Scaling to Exascale: Intra- and Inter-node Asynchronous Tasking in OpenSHMEM

- At exascale, asynchronous APIs are key for intra-, inter-node latency hiding
- Task-parallel APIs are a more general way to express parallelism
- This work explores integration of intra- and inter-node task spawning with the existing OpenSHMEM execution model

Related Publications:

In the Third Workshop on OpenSHMEM and Related Technologies (OpenSHMEM 2016), August 2016, Hanover, Maryland, USA:

Max Grossman, Vivek Kumar, Zoran Budimlic, and Vivek Sarkar. *"Integrating Asynchronous Task Parallelism with OpenSHMEM",*

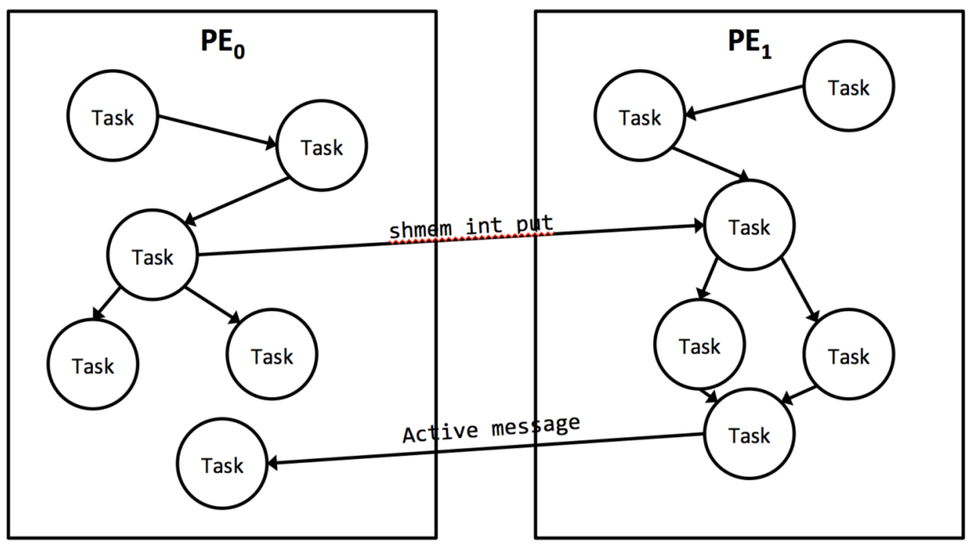
Siddhartha Jana, Tony Curtis, Dounia Khaldi, and Barbara Chapman. *"Increasing Computational Asynchrony in OpenSHMEM with Active Messages"*

Proposed Extensions to the OpenSHMEM Programming Model

Intra-node Tasking

- Support asynchronous tasks, point-to-point task synchronization w/ futures, tasks triggered by remote PUTs

Capability	API
Async. Task Creation	`shmem_task`
Bulk Task Creation	`shmem_parallel_for`
Bulk Task Sync.	`shmem_task_scope_begin/end`
Point-to-point Sync.	`shmem_task_future/await`
Data-driven Tasks	`shmem_task_when`



Abstract execution model. Intra-node dependencies are expressed using futures, inter-node dependencies expressed using RDMA and active messages.

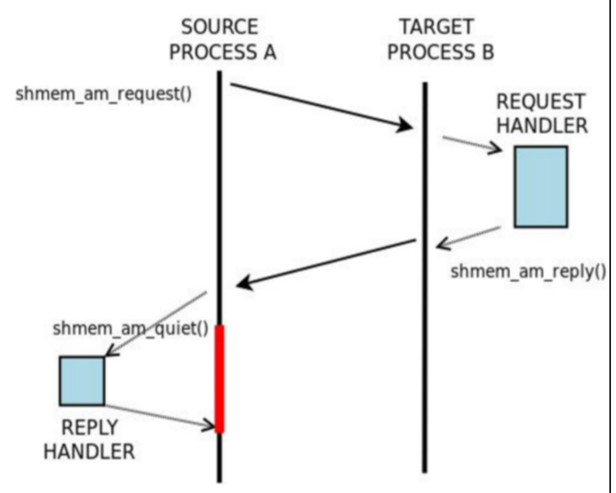
Inter-node Tasking

Challenges in current distributed memory programming models:

- Large data payloads
- Poor computation communication overlap
- Synch. among processes becomes expensive

Data movement becomes a challenge! Need to move computation closer to where the data resides using active messages.

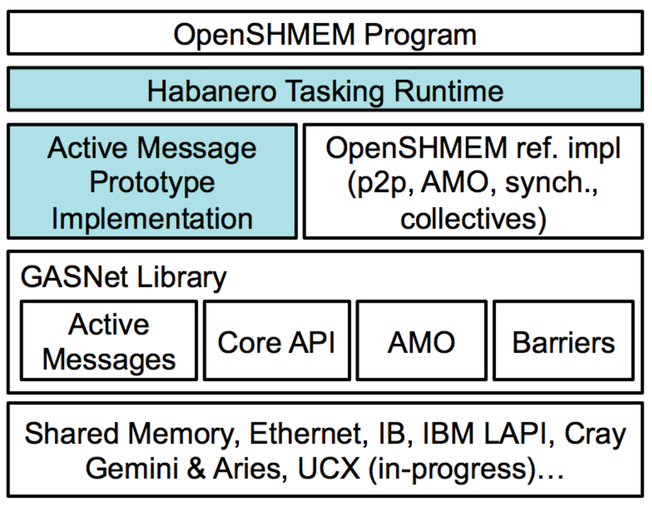
```
/** Handler Function Signature **/
void user_function_name (data_buffer, buffer_size,
source_rank, ...)

/** (De)Registration of Active Message handlers **/
void shmemx_am_attach (handler_id, function_ptr)

/** Initiating Active Messages **/
void shmemx_am_request (target_rank, handler_id,
source_addr, nbytes)
void shmemx_am_reply (handler_id, source_addr,
Nbytes, …)
void shmemx_am_quiet();

/** Handler-safe locking mechanisms **/
void shmemx_am_mutex_lock
(shmemx_am_mutex*)
void shmemx_am_mutex_unlock
(shmemx_am_mutex*)
```



Proposed interface and program control flow

Prototype Integration

- Extends OpenSHMEM reference implementation with AM APIs, coordinate communication/computation using Habanero runtime
  - Re-use GASNet's existing AM interface
- Open source releases:
  - OpenSHMEM Active Message extensions: https://github.com/openshmem-org/openshmem-am
  - Habanero-OSHMEM Integration: https://github.com/habanero-rice/hclib/tree/resource_workers

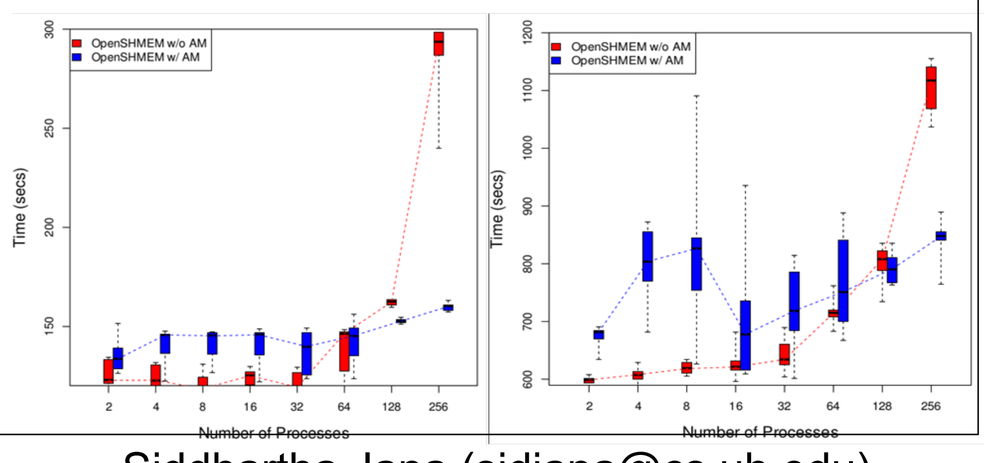


Prototype Evaluation

- Evaluated OpenSHMEM AM extensions using OpenSHMEM version of the Traveling Salesman Problem with and without Active Messages
- Proposed Interface provides locking mechanisms to ensure exclusive access to shared data structures. Its use must be minimal to avoid lock contention overhead (as seen in the plot alongside, for the low PE count)
- For evaluation of intra-node tasking extensions, see the PGAS poster titled, "Experiences Developing Regular and Irregular Applications on OpenSHMEM".

Number of cities = 14
Input data size = 196

Number of cities = 15
Input data size = 225



Max Grossman (jmg3@rice.edu),
Vivek Kumar, Zoran Budimlic, Vivek Sarkar

Howard Pritchard,
Jeff Kuehn

Siddhartha Jana (sidjana@cs.uh.edu),
Tony Curtis, Dounia Khaldi, Barbara Chapman