Multi-core Aware Optimization for MPI Collectives

Bibo Tu[1,2], Ming Zou[1,2], Jianfeng Zhan[1], Xiaofang Zhao[1] and Jianping Fan[1]

[1]*Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190*

[2]*Graduate University of Chinese Academy of Sciences, Beijing 100049*

{tbb, zm, jfzhan}@ncic.ac.cn, {zhaoxf, fan}@ict.ac.cn

***Abstract*—MPI collective operations on multi-core clusters should be multi-core aware. In this paper, collective algorithms with hierarchical virtual topology focus on the performance difference among different communication levels on multi-core clusters, simply for intra-node and inter-node communication; Furthermore, to select befitting segment sizes for intra-node collective communication can cater to cache hierarchy in multi-core processors. Based on existing collective algorithms in MPICH2, above two techniques construct portable optimization methodology over MPICH2 for collective operations on multi-core clusters. Conforming to above optimization methodology, multi-core aware broadcast algorithm has been implemented and evaluated as a case study. The results of performance evaluation show that the multi-core aware optimization methodology over MPICH2 is efficient.**

I. INTRODUCTION

In the new Top500 supercomputer list published in November 2007, multi-core clusters have been the most popular platforms in parallel computing [1]. Intuitively, multi-core processors can speedup application performance by dividing the workload to different cores. However, applications on multi-core clusters have not gotten optimal performance. Initially, applications are likely to treat multi-core processors, also called Chip Multiprocessor (CMP), simply as conventional symmetric multiprocessors (SMPs). However, chip multiprocessors with shared cache offer unique capabilities that are fundamentally different from SMPs, only with shared main memory. Accordingly, applications should be multi-core aware.

For many scientific applications, the communication cost dominates overall execution time, so communication middleware (e.g. MPI) should also be multi-core aware. MPI has emerged as one of the primary programming paradigms for writing efficient parallel applications. It provides for a plethora of communication primitives with operations geared towards point-to-point and collective communications. When compared to the point-to-point performance, collective operation performance is often overlooked. However, profiling study [2] showed that some applications spend more than 80% of a transfer time in collective operations. Thus, improving the performance of collective operations is the key to enabling very high parallel speed-ups for parallel applications. Significant research has been carried out in the past for improving collective communication. Some work on collective communication focused on developing optimized algorithms for particular architectures, such as hypercube, mesh, torus or fat tree, with an emphasis on minimizing link contention, node contention, or the distance between communicating nodes [3, 4]. Automatically tuned collective communication algorithms under different conditions (message size, number of processes) have been developed [5, 6]. Another work on collective communication focused on exploiting hierarchy in parallel computer networks to optimize collective operation performance. One is to optimize MPI collective communication for WAN distributed environments (e.g. MagPIe) [7, 8], the goal is to minimize communication over slow wide-area links at the expense of more communication over faster local-area connections. The other is to optimize MPI collective communication for LAN environments through modifying MPICH ADI layer, target for SMP clusters, e.g. MPI-StarT [9]. Utilizing shared memory for implementing collective communication has been a well studied problem in the past. Some propose using remote memory operations across the cluster and shared memory within the cluster to develop efficient collective operations for clusters of SMPs [10, 11]. With the help of operating system and network drivers, minimizing the cost of memory copy to improve intra-node communication has also been applied in intra-node communication on more-core clusters recently [12]. Some above MPI-level algorithms have successfully been implemented in MPICH2 [13]. However, the collective algorithms currently employed don't perform optimally on the new multi-core clusters.

The optimal implementation of a collective for a given system mainly depends on virtual topology (e.g. flat-tree, binary tree, binomial tree etc.) and message sizes. Memory hierarchy on multi-core clusters gets more complicated, so exploiting hierarchy in different levels of communication (typically for three communication levels: intra-CMP, inter-CMP and inter-node, simply for two communication levels: intra-node and inter-node) on multi-core clusters to optimize collective operation performance, i.e. topology-aware algorithms for collective operations, will be essential, which is inspired by MagPIe and MPI-StarT; Furthermore, to select befitting segment sizes for intra-node collective communication can cater to cache hierarchy in multi-core processors (e.g. private L1 cache and shared L2 cache in Intel multi-core platform). The former emphasizes on the performance difference (communication hierarchy) in different levels of communication, the latter tries to improve cache hit ratio in intra-node collective communication. They together construct portable optimization methodology over MPICH2 for collective operations on multi-core clusters. As a case study, multi-core aware broadcast algorithm has been implemented and evaluated, conforming to above optimization methodology.

978-1-4244-2640-9/08/$25.00 © 2008 IEEE

Accepted as a poster presentation
2008 IEEE International Conference on Cluster Computing

The rest of the paper is organized as follows: In Section II, we propose optimization methodology for collective operations on multi-core clusters. As a case study, multi-core aware broadcast algorithm and its performance evaluation are given in Section III and IV, respectively. And finally we conclude and point out future work directions in Section V.

II. METHODOLOGY

With the emergence of multi-core processors, memory hierarchy on multi-core clusters becomes more and more complicated, so CMPs offer unique capabilities that are fundamentally different from SMPs [14]:

i) The inter-core communication performance (bandwidth and latency) on a CMP can be many times better than is typical for a SMP.
ii) Cache hierarchy organization (e.g. shared L2 cache on Intel Quad- and Dual-Core Xeon) on CMPs makes inter-core memory access far faster, if data buffers are in the cache, than is typical for a SMP, only with shared main memory.

For collective operations on multi-core clusters, some prominent communication characteristics differently from conventional SMP clusters are brought, which can be seen from recent studies [15, 16].

Prominent intra-node communication performance: For SMP clusters, the performance of intra-node communication only with shared main memory can't absolutely excel that of inter-node communication, because of memory copy cost of intra-node communication and some zero-copy and high performance schemes of inter-node communication, e.g. InfiniBand and RDMA-enabled interconnects. However, for multi-core clusters, the performance of intra-node communication is many times better than that of inter-node communication because data can be shared through L2 cache as the example of Intel Quad-core Xeon [15, 16].

Equal chance between intra-node and inter-node communication: For SMP clusters, the most of message distribution is distributed point-to-point communication (inter-node communication); while an average about 50% messages are transferred through the intra-node communication in message distribution experiments of parallel application on a multi-core cluster [15]. With the ever-increased number of computing cores in a CMP, especially for future many-core processors, the chance of intra-node communication will be more dominant.

Consequently, communication hierarchy on multi-core clusters and cache utilization in intra-node communication must be adequately considered for optimal MPI collective operations.

MPI collective operations can be classified as either one-to-many/many-to-one (asymmetrical virtual topology, the root process exists) or many-to-many (symmetrical virtual topology, all processes are equal peers) operations. For example, Broadcast, Reduce, Scatter(v), and Gather(v) follow one-to-many communication pattern, while Barrier, Alltoall, Allreduce, and Allgather(v) employ many-to-many communication. Generalized version of the one-to-many/many-to-one type of collectives can be expressed as i) receive data from preceding node(s), ii) process data (optional), iii) send data to succeeding node(s). The data flow for this type of algorithms is unidirectional. Based on some parameters, including message sizes, number of processes and the location of the root node (where applicable), different virtual topologies can be used to determine the preceding and succeeding nodes in the algorithm.

To optimize collective operations on multi-core clusters, our methodology focuses on communication hierarchy on multi-core clusters and cache utilization in intra-node collective communication, which is presented as follows:

Hierarchical Virtual Topology: Our algorithms are designed for a two layer hierarchy, respectively inter-node communication and intra-node communication. Concretely, asymmetrical algorithms perform two macro steps. In a one-to-many operation, the root first sends its data to head processes of other nodes (the first process in every node), and second, the head process locally sends to other processes in the same node. In the many-to-one case, processes in the same node first send to their head process and then all head processes in different nodes send to the root. Symmetrical algorithms perform three macro steps. First, processes in every node send to their head process. Second, all head processes perform an all-to-all exchange, and third, every head process sends to their other processes.

Cache–aware Intra-node Collective Communication: In order to improve the performance of intra-node collective communication, an easy software technique related with message sizes is used to adapt well cache-shared multi-core system: data tiling approach for improving utilization of private L1 cache and shared L2 cache. Data tiling is a cache blocking technique and tries to allow data to stay in the shared cache while being processed by data loops. By reducing the unnecessary cache traffic (fetching and evicting the same data throughout the loops), a better cache hit rate is achieved. Possible candidates for the cache blocking technique include large data sets that get operated on many times. By going through the entire data set and performing one operation, followed by another pass for the second operation, and so on, if the entire data set does not fit into the cache, the first elements in the cache will be evicted to fit the last elements in. Then, on subsequent iterations through the loop, loading new data will cause the older data to be evicted, possibly creating a domino effect where the entire data set needs to be loaded for each pass through the loop. By sub-dividing the large data set into smaller blocks and running all of the operations on each block before moving on to the next block, there is a likelihood that the entire block will remain in cache through all the operations.

Some previous optimization algorithms involved with virtual topology and message sizes have successfully been implemented in MPICH2, while our methodology is constructed on top of them and portable. In fact, existing algorithms in MPICH2 are still reserved in intra-node and inter-node collective communications, respectively. But our methodology is only to reorder their data flows.

III. MULTI-CORE AWARE BROADCAST

As a case study, multi-core aware broadcast algorithm is optimized. In MPI_Bcast in MPICH2, a so-called root process sends a data vector to all other processes. This operation is the simplest case of an asymmetrical, one-to-many algorithm. MPI_Bcast uses multiple algorithms based on the message size. For small messages, binomial tree algorithms are widely employed. Large message broadcast involves a more complex approach of doing a scatter followed by an allgather. According to our portable optimization methodology over MPICH2, we needn't utterly rewrite above MPI_Bcast algorithm, only reorder its data flow.

```
int MCC_MPI_Bcast (void *buffer, int count, MPI_Datatype datatype, int
root, MPI_Comm comm)
{
/* define all node communicators and head processes communicator */
MPI_Comm Node_Comm[MaxNodeNum], Head_Comm;

/* all processes in the same node are grouped into new communicator, i.e.
every node has a communicator, respectively. */
for (i=0; i < Node_Num; i++)
      MPI_Comm_split(comm, isSameNode(), 0, Node_Comm[i]);

/* all head processes (first process with rank 0 in every node) are grouped into
a new communicator */
MPI_Comm_create(comm., isFirstRank(myrank), Head_Comm);

/*Step 1:  inter-node collective communication */
if (isFirstRank(myrank))
   MPI_Bcast(buffer, count, datatype, root, Head_Comm);

/* Step 2: intra-node collective communication */
for (i=0; i < Node_Num; i++) {
   if (isLocalComm(myrank, Node_Comm[i]) {
/* in a loop broadcast call, message is divided into several segments in order
to fit in cache */
      if (messagesize < fitL1CacheSize) /* no segmented */
          MPI_Bcast(buffer, count, datatype, 0, Node_Comm[i]);
      else if (messagesize < fitL2CacheSize) { /* to fit in L1 Cache */
         for (j = 0; j < messagesize/fitL1CacheSize; j++)
            MPI_Bcast(buffer, fitL1CacheSize, datatype, 0, Node_Comm[i]);
      }
      else {  /* to fit in L2 cache */
         for (j = 0; j < messagesize/fitL2CacheSize; j++)
            MPI_Bcast(buffer, fitL2CacheSize, datatype, 0, Node_Comm[i]);
      }
   }
}}
```

Fig. 1 Pseudo-code for multi-core aware broadcast algorithm (MCC_MPI_Bcast)

Pseudo-code for multi-core aware broadcast algorithm (MCC_MPI_Bcast) is presented in Figure 1. First, all processes in the same node are grouped into a new communicator, thus, every node exists a communicator (e.g. Node_Comm[i] in Fig. 1), respectively. All head processes (process with rank 0) in every node communicator are grouped into a new communicator (e.g. Head_Comm in Fig. 1) for inter-node collective communication. Second, all head processes start inter-node broadcast communication. Thus, the root sends message to all nodes. At last, in every node, message is segmented to start intra-node broadcast communication in a loop.

In intra-node collective communication, segment size is determined by data tiling approach for improving L1 cache and L2 cache utilization. In MCC_MPI_Bcast algorithm, for small message, to split message into segments that can easily fit L1 cache (fitL1CacheSize) may hold segment in high performance L1 cache for a long time in one-to-many communication. For large message, message will be split into some segments that can fit into shared L2 cache (fitL2CacheSize) and get operated on many times. Thus, both L1 and L2 cache hit rates get improved.

IV. PERFORMANCE EVALUATION

We evaluated the performance of cache-aware intra-node broadcast communication and multi-core aware broadcast algorithm, compared with MPI_Bcast in MPICH2. Our evaluation system consists of eight nodes connected by Gigabit Ethernet. Each node is equipped with two sets of quad-core 2.0GHz Intel Xeon processor, i.e. eight computing cores per node. Four cores on the same chip share 8M L2 cache and each core has 16KB private L1 cache. The operating system is Linux 2.6, and MPI version is mpich2-1.0.6p1.

Table 1 shows performance results of cache-aware intra-node broadcast communication (8 processes), assuming fitL1CacheSize is respectively equal to 4K, 8K, 16K and fitL2CacheSize is respectively equal to 512K, 1M, 2M. In order to get optimized performance, segment size and the number of broadcast loops need the tradeoff. From Table 1, we can see that: when segment size is 8KB, the performance of intra-node broadcast communication is the best for small message (less than 512KB); when segment size is 512KB, the performance of intra-node broadcast communication is the best for large message (more than 512KB). Consequently, for our evaluation system, when parameters fitL1CacheSize = 8KB and fitL2CacheSize = 512KB, the performance of cache-aware intra-node broadcast communication is optimal.

Hierarchical virtual topology and cache-aware intra-node collective communication improve the performance of broadcast operation (MCC_MPI_Bcast) on multi-core clusters, compared with MPI_Bcast in MPICH2, which is seen from Figure 2. Broadcast operation with 64 processes (8x8, 8 processes every node, totally 8 nodes) has been tested with different message sizes. The root process is selected at random. From the figure, we can see that the performance difference between MCC_MPI_Bcast and MPI_Bcast is increasing with message sizes, and the percentage of performance improvement is from 16.8% to 24.7%.

TABLE 1
THE COSTS OF CACHE-AWARE INTRA-NODE BROADCAST COMMUNICATION (8 PROCESSES) WITH DIFFERENT MESSAGE SIZES AND SEGMENT SIZES IN MICROSECONDS

Segment / Message	4K	**8K**	16K	**512K**	1M	2M	Not segmented
4K	52	*	*	*	*	*	52
8K	81	58	*	*	*	*	58
16K	133	93	336	*	*	*	336
32K	267	175	624	*	*	*	473
64K	483	436	1103	*	*	*	658
128K	1233	887	2030	*	*	*	1051
256K	2402	1824	3887	*	*	*	1872
512K	4977	3816	7469	3367	*	*	3367
1M	9654	6848	13690	6587	7413	*	7413
2M	18809	13883	28830	13647	14116	14163	14163
4M	38924	27835	53969	26091	26834	27628	27491
8M	76884	56811	98860	52324	52636	53563	57418

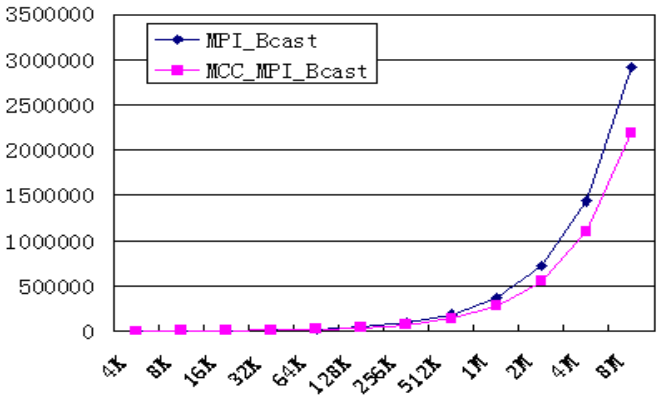

Fig. 2 the costs of multi-core aware broadcast algorithm (64 processes, 8 processes every node) compared with MPI_Bcast in MPICH2 in microseconds

V. CONCLUSION AND FUTURE WORK

Optimizing MPI collective communication on emerging multi-core clusters is the key to obtaining good performance speed-ups for many parallel applications. Communication hierarchy and cache hierarchy are two main characteristics on multi-core clusters. In order to adapt them, we propose the corresponding portable optimization methodology constructed over MPICH2, including hierarchical virtual topology and cache-aware intra-node collective communication. Conforming to it, optimized broadcast algorithm has been implemented and shown good performance. For our future work, we continue to carry out more in-depth design and evaluation of optimal algorithms for other important collective operations.

REFERENCES

[1] TOP500 Team, TOP500 Report for November 2007, http://www.top500.org.

[2] Jelena Pjesivac-Grbovic, Thara Angskun, George Bosilca, etc. Performance Analysis of MPI Collective Operations, Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), 2005.

[3] Ernie Chan, Marcel Heimlich, Avi Purkayastha and Robert van de Geijn, Collective communication: theory, practice, and experience, Concurrency and Computation: Practice & Experience, Volume 19, Issue 13, Pages: 1749 – 1783, 2007

[4] D. Scott. Efficient all-to-all communication patterns in hypercube and mesh topologies. In Proceedings of the 6th Distributed Memory Computing Conference, pages 398–403, 1991.

[5] Sathish S. Vadhiyar, Graham E. Fagg, and Jack Dongarra. Automatically tuned collective communications. In Proceedings of SC99: High Performance Networking and Computing, November 1999.

[6] Ahmad Faraj, Xin Yuan, Automatic Generation and Tuning of MPI Collective Communication Routines, Proceedings of the 19th annual international conference on Supercomputing, Pages: 393 - 402 , 2005.

[7] Nicholas T. Karonis, Bronis R. de Supinski, Ian Foster, William Gropp, etc. Exploiting hierarchy in parallel computer networks to optimize collective operation performance, Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS'2000), pp. 377-384, 2000.

[8] Thilo Kielmann, Rutger F. H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A. F. Bhoedjang, MAGPIE: MPI's Collective Communication Operations for Clustered Wide Area Systems, Symposium on Principles and Practice of Parallel Programming (PPoPP'99), 1999.

[9] Parry Husbands, James C. Hoe, MPI-StarT: Delivering Network Performance to Numerical Applications, Proceedings of the 1998 ACM/IEEE SC98 Conference (SC'98).

[10] V. Tipparaju, J. Nieplocha, and D. K. Panda. Fast Collective operations using Shared and Remote memory Access Protocols on Clusters. In International Parallel and Distributed Processing Symposium, 2003.

[11] M.-S. Wu, R. A. Kendall, and K. Wright. Optimizing Collective Communications on SMP clusters. In ICPP' 2005.

[12] L. Chai, A. Hartono, and D. K. Panda. Designing High Performance and Scalable MPI Intra-node Communication Support for Clusters. In The IEEE International Conference on Cluster Computing, 2006.

[13] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in MPICH. International Journal of High-Performance Computing Applications, (19)1:49–66, Spring 2005.

[14] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro etc. The Landscape of Parallel Computing Research: A View from Berkeley. Electrical Engineering and Computer Sciences, University of California at Berkeley. Technical Report No: UCB/EECS-2006-183, 2006, 12.

[15] Lei Chai, Qi Gao, Dhabaleswar K. Panda. Understanding the Impact of Multi-Core Architecture in Cluster Computing: A Case Study with Intel Dual-Core System. Seventh IEEE International Symposium on Cluster Computing and the Grid(CCGrid'07), pp. 471-478, 2007.

[16] Matthew Curtis-Maury, Xiaoning Ding, Christos D. Antonopoulos and Dimitrios S. Nikolopoulos, An Evaluation of OpenMP on Current and Emerging Multithreaded/Multicore Processors. In IWOMP, 2005.