Updates from GaTech

Max Grossman, Sriraj Paul, Akihiro Hayashi, Vivek Sarkar

Habanero Extreme Scale Software Research Lab
Georgia Institute of Technology

Current Projects in OpenSHMEM

HOOVER: API and Runtime for workloads in dynamic graph modeling and analysis, on top of OpenSHMEM

- Problem domain, data structures, asynchrony
- Personnel: Max Grossman, Vivek Sarkar
- Funding acknowledgement: LANL

NodeJS + HClib: Integration of Javascript and OpenSHMEM

- Programming systems
- Personnel: Sri Raj Paul, Akihiro Hayashi, Vivek Sarkar
- Funding acknowledgement: DoD

Current Projects in OpenSHMEM

HOOVER: API and Runtime for workloads in dynamic graph modeling and analysis, on top of OpenSHMEM

- Problem domain, data structures, asynchrony

NodeJS + HClib: Integration of Javascript and OpenSHMEM

- Programming systems

HOOVER

https://github.com/agrippa/hoover

Iterative dynamic graph modeling/analysis framework.

- Be able to update/mutate graphs
- Then analyze impact those updates have had on the graph.

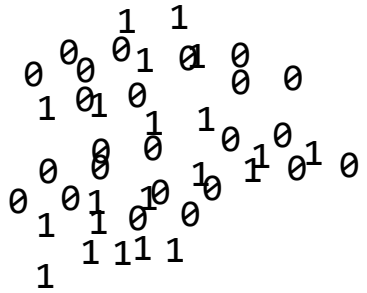
C/C++ library built on OpenSHMEM 1.4 – PGAS-by-design.

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

Users provide callbacks that implement application-specific functionality.

Runtime manages all computation and communication.

Eventually consistent programming model.



HOOVER sucking up your data…

A HOOVER Example

```
void start_iteration(hvr_ctx_t ctx) {
  for (e = 0; e < n_to_add; e++)
    hvr_create_edge(rand_vert_id(), rand_vert_id(),
      BIDIRECTIONAL, ctx);
}

void update_vertex(hvr_vertex_t *vert) {
  uint64_t min_lbl = hvr_vertex_get_uint64(LBL, vert,
    ctx);

  hvr_neighbors_t neighbors;
  hvr_get_neighbors(vert, &neighbors, ctx);

  while (hvr_neighbors_next(&neighbors, &nbr)) {
    uint64_t nbr_lbl = hvr_vertex_get_uint64(LBL, nbr);
    min_lbl = MIN(min_lbl, nbr_lbl);
  }

  hvr_vertex_set_uint64(LBL, min_lbl, vert);
}
```

Example of connected components as a label propagation problem.

`start_iteration`: insert logic at the start of internal runtime iterations.

`update_vertex`: called on every vertex to recompute its state.

`hvr_create_edge`: Create an edge between any two vertices

`hvr_get_neighbors`: Get an iterator over the neighboring vertices.

`hvr_vertex_set/get`: Update/fetch attributes on vertices.

A HOOVER Example

```
int main(int argc, char **argv) {
  shmem_init();

  hvr_ctx_t hvr_ctx;
  hvr_ctx_create(&hvr_ctx);

  for (int v = 0; v < nvertices_per_pe; v++) {
    hvr_vertex_t *vert = hvr_vertex_create(hvr_ctx);

    // Initially each vertex is its own component
    hvr_vertex_set_uint64(LBL, vert->id, vert);
  }

  hvr_init(update_vertex, start_iteration,
    time_limit_s, 1, hvr_ctx);

  hvr_body(hvr_ctx);

  hvr_finalize(hvr_ctx);

  shmem_finalize();
}
```

`hvr_ctx_create`: Allocate a context for the new HOOVER job

`hvr_vertex_create`: Allocate a new vertex.

`hvr_init`: Set up the HOOVER problem by providing callbacks and other parameters.

`hvr_body`: Launch the HOOVER problem.

`hvr_finalize`: Wait for all PEs to complete and clean up runtime resources.

HOOVER Feature Set

The Usual

Callback-based application logic.

Iteration-driven execution – run application-specific logic once per runtime iteration (`start_iteration`).

Data-driven execution – vertices are updated when needed based on changes to neighborhood (`update_vertex`).

Support explicit message passing between vertices (`hvr_send_msg`).

Support creating and deleting vertices and edges.

Support set/get on vertex attributes.

The Unusual

Fully decoupled execution by default.

Ability to force lockstep execution between PEs (`update_coupled`).

PEs may exit the simulation arbitrarily, benefit of PGAS (`should_terminate`).

Support both implicit and explicit edge creation.

Broad diversity of implemented mini-apps and kernels.

HOOVER API

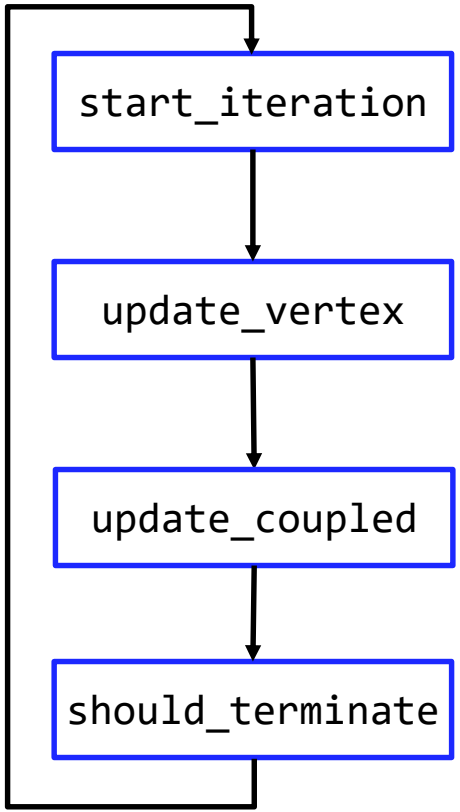
Like other frameworks, callbacks are used to implement application-specific functionality.

`start_iteration`: Hook for logic to be executed at the start of every runtime iteration.

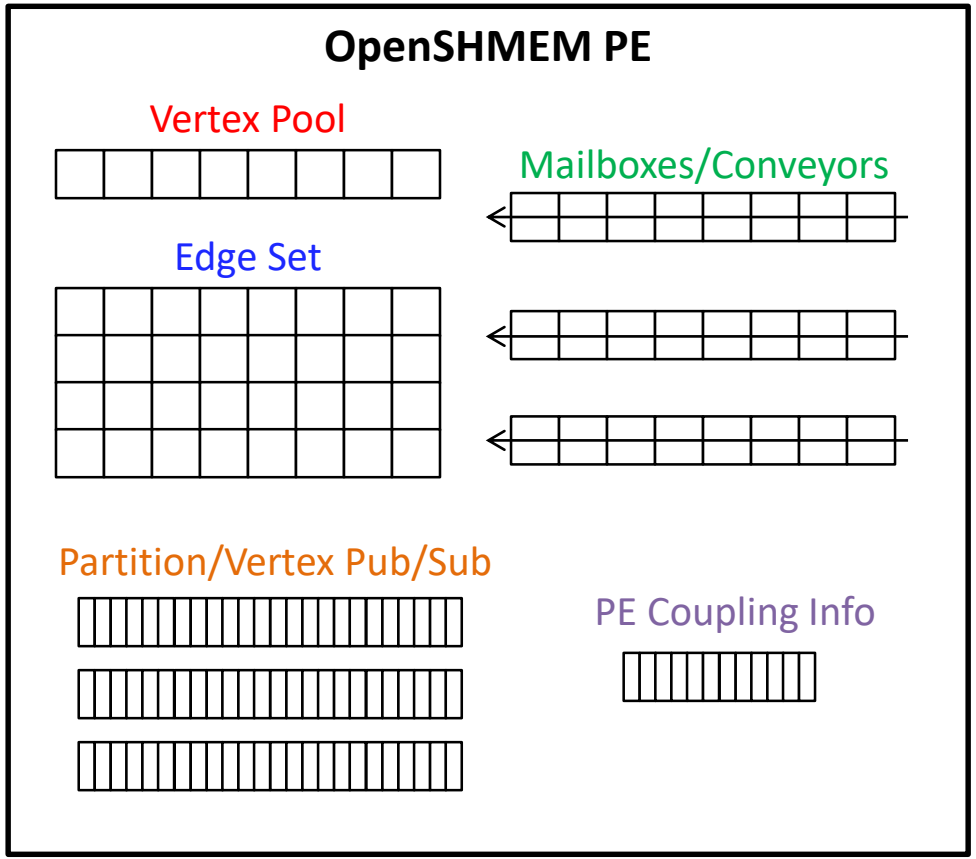
`update_vertex`: Given a vertex and its neighborhood, update its attributes.

`update_coupled:` Update the value shared with coupled PEs.

`should_terminate`: Called at end of iteration, check if this PE will exit.



HOOVER Runtime



Vertex Pool: statically sized pool of all vertices stored on this PE. Includes both locally-owned vertices and mirrored remotely-owned vertices. (hybrid hashmap/linked lists)

Edge Set: CSR matrix storing edge information for each locally-stored vertex.

Mailboxes/Conveyors: Primary inter-PE communication protocol, used to share updates to graph and other metadata.

Partition/Vertex Pub/Sub: Producer-consumer metadata on graph partitions and individual vertices that dictate which PEs receive updates. (bit vectors, AVL trees, etc).

PE Coupling Info: Structures needed to safely handle coupling requests.

Performance Evaluation

Experiments are run on Cori (CraySHMEM 7.7.8) – 1 PE per core (32 PEs per node).

Two streaming graph kernels, measuring performance relative to other frameworks:

- *Update Rates (HORNET, successor to cuSTINGER)*
- *Connected Components (Apache Flink)*
- *Triangle Counting (Apache Flink)*

Two mini-apps, measuring strong scaling of HOOVER:

- *Graph-based anomaly detection* – Stream random vertices into the graph, search for normative patterns, identify anomalies as patterns that are similar but not identical to normative.
- *Community detection* – Clique percolation method

Results should be read with a grain of salt – difficult to perform one-to-one comparisons with HOOVER, and these numbers are all work-in-progress.

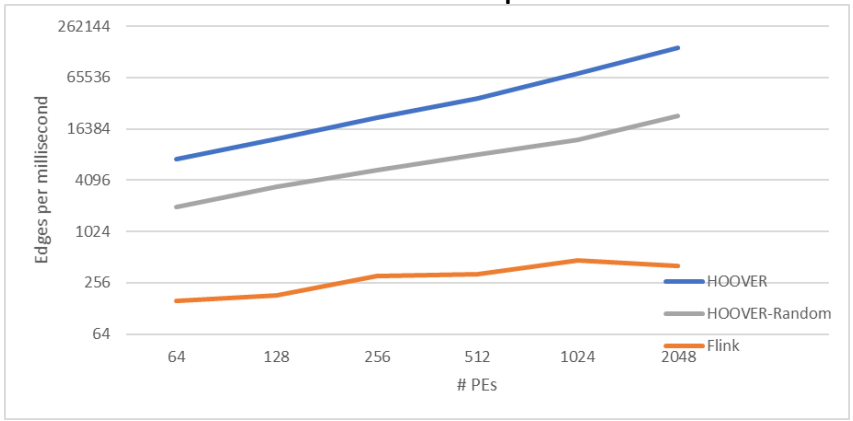
Other implemented applications include infectious disease modeling, n-body simulations, graph convolutional networks, mosquito-borne illness modeling.

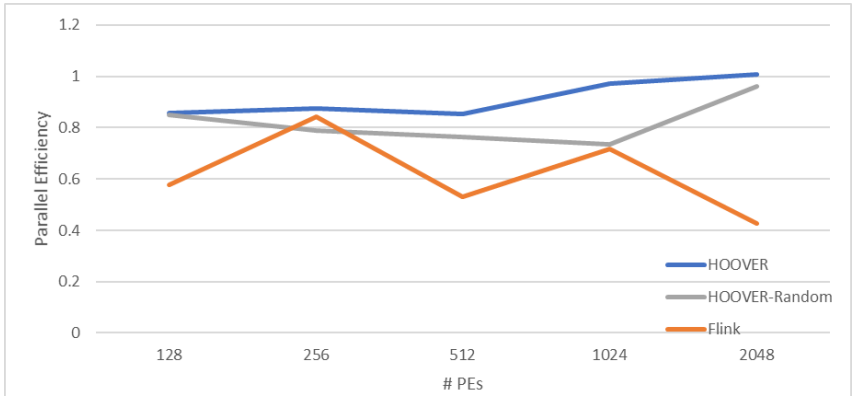
Performance Evaluation

Throughput

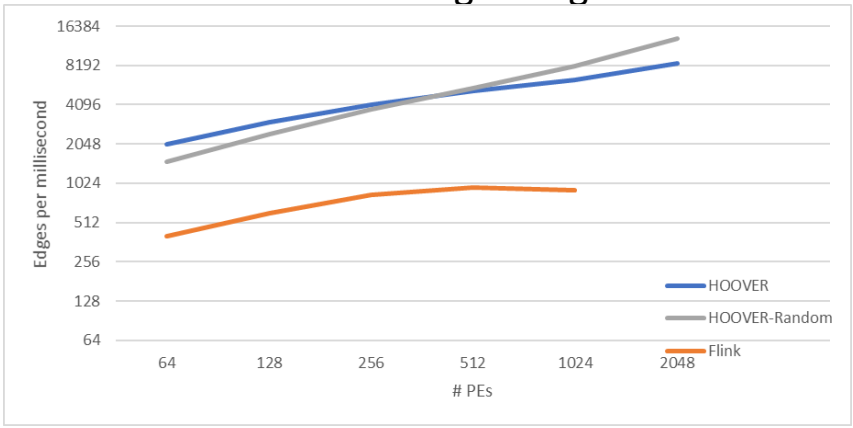
Scaling Efficiency

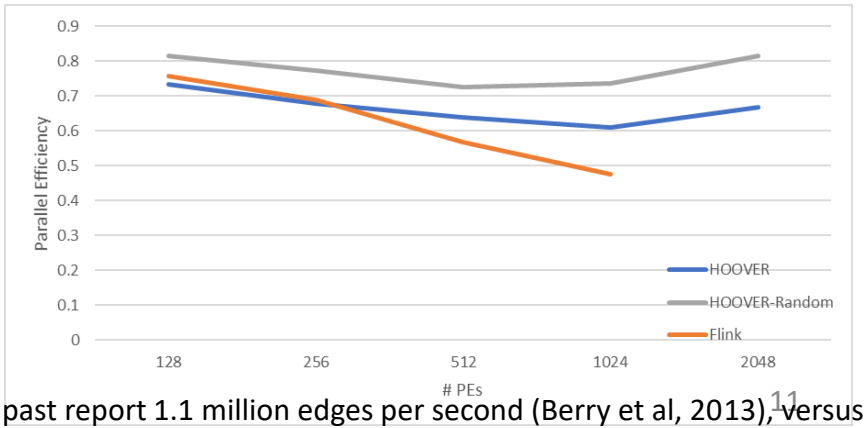
Connected Components





Counting Triangles

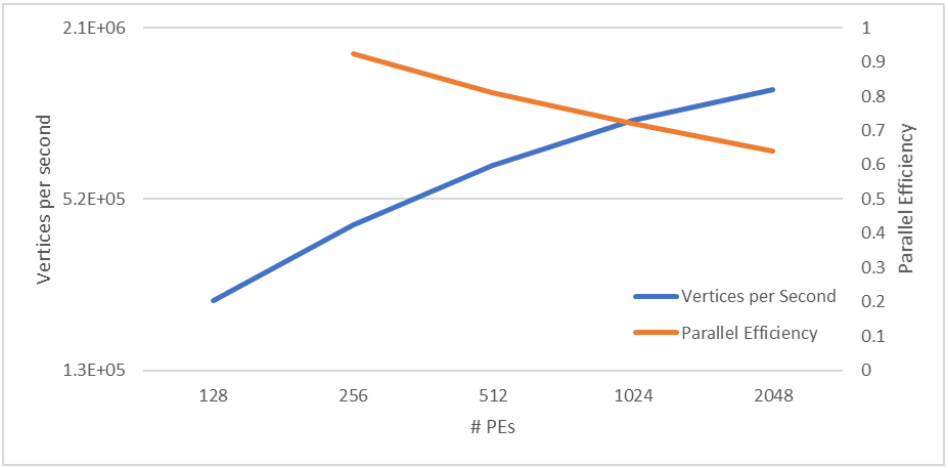




Streaming Connected Components: Similar benchmarking studies in the past report 1.1 million edges per second (Berry et al, 2013), versus 17.7 million in the worst case and 371.5 million edges per second in the best case for HOOVER and scalability beyond that.

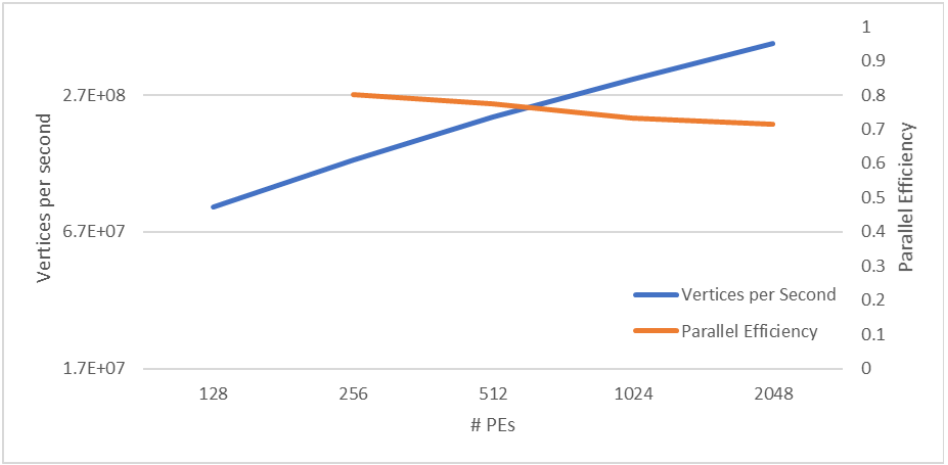
Performance Evaluation

Graph-Based Anomaly Detection



Based on “Mining for Structural Anomalies in Graph-based Data”, William Eberle and Lawrence Holder.

Community Detection



Clique Percolation Method

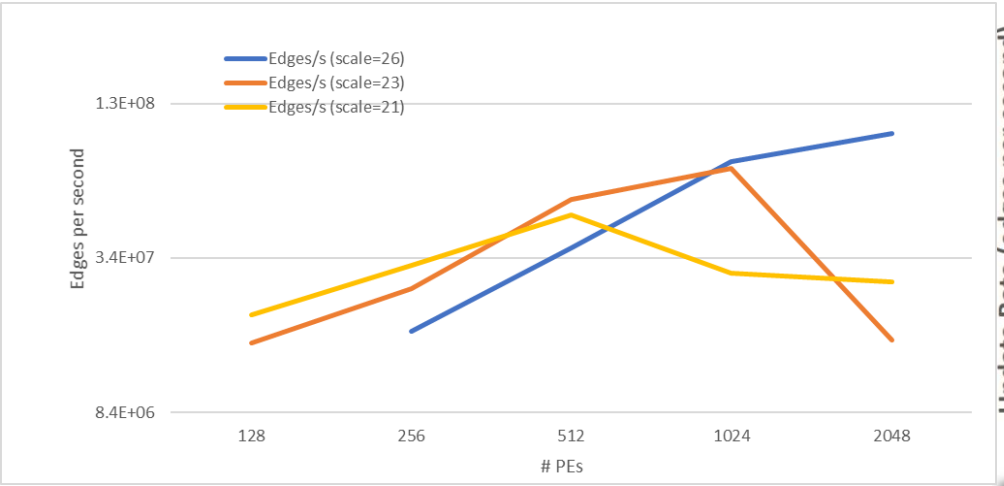
Run each application for a fixed number of seconds and then measure average throughput over all of execution.

Performance Evaluation

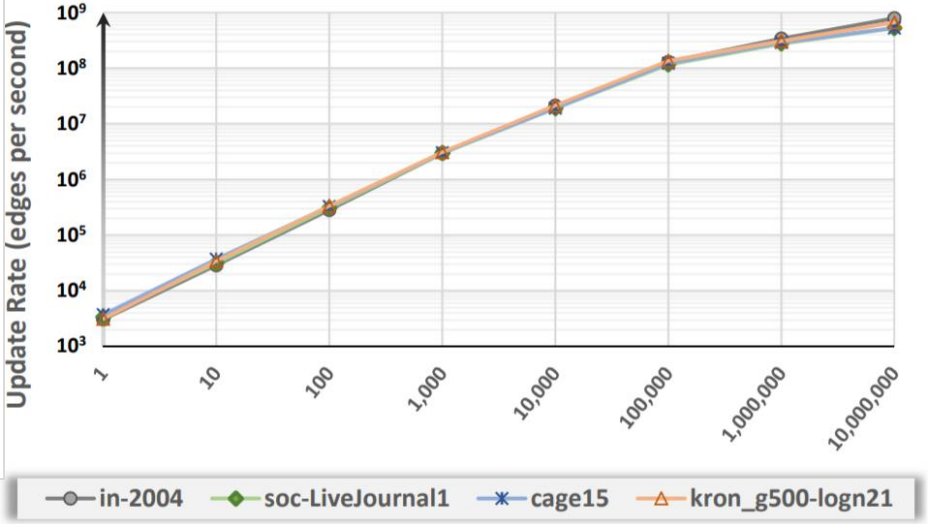
Update Rates benchmark from HORNET paper

Streaming insert of edges for graph construction on Kronecker graph from Graph 500.

A worst case scenario for HOOVER (many, many cross-PE edges requiring communication and causing immediately fully coupled execution).



HOOVER on Cori Haswell



HORNET on an NVIDIA P100

Current Projects in OpenSHMEM

HOOVER: API and Runtime for workloads in dynamic graph modeling and analysis, on top of OpenSHMEM

- Problem domain, data structures, asynchrony

NodeJS + HClib: Integration of Javascript and OpenSHMEM

- Programming systems

NodeJS: Server Side Javascript

Node.js provides a server-side JavaScript runtime environment

Primarily build on top of V8 Javascript Engine

Uses libuv to implement event loop

Event based interface to blocking OS operations

e.g. file, network …

Collaboration between IDA (Jason DeVinney, Bill Carlson) and GaTech (Sri Raj Paul, Akihiro Hayashi).

S.R. Paul, K. Chen, A. Hayashi, M. Grossman, V. Sarkar, J. DeVinney, B. Carlson. “A Unified Runtime for PGAS and Event-Driven Programming”. ESPM2 2018.

IDA

Can We Unify PGAS and Event-Driven Asynchrony

PGAS: Distributed data accesses with semantics of NUMA shared memory systems

Event-driven asynchrony: Very high level event driven language designed for latency tolerance+hiding in web applications (i.e. concurrency, not parallelism)

Can they be unified?

IDA

S.R. Paul, K. Chen, A. Hayashi, M. Grossman, V. Sarkar, J. DeVinney, B. Carlson. “A Unified Runtime for PGAS and Event-Driven Programming”. ESPM2 2018.

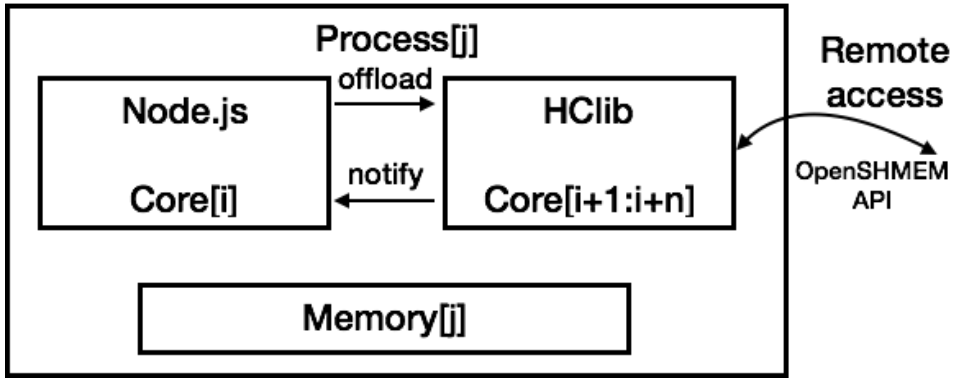
Example Code

```
for (let pe = 0; pe < npes; pe++) {

    for (let i = 0; i < N; i++) {
        long_g_async_promise(arr+i, pe).then(
            function(val) {
                sum += f(val, my_pe());
            });
    }

}
```

S.R. Paul, K. Chen, A. Hayashi, M. Grossman, V. Sarkar, J. DeVinney, B. Carlson. “A Unified Runtime for PGAS and Event-Driven Programming”. ESPM2 2018.

IDA

Implementation: Node.js + Hclib + OpenSHMEM Runtime



Computation worker: Executes Node.js

Offloads communication calls to communication worker

Communication worker: Invokes OpenSHMEM calls

Notifies computation workers when the communication operation finishes

S.R. Paul, K. Chen, A. Hayashi, M. Grossman, V. Sarkar, J. DeVinney, B. Carlson. “A Unified Runtime for PGAS and Event-Driven Programming”. ESPM2 2018.

IDA

Acknowledgements