Efficient Designs for Distributed MPI Neighborhood Collectives

S. Mahdieh Ghazimirsaeed
Ghazimirsaeed.3@osu.edu
Postdoctoral Researcher
Department of Computer Science and Engineering
The Ohio State University
Outline

• Introduction
  – MPI
  – Neighborhood Collectives

• Optimizing Neighborhood Collectives

• Experimental Results

• Conclusion and Future Work
Introduction

• HPC is the key in tackling the problems in different domains
  – Communication is the major bottleneck in achieving performance

• Message Passing Interface (MPI)
  – Most popular parallel programming model in HPC
  – Provides communication APIs
    • Point-to-point
    • One-sided (RMA)
    • Collective (broadcast, all-to-all, etc.)
    • Neighborhood collective (allgather, alltoall, etc.)
Neighborhood Collectives

- **Sparse** collective communications
- **Add communication functions to process topologies**
  - Pattern is defined by the process topology graph
  - Only send/receive to/from neighbors
- **Benefits:**
  - User-defined collectives
  - Support for widely used patterns
  - Better scalability

Can we design non-trivial algorithms to optimize neighborhood collectives?
Neighborhood Collectives – Prior Art

- Well-known MPI libraries perform neighborhood collective operations using a naive approach
- Kandalla et al. designed neighborhood collectives using a network-offload approach
- Hoefler and Schneider proposed an algorithm that balances communications by offloading them from high-outdegree processes to those having lower outdegrees
- Traff et al. proposed a specification and message combining algorithms for isomorphic neighborhoods, with all processes having the same neighborhood structure.
- Mirsadeghi et al. proposed a pair-wise message combining mechanism for distributed graph topologies
Prior work: Optimizing Neighborhood Collectives for Small Messages

- **Objective:**
  - Focus on small-message communications
  - Startup latency
    - Decrease the number of communication stages

- Two design principles
  1. Increase the number of senders in each stage
  2. Increase the number of messages transferred in each send operation ➔ **Message combining**
Prior work: Optimizing Neighborhood Collectives for Small Messages

- Exploit **common neighborhoods** for message combining
- Friend groups and common neighbors
  - Classify the processes into friend groups of size $k$, which have more than $\theta$ neighbors in common
  - Each friend process is responsible for transferring combined messages to $m/k$ of the common neighbors, after exchanging its message with the friends in the group
Optimizing Neighborhood Collectives for Large Messages

- Fully **distributed** algorithm

- Using **no prior knowledge** about the topology
  - Targeting MPI distributed graph topology

- **Focus on large messages**
  - Reduce the number of inter-node communication

- **Two phases:**
  1. **Communication Pattern** design from the topology
  2. **Communication Schedule** design from the communication pattern
Motivation

• A process attempts to send the same message to multiple processes on the remote node:
  1. sender process directly sends the data to the remote processes
  2. Sender chunks the large message and then it sends each chunk to remote processes. Then processes in the remote node perform an Allgather.
Motivation (Cont.)

• How can we redesign neighborhood collectives that take the underlying hardware architecture into account to achieve low latency?

• How can a neighborhood collective efficiently distribute the data between remote processes, in other words, how can we design a neighborhood collective which improves load-balance between all the processes?

• How can we create communication patterns that locally describe the specific communications of each involved process and at the same time support any kind of process topologies?
The Proposed Communication Pattern Design

Part of a general virtual topology graph

The hypergraph associated with sample topology graph
The Proposed Communication Schedule Design

• Uses use the derived communication pattern to extract an efficient communication schedule for the neighborhood Allgather operation
• The communication pattern is created only once for each topology graph
• The communication schedule is used each time a neighborhood collective operation is called
• Three levels of communication:
  – $p$ Communicates with intranode incoming and outgoing neighbors
  – $p$ Scatter the data between Inter-node outgoing neighbors
  – Allgather between Inter-node outgoing neighbors of $p$
Experimental Setup

- The InfiniBand-enabled Frontera cluster in Texas Advanced Computing Center
  - 8008 compute nodes
  - Dual-socket Intel Xeon Platinum 8280
  - 56-core processors operating at 2.70 GHz with 192GB RAM
  - MVAPICH2-2.3.2
Microbenchmark Results and Analysis

- **Moore Neighborhood**

![Speedup in Moore with 1,792 processes](image1)

![Speedup in Moore with 28,672 processes](image2)
Microbenchmark Results and Analysis

- Random Sparse Graph
Application Results and Analysis

- MILC Application Kernel

Overall Speedup for MILC Kernel

<table>
<thead>
<tr>
<th>Number of Processes</th>
<th>8k</th>
<th>16k</th>
<th>32k</th>
<th>64k</th>
<th>128k</th>
<th>256k</th>
<th>512k</th>
<th>1M</th>
<th>2M</th>
<th>4M</th>
<th>8M</th>
</tr>
</thead>
<tbody>
<tr>
<td>896</td>
<td>3.5</td>
<td>3.3</td>
<td>3.1</td>
<td>2.9</td>
<td>2.7</td>
<td>2.5</td>
<td>2.3</td>
<td>2.1</td>
<td>2.0</td>
<td>1.9</td>
<td>1.8</td>
</tr>
<tr>
<td>1792</td>
<td>3.4</td>
<td>3.2</td>
<td>3.1</td>
<td>2.9</td>
<td>2.7</td>
<td>2.5</td>
<td>2.3</td>
<td>2.1</td>
<td>2.0</td>
<td>1.9</td>
<td>1.8</td>
</tr>
<tr>
<td>3584</td>
<td>3.3</td>
<td>3.1</td>
<td>3.0</td>
<td>2.8</td>
<td>2.6</td>
<td>2.4</td>
<td>2.2</td>
<td>2.1</td>
<td>2.0</td>
<td>1.8</td>
<td>1.7</td>
</tr>
<tr>
<td>7168</td>
<td>3.2</td>
<td>3.0</td>
<td>2.9</td>
<td>2.7</td>
<td>2.5</td>
<td>2.3</td>
<td>2.1</td>
<td>2.0</td>
<td>1.9</td>
<td>1.8</td>
<td>1.7</td>
</tr>
<tr>
<td>14336</td>
<td>3.1</td>
<td>2.9</td>
<td>2.8</td>
<td>2.6</td>
<td>2.4</td>
<td>2.2</td>
<td>2.0</td>
<td>1.9</td>
<td>1.8</td>
<td>1.7</td>
<td>1.6</td>
</tr>
<tr>
<td>28672</td>
<td>3.0</td>
<td>2.8</td>
<td>2.7</td>
<td>2.5</td>
<td>2.3</td>
<td>2.1</td>
<td>2.0</td>
<td>1.9</td>
<td>1.8</td>
<td>1.7</td>
<td>1.6</td>
</tr>
</tbody>
</table>

Communication Speedup for MILC Kernel

<table>
<thead>
<tr>
<th>Number of Processes</th>
<th>8k</th>
<th>16k</th>
<th>32k</th>
<th>64k</th>
<th>128k</th>
<th>256k</th>
<th>512k</th>
<th>1M</th>
<th>2M</th>
<th>4M</th>
<th>8M</th>
</tr>
</thead>
<tbody>
<tr>
<td>896</td>
<td>3.5</td>
<td>3.3</td>
<td>3.1</td>
<td>2.9</td>
<td>2.7</td>
<td>2.5</td>
<td>2.3</td>
<td>2.1</td>
<td>2.0</td>
<td>1.9</td>
<td>1.8</td>
</tr>
<tr>
<td>1792</td>
<td>3.4</td>
<td>3.2</td>
<td>3.1</td>
<td>2.9</td>
<td>2.7</td>
<td>2.5</td>
<td>2.3</td>
<td>2.1</td>
<td>2.0</td>
<td>1.9</td>
<td>1.8</td>
</tr>
<tr>
<td>3584</td>
<td>3.3</td>
<td>3.1</td>
<td>3.0</td>
<td>2.8</td>
<td>2.6</td>
<td>2.4</td>
<td>2.2</td>
<td>2.1</td>
<td>2.0</td>
<td>1.8</td>
<td>1.7</td>
</tr>
<tr>
<td>7168</td>
<td>3.2</td>
<td>3.0</td>
<td>2.9</td>
<td>2.7</td>
<td>2.5</td>
<td>2.3</td>
<td>2.1</td>
<td>2.0</td>
<td>1.9</td>
<td>1.8</td>
<td>1.7</td>
</tr>
<tr>
<td>14336</td>
<td>3.1</td>
<td>2.9</td>
<td>2.8</td>
<td>2.6</td>
<td>2.4</td>
<td>2.2</td>
<td>2.0</td>
<td>1.9</td>
<td>1.8</td>
<td>1.7</td>
<td>1.6</td>
</tr>
<tr>
<td>28672</td>
<td>3.0</td>
<td>2.8</td>
<td>2.7</td>
<td>2.5</td>
<td>2.3</td>
<td>2.1</td>
<td>2.0</td>
<td>1.9</td>
<td>1.8</td>
<td>1.7</td>
<td>1.6</td>
</tr>
</tbody>
</table>

Communication Overlap for MILC Kernel

- Default
- Proposed Design

<table>
<thead>
<tr>
<th>Message Size</th>
<th>8k</th>
<th>16k</th>
<th>32k</th>
<th>64k</th>
<th>128k</th>
<th>256k</th>
<th>512k</th>
<th>1M</th>
<th>2M</th>
<th>4M</th>
<th>8M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overlap(%)</td>
<td>0</td>
<td>50</td>
<td>100</td>
<td>150</td>
<td>200</td>
<td>150</td>
<td>100</td>
<td>50</td>
<td>0</td>
<td>50</td>
<td>100</td>
</tr>
</tbody>
</table>
Application Results and Analysis

- Sparse Matrix-Matrix Multiplication

<table>
<thead>
<tr>
<th>Matrices</th>
<th>beaflw</th>
<th>lock1074</th>
<th>rkat7_mat5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rows</td>
<td>497</td>
<td>1074</td>
<td>694</td>
</tr>
<tr>
<td>Columns</td>
<td>507</td>
<td>1074</td>
<td>738</td>
</tr>
<tr>
<td>Density</td>
<td>0.22</td>
<td>0.04</td>
<td>0.07</td>
</tr>
<tr>
<td>Problem</td>
<td>Economic</td>
<td>Structural</td>
<td>Combinatorial</td>
</tr>
<tr>
<td>Speedup</td>
<td>9.53</td>
<td>1.25</td>
<td>1.29</td>
</tr>
</tbody>
</table>
Conclusion and Future Work

• **Conclusion**
  – Novel designs to efficiently increase the performance of large message neighborhood collectives
  – Modeling the communication pattern as a distributed hypergraph and determining the weight of hyperedges based on the virtual and physical topology of the processes
  – Load-aware: fair distribution of workload between processes

• **Future Work**
  – Add support for efficient communication for deep learning applications
  – Modify deep learning applications to take advantage of neighborhood collectives and consequently our design.