td>
<td></td>
<td></td>
<td>int b; int c;</td>
</tr>
<tr>
<td></td>
<td>Part 0</td>
<td></td>
<td></td>
<td>Part N-1</td>
</tr>
</tbody>
</table>

Managed by HCL in virtual global address.

Managed by regular C++ mechanisms

Partition can be owned by process / thread
**HCL: Hermes Container Library**

- **Data Distribution**
  - Data has a well-defined owner but can be accessed by any unit.
  - Access model is based on locality of data (inter-node or intra-node)
  - Follows *owner computes execution* model.

- **HCL:**
  - Unified access to node-local and remote data in global address space.
  - Enable hybrid access model for maximizing performance.
HCL: Core design elements

- RPC over RDMA
- High performance commutation protocol
- Hybrid Access Model: Optimize inter-node and intra-node access
- Persistent storage: Extend PGAS to NVRAM and NVMe SSD.
- Event-driven design: Asynchronous remote executions
RPC over RDMA

- **RDMA Infrastructure**
  - Utilize RDMA work-queue for event-based RPC protocol on NIC core.
  - Use RDMA one-sided to send instructions and
  - Use RDMA one-sided to pull data from server.

- **Additional Resources**
  - Memory Buffers for request and response.
  - Generic client and server stub to encode and decode m
RPC over RDMA Overhead Analysis

Key Observations

- NIC core in BCL has higher utilization due to remote CAS operation. HCL performs local CAS which is faster.
- Memory allocation in HCL is dynamic.
- Network utilization is better because RPC protocol packages multiple instructions as opposed to executing multiple remote instructions.
HCL offers distributed data structures
- Custom distribution schemes (std::hash)
- Custom data ordering schemes (std::less)
- Example: hcl::set<T>

```cpp
1 std::vector sort(const std::vector& data) {
2   auto set = hcl::set<int>();
3   for (auto& val : data) {
4     set.insert(val);
5   }
6   return set.local();
7 }
```

Initializes HCL set and create empty partition all over the address space based on std::hash function defined.

Each process inserts its set of values into the set. The data is distributed based on std::hash and ordered based on std::less defined.

Each Process gets its local sorted set. This is globally sorted between process already.

```
$ mpirun -n 4 ./sort_set 20
0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
```
Accessing Local Data

Use `.size` as a shot hand for `.local.size` and returns the number of local elements.

```
auto array = hcl::array<int>(20);
for (int i=0; i<array.size(); ++i) {
  array.local[i]=hcl::my_rank;
}
MPI_Barrier(MPI_COMM_WORLD);
if (hcl::my_rank == 0) {
  for (auto a: array) {
    cout << (int) a << " ";
  }
  cout << endl;
}
```

$ mpirun -n 4 ./local_array 20
0 0 0 0 1 1 1 1 2 2
2 2 3 3 3 3 4 4 4 4
Hybrid Access Model Performance

- **Key Observation**
  - Intra-node access is **order of magnitude faster** than inter-node
    - Due to lower network than in-memory performance.
  - As **size of data increases**, the bandwidth achieved is closer to theoretical peak of hardware.
## HCL: Data Structure Overview

<table>
<thead>
<tr>
<th>Container</th>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hcl::unordered_map</td>
<td>bool insert(const K &amp;key, const V &amp;val)</td>
<td>Insert item into hash table</td>
</tr>
<tr>
<td></td>
<td>bool find(const K &amp;key, V &amp;val)</td>
<td>Find item in table, return val.</td>
</tr>
<tr>
<td>hcl::map</td>
<td>bool insert(const K &amp;key, const V &amp;val)</td>
<td>Insert item into ordered map</td>
</tr>
<tr>
<td></td>
<td>bool find(const K &amp;key, V &amp;val)</td>
<td>Find item in map, return val.</td>
</tr>
<tr>
<td>hcl::unordered_set</td>
<td>bool insert(const K &amp;key)</td>
<td>Insert item into hash set</td>
</tr>
<tr>
<td></td>
<td>bool find(const K &amp;key)</td>
<td>Find item in set, return if exists.</td>
</tr>
<tr>
<td>hcl::set</td>
<td>bool insert(const K &amp;key)</td>
<td>Insert item into ordered set</td>
</tr>
<tr>
<td></td>
<td>bool find(const K &amp;key)</td>
<td>Find item in set, return if exists.</td>
</tr>
<tr>
<td>hcl::queue</td>
<td>bool push(const T &amp;val)</td>
<td>Push element into queue</td>
</tr>
<tr>
<td></td>
<td>bool pop(const T &amp;val)</td>
<td>Pop element from queue</td>
</tr>
</tbody>
</table>
Key Observations

- HCL data structures achieve order of magnitude higher performance than BCL.
- The data ordering reduces throughput by 20-40%.
HCL with Application Workloads

Key Observations

- Enables higher performance for all real workloads.
- Over 10x performance improvement.
Conclusions

- We propose a new RPC over RDMA protocol which can be a high-performance communication fabric for distributed data structures.
- We implemented several STL-like data structures to enable efficient programmability.
- We showcase that HCL can accelerate applications with its data structure by 2x to 12x.
Hariharan Devarajan, hdevarajan@hawk.iit.edu

Q & A