A Portable Implementation of Partitioned Point-to-Point Communication Primitives

Purushotham Bangalore, Andrew Worley, Derek Schafer
Ryan Grant, Anthony Skjellum, Sheikh Ghafoor

Tennessee TECH
THE UNIVERSITY OF ALABAMA AT BIRMINGHAM
Sandia National Laboratories
THE UNIVERSITY OF TENNESSEE AT CHATTANOOGA

Motivation

Partitioned communication:

- Is a recent addition to the MPI 4.0 standard
- Has few, if any, public implementations
- Could have large impact on multi-threaded programs

Goals

We developed a portable library that:

- Implements the partitioned communication API
- Works on top of existing MPI implementations
- Is designed to be used as a early model for potential test applications and algorithms
- Builds experience with the new communication model

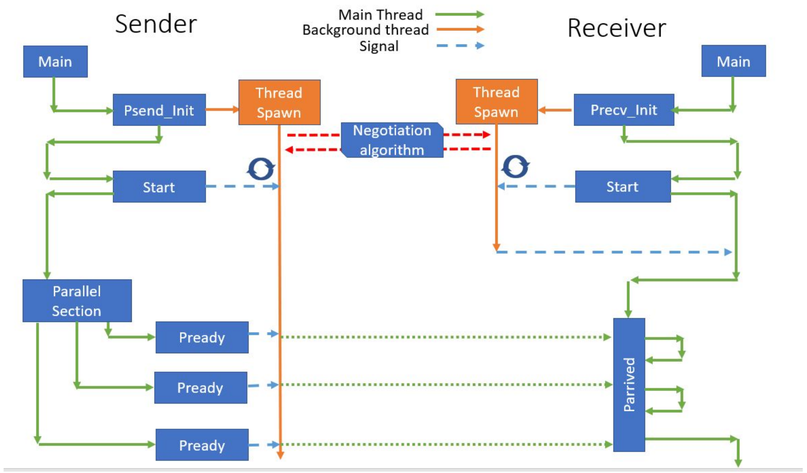
Library Structure

- The library contains a few different versions of the core structure to allow for testing with various features enabled or disabled.
- Follows the similar semantics as persistent point to point communication.

Basic Execution

Sender	Receiver
MPI_Psend_Init()	MPI_Precv_Init()
MPI_Start()	MPI_Start()
...*Prepare partition(s)*...	...
MPI_Pready()	...*Wait for partition(s)*…
…	MPI_Parrived()
MPI_Wait()	MPI_Wait()
MPI_Request_free()	MPI_Request_free()

Library Control Flow



Internal Design

New Request Object

Why? Existing request structures do not support tracking of partial completion states. By creating a new structure, we are able to track each of the partitions and while maintaining a format similar to the approved standard. Functions to interact with the new request object are included in the library.

Synchronization between Sender and Receiver

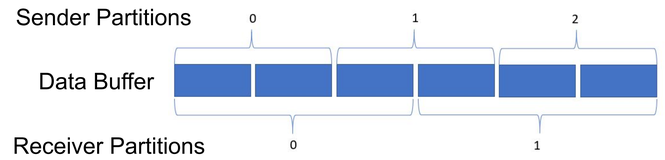
Why? As the library is constructed using existing point to point protocols, it is necessary to balance the internal sends and receive requests. The synchronization process also grants additional functionally in that the messages are not as tightly bound to the provided number of partitions. This allows for the support of different partitions at sender and receiver as well the possibility of aggregation for improved bandwidth utilization.

How? - Due to portability limitations, this is done using an additional communication, which requires blocking on at least one of the ends. As the initiation processes are required to be non-blocking, this is offloaded to a background thread.

Data Remapping at Receiver

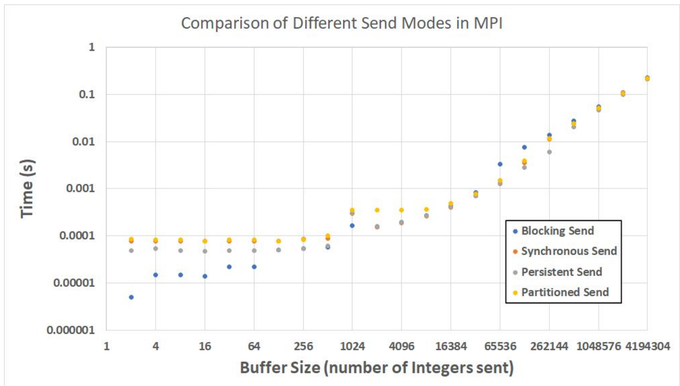
Why? - Abstracts the final buffer from the internal communication requests.

How? - Simple mapping using offsets



Results

Benchmarking involved comparing library to other other send routines:
Blocking Send, Synchronous Send, Persistent



A single partitioned send is comparable to a persistent send.

Ongoing Development

Limitation	Future Work
Single continuous datatype	Derived datatypes
Datatypes need to be identical on each side	Different but compatible datatypes on each side
Synchronization options limited	Additional synchronization algorithms
	Psync* integration
	Support for Info keys