
# HabaneroUPC++: A Compiler-free PGAS Library

**Vivek Kumar**[1], Yili Zheng[2], Vincent Cavé[1], Zoran Budimlić[1], Vivek Sarkar[1]

1 Rice University

2 Lawrence Berkeley National Laboratory

# Outline

- Background
- Motivation and Insights
- HabaneroUPC++ Programming Model
- Implementation
- Results
- Summary



RICE BERKELEY LAB

# Outline

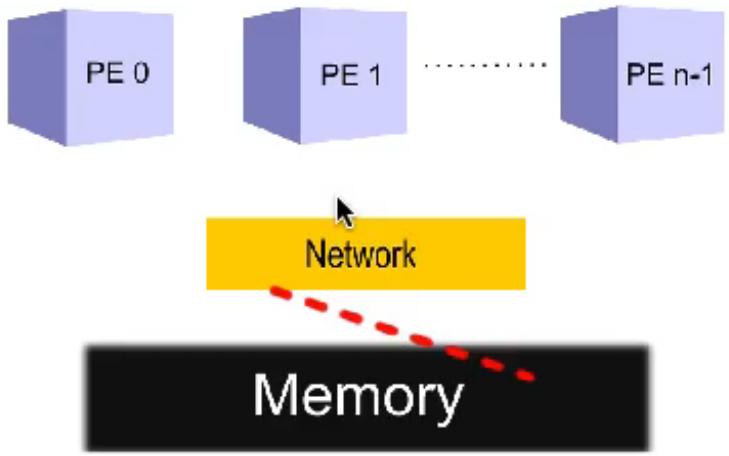
- Background
- Motivation and Insights
- HabaneroUPC++ Programming Model
- Implementation
- Results
- Summary



RICE BERKELEY LAB

# Parallel Programming Models

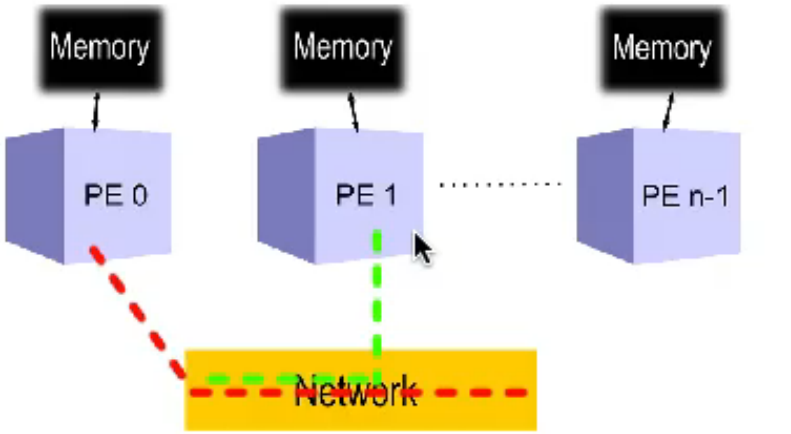
- Shared memory model
  - Cilk, Habanero-C, OpenMP and TBB




*Source: http://cnx.org/contents/82d83503-3748-4a69-8d6c-50d34a40c2e7@7*

# Parallel Programming Models

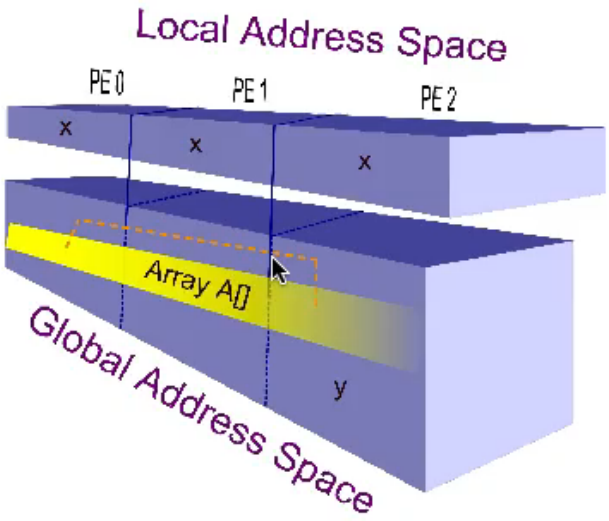
- Distributed memory model
  - Processes communicating using messages (MPI)




*Source: http://cnx.org/contents/82d83503-3748-4a69-8d6c-50d34a40c2e7@7*

# Parallel Programming Models

- Partitioned global address space (PGAS)
  - Strikes balance between shared and distributed memory models (CAF, Titanium and UPC)




*Source: http://cnx.org/contents/82d83503-3748-4a69-8d6c-50d34a40c2e7@7*

# Parallel Programming Models

- Asynchronous PGAS (APGAS)
  - PGAS + dynamic tasking (Chapel and X10)
  - Places with thread pools using work-stealing

# Limitations to Language Based PGAS / APGAS Approaches

- Compiler maintenance and development costs
- Special debugging support
- Not easy to add / introduce new features
- Lacks the features available in mainstream programming languages (C++)
  - Lambda functions in C++11
  - Type inference in C++11
- Learning curve for new programmers (Chapel / X10)

# Outline

- Background
- Motivation and Insights
- HabaneroUPC++ Programming Model
- Implementation
- Results
- Summary



RICE BERKELEY LAB

# Motivation

- A compiler-free approach for APGAS programming
- Use a modern mainstream programming language (C++)
- Rely on the compilers from major compiler construction organizations (GNU, Clang, etc.)



RICE
BERKELEY LAB

# Related Work

- Chapel / X10
  - New language and compiler supported (async-finish)
  - Dynamic load balancing capabilities
- UPC++ (IPDPS 2014)
  - Compiler free approach for writing UPC programs
  - C++ templates (C++11 optional)
  - Does not provide dynamic load balancing
- UPC work-stealing task library (PGAS 2011)
  - Only **inter**-*place* dynamic local balancing
  - Does not support async-finish style programming



RICE BERKELEY LAB

# Insights

- HabaneroUPC++ library
  - Inter-mixing Habanero programming model with UPC++
- Use C++11 features to map user code to runtime
  - Lambda functions
  - Type inference

# Outline

- Background
- Motivation and Insights
- HabaneroUPC++ Programming Model
- Implementation
- Results
- Summary




# C++11 Lambda Functions

```
// create lambda
auto func = [   ] (argument_list) ➔ return_type {
                    Statements;
                };

// execute lambda
func (argument_list);
```



RICE BERKELEY LAB

# HabaneroUPC++ Programming Model

- Program execution starts in SPMD mode
- Places can launch local and remote asynchronous tasks
  - Local tasks => Habanero model
  - Remote tasks => UPC++
- Each place has a fixed pool of workers
  - Work-stealing load-balancing (*intra*-place only)

# **Intra-*place*** Dynamic Tasking in HabaneroUPC++

- Habanero-C async-finish syntax

```
finish {
  async [IN (var1, var2, ...)]  [OUT (var1, var2, ...)]  [INOUT (var1, var2, ...)]
            Statements;
}
```

- HabaneroUPC++ async-finish syntax

```
finish ( [capture_list1] ( ) {
  async ( [capture_list2] ( ) {
            Statements;
  });
});
```



RICE

BERKELEY LAB

# UPC++ Support for finish-async

*IPDPS 2014*

- Remote function invocation

```
// does not wait for nested async
finish {
    // cannot capture closure
    async ( dest_place ) [ ] (argument_list) {
        Statements;
    });
}
```

- Remote copy

```
async_copy ( src, dest, count );
async_copy_fence ( );          // wait for all previous async_copy
```



RICE BERKELEY LAB

# Inter-*place* Dynamic Tasking in HabaneroUPC++

```
finish_spmd ( [capture_list1] () {

        // Any Habanero dynamic tasking constructs

        // Remote function invocation
        asyncAt ( destPlace, [capture_list2] ( ) {
                Statements;
        });

        // Remote copy
        asyncCopy ( src, dest, count, ddf=NULL );
        asyncAwait(ddf, ….); // local
});
```



RICE

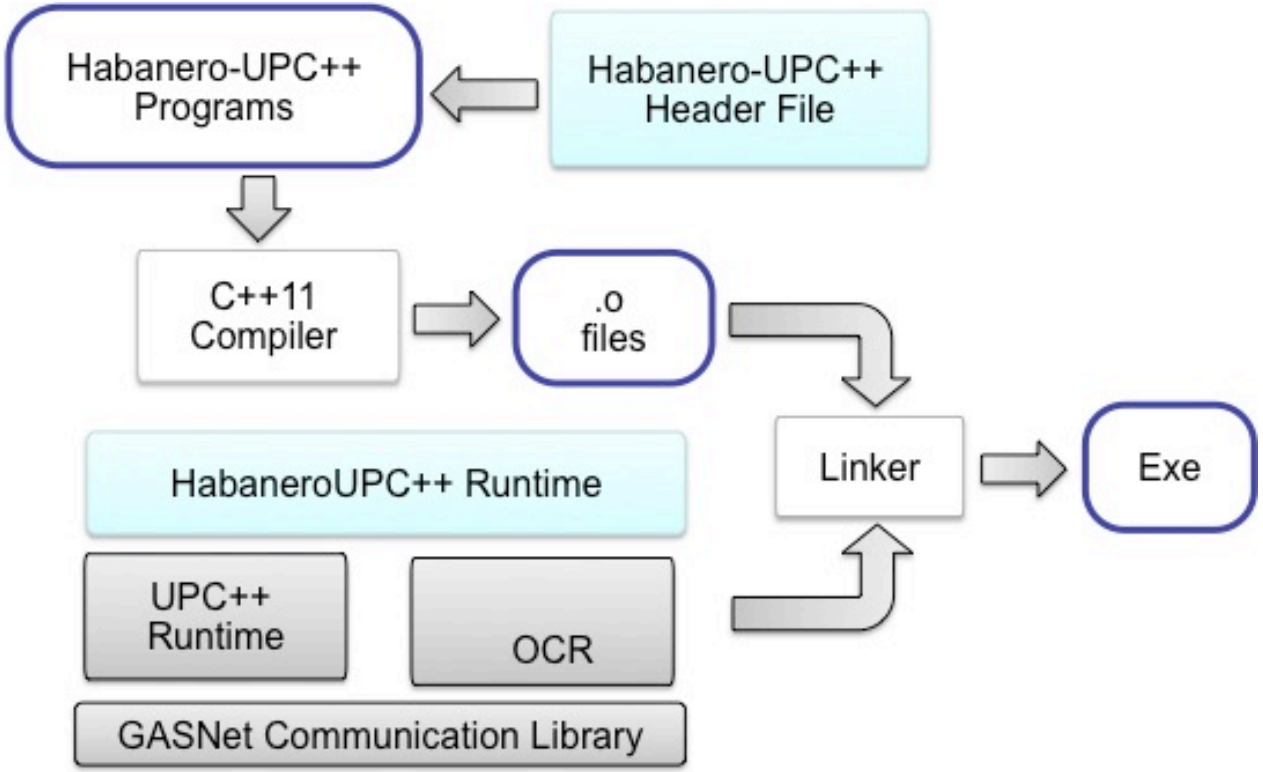
BERKELEY LAB

# Outline

- Background
- Motivation and Insights
- HabaneroUPC++ Programming Model
- Implementation
- Results
- Summary




# Habanero-UPC++ Software Stack



# HabaneroUPC++ Runtime

- Mapping lambda function to runtime
- Integrating OCR with UPC++
- Termination detection




# Mapping Lambda Functions to Runtime

```
void async ( std::function <void()> lambda ) {

    pass_to_OCR ( local_async, heap_allocated_lambda );




}
```



RICE BERKELEY LAB

# Mapping Lambda Functions to Runtime

```
void async ( std::function <void()> lambda ) {




}
```



RICE BERKELEY LAB

# Mapping Lambda Functions to Runtime

```
template < typename T >
void asyncAt ( int dest_place, T lambda ) {




}
```




# Mapping Lambda Functions to Runtime

```
template < typename T >
void asyncAt ( int dest_place, T lambda ) {
    if ( dest_place == my_place ) {
      pass_to_OCR ( local_async, heap_allocated_lambda );
    }
    else {



    }
}
```

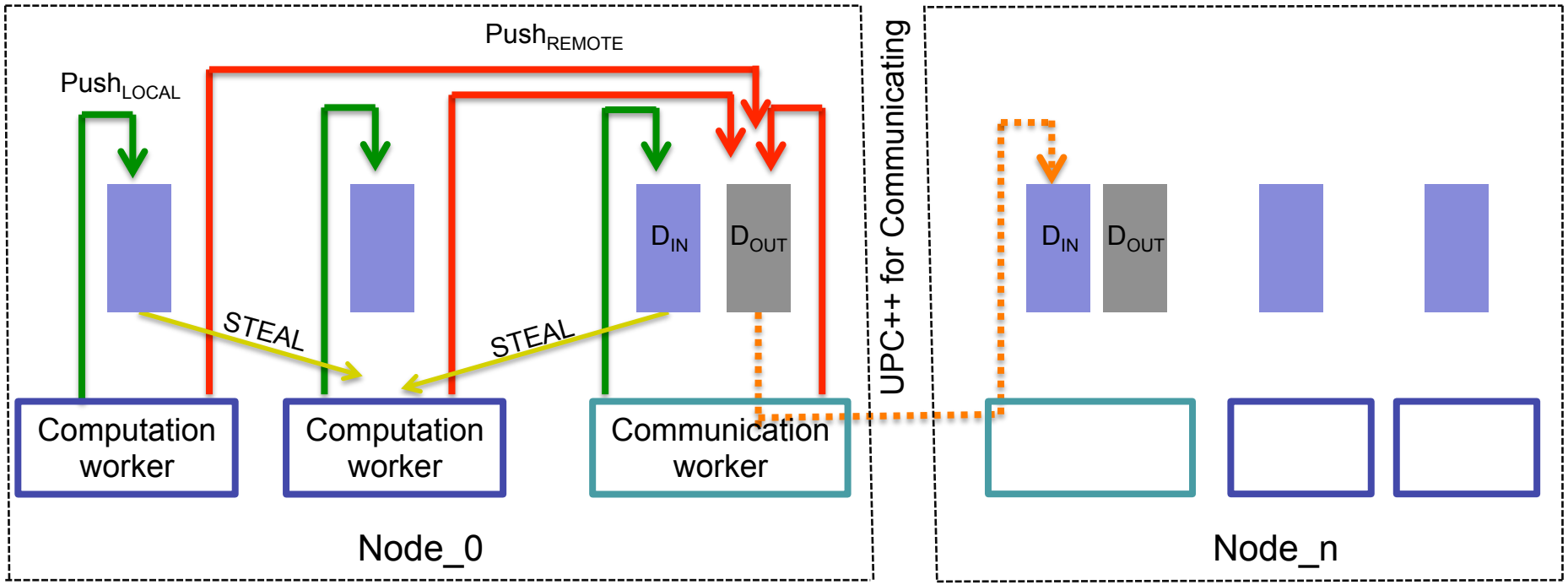
# Mapping Lambda Functions to Runtime

```
template < typename T >
void asyncAt ( int dest_place, T lambda ) {
    if ( dest_place == my_place ) {
      pass_to_OCR ( local_async, heap_allocated_lambda );
    }
    else {
      shared_lambda = allocate_memory_global_address_space (sizeof (T));
      copy ( shared_lambda, lambda );
      pass_to_OCR ( remote_async, shared_lambda );
    }
}
```



RICE BERKELEY LAB

# Integrating OCR with UPC++




Communication worker (SPMD)

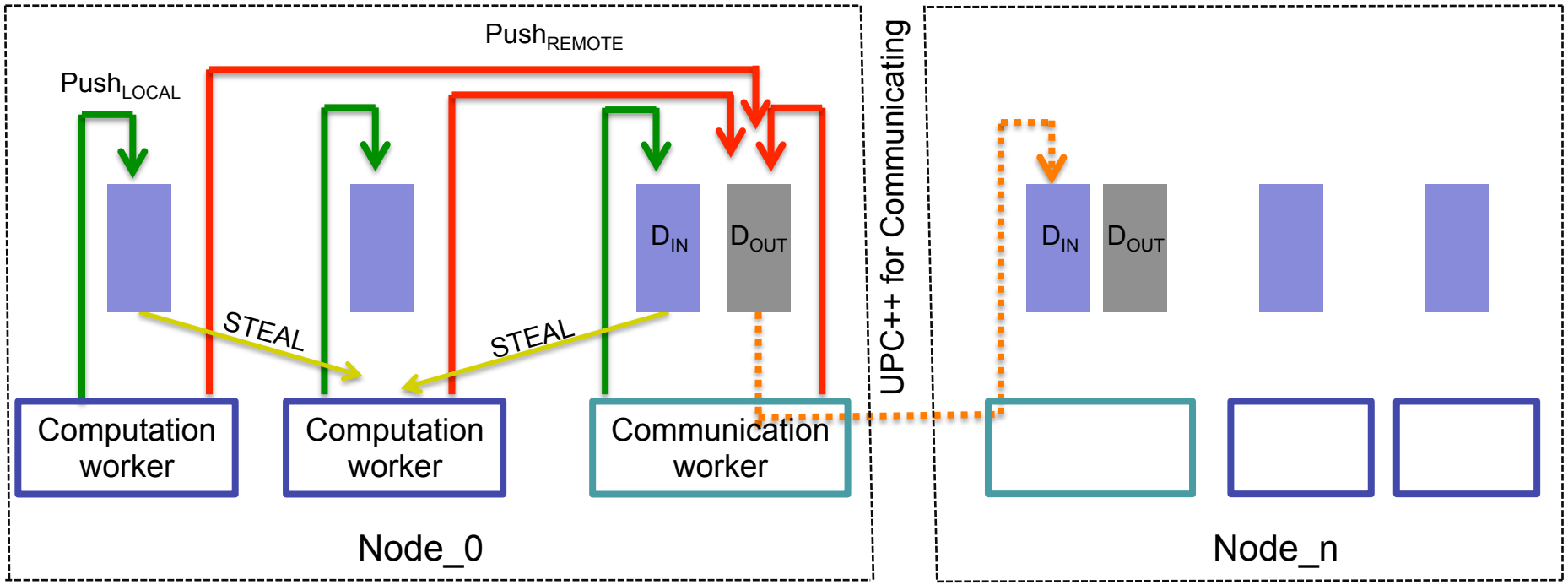
```
main ( ) {
    finish_spmd ([=]() {
        local;
        remote;
    });
}
```



RICE

BERKELEY LAB

# Termination Detection




Communication worker (SPMD)

```
main ( ) {
    finish_spmd ([=]() {
        local;
        remote;
    });
}
```

```
while ( true ) {
    pop_execute_tasks_from_out_deq ( );
    push_incoming_remote_task_in_deq ( );
    count_pending_tasks_my_place ( );
    allReduce ( sum_global_tasks_count );
    if ( global_pending_count == 0 ) break;
}
```



RICE

BERKELEY LAB

# Outline

- Background
- Motivation and Insights
- HabaneroUPC++ Programming Model
- Implementation
- Results
- Summary

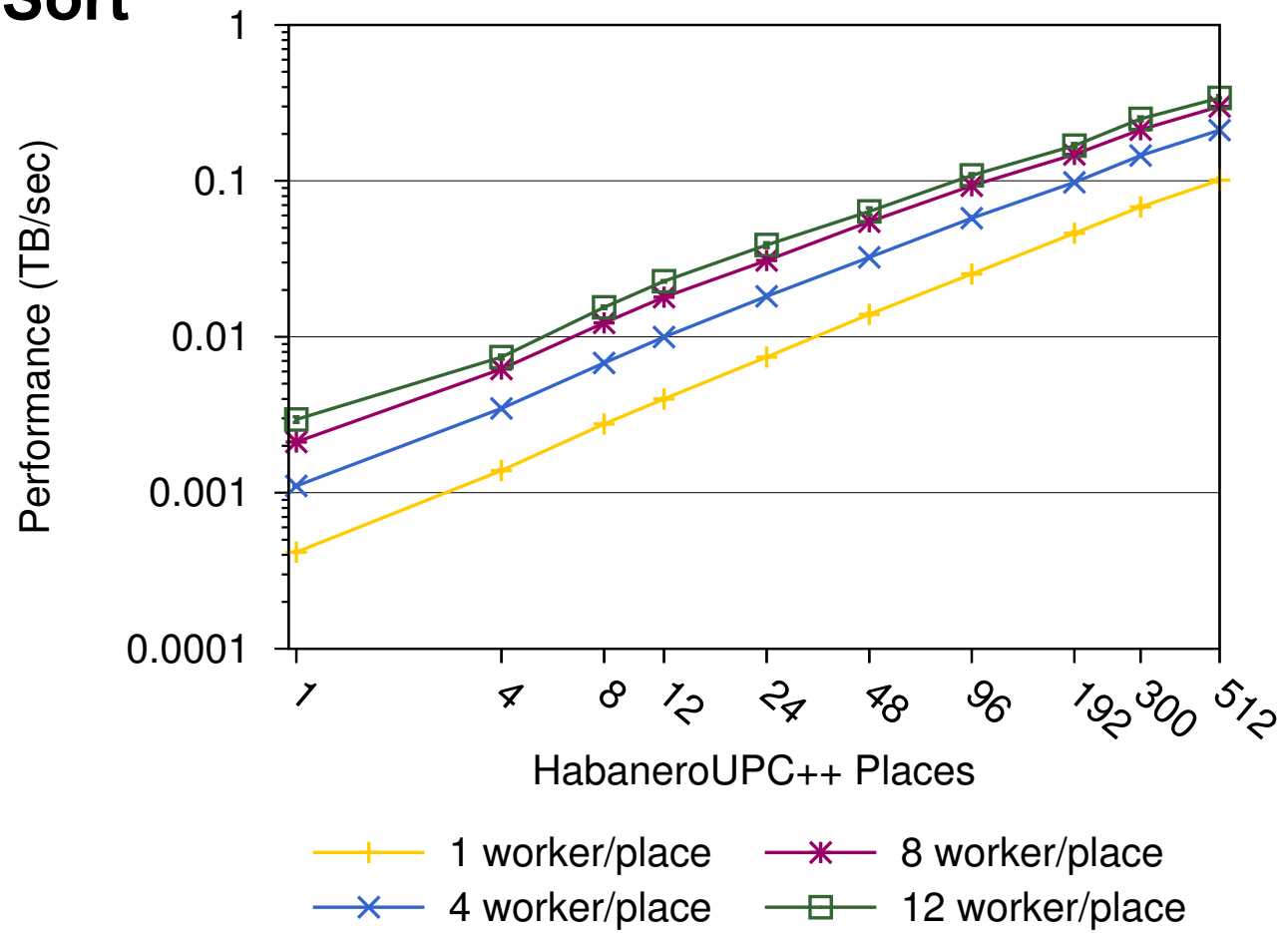



# Experimental Setup

- Benchmarks
  - Sample sorting  (distributed array of integers)
  - LULESH (hydrodynamics simulation)
- Machine
  - Edison supercomputer at NERSC
  - 1 Node = 2 × 12 CPUs
  - 1 place / socket
- Compilers
  - GCC compiler 4.9.0 (C++11 support ≥ GCC 4.7.0)

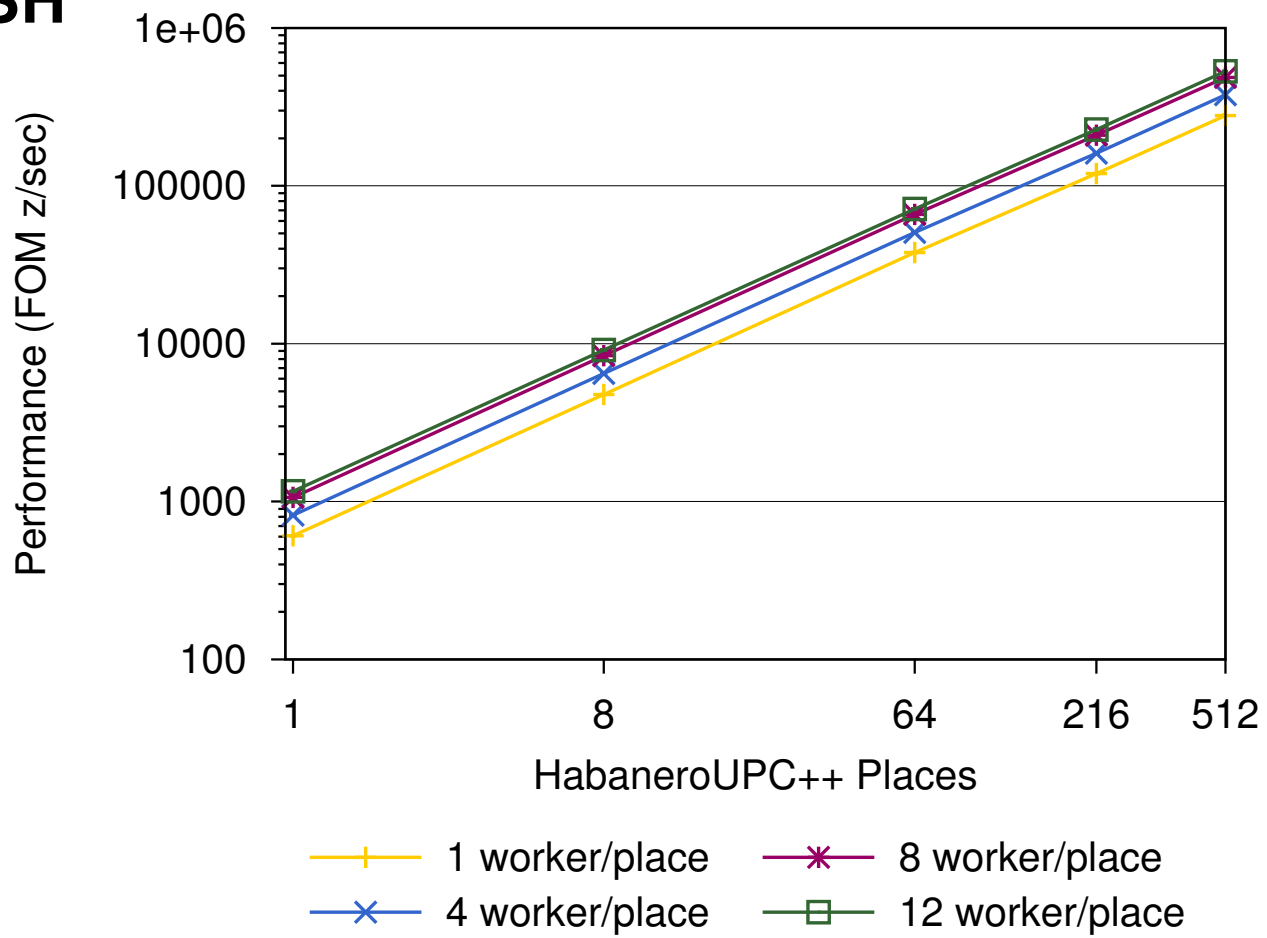
# Weak Scaling Result

**Sample Sort**



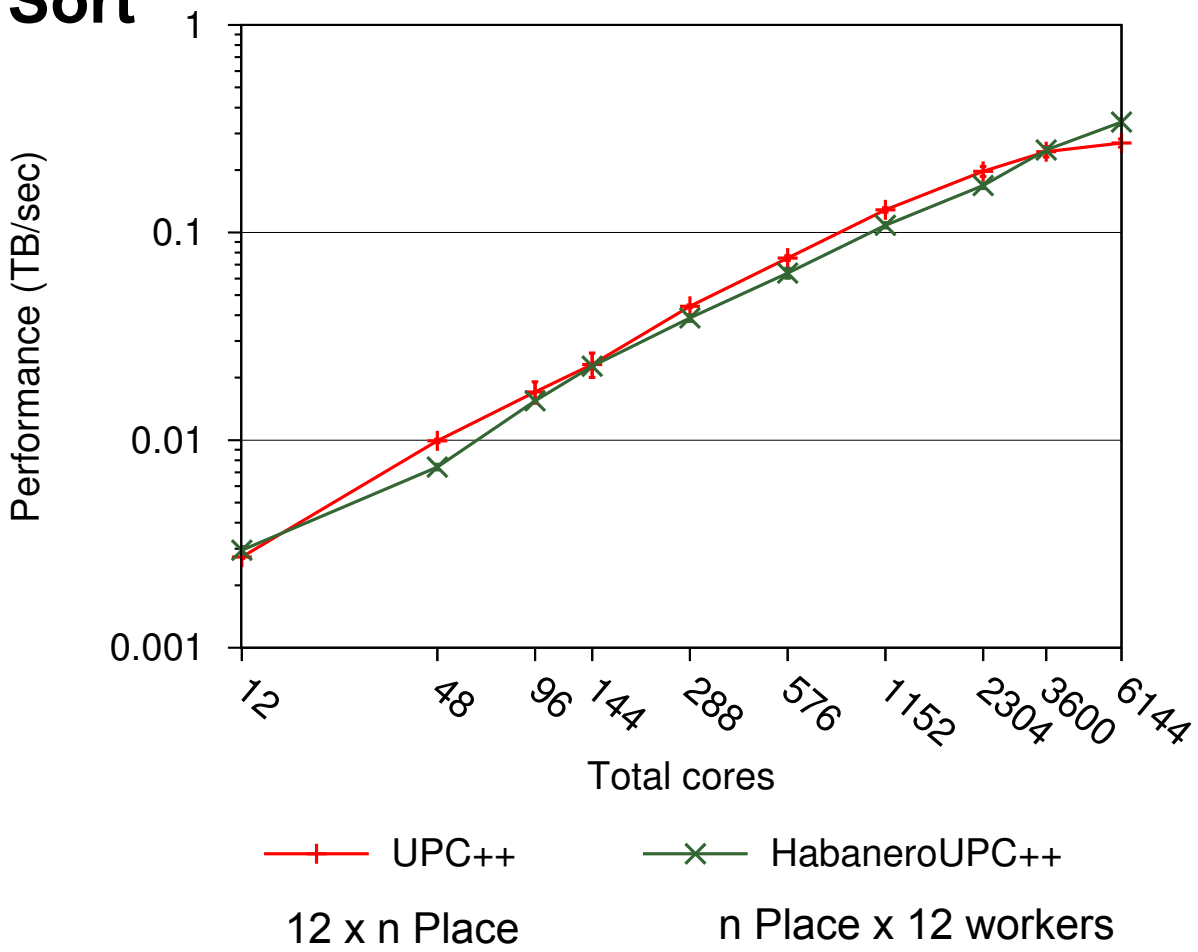
# Weak Scaling Result

**LULESH**



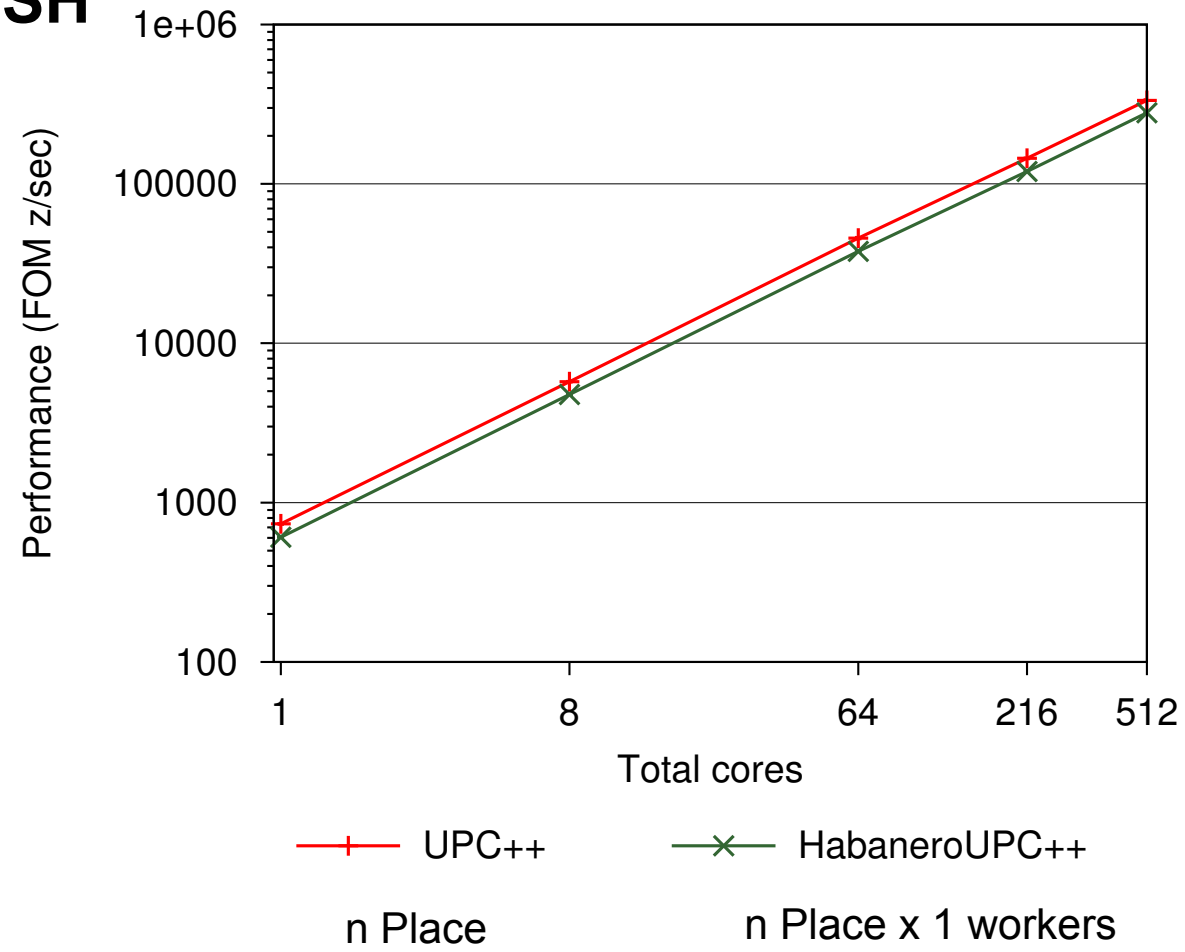
# Performance Comparison

**Sample Sort**



# Performance Comparison

**LULESH**




n Place

n Place x 1 workers

# Outline

- Background
- Motivation and Insights
- HabaneroUPC++ Programming Model
- Implementation
- Results
- Summary




# Summary

- HabaneroUPC++
  - C++11 based compiler-free PGAS library
  - Intermixes Habanero asynchronous task programming model in UPC++
  - **Intra**-*place* work-stealing using OCR
  - **Inter**-*place* communications using UPC++
  - Encouraging results with Sorting and LULESH

Download: http://habanero-rice.github.io/habanero-upc/



RICE

BERKELEY LAB