SCALABLE COMPUTING
SOFTWARE LABORATORY

ILLINOIS INSTITUTE
OF TECHNOLOGY

HCL

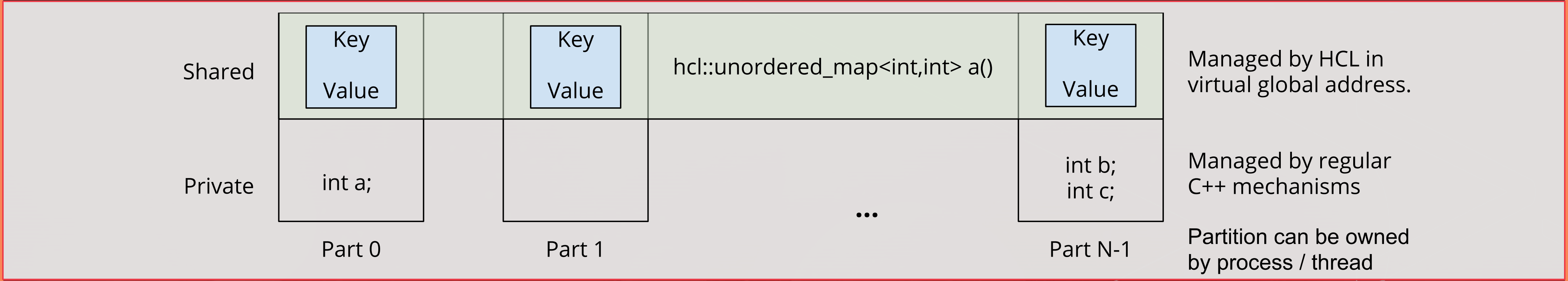
HCL: Distributing Parallel Data Structures in Extreme Scales

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 (paritition global address space) programming system
    - No custom (pre-)compiler
    - Extends data structures with persistent Memory or SSD.
- PGAS Terminology – ShMem Analogy



Shared

Key
Value

Key
Value

hcl::unordered_map<int,int> a()

Key
Value

Managed by HCL in virtual global address.

Private

int a;

int b;
int c;

...

Part 0

Part 1

Part N-1

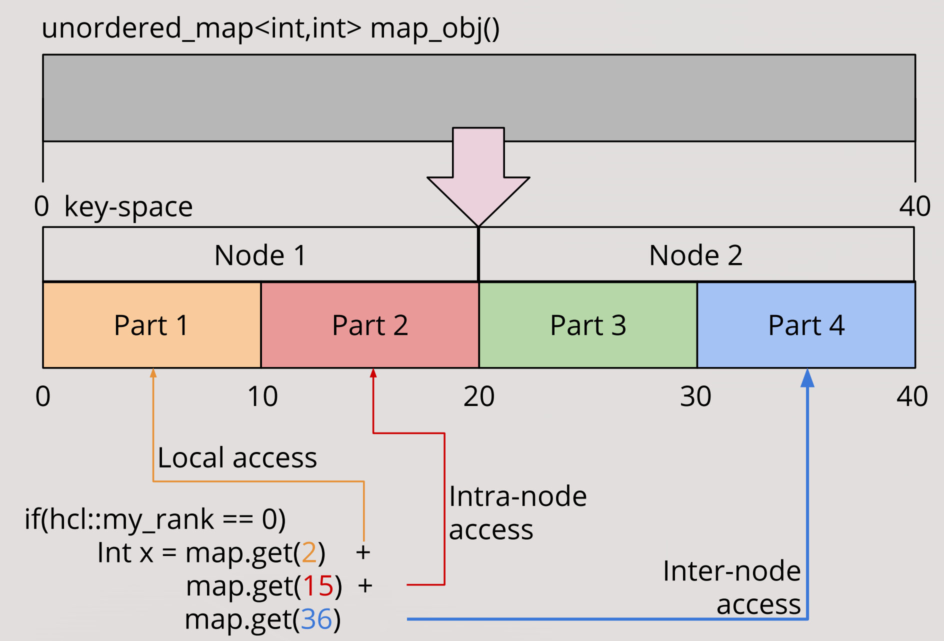
Managed by regular C++ mechanisms

Partition can be owned by process / thread

Hariharan Devarajan, hdevarajan@hawk.iit.edu

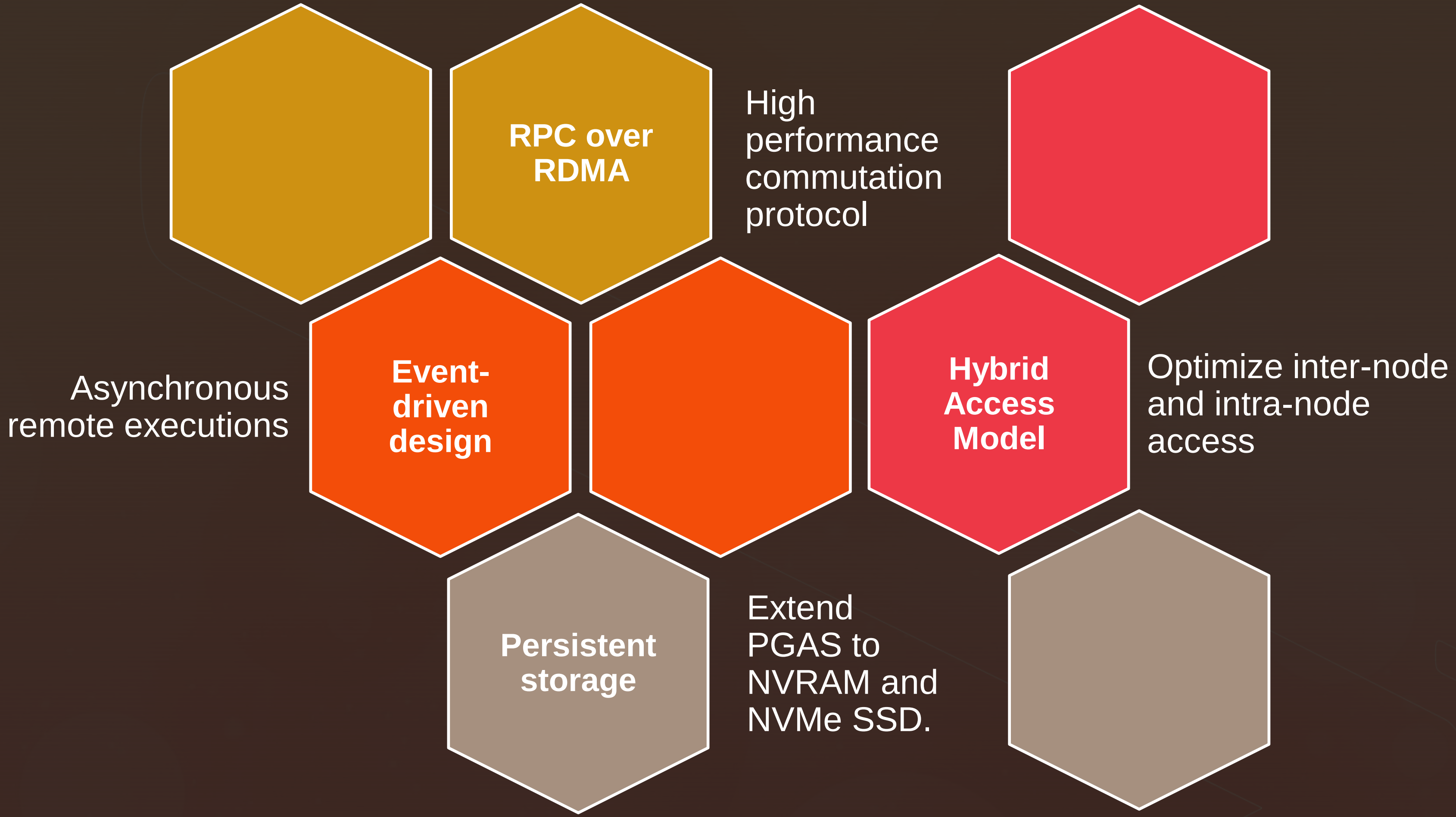
HCL: Hermes Container Library

- 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.



```
if(hcl::my_rank == 0)
    Int x = map.get(2)   +
            map.get(15)  +
            map.get(36)
```

HCL: Core design elements



RPC over RDMA

High performance commutation protocol

Event-driven design

Asynchronous remote executions

Hybrid Access Model

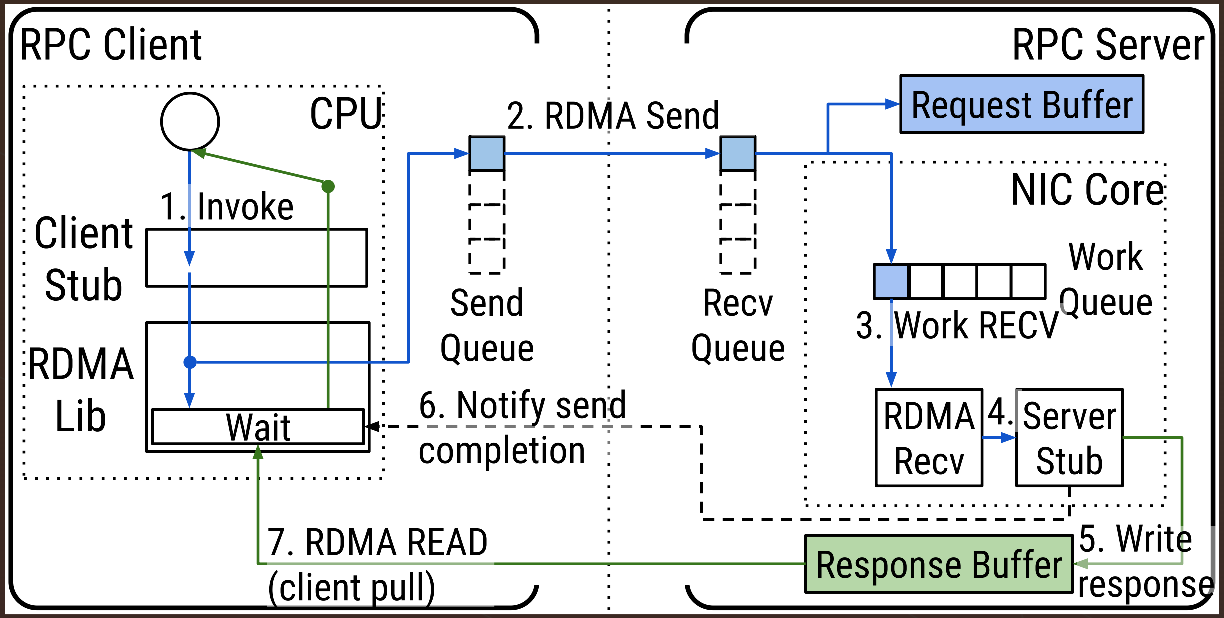
Optimize inter-node and intra-node access

Persistent storage

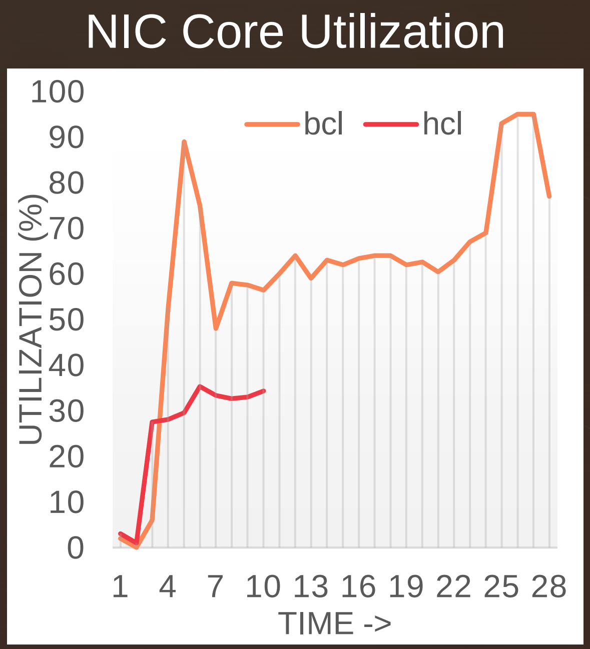
Extend PGAS to NVRAM and NVMe SSD.

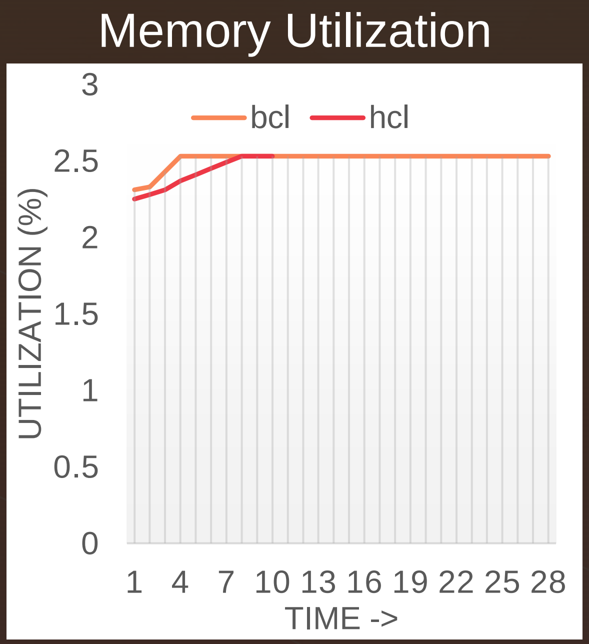
RPC over RDMA

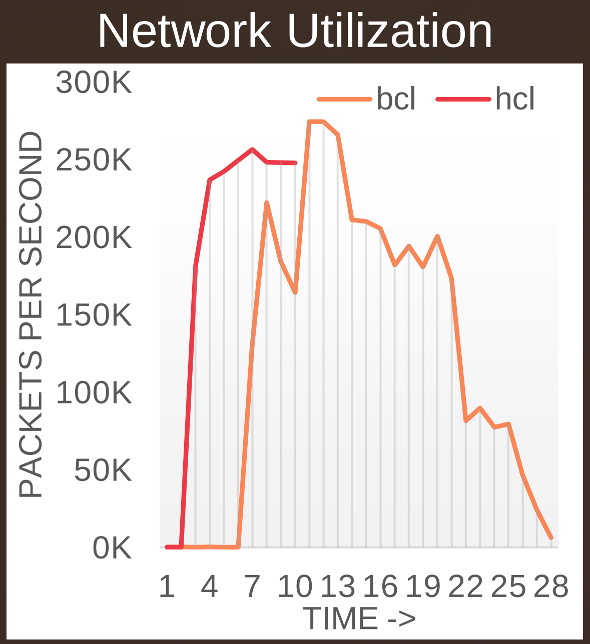
- RDMA Infrastructure
  - Utilize RDMA work-queue for event-based RPC protocol on NIC core.
  - Use RDMA one-sided to send instructions and
  - Use RDMA one-sided to pull data from server.
- Additional Resources
  - Memory Buffers for request and response.
  - Generic client and server stub to encode 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.

Distributed Data Structures

- HCL offers distributed data structures
  - Custom distribution schemes (std::hash)
  - Custom data ordering schemes (std::less)
  - Example: hcl::set<T>

```
std::vector sort(const std::vector& data) {
  auto set = hcl::set<int>();
  for (auto& val : data) {
      set.insert(val);
  }
  return set.local()
}
```

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

.local is a proxy object for accessing local partition of the HCL data structure.

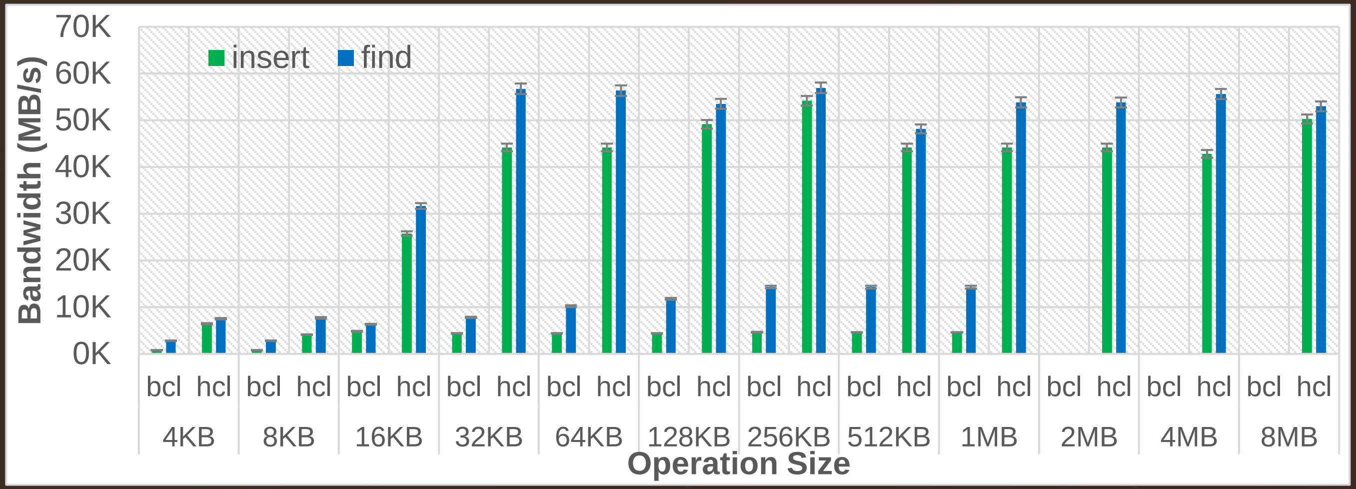
```
auto array = hcl::array<int>(20);
for (int i=0; i<array.lsize(); ++i) {
    array.local[i]=hcl::my_rank;
}
MPI_Barrier(MPI_COMM_WORLD);
if(hcl::my_rank == 0){
    for(auto a:array)
        cout << (int) a <<"  ";
    cout << endl;
}
```

```
$ 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
```

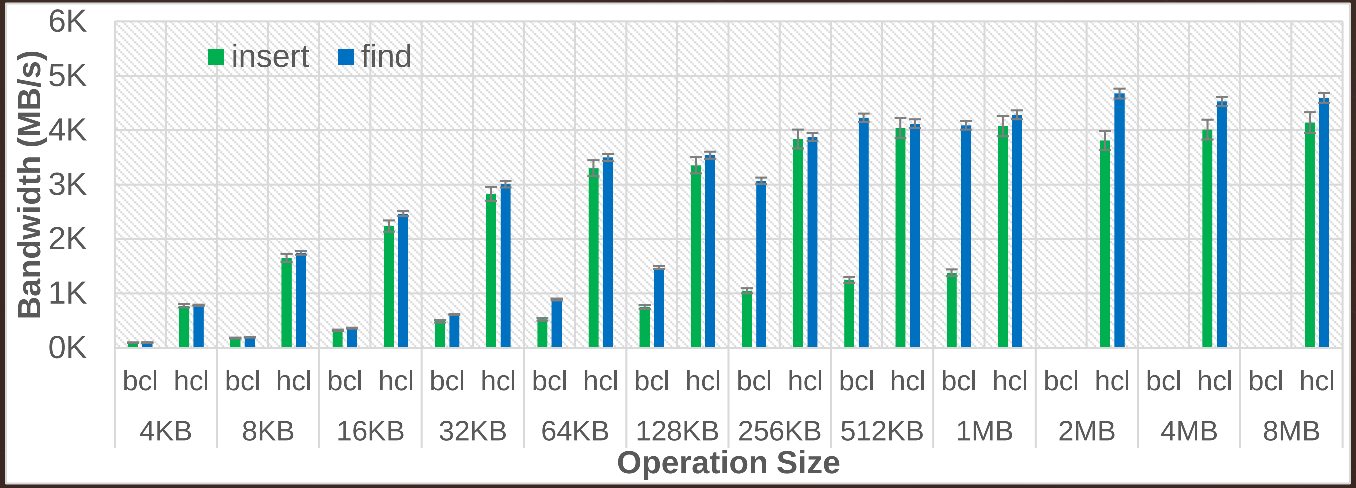
Hariharan Devarajan, hdevarajan@hawk.iit.edu

Hybrid Access Model Performance

Intra-node



Inter-node



- 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.

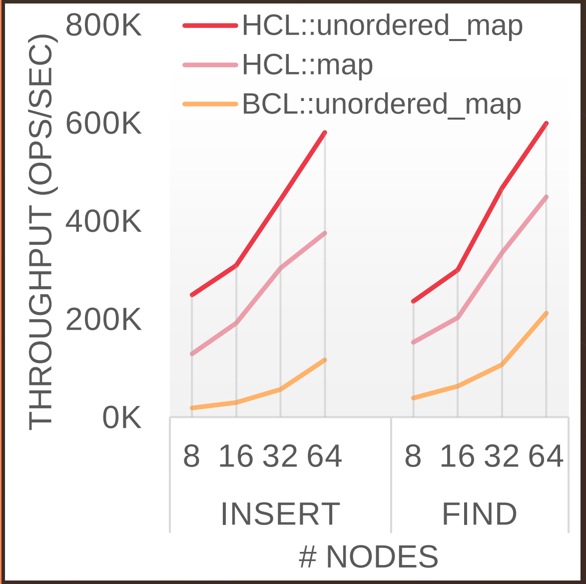
HCL: Data Structure Overview

Container	API	Description
hcl::unordered_map	bool insert(const K &key, const V &val)	Insert item into hash table
	bool find(const K &key, V &val)	Find item in table, return val.
hcl::map	bool insert(const K &key, const V &val)	Insert item into ordered map
	bool find(const K &key, V &val)	Find item in map, return val.
hcl::unordered_set	bool insert(const K &key)	Insert item into hash set
	bool find(const K &key)	Find item in set, return if exists.
hcl::set	bool insert(const K &key)	Insert item into ordered set
	bool find(const K &key)	Find item in set, return if exists.
hcl::queue	bool push(const T &val)	Push element into queue
	bool pop(const T &val)	Pop element from queue

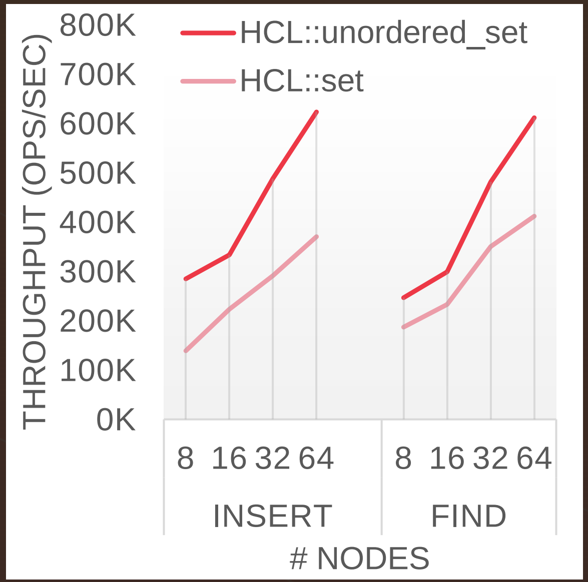
Hariharan Devarajan, hdevarajan@hawk.iit.edu

HCL data structure performance

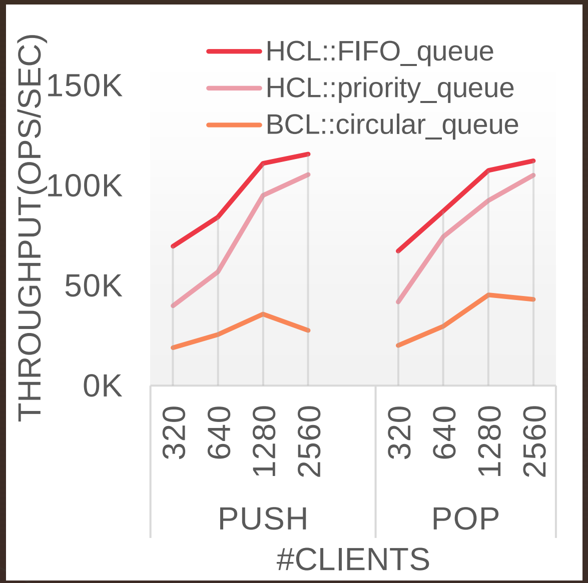
unordered_map/map



unordered_set/set



queue/priority_queue



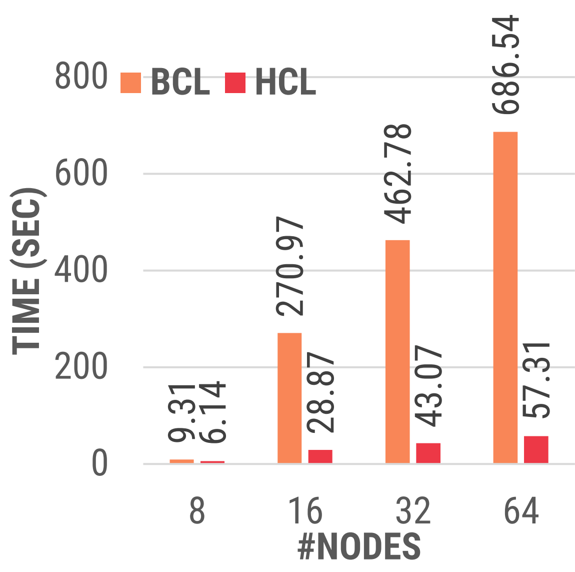
Key Observations

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

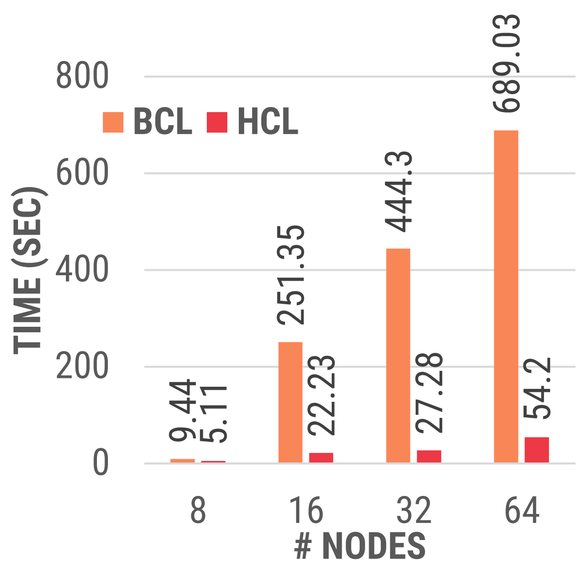
Hariharan Devarajan, hdevarajan@hawk.iit.edu

HCL with Application Workloads

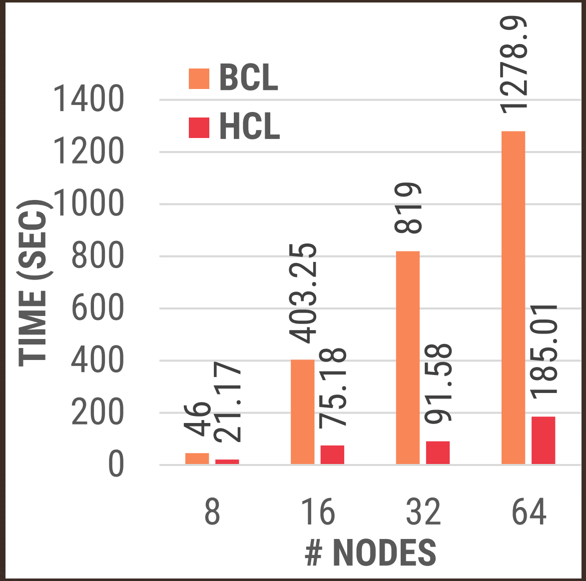
ISx Benchmark



Genome Contig Generation



Genome k-mer counting



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

Hariharan Devarajan, hdevarajan@hawk.iit.edu

Q & A