Optimizing Collective Performance in OpenSHMEM

Jithin Jose, Krishna Kandalla, Jie Zhang, Sreeram Potluri and Dhabaleswar K. (DK) Panda

Network-Based Computing Laboratory
Department of Computer Science and Engineering
The Ohio State University, USA

Outline

• Introduction & Motivation
• Problem Statement
• Design
• Performance Evaluation
• Conclusion & Future work

Introduction

• InfiniBand has become one of the most widely used interconnects in HPC
  – 41% of the systems in the June 2013 Top500 list
• MPI – the most widely used programming model for scientific parallel applications
• MPI Libraries (MVAPICH2, OpenMPI, IntelMPI) have been optimized to the hilt for InfiniBand clusters
• Partitioned Global Address Space (PGAS) models - OpenSHMEM, Unified Parallel C (UPC) – an attractive alternative

OpenSHMEM

• PGAS Models
  – Shared memory abstraction over distributed systems
  – Expose/manage locality of data for performance
  – Light weight one-sided communication
  – Easier to express irregular communication patterns
• OpenSHMEM (http://openshmem.org/)
  – Library based PGAS model
  – Open specification to standardize various SHMEM implementations
    • Memory management
    • Data transfers and Atomics
    • Collective Communication
    • Locks and Synchronization

Collective Communication Operations in MPI/PGAS

• Collective communication primitives offer a flexible, portable way to implement group communication operations
• Used across various scientific applications
• Supported across both MPI and PGAS models
• High-performance MPI implementations have incorporated optimizations for modern architectures
  – Optimized using multi-core-aware, network-aware, kernel assisted mechanisms and optimized algorithms
Collective Communication in OpenSHMEM

- OpenSHMEM collective standards are relatively new
- Linear and Tree algorithm based collective implementations available in OpenSHMEM reference implementation
- Current reference implementations need more work in optimizing collective communication primitives
  - Needs improvement in performance and scalability
- Can we leverage the entire gamut of designs that are available in high performance MPI implementations?

Outline

- Introduction & Motivation
- Problem Statement
- Design
- Performance Evaluation
- Conclusion & Future work

Problem Statement

- Can we improve OpenSHMEM collective operations by efficiently mapping them on to MPI collectives?
- Can we design a light-weight and scalable interface to improve the OpenSHMEM collectives performance through leveraging MPI-level designs?
- What are performance benefits of our proposed approach across various micro-benchmarks and OpenSHMEM applications?

Collectives in OpenSHMEM

- `shmem_broadcast`
  - Copy a block of data from one PE (similar to rank in MPI) to one or more target PEs (equivalent to MPI_Bcast)
- `shmem_collect`
  - Concatenate elements from the source array to a target array over the specified PEs (equivalent to MPI_Allgather)
- `shmem_reduce`
  - Perform an associative binary operation over the specified PEs (equivalent to MPI_Allreduce)
- `shmemBarrier`
  - No PE may leave the barrier prior to all PEs entering the barrier (similar to MPI_Barrier)

Defining non-global groups OpenSHMEM

- OpenSHMEM Collectives specifies group dynamically – “Active Set”
  - void `shmem_broadcast64(void *target, const void *source, size_t nelems, int PE_root, int PE_start, int logPE_stride, int PE_size, long *pSync);
  - Eg: Collective among even ranked processes
    - PE_Start = 0
    - PE_Stride = 1 (log (base 2) of stride between PEs)
    - PE_Size = 4

Outline

- Introduction & Motivation
- Problem Statement
- Design
  - Design Challenges
  - Detailed Design
- Performance Evaluation
- Conclusion & Future work
Defining non-global groups in MPI

- MPI uses communicator
  - Communicators are created prior to collective call
    - int MPI_Bcast( void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
  - Communicator Creation in MPI
    - int MPI_Comm_create(MPI_Comm comm, MPI_Group group, MPI_Comm *newcomm)
    - Collective Operation among all processes in comm (in MPI-2)
    - Costly operation

Challenges in designing OpenSHMEM collectives over MPI

- MPI Collectives are based on “Communicators”, but OpenSHMEM collectives based on “Active Set”
- Higher cost of creating communicator in MPI
  - Cannot create communicator for each collective operation
- Need for non-global collective communicator creation
  - OpenSHMEM collectives involve only the processes in the “Active Set”

MVAPICH2/MVAPICH2-X Software

- High Performance open-source MPI Library for InfiniBand, 10Gig/LiWARP and RDMA over Converged Enhanced Ethernet (RoCE)
  - MVAPICH (MPI-1), MVAPICH2 (MPI-3.0), Available since 2002
  - MVAPICH2-X (MPI, OpenSHMEM, UPC, MP + OpenSHMEM/UPC), Available since 2012
  - Used by more than 2,085 organizations (HPC Centers, Industry and Universities) in 70 countries
  - More than 183,000 downloads from OSU site directly
  - Empowering many TOP500 clusters
    - 6th ranked 462,462-core cluster (Stampede) at TACC
    - 19th ranked 125,980-core cluster (Pleiades) at NASA
    - 21st ranked 73,278-core cluster (Truetime 2.0) at Tokyo Institute of Technology and many others
  - Available with software stacks of many IB, HSE and server vendors including Linux Distros (RedHat and SuSE)
    - http://mvapich.cse.ohio-state.edu
  - Partner in the U.S. NSF-TACC Stampede System

Group-based Communicator Creation

- Communicator creation until MPI-2 required involvement of all processes in the parent communicator
- MPI-3 has introduced routine for creating group-based collective communicator
  - MPI_Comm_create_group(MPI_Comm comm, MPI_Group group, int tag, MPI_Comm *newcomm)
- Algorithm is collective only on processes that are members of group
- Implemented as an Allgather, complexity is log(N)

Design Overview

- Design Components
  - Light-weight group-based communicator creation
  - Communicator Cache

Communicator Cache

- Communicator cache caches non-global communicators
- Cache lookup for every non-global communicator creation request
- If cache hit, communicator is reused; otherwise communicator is created and added to Cache
- Limit on number of cache entries
- Communicator is freed during finalize operation
Outline

- Introduction & Motivation
- Problem Statement
- Design
- Performance Evaluation
  - Experiment Setup
  - Micro-benchmark Results
  - Application Evaluations
- Conclusion & Future work

shmem_broadcast Performance across Implementations

- MV2-X Linear and tree versions perform better than UH-SHMEM (uses GASNet as runtime)
- Proposed approach achieves better latency
  - 4byte: Proposed - 1.41us, MV2-X Tree – 79us, UH-Tree – 90us
  - 22X better performance compared to MV2X Tree at 128 processes
  - Latencies similar to MPI_Bcast results

shmem_collect Performance

- Achieves better performance than the linear algorithm
  - At 512 procs, Latency for 4 byte collect: Linear – 59 ms, Proposed - 45 us
  - No tree based algorithm exists for collect
- Similar performance as that of MPI_Allgather performance

shmem_reduce Performance

- Proposed design achieves better performance than the linear and tree algorithm
  - At 512 procs, Latency for 4 byte reduce: Linear – 30 ms, Tree – 12 ms, Proposed - 10 us
- Similar performance as that of MPI_AllReduce performance

Experiment Setup

- Cluster A (TACC Stampede)
  - Intel Sandybridge series of processors using Xeon dual 8 core sockets (2.7GHz) with 32GB RAM
  - Each node is equipped with FDR ConnectX HCA
  - (54 Gbps data rate) with PCI-Ex Gen3 interfaces
- Cluster B
  - Intel Westmere Dual quad-core processor (2.67GHz) with 12GB RAM
  - Each node is equipped with QDR ConnectX HCA (32Gbps data rate) with PCI-Ex Gen2 interfaces
- Software Stacks
  - OpenSHMEM: MVAPICH2-X OpenSHMEM (v1.9, denoted as MV2X); OpenSHMEM Reference Implementation (v1.0d, denoted as UH) using GASNet (v1.20.2) as runtime
  - MPI: MVAPICH2 1.9

Statement

- Message Size
  - 4 byte
  - 8K
  - 16K
  - 32K
  - 64K
  - 128K
  - 256K
  - 512K
  - 1M
  - 2M
  - 4M
  - 8M
  - 16M
  - 32M
  - 64M
  - 128M
  - 256M
  - 512M
  - 1G
  - 2G
  - 4G
  - 8G
  - 16G
  - 32G
  - 64G
  - 128G
  - 256G
  - 512G
  - 1T
  - 2T
  - 4T
  - 8T
  - 16T
  - 32T
  - 64T
  - 128T
  - 256T
  - 512T
  - 1P
  - 2P
  - 4P
  - 8P
  - 16P
  - 32P
  - 64P
  - 128P
  - 256P
  - 512P
**Proposed design achieves better performance than the linear and tree algorithm**
- At 512 procs, latency for barrier operation: Linear ~ 2.1 ms, Tree ~ 44 us, Proposed ~ 12 us
- Similar performance as that of MPI_Barrier performance

**Shmem_barrier Performance**

- 2D Heat Transfer Application
  - Repeats Gauss-Seidel kernel until the value convergence
  - Uses shmem_barrier for synchronization, and shmem_reduce and shmem_broadcast in Gauss-Seidel kernel and for calculating convergence
  - Execution time (512 procs): Linear ~ 179s, Tree ~ 136s, Proposed ~ 127s

**Application Evaluation – 2D Heat Transfer**

- Non-COMM-WORLD Benchmark
  - Broadcast among even ranked processes
  - At 512 procs, latency for 4 byte broadcast: Linear ~ 2ms, Tree ~ 47 us, Proposed ~ 5 us
- Similar performance as that of MPI_Bcast performance

**Shmem_broadcast Performance (Even ranks)**

- Graph500 Benchmark
  - Level synchronized graph traversal algorithm
  - Vertices are kept in queues – 'new_queue' and 'curr_queue'
  - Queue lengths are calculated across all processes to decide whether to proceed to next level
  - Uses shmem_reduce for identifying queue length
- Execution Time (4,096 procs):
  - Linear ~ 10s, Tree ~ 4s, Proposed ~ 1.8 s

**Application Evaluation – Graph500**

**Outline**

- Introduction & Motivation
- Problem Statement
- Design
- Performance Evaluation
- Conclusion & Future work

**Conclusion**

- Proposed a high performance, light weight caching mechanism to map OpenSHMEM collectives over MPI
- OpenSHMEM implementations can directly leverage the advanced designs that are available in MPI
- Orders of magnitude improvement in performance in microbenchmark evaluations
- 7% improvement for 2D Heat Transfer Modeling OpenSHMEM Application for 512 processes
- 57% improvement for Hybrid Graph500 application for 4,096 processes
- Available in MVAPICH2-X v2.0a release
- Future Work: Design a light-weight communicator at the runtime layer
Thank You!
{jose, kandalla, zhanjie, potluri, panda}@cse.ohio-state.edu

Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/
MVAPICH Web Page
http://mvapich.cse.ohio-state.edu/