Increasing Computational Asynchrony in OpenSHMEM with Active Messages

Siddhartha Jana (Sid), Tony Curtis,
Dounia Khaldi, Barbara Chapman

August 2, 2016
Baltimore, MD

Outline

- Challenges in current distributed memory programming models
- Active Messages as a means to introduce computational asynchrony
- Potential applications and past implementations
- Proposed interface for OpenSHMEM
- Prototype implementation
- Prototype evaluation
- Case study
- Lessons learned and future work

Challenges in current distributed memory programming models

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

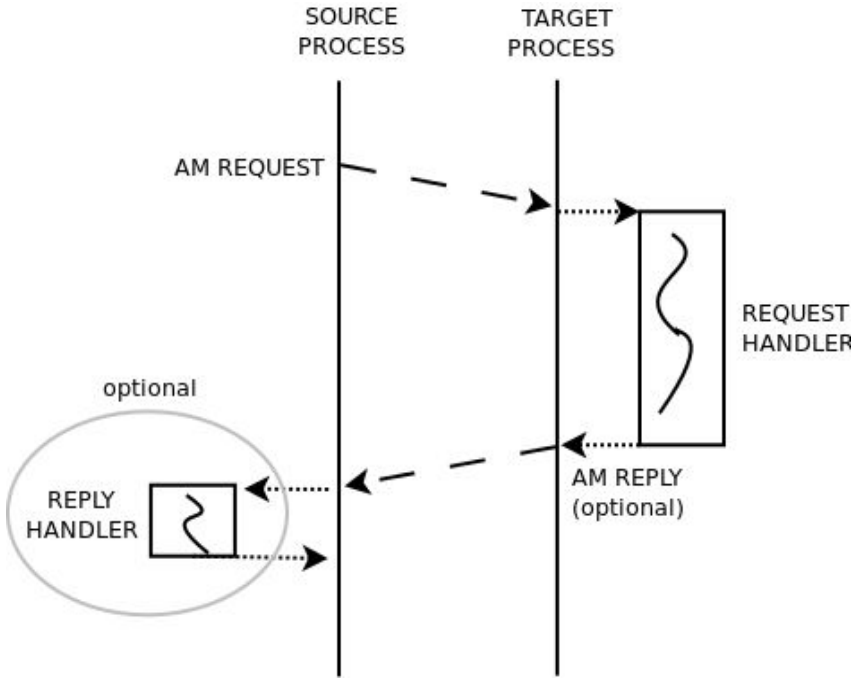
Data movement becomes a challenge
There is a need to move computation closer to data

ASCR report by US DOE Exascale workshop states:
"... The increased variation in execution speed of various components, due to error recovery and power management, will require codes that are more tolerant to noise; hence, more asynchronous."

** Amarasinghe, S., Hall, M., Lathin, R., Pingarli, K., Quinlan, D., Sarkar, V., Shalf, J., Lucas, R., Yelick, K.: ASCR Programming Challenges for Exascale Computing (2011)

Active Messages (AM)

- Original proposal: T. von Eicken et al., 1992 **
- Launch tasks on remote processes
- Single AM request =
  (handler id) + (target process id) + (optional data payload)
- Minimum code injection overhead
- The source process does not have to wait for the remote handler to complete execution.



** Eicken et al., “Active Messages: a Mechanism for Integrated Communication and Computation”, 1992

Potential Applications and Implementations

Applications that stand to benefit:

- Design characteristics:
  - Computation on large data structures
  - Computation over remotely inaccessible data structures

- Communication patterns:
  - Parallel graph computations
  - Irregular data accesses (halo exchanges)

Existing approaches:

- Low-level libraries: GASNet (LBNL), OpenUCX, LAPI (IBM), PAMI (IBM).
- Explicit launching of computation units: ParalleX (parcels), UPC++ (function shipping), Charm++ (entry methods), Chapel (begin-at), CAF 2.0 (spawn)
- Dynamic work-scheduling of tasks among processes: X10, AsyncSHMEM (Max's talk), Titanium, Chapel, Habanero-Java, Habanero-C, Habanero-UPC, Habanero-UPC++

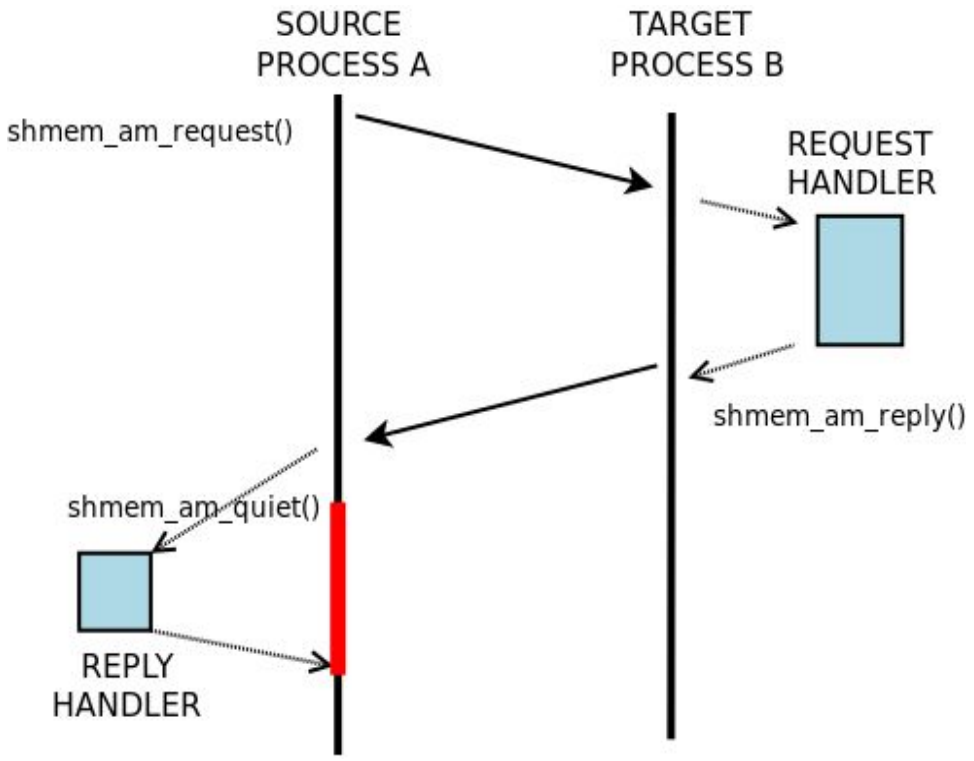
Introducing Active Messages within OpenSHMEM

/** 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)

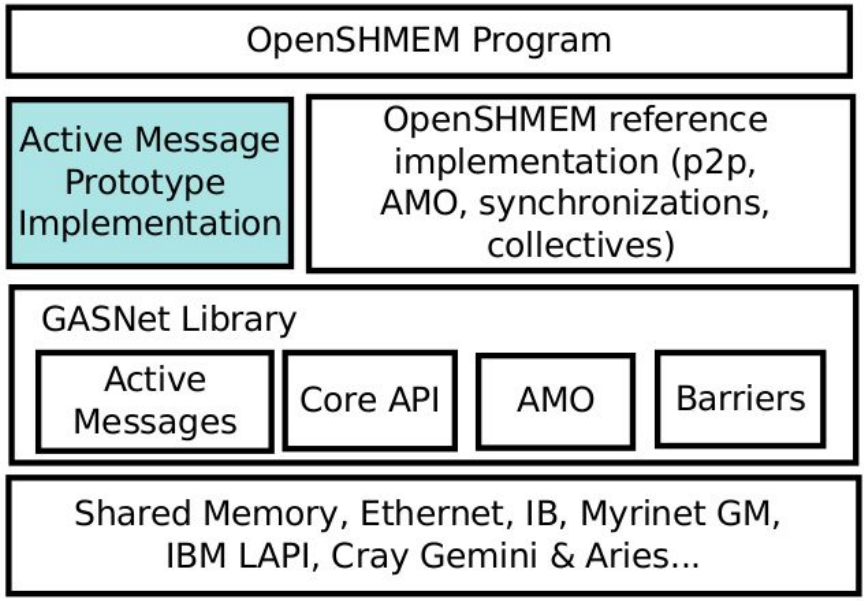
/** Initiating Active Messages **/
void **shmemx_am_request** (target_rank, handler_id, source_addr, nbytes)
void **shmemx_am_reply** (handler_id, source_addr, Nbytes, …)
void **shmemx_am_quiet**();

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



Prototype implementation

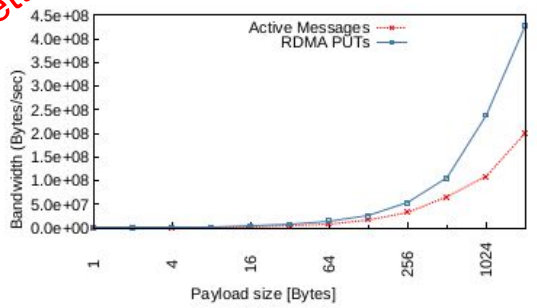
- Implemented within the OpenSHMEM reference implementation
- Reuses GASNet's existing Active Messaging interface
- Currently hosted at github.com/openshmem-org/openshmem-am
- Microbenchmark suite at github.com/openshmem-org/openshmem-am-testsuite.git



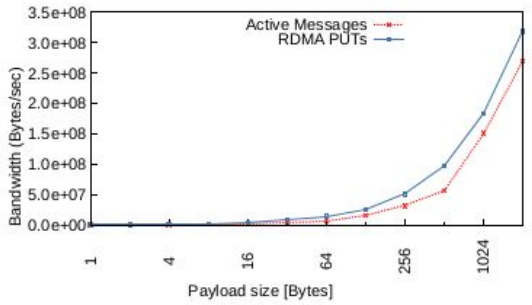
Active Messages is not a replacement of point-to-point RMA

Bandwidth and Message Rate behavior

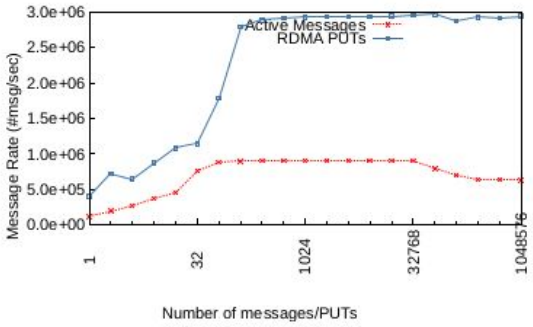
Higher is better



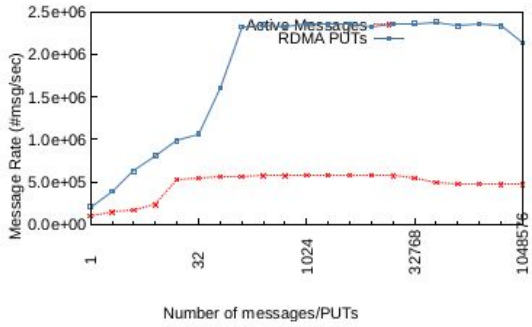
(a) Inter-node Unidirectional Bandwidth (bytes/sec)



(b) Inter-node Bidirectional Bandwidth (bytes/sec)



(c) Inter-node Unidirectional Message Rate (msg/sec)

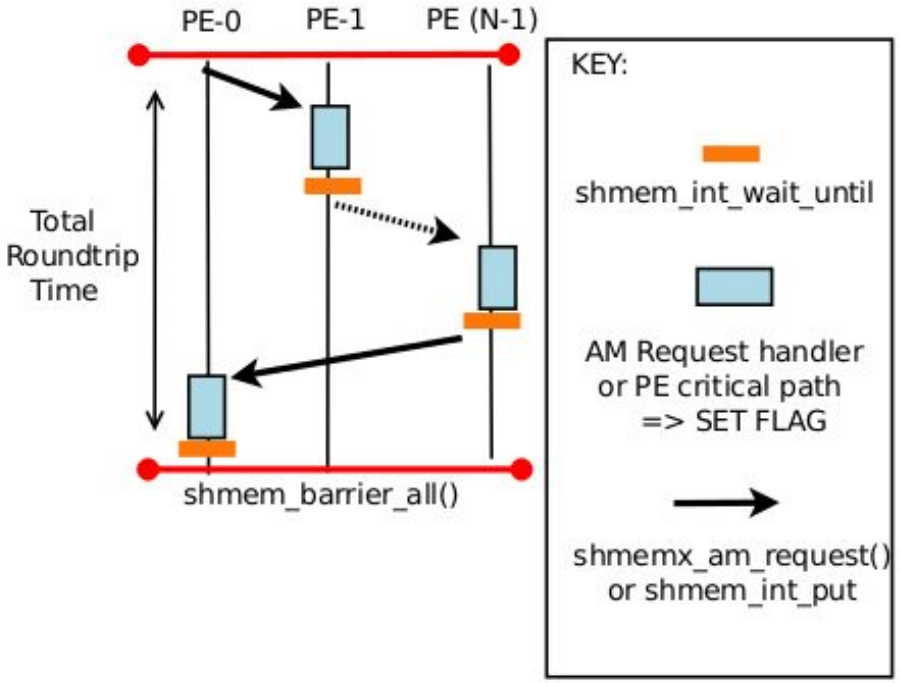


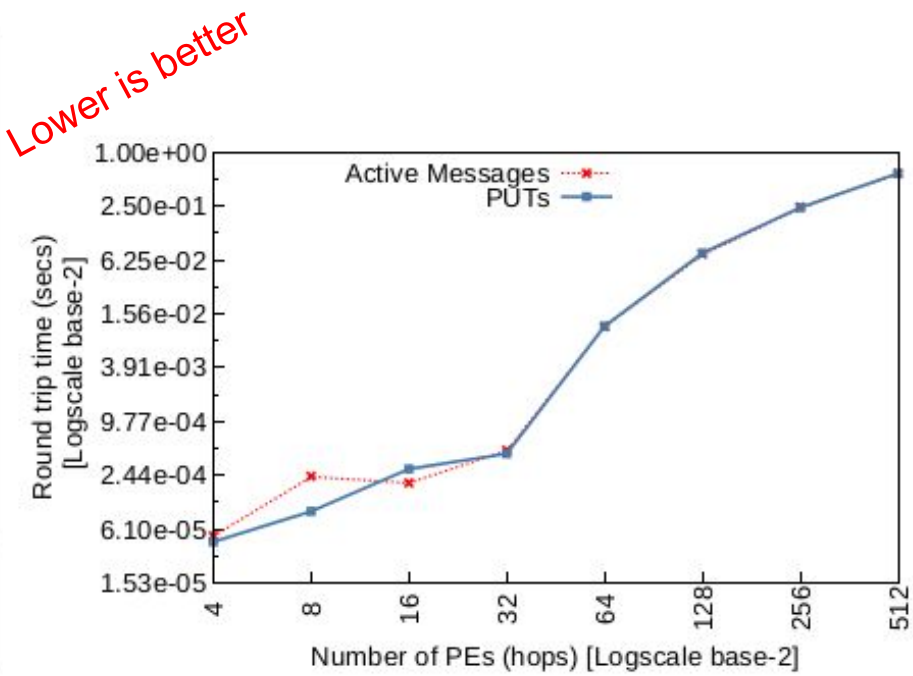
(d) Inter-node Bidirectional Message Rate (msg/sec)

Takeaway:

- Closer-to-metal performance achievable with standard OpenSHMEM point-to-point operations

Token-ring latency of the prototype implementation





Takeaway:

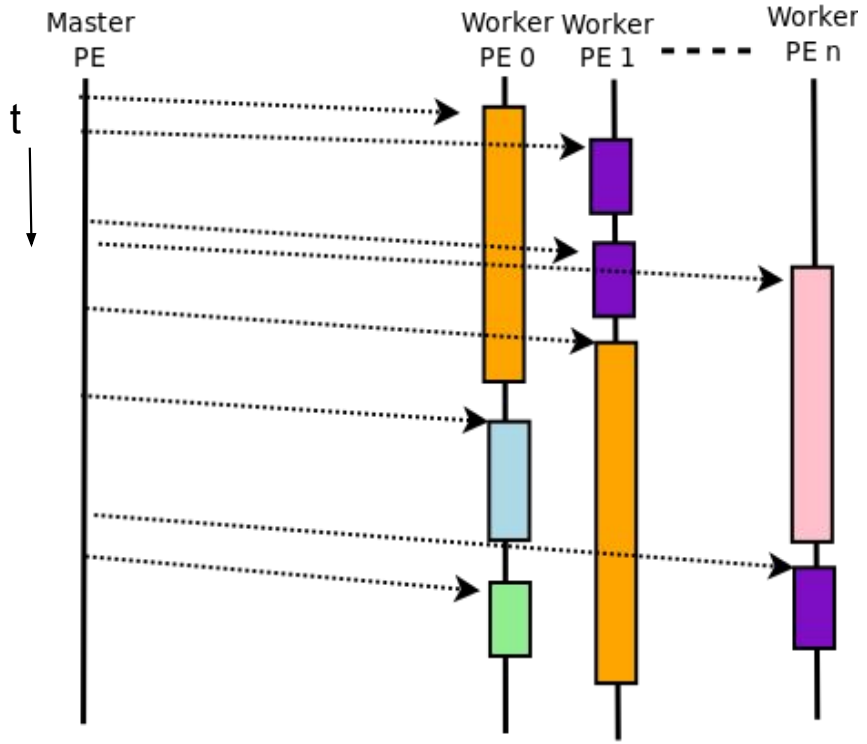
- Better suited for triggering specific events on remote PEs
- Better productivity with no significant loss in performance.

Launching Asynchronous Tasks

Varying Workloads and Minimal Synchronization

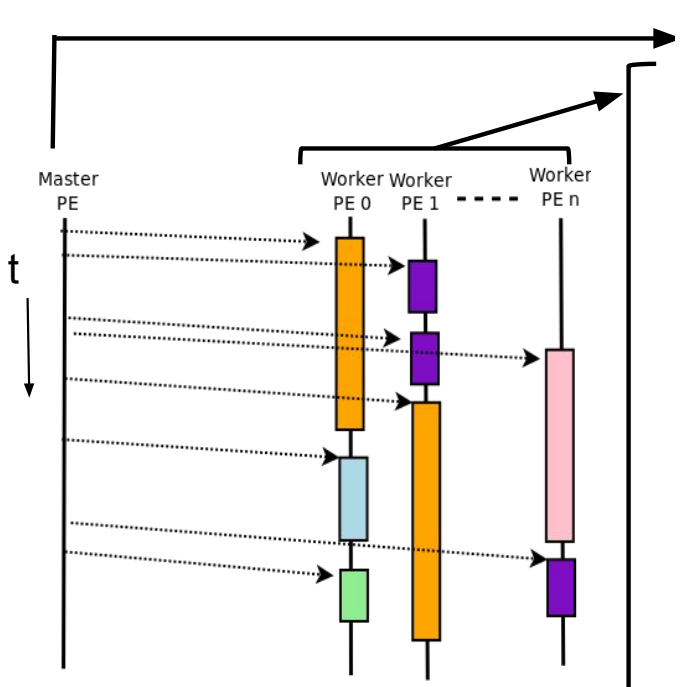
Master-worker communication in Graph based applications like:

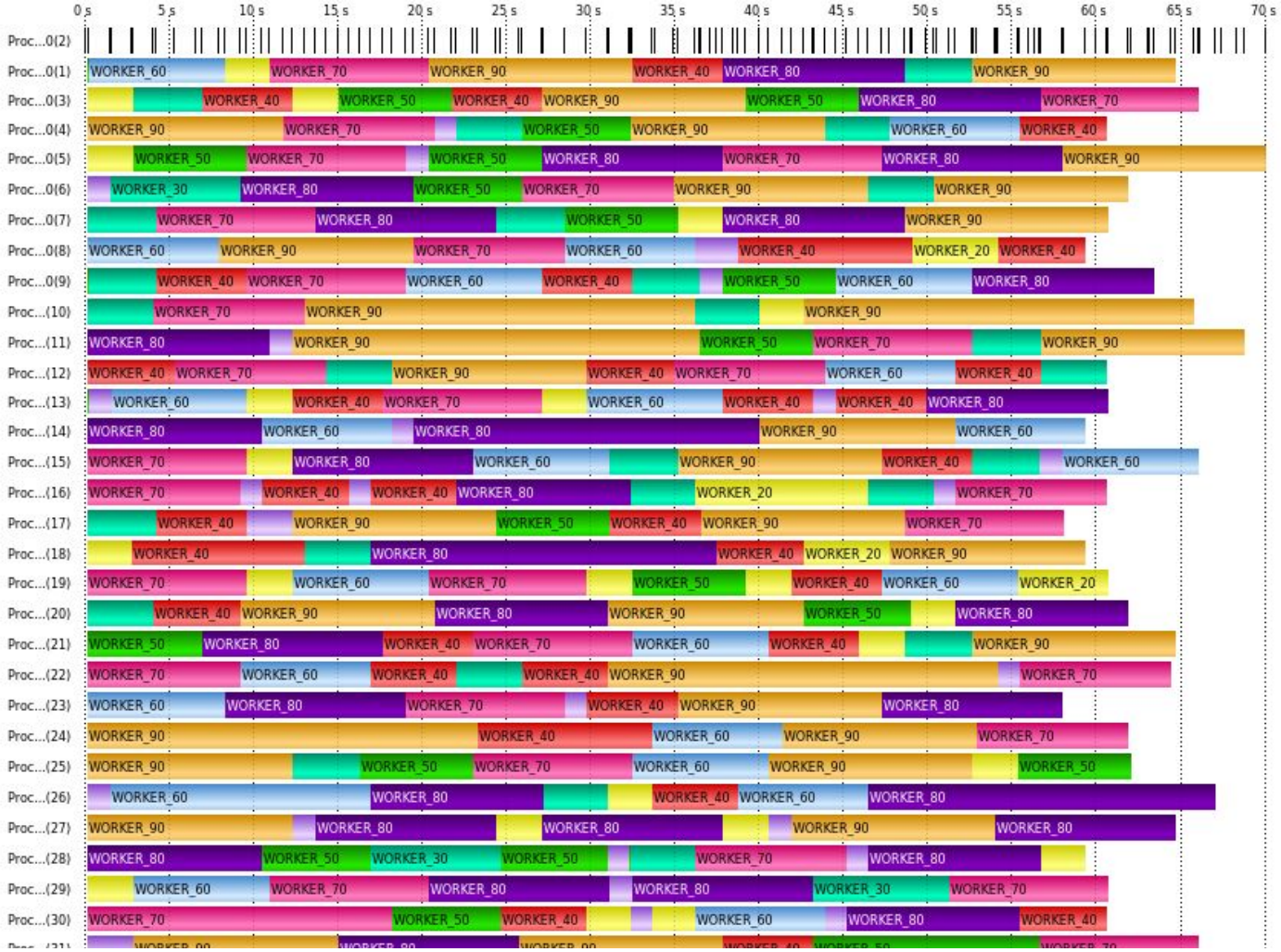
- Travelling Salesman Problem (shortest path)
- Minimum Spanning Tree



Launching Asynchronous Tasks

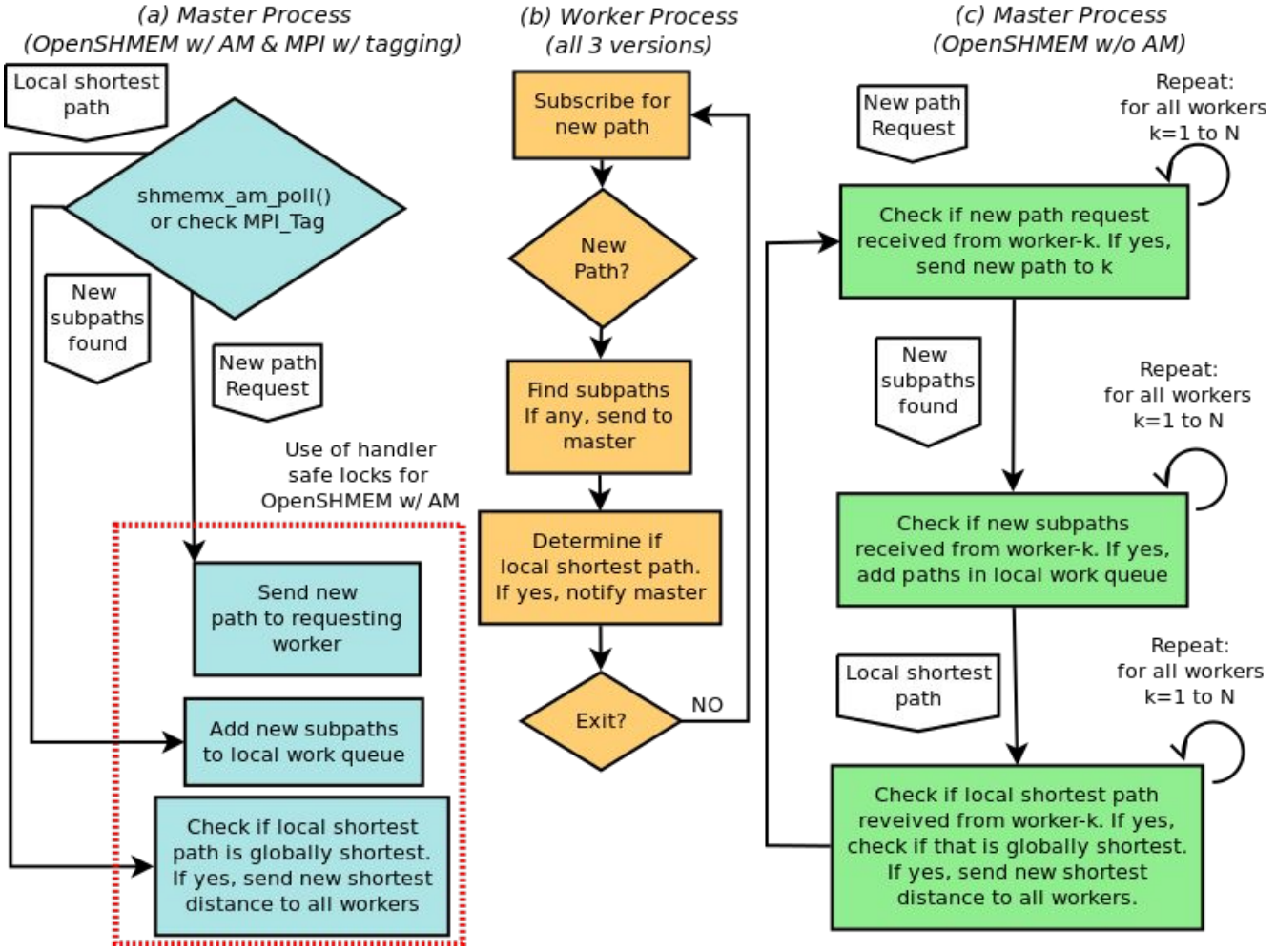
Visualization of Asynchrony using Vampir





Case study: Traveling Salesman Problem (TSP)

Flow diagrams of OpenSHMEM Versions with and without Active Messages



A closer look at TSP using Active Messages

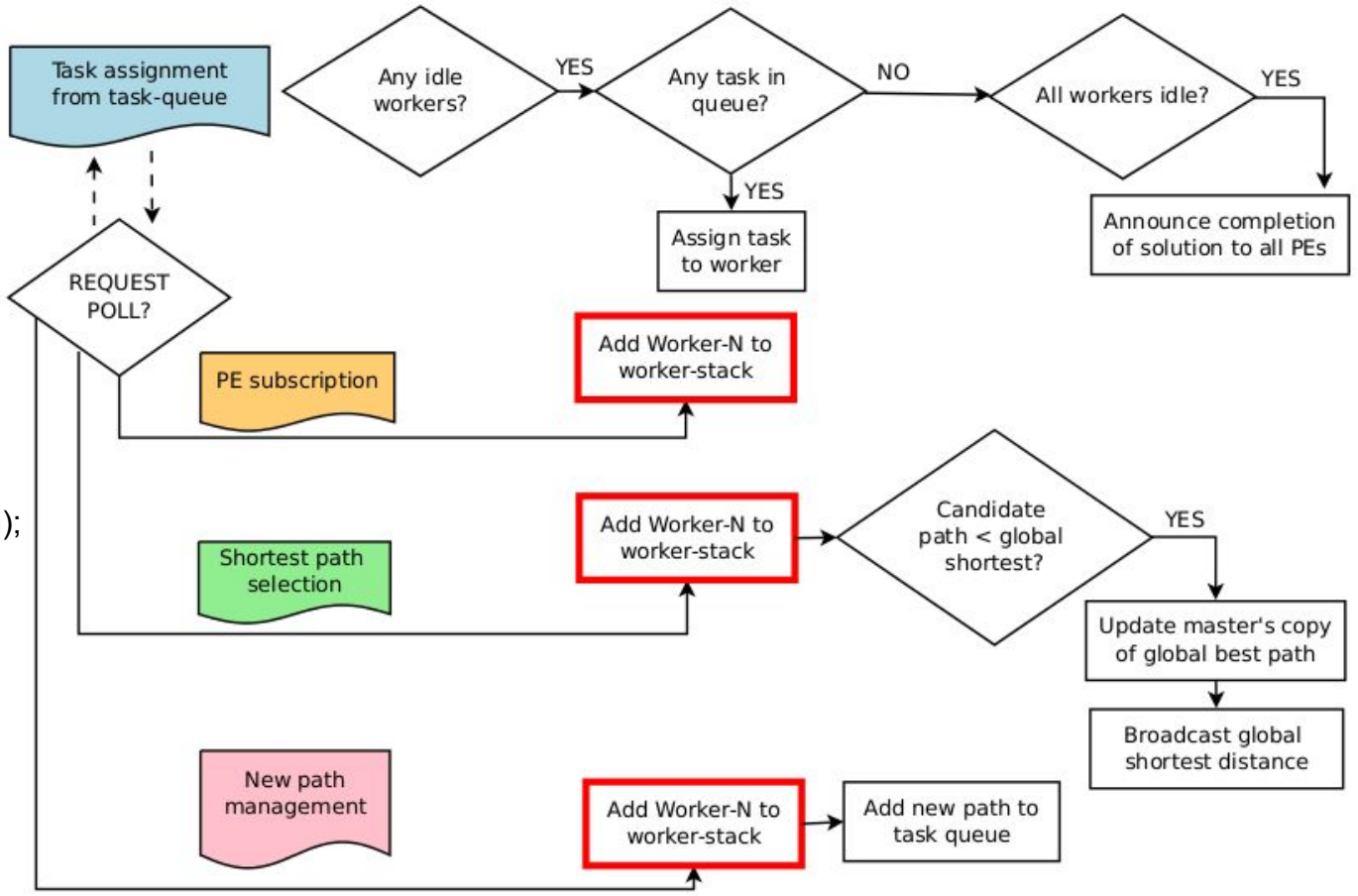
Using handler-safe locks

```
void handler_subscribe()
{...}

void handler_shortestpath()
{...}

void handler_newpath()
{...}

int main()
{
 shmem_init();
 shmemx_am_attach(..&handler_subscribe);
 shmemx_am_attach(..&handler_shortestpath);
 shmemx_am_attach(..&handler_newpath);
 shmemx_am_mutex_init(...);
 while(1) {
    if (!assign_task())
       break;
    if(shortestpath)
       updatepath().
 }
  …. /* cleanup */ ....
}
```

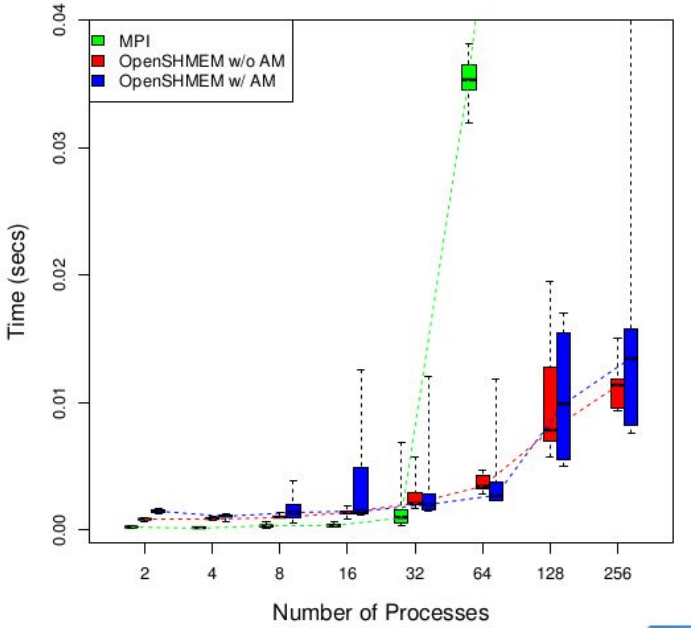


Case study: Traveling Salesman Problem

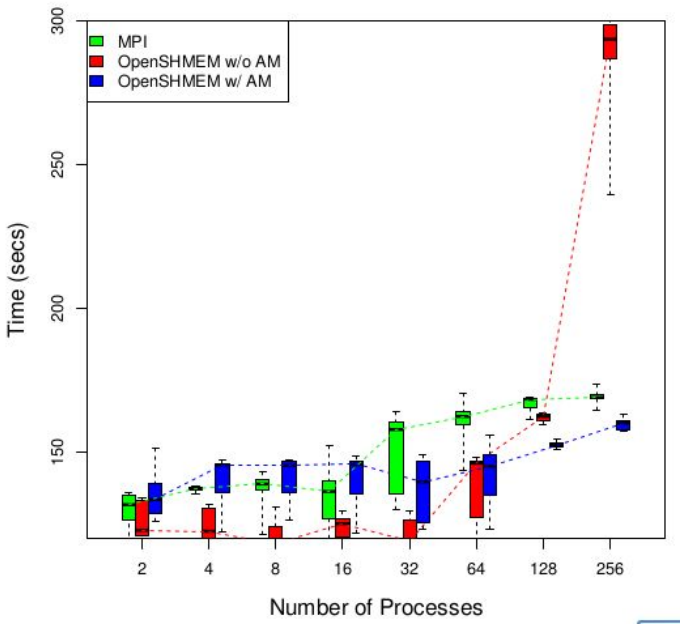
Comparison between OpenSHMEM versions with and without Active Messages

Lower is better

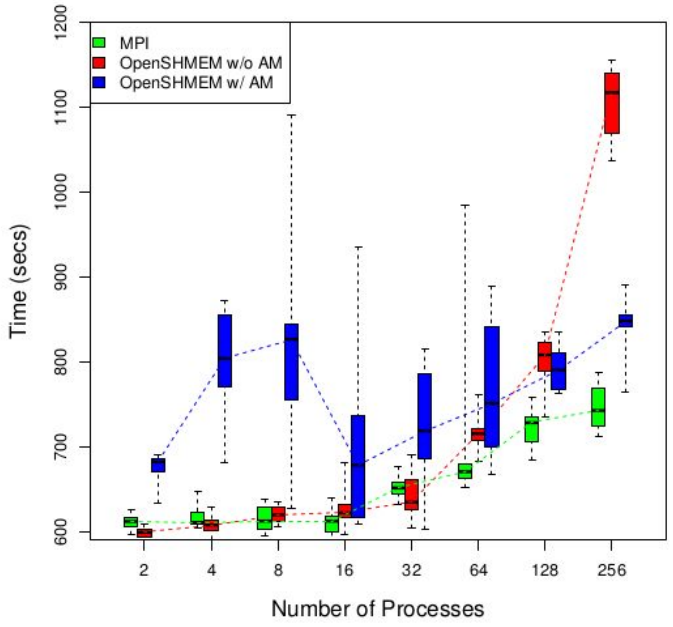
Input data size = 16



Input data size = 196



Input data size = 225



Takeaway:

- Use of locking mechanisms take a toll on performance for small input data size and PE count.
- Avoid designs with multiple handlers sharing same data structures

Lessons learned: Pros and cons

Pros:

- Asynchronous execution w.r.t. source – fire and forget
- Minimal synchronization overhead at the source
- Helps injection of codepath in remote processes

Cons:

- Sharing data structures may require locking mechanisms
- Inefficient for small number of cores and problem sizes

Future Work

- Initiate a redmine ticket and launch discussion within the OpenSHMEM community
- Identify different types of Active Message requests based on completion semantics
- Handler can either run independent of the remote process or within an interrupt handler (depending on the implementation)
- Collective operations like broadcast operations over an active set
- Explore more irregular application benchmarks are a good fit

Increasing Computational Asynchrony in OpenSHMEM with Active Messages

Acknowledgements

- Funding support by DOD subcontract# 346023
- Research support by LANL
- Compute resources support by NSF contract# CRI-0958464

Contacts:

- Sid Jana sidjana@cs.uh.edu
- Tony Curtis anthony.curtis@stonybrook.edu
- Dounia Khaldi dounia.khaldi@stonybrook.edu
- Barbara Chapman barbara.chapman@stonybrook.edu