A Portable Implementation of Point-to-Point Partitioned Communication

By

Andrew Worley, Prema Soundararajan, Derek Schafer
Ryan Grant, Mathew Dossnjh, Purushotham Bangalore
Anthony Skjellum, Sheikh Ghafoor
Partitioned Communication Model
Partition Aggregation

Begin Sending  Begin Sending  Begin Sending

Begin Sending  Begin Sending  Sending
### Partitioned Communication API

<table>
<thead>
<tr>
<th>Approved MPI 4.0 Functions</th>
<th>C Language Binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Psend_init</td>
<td>void *buf, int partitions, MPI_Count count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Info info, MPI_Request *request</td>
</tr>
<tr>
<td>MPI_Precv_init</td>
<td>void *buf, int partitions, MPI_Count count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Info info, MPI_Request *request</td>
</tr>
<tr>
<td>MPI_Pready</td>
<td>int partition, MPI_Request *request</td>
</tr>
<tr>
<td>MPI_Pready_range</td>
<td>int partition_low, int partition_high, MPI_Request *request</td>
</tr>
<tr>
<td>MPI_Pready_list</td>
<td>int length, int array_of_partitions[], MPI_Request *request</td>
</tr>
<tr>
<td>MPI_Parrived</td>
<td>MPI_Request *request, int partition, int *flag</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proposed MPI 4.1 Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Pbuf_prepare</td>
<td>MPI_Request request</td>
</tr>
<tr>
<td>MPI_Pbuf_prepareall</td>
<td>int count, MPI_Request requests[]</td>
</tr>
</tbody>
</table>

- **Initialize Operation**
- **Signal Partition Read**
- **Check for Partition Arrival**
- **Robust Synchronization**
In practice

• Break buffer into partitions.
• Threads signal when ready to transmit.
• Library decides transmission method
• Data is arrives as piecewise at receiver
Portable Layered Library

• Features
  • Runs on top of MPI 3.0 complaint implementations
  • MPIX versions of Partitioned Communication API
  • Uses new Request Object for tracking internal messages
  • Basic Static Aggregation Functionality

• Purpose
  • Early testing and Prototyping
Tracking Partition Tracking via MPIX_Request

- Multiple Requests nested inside a MPIX_Request object.
- Each internal request starts as soon as possible
Synchronization

• Non-local operation required by a local function
• Need to separate progress from function return.
• Solution: off-load to a progress engine.
  • Internal Implementations can use an internal progress engine
  • MPIPCL spawns a thread to act as a Progress Engine.
Benefits & Complications of Static Aggregation

What do we mean by static aggregation?
A single constant mapping between the partition contents and the internally generated messages

Pros:
• Easier Mapping and Tracking
• Single synchronization required.

• Cons
• Synchronization needed at initiation
• Message aggregation cannot be modified based on dynamic conditions.
Testing

• External MPI PCL Library
  • Intel MPI
  • OpenMPI

• OpenMPI Integrated Version
  • Some modifications required.
  • Used internal progress engine
  • Changed how some of the functions worked.
Performance Vs. Existing Communications

OpenMPI

Intel MPI
Layered vs. Internal

The graph compares execution time (in seconds) for different buffer sizes in integers, varying the number of partitions. The graph includes data for 1441440 and 2882880 integers, with results from MPI-CL and OMPI implementations.
Performance Results - MiniFE

Solve Time

Communication Time
Conclusion

• The library successfully allows access to the partitioned communication API.

• Performance is not negatively effected even with lack of optimizations.