The OpenSHMEM Programming Library

Md Abdullah Shahneous Bari[1], Delafrouz Mirfendereski[1], Tony Curtis[1] and Dr. Barbara Chapman (PI)[1,2]
[1]Exascale Programming Model Laboratory, Stony Brook University
(https://you.stonybrook.edu/exascallab/)
[2]Brookhaven National Lab

http://www.openshmem.org/

OpenSHMEM

INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

Experimental Feature

Eureka Programming Model

Motivation

- A wide range of problems, such as combinatorial optimization, constraint satisfaction, image matching, genetic sequence similarity, iterative optimization methods, can be reduced to tree or graph search problems.
- Algorithms for these problems employs a common pattern, a eureka event.
  - A point in the program which announces that a result has been found.
- Reduces computation time by avoiding further exploration of a solution
- How can we make the eureka method available efficiently in a parallel programming model?

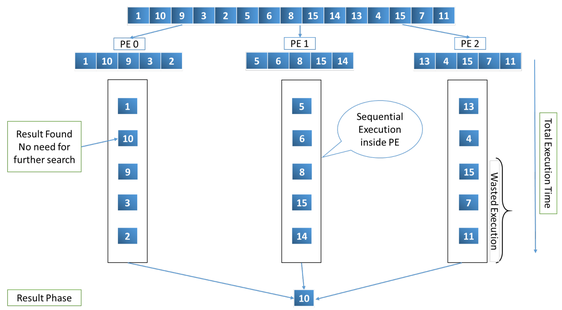

Figure 1. Workflow of a simple parallel search algorithm, searching for 10 in an array using 3 PEs [using traditional programming model]

Eureka Framework

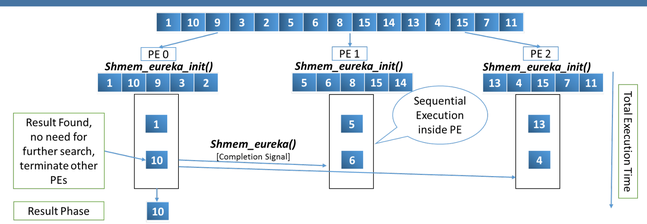

Figure 2. Workflow of a simple parallel search algorithm, searching for 10 in an array using 3 PE [using Eureka Framework]

- Provides API to handle eureka events efficiently
- Allows a worker that encountered eureka event to send early completion signal to other workers
- Improves performance by avoiding unnecessary computation
- Implemented as an extension to the OpenSHMEM library
- API description
  - ***shmemx_eureka_init():*** Initializes a eureka region
  - ***shmemx_eureka():*** Signals a eureka event, which in turn sends completion signals to other processing elements (PEs)

OpenSHMEM

Motivation

- Maintaining performance at scale for applications with irregular and dynamic communication patterns (e.g. irregular graph, multi dimensional fast Fourier transform) using traditional distributed programming models such as MPI is challenging [2]
- The Partitioned Global Address Space (PGAS) programming models present an alternative approach to improve programmability.
- The lower overhead in one-sided communication and the global view of data in PGAS models have the potential to increase the performance at scale.
- OpenSHMEM is a standardized PGAS library

About

- PGAS Parallel Programming Library
- SPMD (Single Program Multiple Data) programming model
- Provides global view of memory in a distributed programming environment
- One sided communication compared to traditional two sided communication (e.g. MPI)
- Work is divided into multiple processes known as processing elements (PEs)
- OpenSHMEM routines supply remote data transfer, work-shared broadcast and reduction, barrier synchronization, and atomic memory operations across PEs
- For communication OpenSHMEM uses RDMA (Remote Direct Memory Access) **1-sided** put/get instead of matched pairs (e.g. MPI send/recv)
- Less communication overhead
- OpenSHMEM provides specification for a standardized API and an execution model
- Exascale Programming Model Laboratory at Stony Brook University is responsible for the reference implementation [1]
- Other implementations also available:
  - SGI's SHMEM
  - Sandia OpenSHMEM
  - Mellanox ScalableSHMEM
  - MVAPICH2-X includes OpenSHMEM
  - OpenSHMEM in Open-MPI and
  - OpenSHMEM over MPI-3

Experimental Feature

Multi-threading

Motivation

- Although all the PEs work in parallel to solve the same problem, each PE runs in a single-threaded mode.
- There is more support for parallelism in hardware and clusters today.
- Nested parallelism can result in better use of resources.
- It will also provide a more flexible API for users.

It is possible to start multiple threads in a PE in OpenSHMEM using known models such as OpenMP, but to better address the above issues it has been suggested to add built-in support for multi-threading inside PEs.

Design and Implementation

The community recommended API: It has been suggested to use ***shmemx_init_thread()*** instead of ***shmem_init()*** which allows PEs to start in different modes regarding threads. These modes define whether the PEs run in one or more threads, as well as the communication pattern between threads of a PE with other PEs.

Our attempt to implement: Here we are trying to take an almost similar approach, the only difference is we want to allow each PE to enter a multi-threaded region independent from other PEs.

- ***void shmemx_thread_init(void(*f)(void*), void *args, int num_th):*** the calling PE would start the multi-threading mode with `threads` calling **`f(args)`**.
- ***void shmemx_thread_finalize(void):*** After calling this function inside some or all the PEs, they will continue running in single-threaded mode.

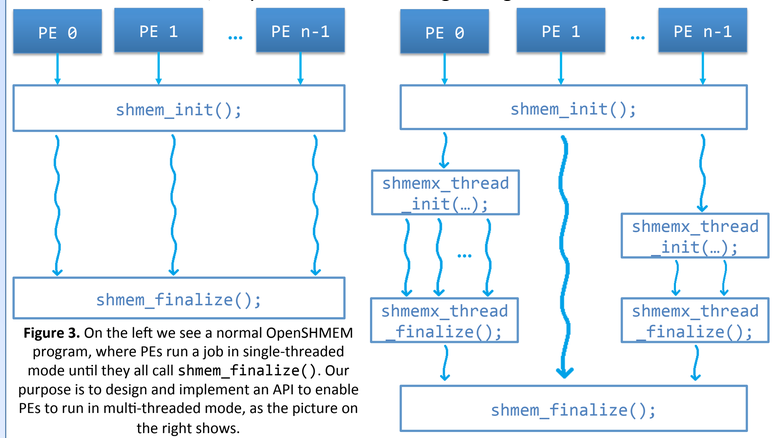

Figure 3. On the left we see a normal OpenSHMEM program, where PEs run a job in single-threaded mode until they all call `shmem_finalize()`. Our purpose is to design and implement an API to enable PEs to run in multi-threaded mode, as the picture on the right shows.

Future Work:

- How to handle the results of calling **`f(args)`**.
- How to enable threads to communicate.
- What happens when a thread calls a function that affects other PEs.
- On-going discussion about different modes of multi-threading.

Interested? Contact

Md Abdullah Shahneous Bari
Exascale Programming Model Laboratory
Email: *mshahneousbari@cs.stonybrook.edu*

Delafrouz Mirfendereski
Exascale Programming Model Laboratory
Email: *dmirfenderes@cs.stonybrook.edu*

Tony Curtis (Project Lead)
Exascale Programming Model Laboratory
Email: *anthony.curtis@stonybrook.edu*

Sponsors

Los Alamos NATIONAL LABORATORY EST.1943

U.S. DEPARTMENT OF ENERGY

References

1. Reference Implementation [https://github.com/openshmem-org/openshmem/]
2. Basumallik, A., Eigenmann, R.: Optimizing Irregular Shared-memory Applications for Distributed-memory Systems. In: Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2006 (2006)