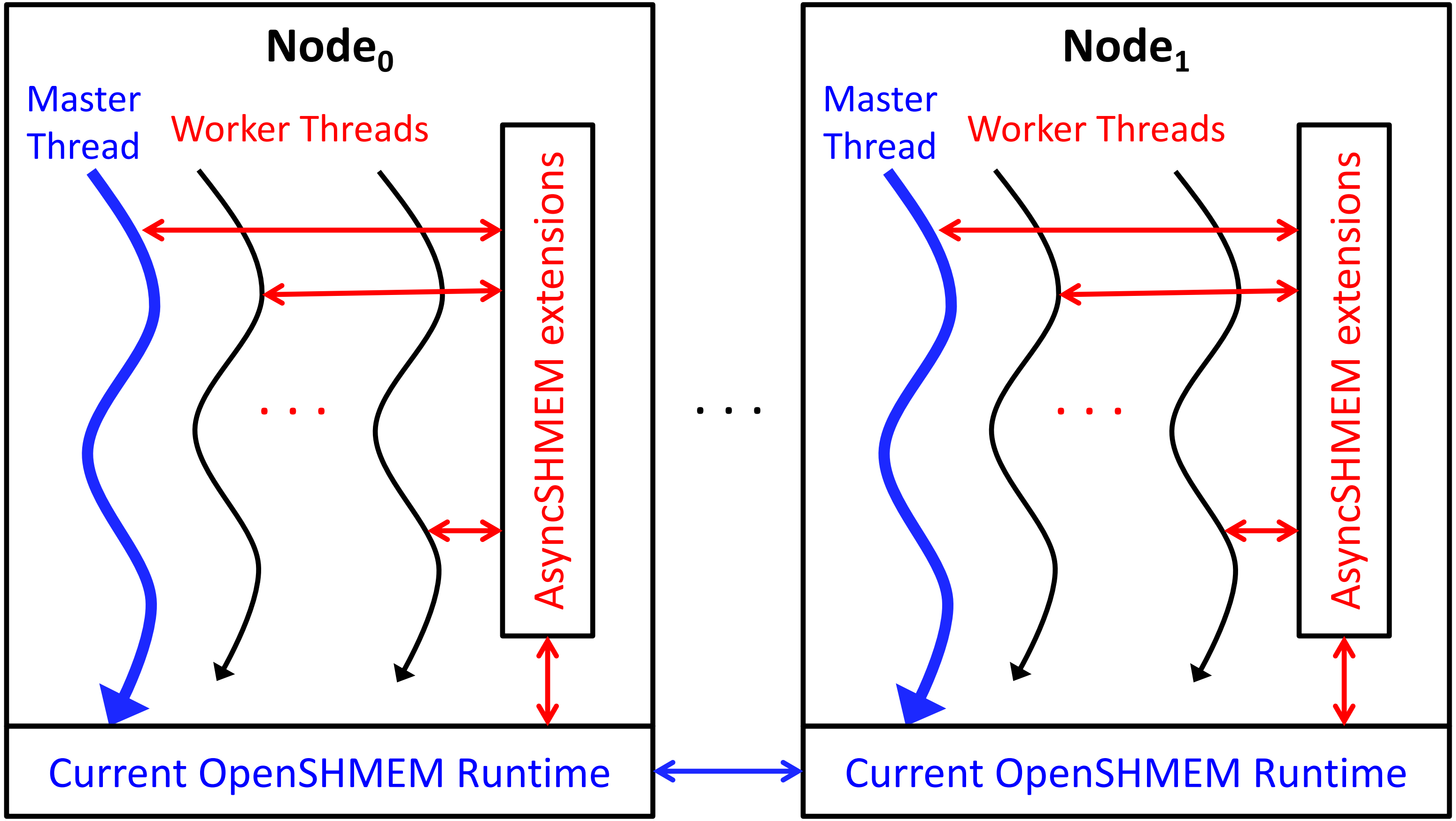
HOOVER: Distributed, Flexible, and Scalable Streaming Graph Processing on OpenSHMEM

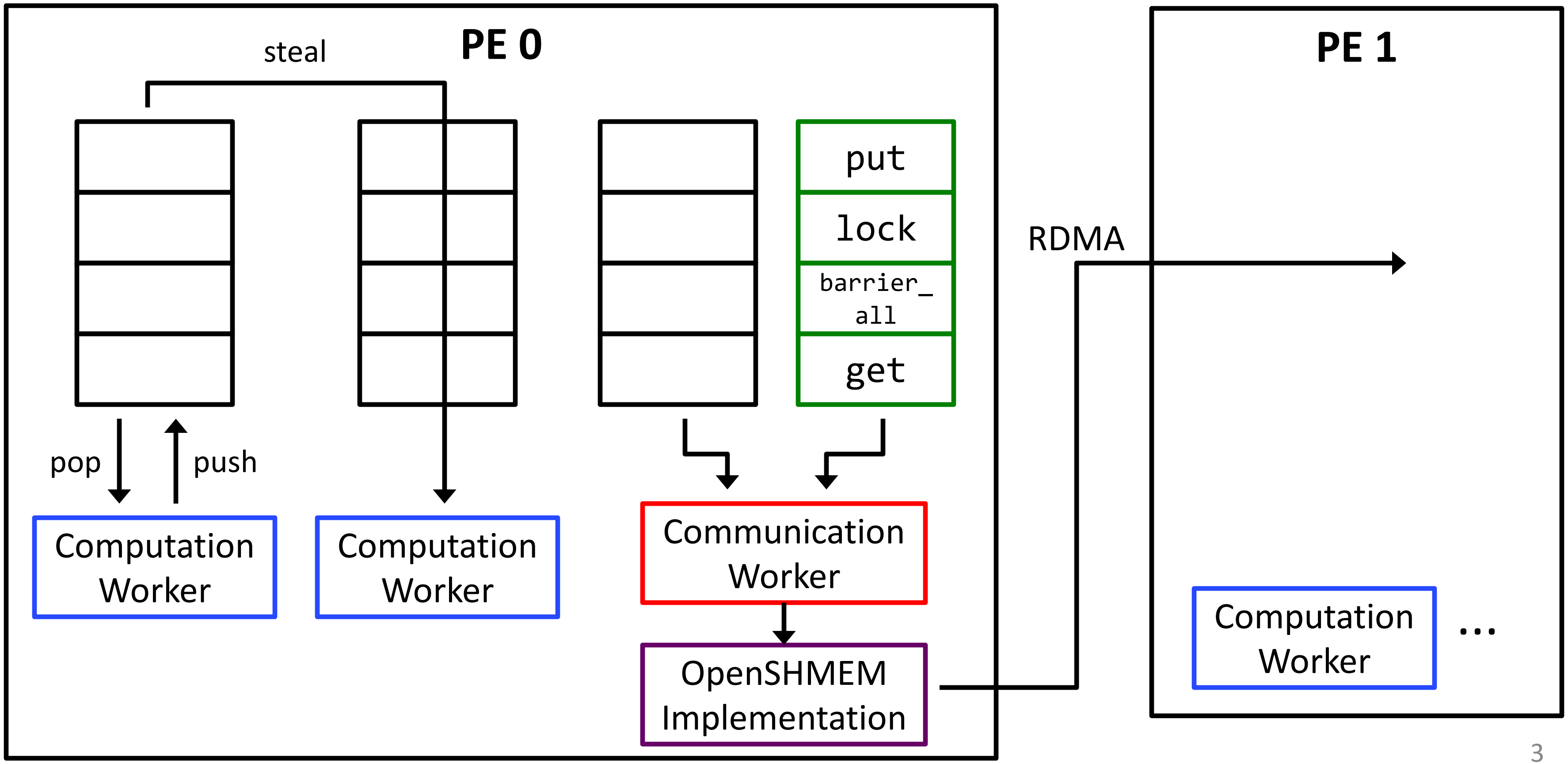
Max Grossman, Vivek Sarkar

Rice University, Georgia Institute of Technology

Past Work: AsyncSHMEM



Offload Runtime



Creating an asynchronous task: shmem_task()

```
void shmem_task(void (*body)(void *), void *data);
```

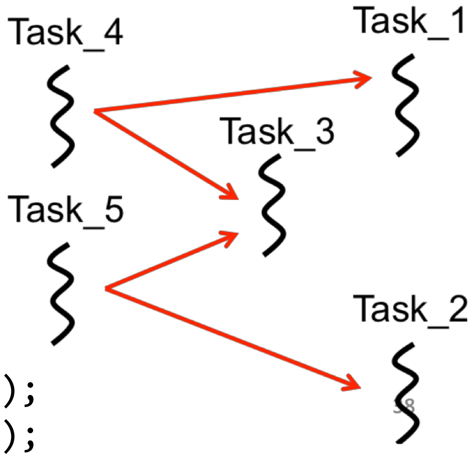
Creates an asynchronous task defined by body (like “begin” construct in Chapel)

```
void foo(void *data) {// Body of child task
    . . .
}

void entrypoint(void *args) { // Body of root task
    shmem_task(foo, NULL);
}

int main(int argc, char** argv) {
    shmem_worker_init(entrypoint, NULL);
}
```

Example of DAG parallelism using futures

Futures enable more complex dependency graphs than fork-join tasks

```
void task_4(void *task4_prom) {
    // some computation
    shmem_satisfy_promise((shmem_promise_t *)task4_prom);
}
void task_5(void *task5_prom) {
    // some computation
    shmem_satisfy_promise((shmem_promise_t *)task5_prom);
}

shmem_task_await(task_1, args, shmem_future_for_promise(task4_prom));
shmem_task_await(task_2, args, shmem_future_for_promise(task5_prom));
shmem_task_await(task_3, args, shmem_future_for_promise(task4_prom),
        shmem_future_for_promise(task5_prom));
shmem_task(task_4, task4_prom);
shmem_task(task_5, task5_prom);
```



API Extensions: Communication-Driven Tasks

```
void shmem_int_task_when(int *ivar, int cond, int value,
        void (*body)(void *), void *data);
```

Create an asynchronous task when the specified condition is satisfied on the specified location in the symmetric heap. Analogous to `shmem_int_wait_until`, except that this call never blocks.

```
void shmem_int_task_when_any(int **ivars, int cond, int *values,
        void (*body)(void *), void *data);
```

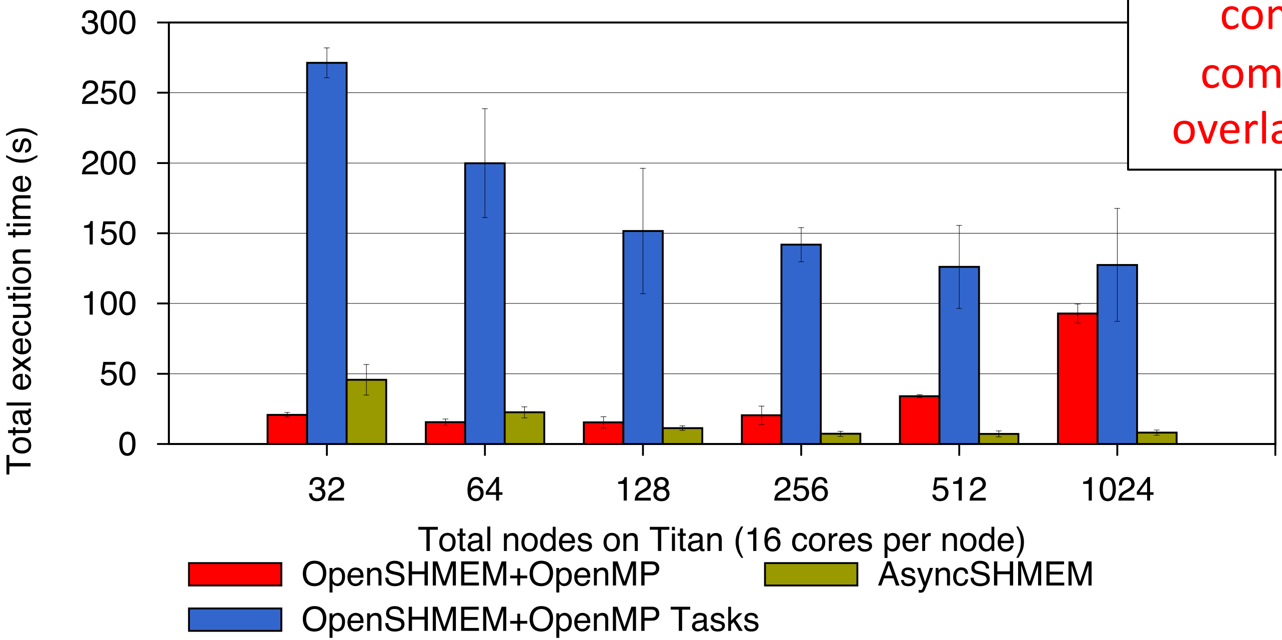
Same as `shmem_int_task_when`, but allows waiting for any of multiple conditions.

Communication-driven tasks allow remote communication to trigger asynchronous task creation on a PE.

Analogous to existing `shmem_wait` APIs, but these APIs do not block, and also offer single- and multi-condition variants.

UTS results

UTS (T1XXL) – Offload approach



AsyncSHMEM integration improves computation-communication overlap, scalability

Target Class of Problems

Streaming, dynamic graphs with edge and vertex additions/removals.

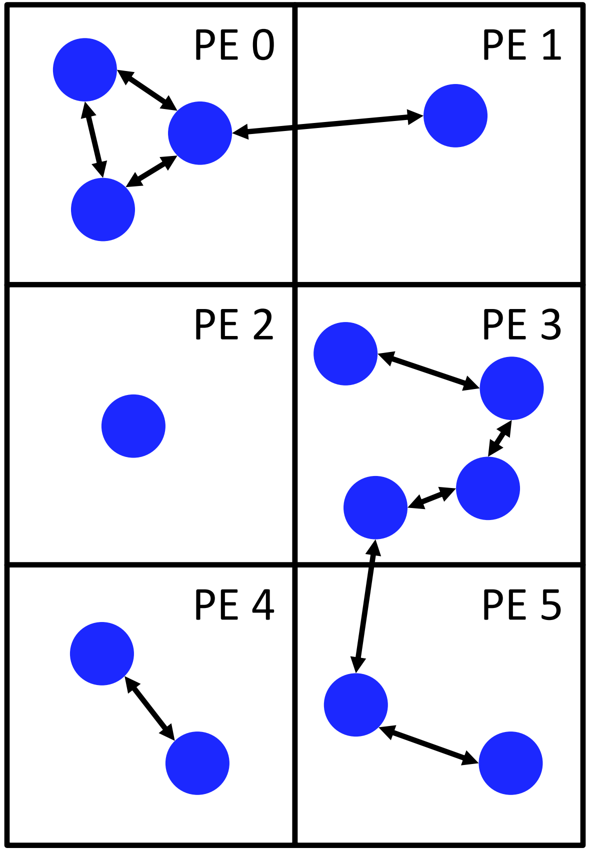
Graph is naturally partitioned across PEs.

- Partitions of graph in different PEs have few cross-PE edges.
- PEs are naturally de-coupled by properties of the graph.

As graph evolves over time, inter-PE connectivity may grow.

Based on connectivity, PEs may choose to begin lockstep execution to enable closer sharing of data.

- May lead to a few islands of PEs, or one global cluster.



HOOVER

First published at OpenSHMEM 2017

Iterative dynamic graph modeling and analysis framework.

- Be able to update/mutate graphs
- Then analyze impact those updates have had on structure or other properties.

C/C++ library built on OpenSHMEM 1.4 – PGAS-by-design.

Emphasis on de-coupled execution – communication is always one-sided and as localized as possible.

Runtime manages all computation and communication.

Users provide callbacks that implement application-specific functionality (similar to other graph frameworks, but better supporting more sophisticated applications).

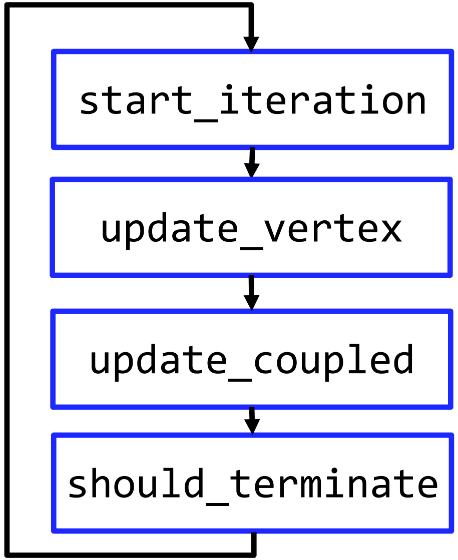
HOOVER sucking up your dynamic data…

HOOVER API Example – Fraud Detection

Vertices represent transactions.

Vertex attributes may be source acct, destination acct, amount, etc.

Edges represent similarities/relations between transactions.




```
start_iteration(vertex_iterator, ctx) {
  Ingest data from external data streams;

  Inject into HOOVER graph as vertices;
}

update_vertex(vertex, neighbors) {
  Identify normative graph patterns from each vertex;
}

update_coupled(vertex_iterator, ctx) {
  Identify anomalies based on global normative patterns;

  Enter lockstep with PEs that share anomalies;
}

should_terminate(vertex_iterator, ctx) {
  Print diagnostics, decide whether to exit;
}
```

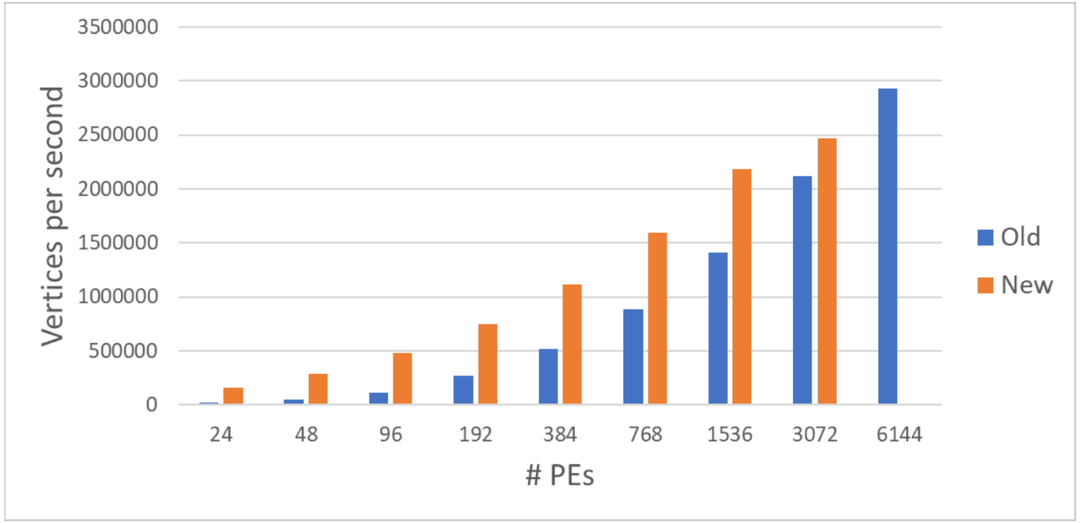
Updates Since OpenSHMEM 2018

Complete refactor of the runtime and programming model.

- >6000 LOC added, >4700 LOC deleted
- Large throughput improvements thanks to runtime refactoring
- Move to iterate-to-convergence programming model
- Port existing applications, benchmarks, and tests to the new model

Contribution of mosquito-borne illness model by Wenbin Liu (SBU).

Extensions and improvements to graph-based anomaly detection application.



Throughput scaling of graph-based anomaly detection on Edison.
Old = October 2018. New = November 2018.

Ongoing Work

Wes Suttle (SBU): Using HOOVER as a use case for exploring fault tolerance in OSSS OpenSHMEM.

Max Grossman (Rice):

- Continued performance improvements
- Experiment with active messages (supported by work by Jack Snyder, Duke University)
- Comparison to other graph modeling frameworks (e.g. GraphX)
- Cross-OpenSHMEM implementation performance comparisons
- Additional application development (Long Distance Leonard Jones)
- Multi-threading support
- GPU support

Conclusions

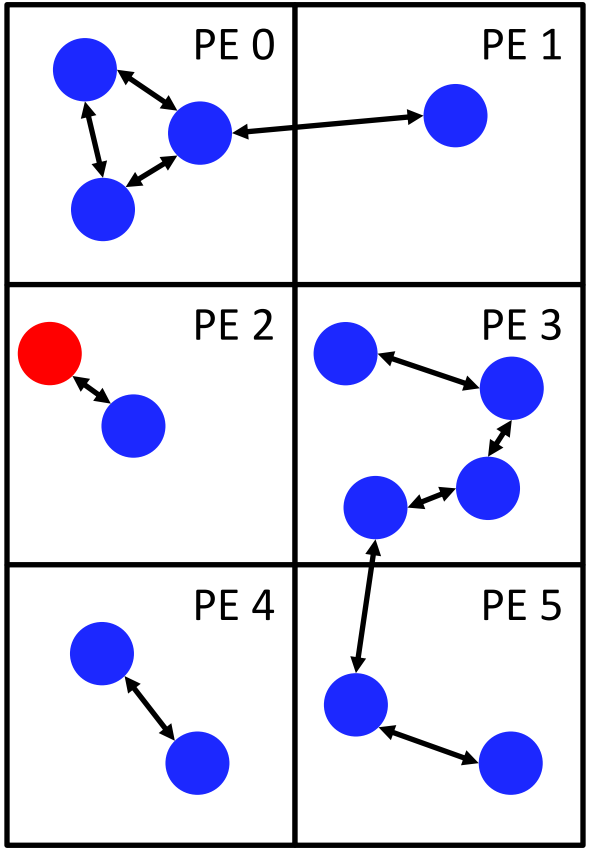
HOOVER: an iterative dynamic graph modeling and analysis framework.

Emphasis on de-synchronized, de-coupled execution – one-sided and PGAS by default.

This adds complexity to the programming model and runtime.

But enables scalability in a way that bulk synchronous models can't.

Github: https://github.com/agrippa/hoover
Contact: max.grossman@rice.edu



Acknowledgements

Los Alamos NATIONAL LABORATORY EST.1943

DEPARTMENT OF DEFENSE UNITED STATES OF AMERICA

RICE

Georgia Tech

Stony Brook University