Initial Study of Multi-Endpoint Runtime for MPI+OpenMP Hybrid Programming Model on Multi-Core Systems [*]

Miao Luo, Xiaoyi Lu, Khaled Hamidouche, Krishna Kandalla, Dhabaleswar K. (DK) Panda

Dept. of Computer Science and Engineering
The Ohio State University
{luom, luxi, hamidouc, kandalla, panda}@cse.ohio-state.edu

Abstract

State-of-the-art MPI libraries rely on locks to guarantee thread-safety. This discourages application developers from using multiple threads to perform MPI operations. In this paper, we propose a high performance, lock-free multi-endpoint MPI runtime, which can achieve up to 40% improvement for point-to-point operation and one representative collective operation with minimum or no modifications to the existing applications.

1. Introduction

MPI/OpenMP hybrid programming model is widely regarded as suitable model for scaling parallel applications on emerging multi-/many-core computing architectures. In this model, applications can be deployed with one MPI process per compute node or CPU socket, with OpenMP "threads" running on other compute cores to accelerate computation. However, previous studies demonstrated that there are several challenges associated with combining these two models (MPI and OpenMP) to fully leverage the performance benefits of a hybrid model [2–4, 6]; including overhead from locks and waste of communication/computation resources. It is important to design a new MPI runtime and understand its impact in accelerating MPI/OpenMP hybrid applications on modern multi-core systems. In this paper, we propose a high performance, lock-free multi-endpoint MPI runtime and discuss how to deliver its benefits to applications for point-to-point and collective operations.

* This research is supported in part by National Science Foundation grants #OCI-0926691, #OCI-1148371 and #CCF-1213084

PPoPP '14, February 15–19, 2014, Orlando, Florida, USA.
Copyright is held by the owner/author(s).
ACM 978-1-4503-2656-8/14/02.
http://dx.doi.org/10.1145/2555243.2555287

2. Design of Multi-endpoint MPI Runtime

2.1 Lock-free Communication Routines

In order to achieve lock-free communication routines, we identify the following two critical components that need to be re-designed according to multi-threading requirements:

Request Handling: Instead of sharing the same request memory region across different threads with locks, our proposed design pre-allocates request memory objects according to the number of threads that would be able to call MPI functions. Each thread is associated with a thread-ID as a thread local storage parameter (LST). Any request-related operations should access the corresponding request memory object according to a specific thread-ID.

Communication Resources Management: In order to launch send/receive operations without lock protection, the proposed multi-treading runtime establishes necessary connection resources according to the number of endpoints. These resources are stored in the form of a table. An Endpoint-ID serves as the index key for accessing the table. Particularly, the communication resources instantiate the concept of endpoint in the proposed runtime.

Based on the proposed multi-endpoint runtime, we are able to remove locks in the critical routines. However, it is not straight-forward to deliver the benefits of multi-endpoint runtime to applications. It is necessary to explore design alternatives to provide the benefits of the proposed multi-endpoint runtime to real applications, in a transparent manner, or with minimal modifications.

2.2 Optimization for Point-to-Point Operations

In the master-only model, MPI non-blocking send/receive operations are called after the OpenMP threads joined, or only the master thread makes MPI function calls through "pragma omp master" directive. Simple modifications are required for such programs in order to take advantage of the proposed multi-endpoint runtime. The application developers only need to add OpenMP pragma, "pragma omp parallel", outside of the MPI functions with the original input parameters. Inside the runtime, the number of endpoints E for communication requests is decided by the message size, according to a pre-defined tuning table. Each endpoint is responsible for sending its portion of size $\frac{n}{E}$ independently for the original message of size n. Threads that are not assigned with a communication task just bypass and wait at the barrier or continue with the next unrelated communication request.

2.3 Optimizations for Collective Operations

This section discusses our initial study of multi-endpoint based algorithm optimizations for the pair-wise exchange based MPI_Alltoallv operation, with no modification to applications.

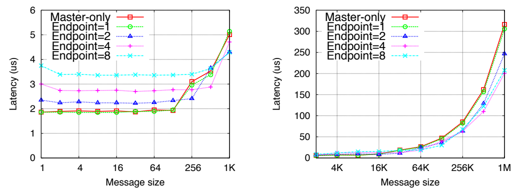
Pair-wise exchange algorithm is widely utilized to implement the MPI_Alltoallv operation. $p-1$ send/receive operations are required for every MPI rank with p processes. In the single-endpoint runtime, all the send/receive requests are handled by a single thread. The multi-endpoint runtime can optimize the algorithm by distributing these requests among a set of endpoints. In the new algorithm, all the p MPI ranks can be divided into E groups, where E is the number of active endpoints. In each node, the thread with Endpoint-ID equal to Group ID operates as the "receiver"; and, all the threads should perform as "sender" to the destinations with a Group ID equal to the Endpoint-ID of the "sender". The new "Group" algorithm successfully distributes the send requests from a single endpoint across multiple endpoints.

3. Experimental Results

We implement the new design based on MVAPICH2 1.9 [5] and use it for the comparison. We carry out the evaluations on the "Stampede" computing system from TACC [1]. Each compute node includes two Xeon E5-2680 processors with a total of 16 cores and 32GB memory per node. Nodes are interconnected with Mellanox FDR InfiniBand. We use 256 compute nodes (4,096 cores) for collective evaluations.

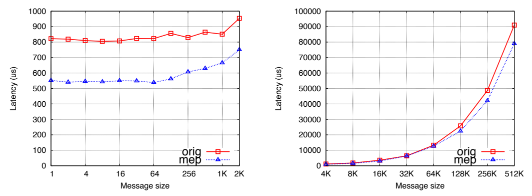
3.1 Point-to-Point Micro-Benchmark Evaluations

We compare the performance across different numbers of active endpoints with the default single-endpoint runtime in Figure 1. The best number of active endpoints varies according to message size. The multi-endpoint optimization can reduce the latency of the master-only mode by up to 40%. A tuning table inside the multi-endpoint runtime is configured according to the evaluation results. It helps the runtime to choose the most suitable number of active endpoints according to the message size.



(a) Small message (b) Large message

Figure 1. Evaluations for point-to-point



(a) Small message (b) Large message

Figure 2. Evaluations: alltoallv

3.2 Collective Micro-Benchmark Evaluations

We compare our multi-endpoint optimized "Group" algorithm, with the process-based alltoallv operation ("orig") in Figure 2. The multi-endpoint runtime automatically adjusts the number of active endpoints according to the predefined tuning table. The multi-endpoint runtime ("mep") can achieve 30% improvement for small messages, which is mainly from the overlapped send startup time. For large messages, the new runtime can achieve up to 14% improvement from a better utilization of network bandwidth.

4. Conclusions and Future Work

This paper proposed an initial study of a lock-free, multi-endpoint runtime for MPI/OpenMP hybrid applications on emerging multi-core systems. This design can efficiently accelerate hybrid parallel applications with minimal modifications for point-to-point and no modification for collective operations. Preliminary results show that our framework can improve the performance of point-to-point and collective operations by up to 40% with 256 compute nodes (4,096 cores). For future work, we plan to investigate our multi-endpoint runtime design for heterogeneous architectures and explore more efficient collective algorithms based on the multi-endpoint design.

References

[1] Stampede at Texas Advanced Computing Center. http://www.tacc.utexas.edu/resources/hpc/stampede.

[2] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, and R. Thakur. Fine-Grained Multithreading Support for Hybrid Threaded MPI Programming. *Int. J. High Perform. Comput. Appl.*, 24 (1):49–57, Feb. 2010.

[3] S. Bova, C. Breshears, H. Gabb, B. Kuhn, B. Magro, R. Eigenmann, G. Gaertner, S. Salvini, and H. Scott. Parallel Programming with Message Passing and Drectives. *Computing in Science Engineering*, 3(5):22–37, 2001.

[4] Z. Lan, V. Taylor, and G. Bryan. Dynamic Load Balancing for Structured Adaptive Mesh Refinement Applications. In *International Conference on Parallel Processing*, 2001.

[5] MVAPICH2. http://mvapich.cse.ohio-state.edu/.

[6] R. Rabenseifner, G. Hager, and G. Jost. Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes. In *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*, pages 427–436, 2009.