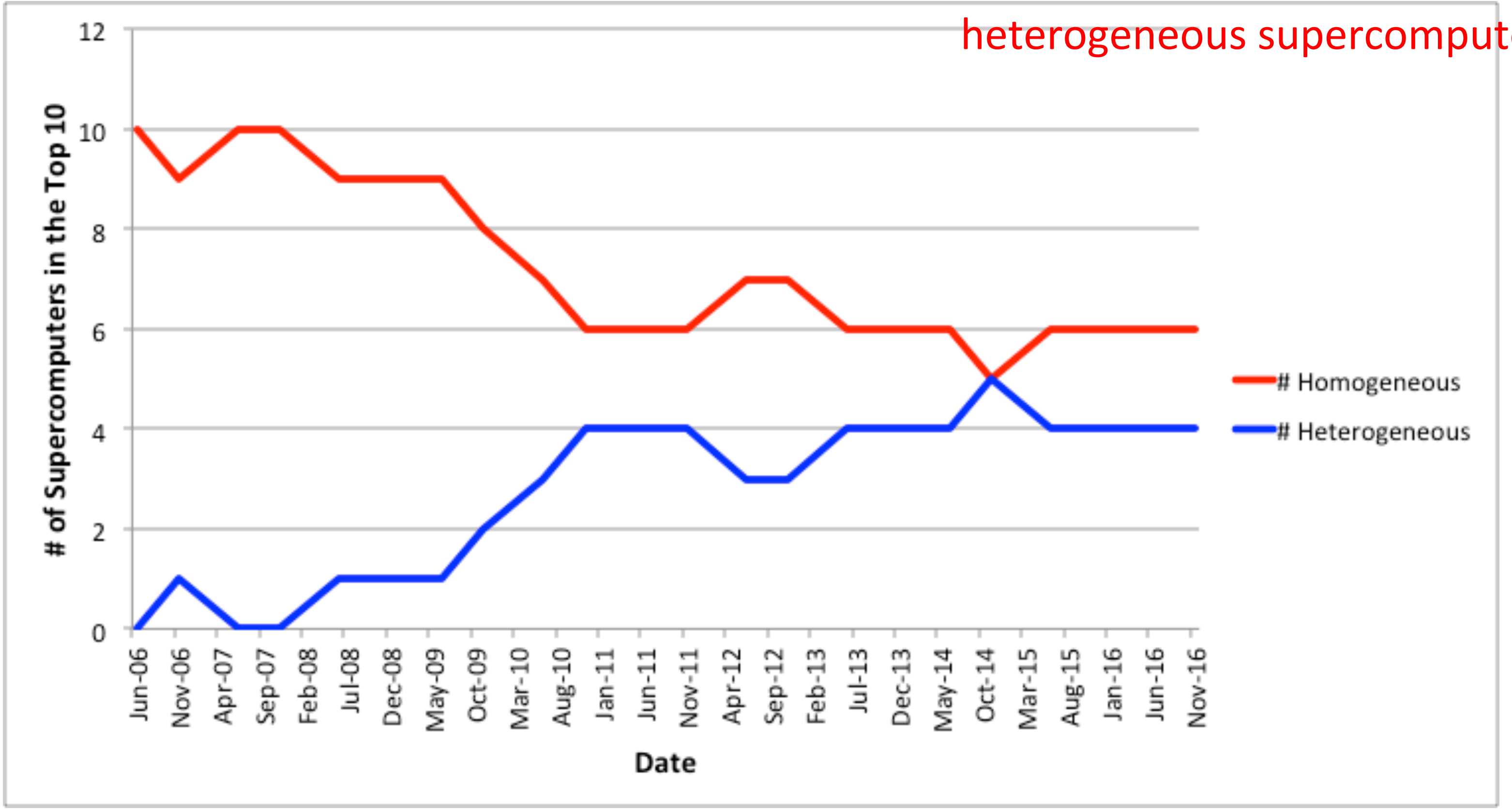
A Pluggable Framework for Composable HPC Scheduling Libraries

Max Grossman[1], Vivek Kumar[2], Nick Vrvilo[1], Zoran Budimlic[1], Vivek Sarkar[1]
[1]Habanero Extreme Scale Software Research Group, Rice University
[2]IIIT-Delhi

AsHES 2017 - May 29 2017

Top10

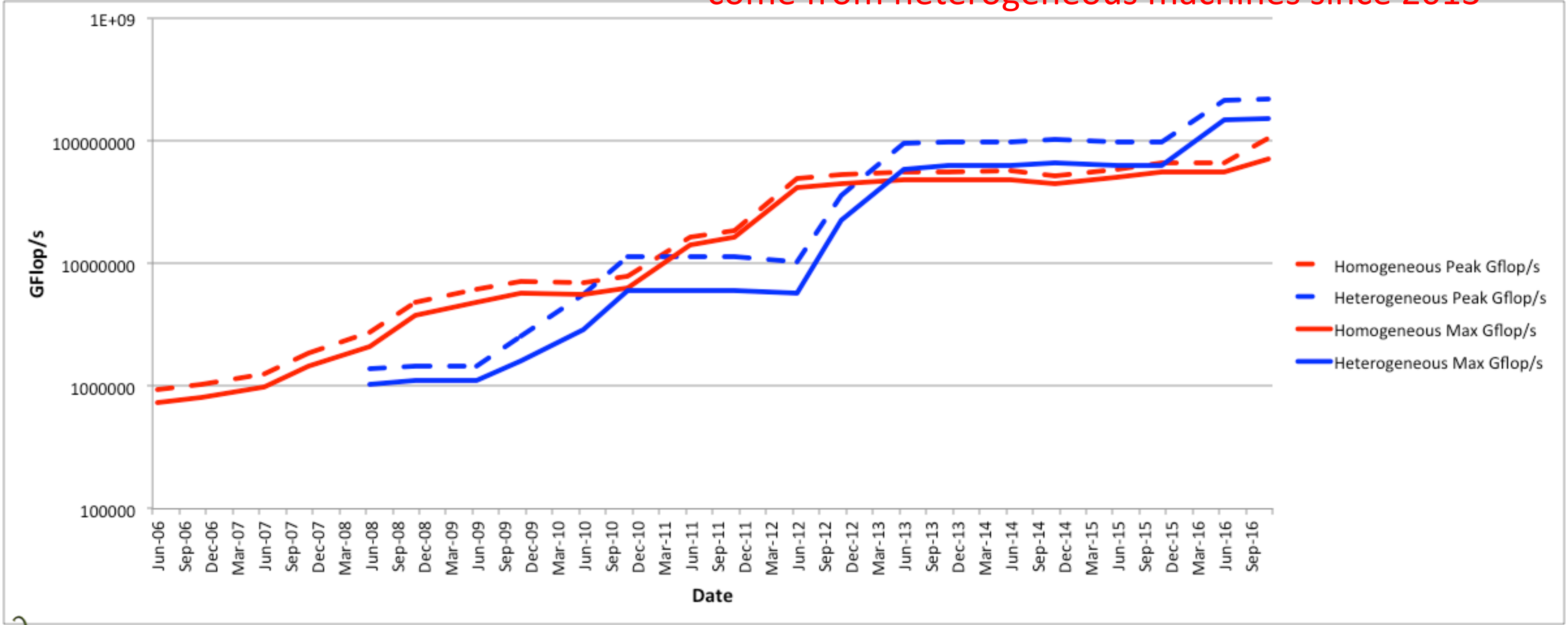
Past decade has seen more heterogeneous supercomputers



https://www.top500.org

Top10

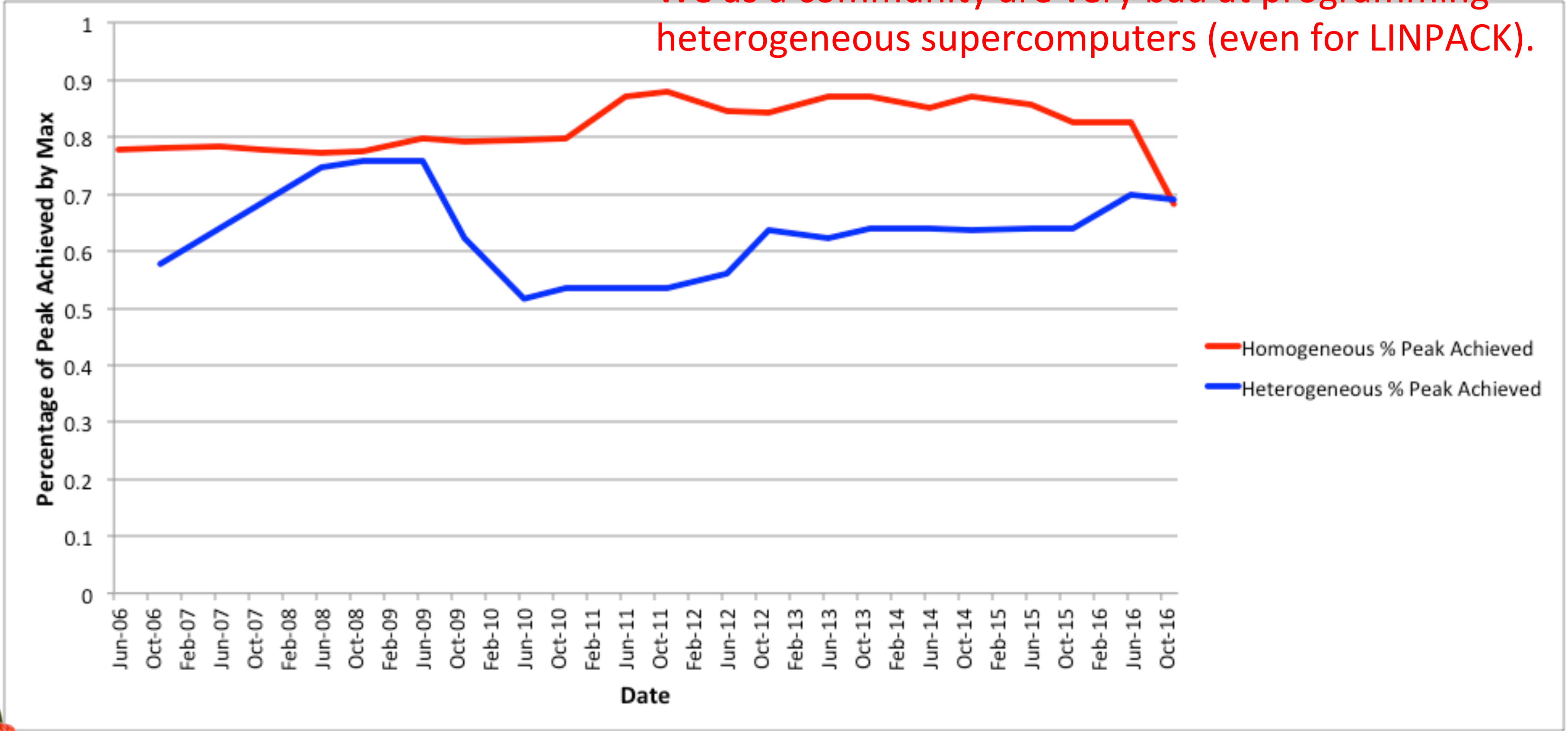
Majority of Top10 Peak and Achieved GFlop/s has come from heterogeneous machines since 2013



https://www.top500.org

Top10

We as a community are very bad at programming heterogeneous supercomputers (even for LINPACK).



https://www.top500.org

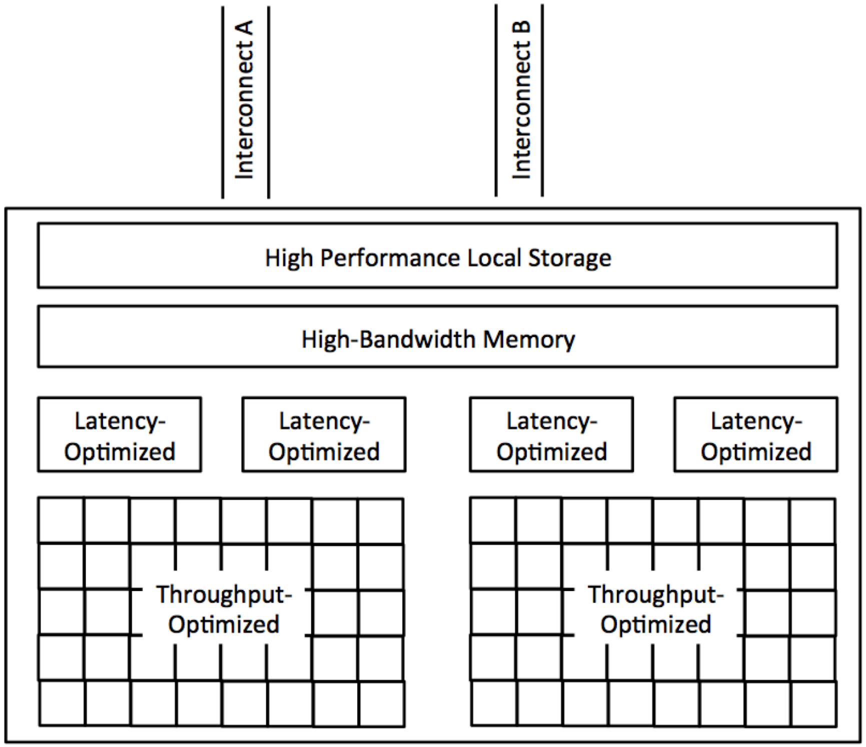
How Do We Define Heterogeneity?

For the past decade, “heterogeneous computing” == “GPUs”

- Dealing with GPUs has taught us a lot about software heterogeneity

But heterogeneity is on the rise everywhere in HPC:

- Hardware: memory, networks, storage, cores
- Software: networking libraries, compute libraries, managed runtimes, domain libraries, storage APIs



Depiction of the abstract platform motivating this work.

Heterogeneous Programming in Practice

OpenCL

OpenACC
Directives for Accelerators

NVIDIA CUDA

OpenMP

Trilinos

cuDNN

OpenMP

Trilinos

pthreads

QThreads

OPEN MPI

Open SHMEM

BERKELEY UPC UNIFIED PARALLEL C

Heterogeneous Programming in Research

Legion: Hide all heterogeneity from user, rely on runtime to map problem to hardware efficiently, implicit dependencies discovered by runtime.

Parsec, OCR: Explicit dataflow model.

HCMPI, HCUPC++, HC-CUDA, HPX: Task-based runtimes that create dedicated proxy threads for managing some external resource (e.g. NIC, GPU).

HiPER: Generalize a task-based, locality-aware, work-stealing runtime/model to support non-CPU resources.

- Retain the appearance of legacy APIs
- Composability, extensibility, compatibility are first-class citizens from the start.

Outline

HiPER Execution & Platform Model

HiPER Use Cases

- MPI Module
- Composing MPI and CUDA

Performance Evaluation

Conclusions & Future Work

Outline

HiPER Execution & Platform Model

HiPER Use Cases

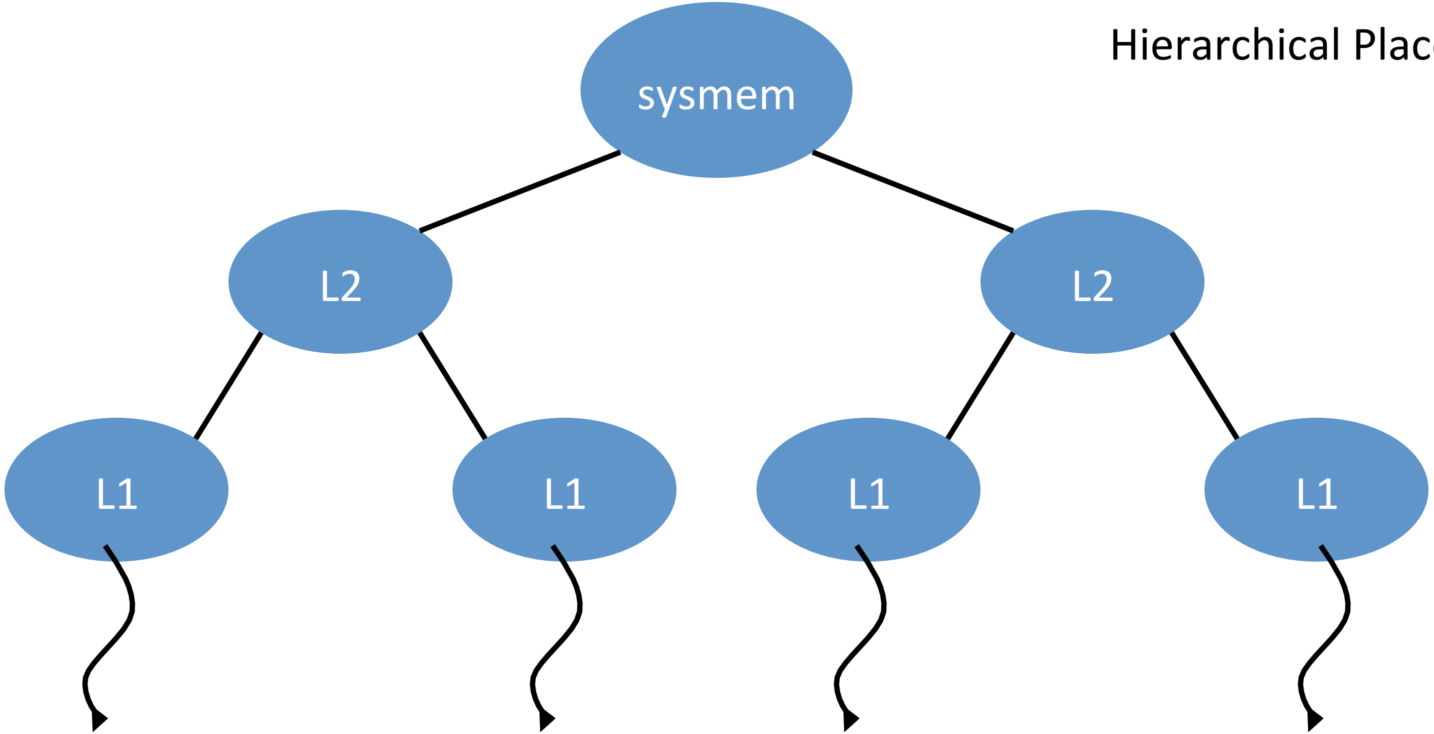
- MPI Module
- Composing MPI and CUDA

Performance Evaluation

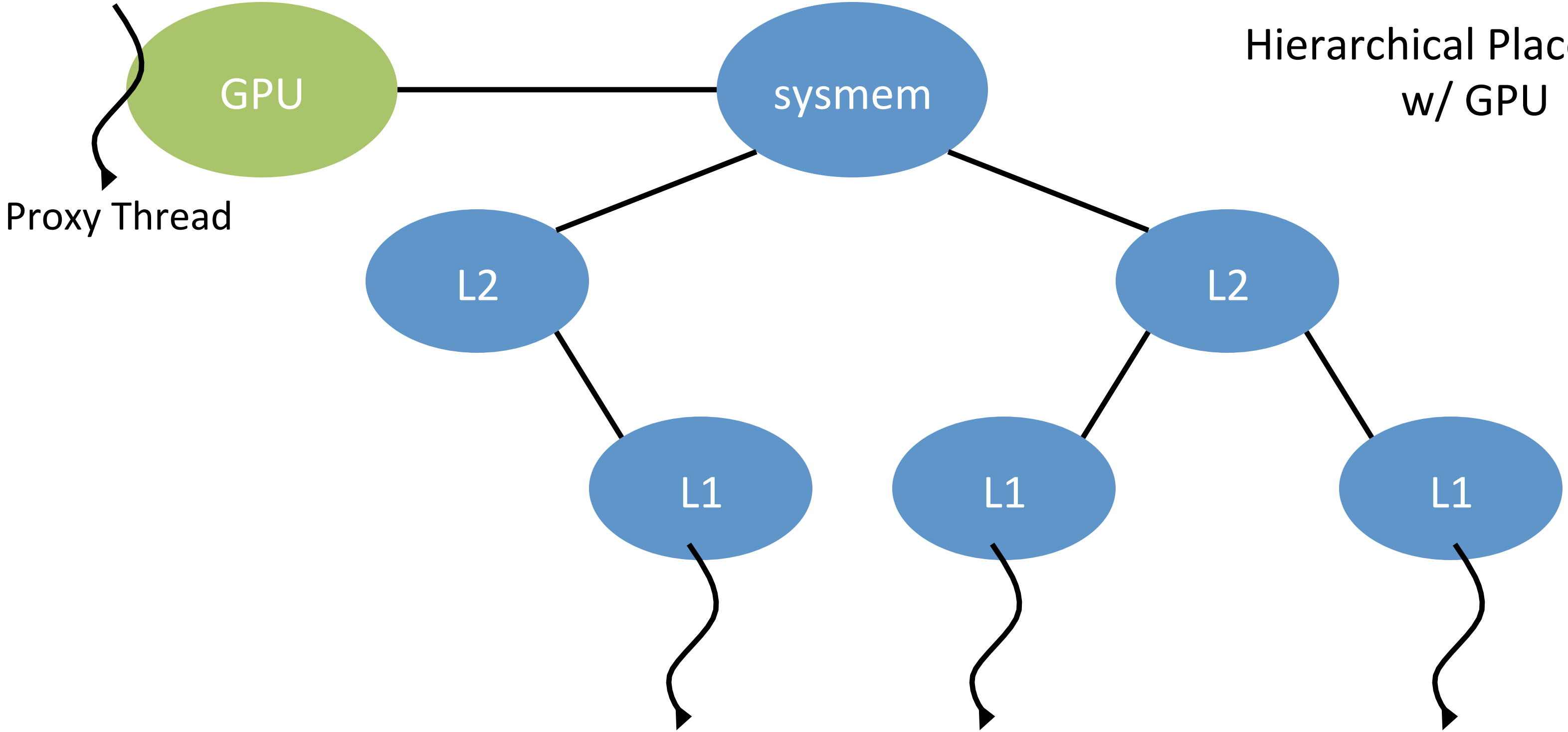
Conclusions & Future Work

HiPER's Predecessors

Hierarchical Place Trees

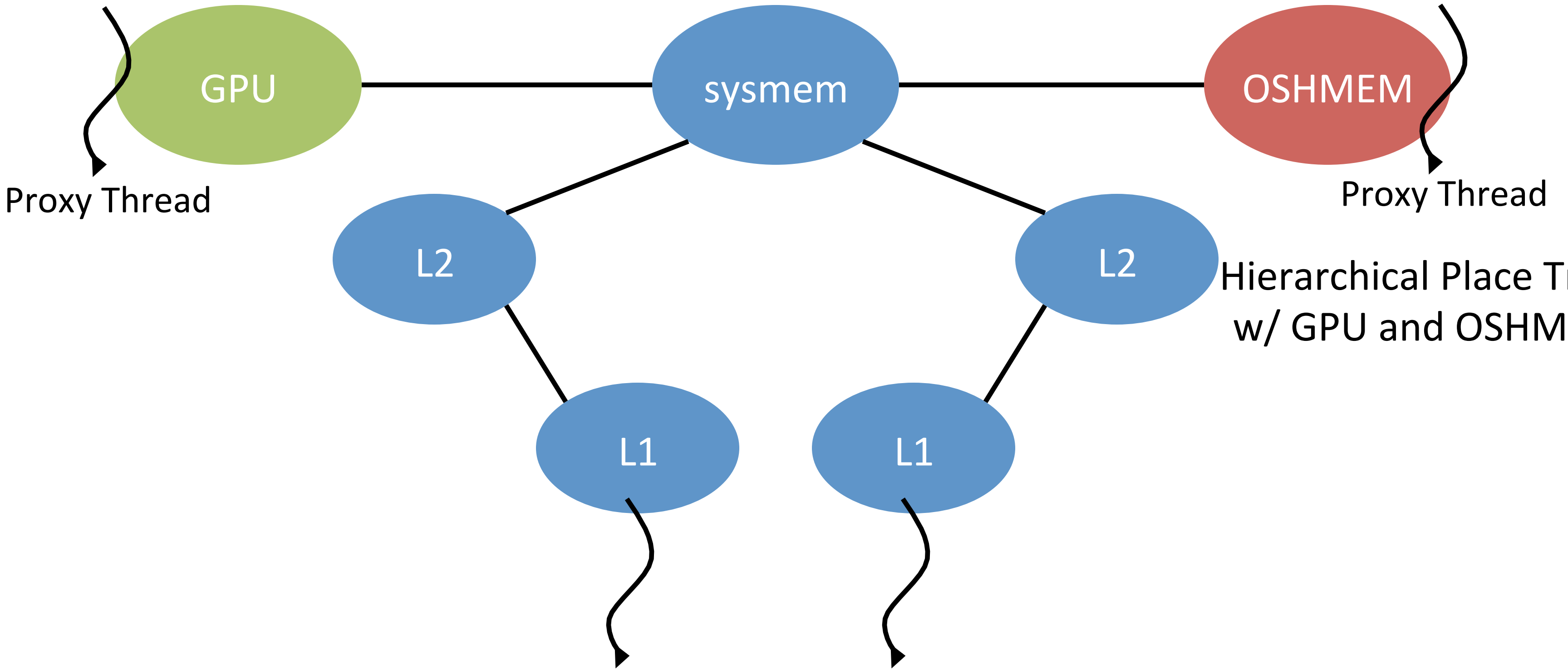


HiPER's Predecessors



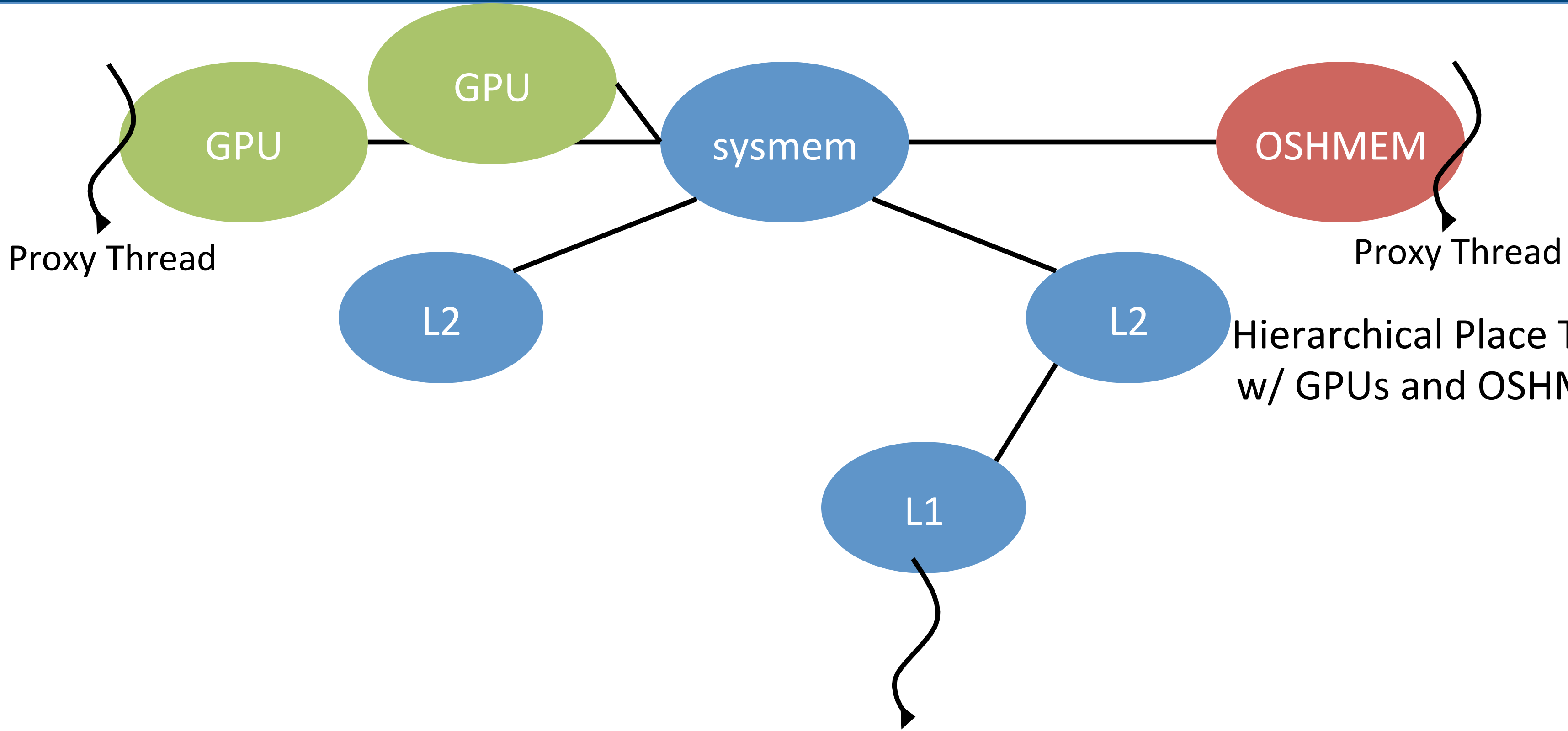
Hierarchical Place Trees
w/ GPU

HiPER’s Predecessors



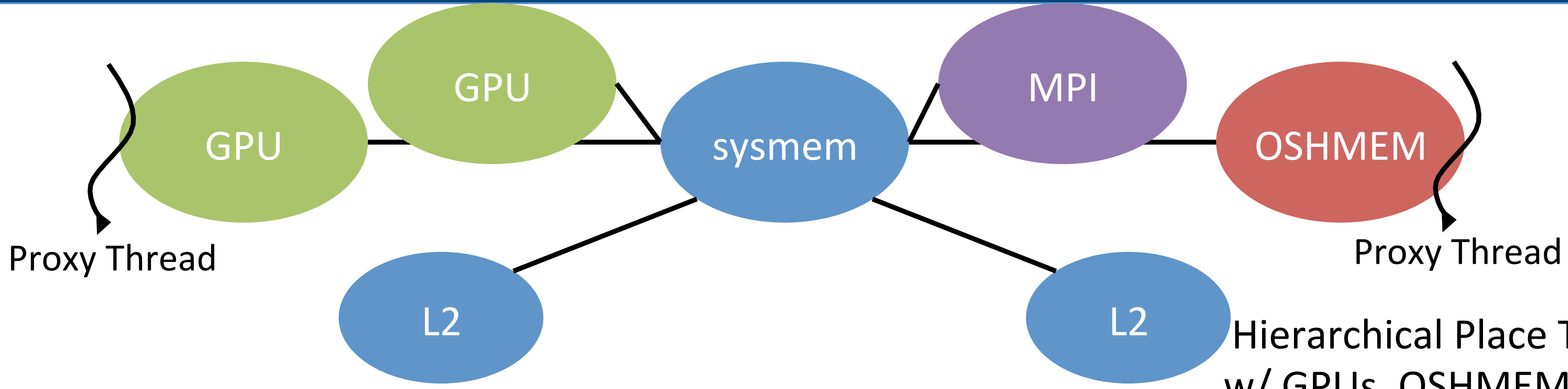
Hierarchical Place Trees w/ GPU and OSHMEM

HiPER's Predecessors



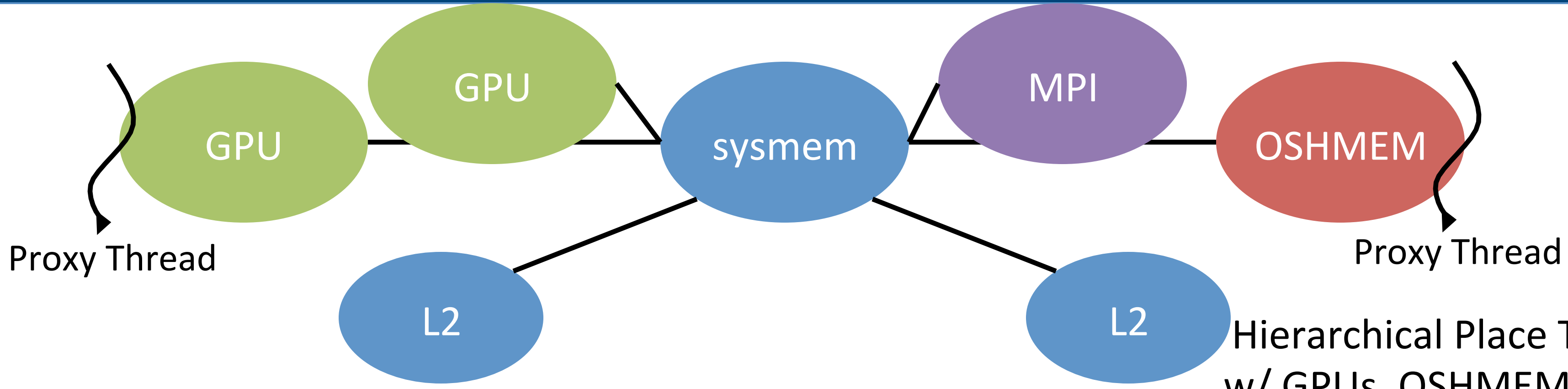
Hierarchical Place Trees w/ GPUs and OSHMEM

HiPER’s Predecessors



Hierarchical Place Trees
w/ GPUs, OSHMEM, MPI

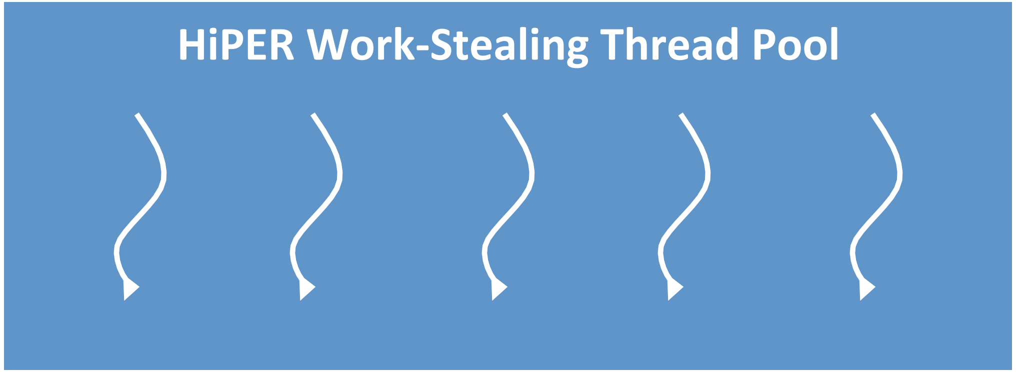
HiPER's Predecessors



Hierarchical Place Trees w/ GPUs, OSHMEM, MPI

- Simple model makes it attractive for many past research efforts, but…
  - Not scalable software engineering
  - Wasteful use of host resources
  - Not easily extendable to new software/hardware capabilities

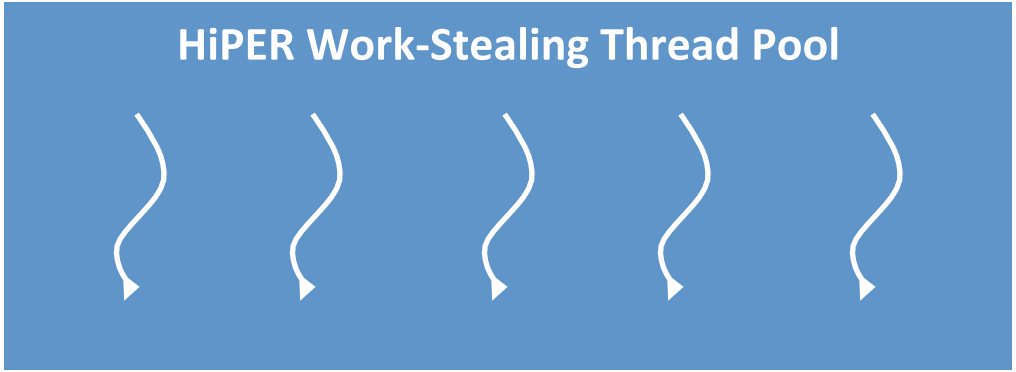
HiPER Platform & Execution Model



HiPER Platform & Execution Model

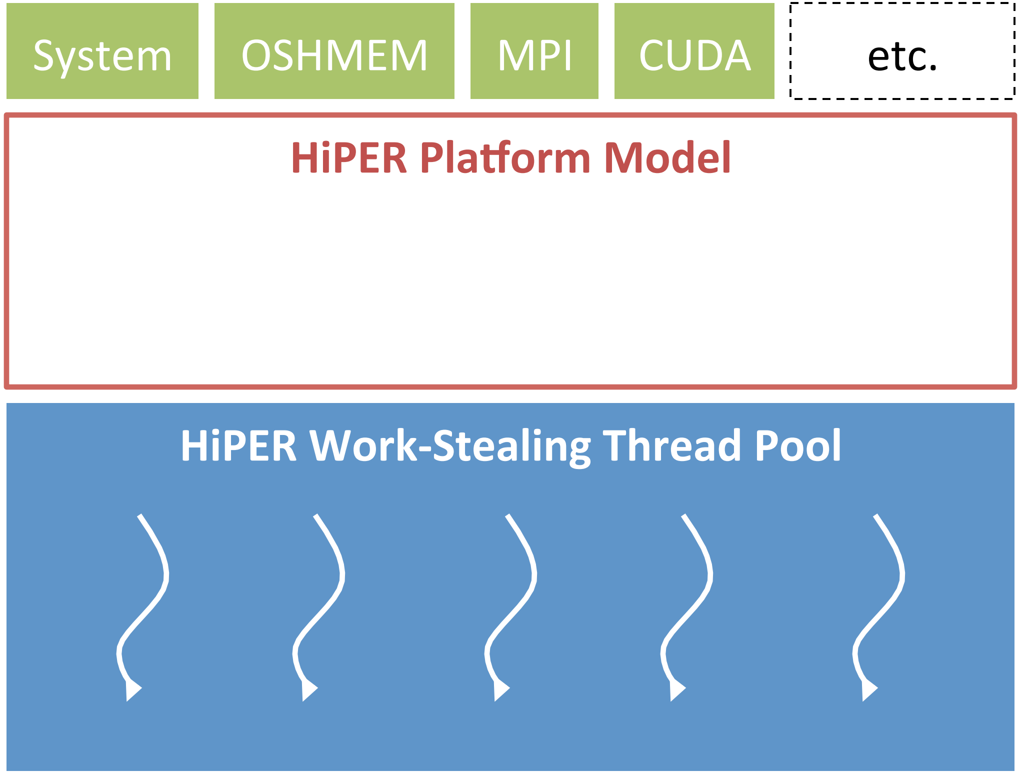
Pluggable Modules → System | OSHMEM | MPI | CUDA | etc.

Modules expose user-visible APIs for work creation.



HiPER Platform & Execution Model

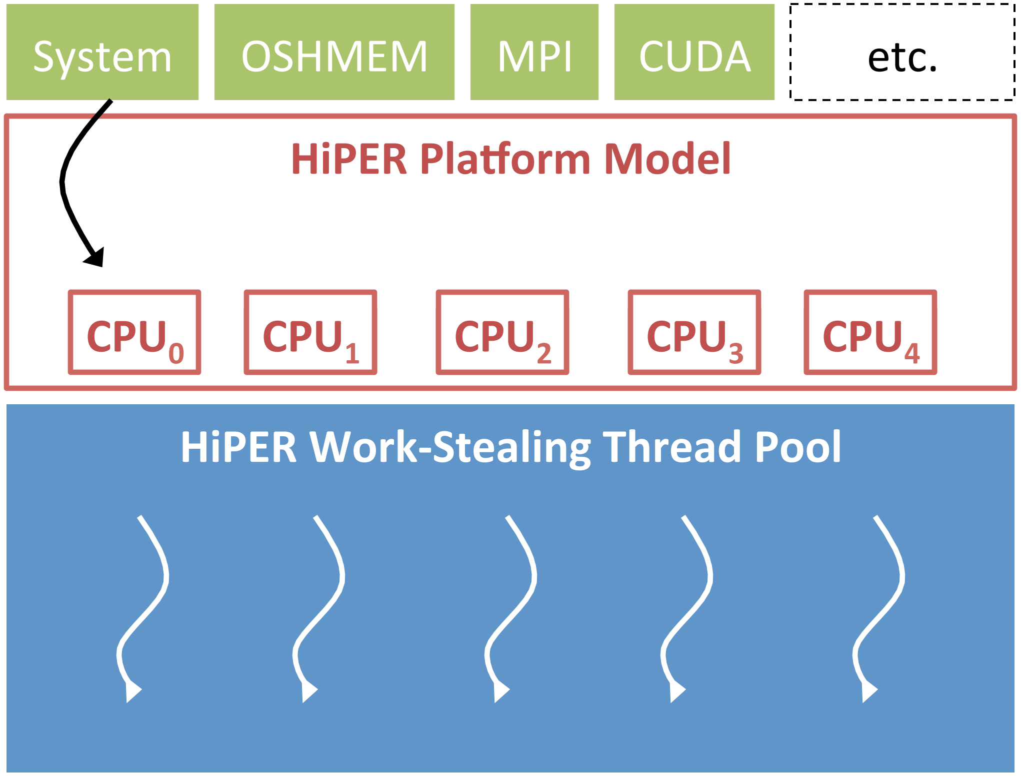
Pluggable Modules



Platform model gives modules somewhere to place work, thread pool somewhere to find work.

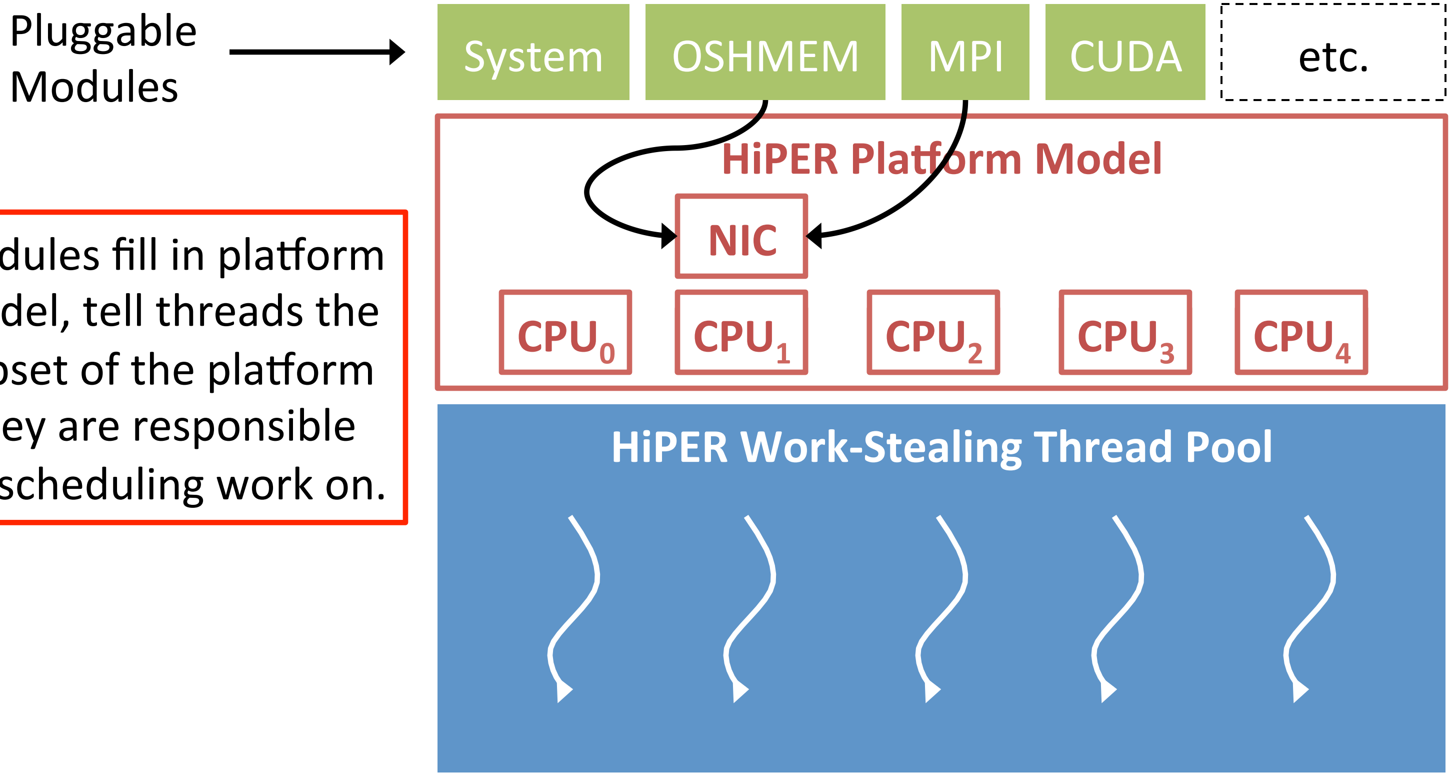
HiPER Platform & Execution Model

Pluggable Modules



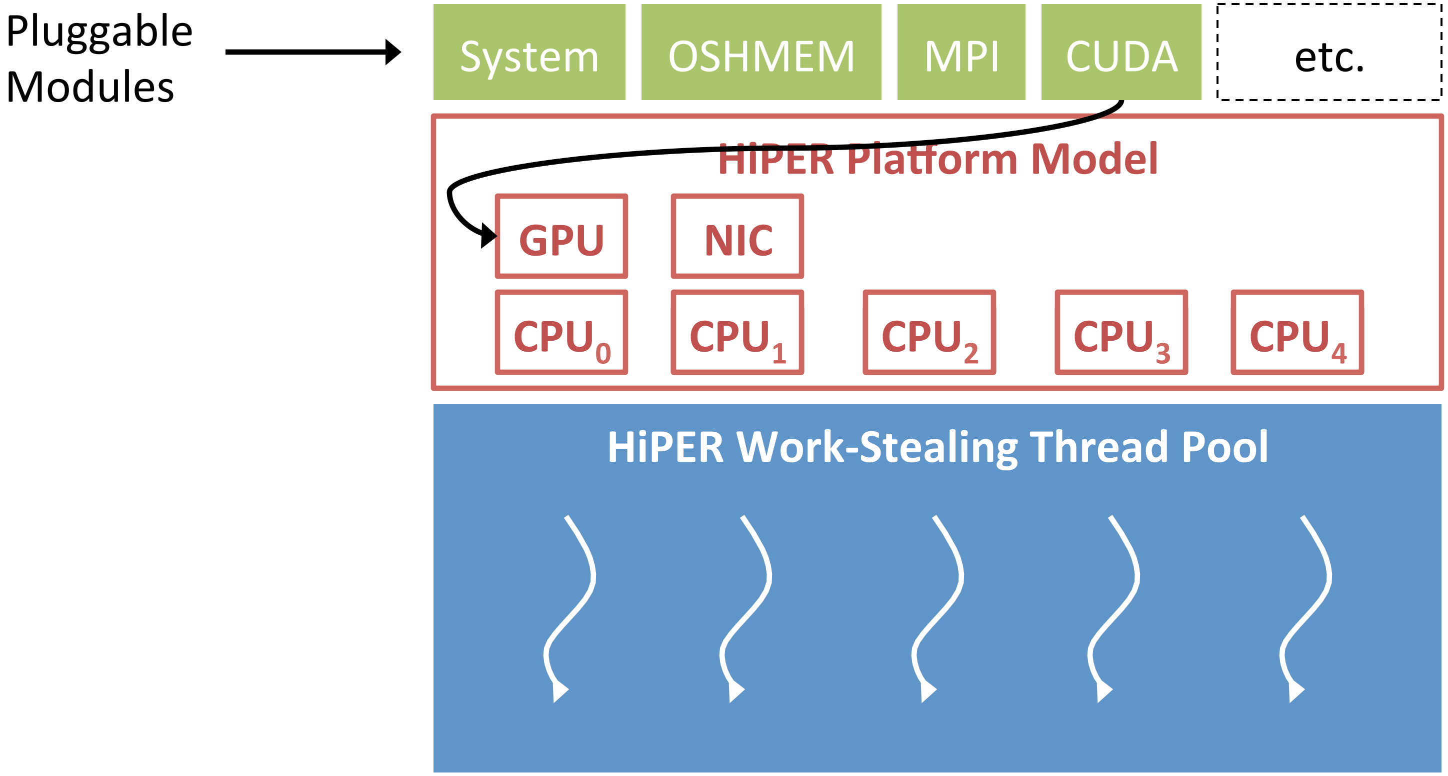
Modules fill in platform model, tell threads the subset of the platform they are responsible for scheduling work on.

HiPER Platform & Execution Model



Modules fill in platform model, tell threads the subset of the platform they are responsible for scheduling work on.

HiPER Platform & Execution Model



Outline

HiPER Execution & Platform Model

HiPER Use Cases
- MPI Module
- Composing MPI and CUDA

Performance Evaluation

Conclusions & Future Work

Fundamental Task-Parallel API

The HiPER core exposes a fundamental C/C++ tasking API.

API	Explanation
`async([] { S1; });`	Create an asynchronous task
`finish([] { S2; });`	Suspend calling task until nested tasks have completed
`async_at([] { S3; }, place);`	Create an async. task at a place in the platform model
`fut = async_future([] { S4; });`	Get a future that is signaled when a task completes
`async_await([] { S5; }, fut);`	Create an asynchronous task whose execution is predicated on satisfaction of fut.

Summary of core tasking APIs. The above list is not comprehensive.

MPI Module

Extends HiPER namespace with familiar MPI APIs

- Programmers can use the APIs they already know and love
- Built on 1) an MPI implementation, and 2) HiPER's core tasking APIs.

Asynchronous APIs return futures rather than MPI_Requests, enabling composability in programming layer with all other future-based APIs:

```
hiper::future_t<void> *MPI_Irecv/Isend(...);
```

Enables non-standard extensions, e.g.:

`hiper::future_t<void> *MPI_Isend_await(...,` `hiper::future_t<void> *await);`	Start an asynchronous send once await is satisfied.
`hiper::future_t<void> *MPI_Allreduce_future(...);`	Asynchronous collectives.

Example API Implementation

```
hiper::future_t<void> *hiper::MPI_Isend_await(..., hiper::future_t<void> *await) {
    // Create a promise to be satisfied on the completion of this operation
    hiper::promise_t<void> *prom = new hiper::promise_t<void>();

    // Taskify the actual MPI_Isend at the NIC, pending the satisfaction of await
    hclib::async_nb_await_at([=] {
        // At MPI place, do the actual Isend
        MPI_Request req;
        ::MPI_Isend(..., &req));

        // Create a data structure to track the status of the pending Isend
        pending_mpi_op *op = malloc(sizeof(*op));
        ...
        hiper::append_to_pending(op, &pending, test_mpi_completion, nic);
    }, fut, nic);

    return prom->get_future();
}
```

Periodic polling function

Composing System, MPI, CUDA Modules

```
// Asynchronously process ghost regions on this rank in parallel on CPU
ghost_fut = forasync_future([] (z) { ... });

// Asynchronously exchange ghost regions with neighbors
reqs[0] = MPI_Isend_await(..., ghost_fut);
reqs[1] = MPI_Isend_await(..., ghost_fut);
reqs[2] = MPI_Irecv(...);
reqs[3] = MPI_Irecv(...);

// Asynchronously process remainder of z values on this rank
kernel_fut = forasync_cuda(..., [] (z) { ... });

// Copy received ghost region to CUDA device
copy_fut = async_copy_await(..., reqs[2], reqs[3], kernel_fut);
```

Outline

HiPER Execution & Platform Model

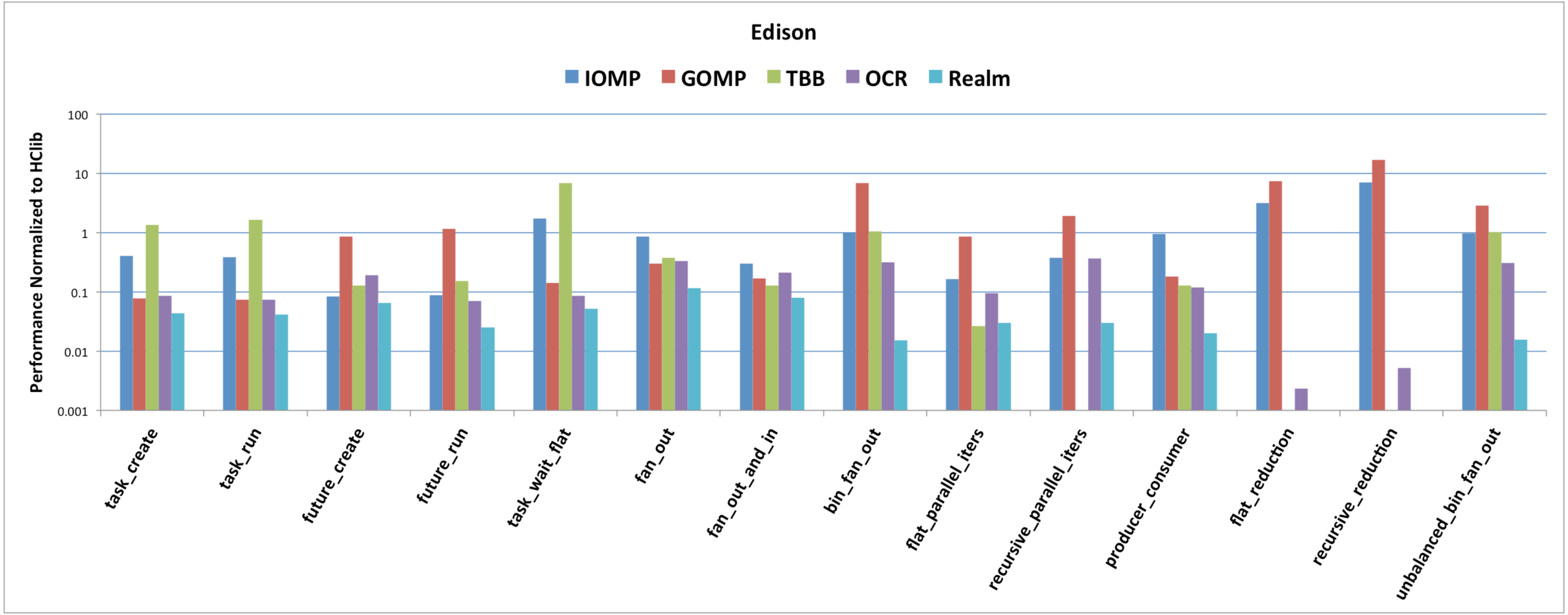
HiPER Use Cases
- MPI Module
- Composing MPI and CUDA

Performance Evaluation

Conclusions & Future Work

Task Micro-Benchmarking

Micro-benchmark performance normalized to HiPER on Edison, higher is better.



https://github.com/habanero-rice/tasking-micro-benchmark-suite

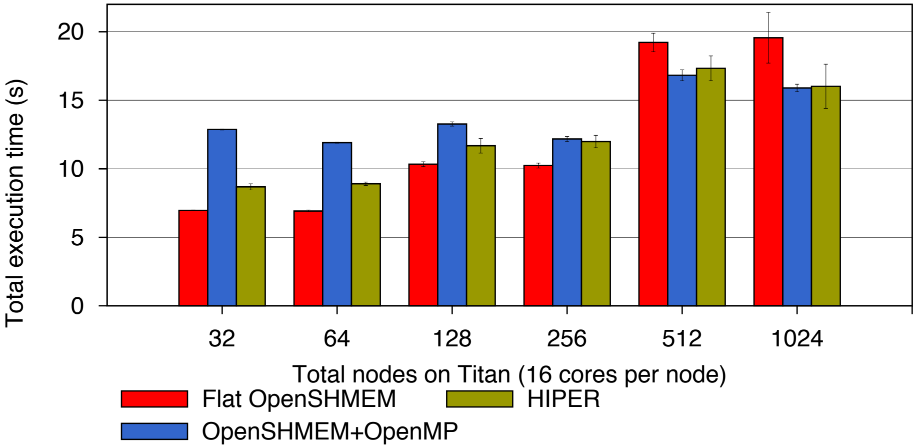
Experimental Setup

Experiments shown here were run on Titan @ ORNL and Edison @ NERSC.

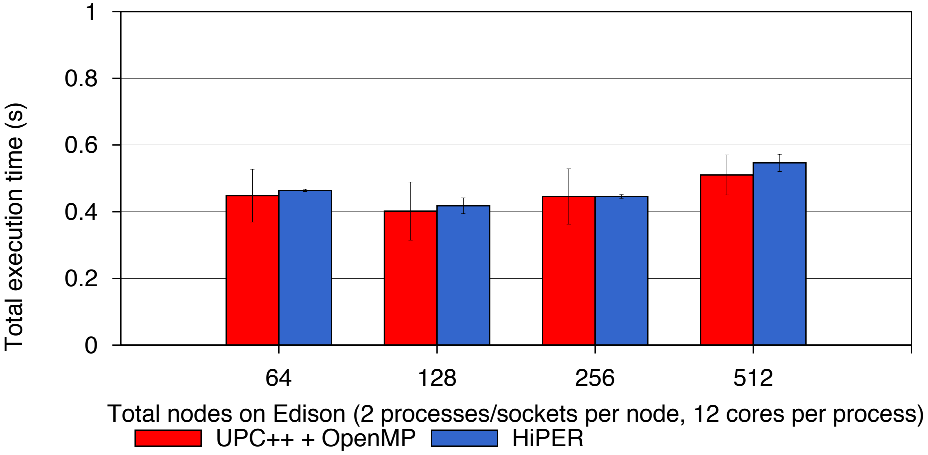
Application	Platform	Dataset	Modules Used	Scaling
ISx	Titan	2^{29} keys per node	OpenSHMEM	Weak
HPGMG-FV	Edison	log2_box_dim=7 boxes_per_rank=8	UPC++	Weak
UTS	Titan	T1XXL	OpenSHMEM	Strong
Graph500	Titan	2^{29} nodes	OpenSHMEM	Strong
LBM	Titan		MPI, CUDA	Weak

HiPER Evaluation – Regular Applications

HIPER is low-overhead, no impact on performance for regular applications



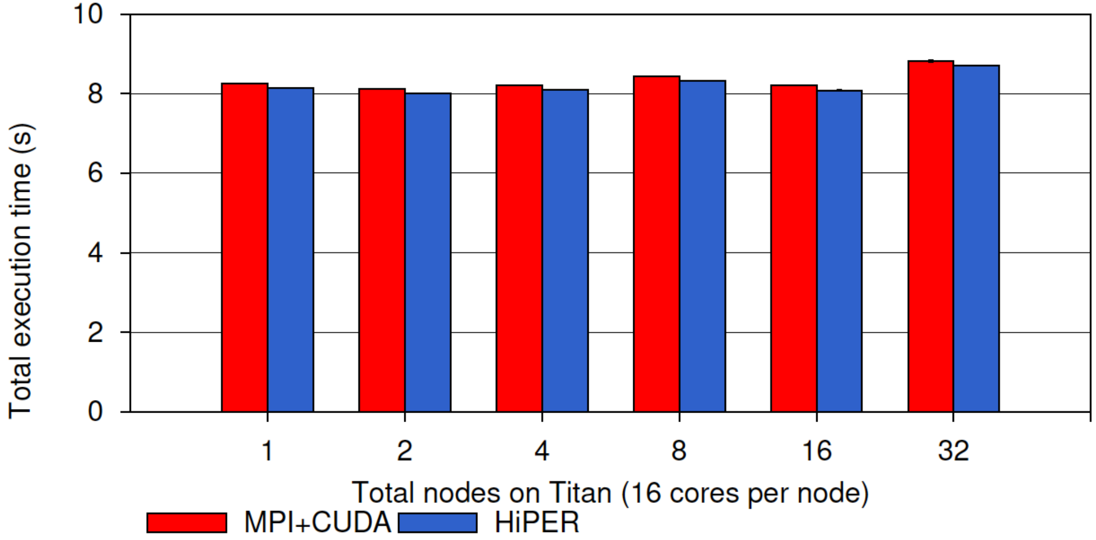
ISx



HPGMG Solve Step

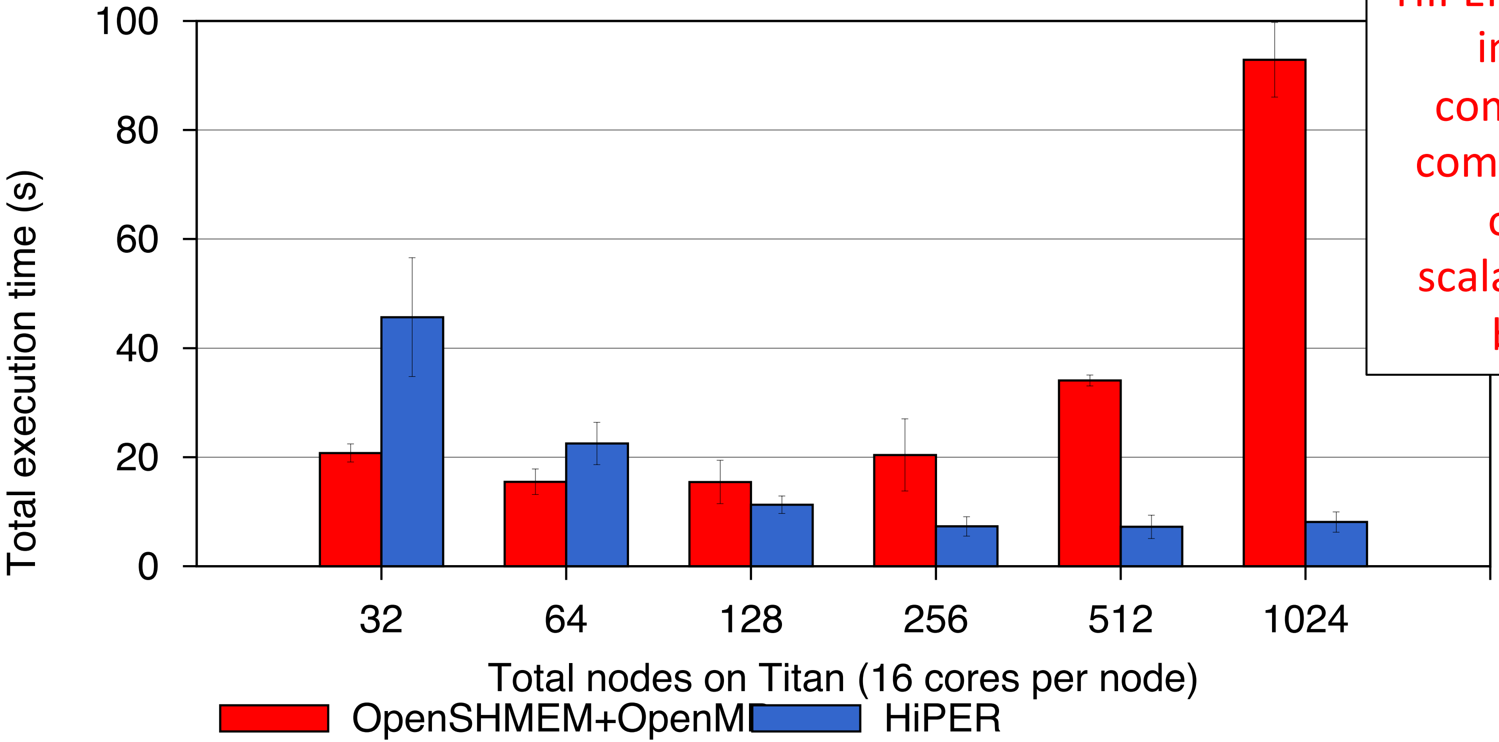
HiPER Evaluation – Regular Applications

~2% performance improvement through reduced synchronization from futures-based programming.



LBM

HiPER Evaluation – UTS



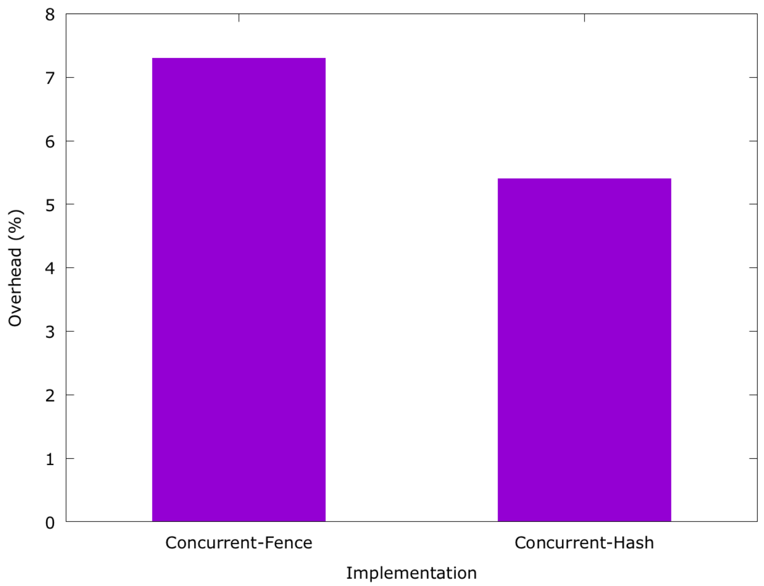
HIPER integration improves computation-communication overlap, scalability, load balance

HiPER Evaluation – Graph500

HiPER used for concurrent (not parallel) programming in Graph500.

Rather than periodic polling, use novel `shmem_async_when` APIs to trigger local computation on incoming RDMA.

Reduces code complexity, hands scheduling problem to the runtime.



Outline

HiPER Execution & Platform Model

HiPER Use Cases

- OpenSHMEM w/o Thread Safety
- OpenSHMEM w/ Contexts

Performance Evaluation

Conclusions & Future Work

HiPER

Working to generalize past work on improving the composability of HPC libraries through tasking. Exploring both improvements at the runtime and API layer.

Drive system requirements using OpenSHMEM, but also currently support composing CUDA, MPI, UPC++.

Future work:

- Continuing work on additional module support (integration with OpenSHMEM contexts)
- Continue to iterate on existing benchmarks
- New application development (Fast Multipole Method)

https://github.com/habanero-rice/hclib/tree/resource_workers
https://github.com/habanero-rice/tasking-micro-benchmark-suite

Acknowledgements

Los Alamos
NATIONAL LABORATORY
EST. 1943

DEPARTMENT OF DEFENSE
UNITED STATES OF AMERICA

RICE

UNIVERSITY of
HOUSTON

Stony Brook
University