Process Mapping for MPI Collective Communications*

Jin Zhang, Jidong Zhai, Wenguang Chen, and Weimin Zheng

Department of Computer Science and Technology, Tsinghua University, China
{jin-zhang02,dijd03}@mails.tsinghua.edu.cn,
{cwg,zwm-dcs}@tsinghua.edu.cn

Abstract. It is an important problem to map virtual parallel processes to physical processors (or cores) in an optimized way to get scalable performance due to non-uniform communication cost in modern parallel computers. Existing work uses profile-guided approaches to optimize mapping schemes to minimize the cost of point-to-point communications automatically. However, these approaches cannot deal with collective communications and may get sub-optimal mappings for applications with collective communications.

In this paper, we propose an approach called OPP (Optimized Process Placement) to handle collective communications which transforms collective communications into a series of point-to-point communication operations according to the implementation of collective communications in communication libraries. Then we can use existing approaches to find optimized mapping schemes which are optimized for both point-to-point and collective communications.

We evaluated the performance of our approach with micro-benchmarks which include all MPI collective communications, NAS Parallel Benchmark suite and three other applications. Experimental results show that the optimized process placement generated by our approach can achieve significant speedup.

1 Introduction

Modern parallel computers, such as SMP (Symmetric Multi-Processor) clusters, multi-clusters and BlueGene/L-like supercomputers, exhibit non-uniform communication cost. For example, in SMP clusters, intra-node communication is usually much faster than inter-node communication. In multi-clusters, the bandwidth among nodes inside a single cluster is normally much higher than the bandwidth between two clusters. Thus, it is important to map virtual parallel processes to physical processors (or cores) in an optimized way to get scalable performance.

For the purpose of illustration, we focus on the problem of optimized process mapping for MPI (Message Passing Interface) applications on SMP clusters in this paper[1].

The problem of process mapping can be formalized to a graph mapping problem which finds the optimized mapping between the communication graph of applications

* This work is supported by Chinese National 973 Basic Research Program under Grant No. 2007CB310900 and National High-Tech Research and Development Plan of China (863 plan) under Grant No. 2006AA01A105.

[1] *MPI ranks* are used commonly to indicate MPI processes. We use terms *MPI ranks* and *MPI processes* interchangeably.

H. Sips, D. Epema, and H.-X. Lin (Eds.): Euro-Par 2009, LNCS 5704, pp. 81–92, 2009.

and the topology graph of the underlying parallel computer systems. Existing research work, such as MPI/SX [1], MPI-VMI [2] and MPIPP [3], addresses this problem by finding optimized process mapping for **point-to-point communications**. However, they all ignore **collective communications** which are also quite sensitive to process mapping.

In this paper, we propose a way to optimize process mapping for collective communications. Our approach called OPP is based on the observation that most collective communications are implemented through a series of point-to-point communications. Thus we can transform collective communications into a series of point-to-point communications according to their implementation in communication libraries. Then we can use the existing framework [3] to find out the optimized process mapping for whole applications.

The contributions of this paper can be summarized as follows:

- A method to find optimized mapping scheme for a given collective operation by decomposing it to a series of point-to-point communications.
- Integration of the above method with existing process mapping research work to obtain optimized process mapping for whole parallel applications which have both point-to-point communications and collective communications.
- We perform extensive experiments with micro-benchmarks, the NAS Parallel Benchmark suite (NPB) [4] and three other applications to demonstrate the effectiveness of our method.

Our paper is organized as follows, in Section 2 we discuss the related work, and Section 3 describes the brief framework about process placement mechanism. Section 4 introduces the method to generate communication topology of parallel applications. And the experimental environment and experimental results are shown in Section 5 and Section 6. We discuss the interaction between process mapping and collective communication optimization and propose an alternative way to deal with the process placement problem in Section 7. Conclusion is finally made in Section 8.

2 Related Works

Various process mapping approaches have been proposed to optimize the communication performance for message passing applications in SMP clusters and multi-clusters [5, 6, 3]. MPICH-VMI [2] proposes a profile-guided approach to obtain the application communication topology, and uses general graph partitioning algorithm to find optimized mapping from parallel processes to processors. But MPICH-VMI requires users to provide the network topology of the target platform. MPIPP [3] makes the mapping procedure more automatically by employing a tool to probe the hardware topology graph so that it can generate optimized mapping without users' knowledge on either applications or target systems. MPIPP also proposes a new mapping algorithm which is more effective for multi-clusters than previous work. Topology mapping on BlueGene/L has been studied in [7, 8] which describe a comprehensive topology mapping library for mapping MPI processes onto physical processors with three-dimensional grid/torus topology. However, none of the above work handles the problem of optimizing process mapping for collective communications, and may get sub-optimal process mapping results for applications with collective communications.

Much work has been done on optimized implementations of collective communications. For example, Magpie [9] is a collective communication library optimized for wide area systems. Sanders et al.[10], Sistare et al.[11] and Tipparaju et al.[12] discuss various approaches to optimize collective communication algorithms for SMP clusters. Some work focuses on using different algorithms for different message size, such as [13, 14]. None of previous work shows how it interacts with existing process placement approaches which are based on point-to-point communications and may also result in sub-optimal mappings.

Our work, to the best of our knowledge, is the first one to obtain optimized process mapping for applications with both collective communications and point-to-point communications.

3 The Framework of Process Placement

In this section, we illustrate the framework of process placement. In general, the process placement algorithm takes parameters from target systems and parallel applications, and outputs the process placement scheme, as illustrated in Figure 1.

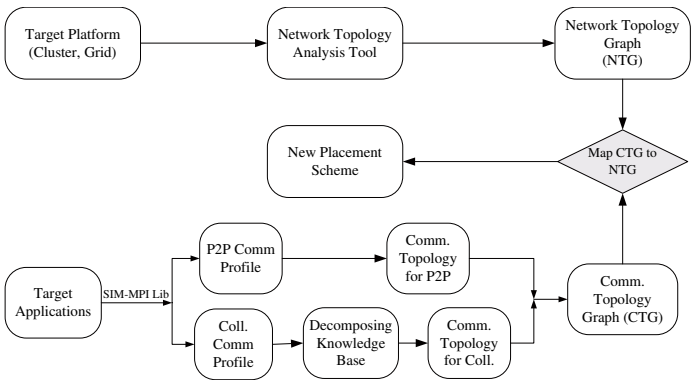

Fig. 1. The framework of our process placement method

Target systems are modeled with network topology graphs (NTGs) describing bandwidth and latency between processors (cores), which are obtained with a network topology analysis tool automatically. The tool is implemented with a parallel ping-pong test benchmark which is similar to the one used in MPIPP [3]. Two $M \times M$ matrices are used to represent network topology graphs, where M is the number of processor cores in the target system: (1) NTG_{bw} describes the communication bandwidth and (2) $NTG_{latency}$ describes the communication latency between each pair of two processor cores. We adopt the method used in MVAPICH [15] to measure and calculate latency and bandwidth between two processor cores.

Parallel applications are characterized with communication topology graphs (CTGs) which include both message count and message volume between any pair of MPI

ranks. We process point-to-point communications and collective communications separately. For point-to-point communications, we use two matrices, CTG_{p2p_count} and CTG_{p2p_volume}, to represent the number or aggregated volume of point-to-point communications between rank i and j in a parallel application respectively (Please refer to [3] for details). For collective operations, we propose a method to translate all collective communications into point-to-point communications, which will be described in detail in Section 4. Now assuming we have translated all collective communications into a series of point-to-point communications, we can generate the following two matrices CTG_{coll_count} and CTG_{coll_volume} in which element (i, j) represents the number or volume of **translated** point-to-point communications from **collective communications** between rank i and j respectively. Then the communication topology of the whole application can be represented by two matrices which demonstrate message count and message volume for both collective and point-to-point communications:

$$CTG_{app\_count} = CTG_{coll\_count} + CTG_{p2p\_count}$$
$$CTG_{app\_volume} = CTG_{coll\_volume} + CTG_{p2p\_volume}$$

We feed the network topology graphs and communication topology graphs to a graph partitioning algorithm to get the optimized process placement. In our implementation, we use the heuristic k-way graph partitioning algorithm proposed in [3] which gives a detailed description for its implementation and performance.

4 Communication Topology Graphs of Collective Communications

In this section, we introduce our approach to decompose collective communications into point-to-point communications. We first use $MPI\_Alltoall$ as a case study, then we show the construction of Decomposition Knowledge Base (DKB) which can be employed to transform every MPI collective communications into point-to-point communications.

4.1 A Case Study: $MPI\_Alltoall$

One implementation of $MPI\_Alltoall$ is the Bruck Algorithm [16], as shown in Figure 2. At the beginning, rank i rotates its data up by i blocks. In each communication step k, process i sends to rank$(i + 2^k)$ all those data blocks whose kth bit is 1, receives data from rank$(i - 2^k)$. After a total of $\lceil \log P \rceil$ steps, all the data get routed to the right destination process. A final step is that each process does a local inverse shift to place the data in the right order.

For an $MPI\_Alltoall$ instance on 8 MPI ranks, we can decompose it in three steps as illustrated in Figure 2. Assuming the message size of each item is 10 byte, then $step\,0$ can be decomposed into 8 point-to-point communications whose message sizes are all 40 bytes. The second and third steps can also be decomposed into 8 point-to-point communications of 40 bytes respectively. Finally, we decompose the $MPI\_Alltoall$ into 24 different point-to-point communications. We aggregate the volume and number of messages between each pair of the MPI ranks and get the communication graphs (CTG_{coll_count} and CTG_{coll_volume}), which are first introduced in Section 3. Figure 3 shows the CTG_{coll_volume} for this $MPI\_Alltoall$ instance.

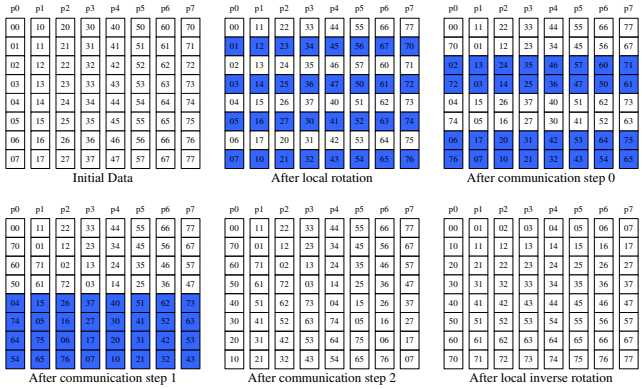

Fig. 2. Bruck Algorithm for $MPI\_Alltoall$ with 8 MPI ranks. The number (ij) in each box represents the data to be sent from rank i to rank j. The shaded boxes indicates the data to be communicated in the next step.

P7	40	40	0	80	0	40	40	0
P6	40	0	80	0	40	40	0	40
P5	0	80	0	40	40	0	40	40
P4	80	0	40	40	0	40	40	0
P3	0	40	40	0	40	40	0	80
P2	40	40	0	40	40	0	80	0
P1	40	0	40	40	0	80	0	40
P0	0	40	40	0	80	0	40	40
	P0	P1	P2	P3	P4	P5	P6	P7

Fig. 3. Decomposition results for Bruck Algorithm of $MPI\_Alltoall$ with 8 MPI ranks. The value of the block (i, j) represents the communication volume between the process i and the process j during the process of collective communications.

4.2 Decomposition Knowledge Base

The previous section shows how we can decompose $MPI\_Alltoall$ to point-to-point communications and generate its communication graphs. The same approach can be applied to other MPI collection communications too.

One of the challenge to decompose MPI collective communications is that they are implemented in different ways for different MPI libraries. What's more, even in the same MPI library, the algorithms used to implement a certain collective communication may depend on the size of messages and the number of processes.

To correctly identify implementation algorithms for each collective communication, we build a Decomposing Knowledge Base (DKB) which records the rules to map collective communications to its implementation algorithms. Through analyzing the MPICH-1.2.7 [18] code and MVAPICH-0.9.2 [15] code manually, we get the underlying

Name	MPICH-1.2.7	MVAPICH-0.9.2
Barrier	Recursive Doubling	Recursive Doubling
Bcast	Binomial Tree (SIZE<12288bytes or NP<8)	Binomial Tree (SIZE<12288bytes and NP≤8)
	Van De Geijn (SIZE<524288bytes, power-of-two MPI processes and NP≥8)	Van De Geijn (the other conditions)
	Ring (the other conditions)	
Allgather	Recursive Doubling (SIZE*NP<524288bytes and power-of-two MPI processes)	Recursive Doubling
	Bruck (SIZE*NP<81920bytes and non-power-of-two MPI processes)	
	Ring (the other conditions)	
Allgatherv	Recursive Doubling (SIZE*NP<524288bytes and power-of-two MPI processes)	Recursive Doubling (TOTAL_SIZE≤262144bytes)
	Bruck (SIZE*NP<81920bytes and non-power-of-two MPI processes)	Ring(the other conditions)
	Ring (the other conditions)	
Gather	Minimum Spanning Tree	Minimum Spanning Tree
Reduce	Rabenseifner (SIZE>2048bytes and OP is permanent.)	Recursive Doubling
	Binomial Tree (the other conditions)	
Allreduce	Recursive Doubling (SIZE≤2048bytes or OP is not permanent.)	Recursive Doubling
	Rabenseifner (the other conditions)	
Alltoall	Bruck (SIZE≤256bytes and NP≥8)	Recursive Doubling (SIZE≤128bytes)
	Isend_Irecv (256bytes≤SIZE≤32768bytes)	Isend_Irecv (128bytes<SIZE<262144)
	Pairwise Exchange (the other conditions)	Pairwise Exchange (the other conditions)
Scatter	Minimum Spanning Tree	Minimum Spanning Tree
Scatterv	Linear	Linear
Gatherv	Linear	Linear
Alltoallv	Isend_Irecv	Isend_Irecv

Fig. 4. Classify algorithms used in MPICH and MVAPICH. SIZE represents the communication size. NP represents the number of MPI processes. OP represents the operation used in MPI_Allreduce or MPI_Reduce and TOTAL_SIZE used for MPI_Allgatherv represents the sum of sizes received from all the other processes (Van de Geijn and Rabenseifner's algorithms are proposed in [13] and [17] respectively.).

collective communication algorithms listed in Figure 4. This table presents collective communication algorithms used in different collective communications. The DKB we built is a essentially similar table which can output the decomposition algorithms used by a specified MPI collective communication depending on the interconnect, the MPI implementation, message sizes and the number of processes. Our current DKB covers MPICH-1.2.7 and MVAPICH-0.9.2 for both Ethernet and Infiniband networks.

With the help of DKB, we can generate the communication topology graph of any collective communications, as shown in Figure 5. We input a collective communication to DKB, which include the type of the collective communication, Root_ID, Send_Buf_Size and Communicator_ID. DKB outputs the algorithm used in this

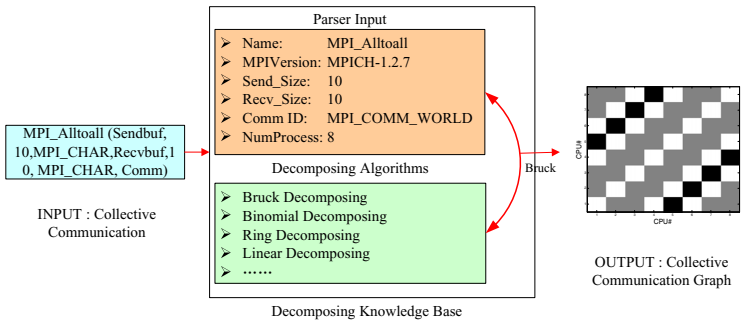

Fig. 5. Usage of decomposing knowledge base

collective communication. We then decompose the collective communication into a series of point-to-point communications based on its implementation. These point-to-point communications can then be used to generate the communication topology graph of this collective communication. We process the collective communications one by one, and aggregate their communication topology graphs to obtain the communication topology graphs for all the collective communications (CTG_{coll_count} and CTG_{coll_volume} defined in Section 3) of the whole application. Then we can perform process mapping according to steps described in Section 3.

5 Experiment Platforms and Benchmarks

5.1 Experiment Platforms

We perform experiments on a 16-node cluster in which each node is a 2-way server with 4GB memory. The processors in this cluster are 1.6GHz Intel Xeon dual core processors with 4MB L2 cache. Linux 2.6.9 and MPICH-1.2.7 [18] are installed on each node. These nodes are connected by a 1Gbps Ethernet. Since there are 64 cores in the cluster, we execute up to 64-rank MPI applications and all applications are compiled with Intel compiler 9.0.

The cluster exhibits non-uniform communication cost between cores. We use our network analysis tool to get the network topology graphs for our experimental network platform. Figure 6 and Figure 7 show the latency and bandwidth between each pair of cores in this cluster. We see that communication inside a node is much faster than communication between nodes.

5.2 Benchmarks and Applications

We use the following benchmarks and applications to verify the effect of our optimized process placement scheme.

1. Intel MPI Benchmark (IMB) [19]
   IMB is a micro-benchmark developed by Intel. We use it to verify that we can find optimized process placement for each MPI collective communication.

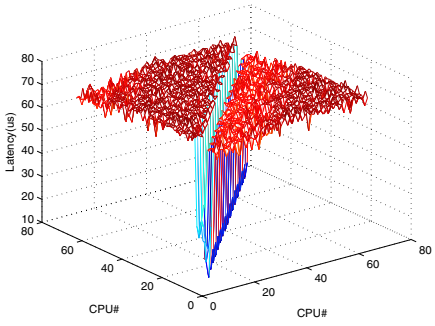

Fig. 6. Latency between cores

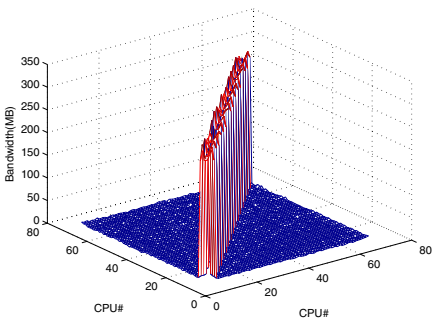

Fig. 7. Bandwidth between cores

2. NAS Parallel Benchmark (NPB) [4]
   The NAS Parallel Benchmarks (NPB) are a set of scientific computation programs. In this paper, we use NPB 3.2 and Class C data set.
3. Three other applications
   - ASP [9]
     A parallel application which solves the all-pairs-shortest-path problem with the Floyd-Warshall algorithm. It has been integrated into Magpie [9].
   - GE [20]
     A message passing implementation of the Gauss Elimination. It is an efficient algorithm for solving systems of linear equations.
   - PAPSM [21]
     This is a parallel application which implements the realtime dynamic simulation of power systems. The application is implemented by a hierarchical Block Bordered Diagonal Form (BBDF) algorithm for power network computation.

6 Experiment Results and Analysis

In this section, we compare our process mapping approach *OPP* with two widely used process placement scheme in MPI, $block$ and $cyclic$ [22], as well as the most up-to-date optimized process mapping approach, *MPIPP* [3]. Each result is the average of five executions and normalized by the result of $block$ scheme.

6.1 Micro Benchmarks

We use IMB[2] to evaluate how OPP outperforms $block$ and $cyclic$ for each individual collective communications. The result of MPIPP is not presented because MPIPP does not deal with collective communications at all.

Figures 8–15 exhibit the results of OPP, $block$ and $cyclic$ process placement for different collective communications on 64 MPI ranks. The results show:

- There are a few collective communications such as $Reduce$, $Allreduce$, on which the $block$ scheme always outperforms the $cyclic$ scheme, while there are a few others such as $Gather$ and $Scatter$, on which the $cyclic$ scheme always outperforms the $block$ scheme.
- For some collective communications, such as $Bcast$ and $Allgather$, neither the $block$ nor $cyclic$ scheme can always outperform the other because their relative performance depends on the size of messages.
- Our proposed approach, *OPP*, can always find the best placement scheme comparing to the $block$ and $cyclic$ for all collective communications on all message sizes.

[2] IMB does not provide the test for MPI_Gather and $MPI\_Scatter$. We design a micro-benchmark to test them.

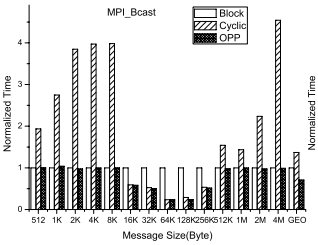

Fig. 8. Bcast (NP=64)

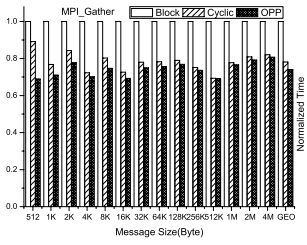

Fig. 9. Gather (NP=64)

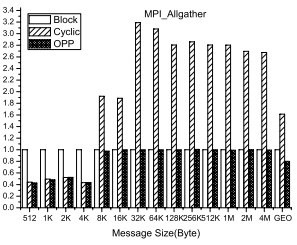

Fig. 10. AllGather (NP=64)

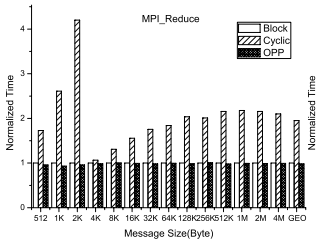

Fig. 11. Reduce (NP=64)

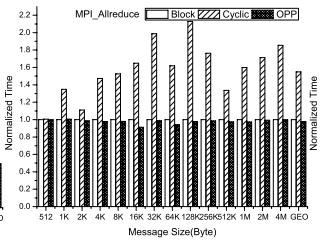

Fig. 12. Allreduce (NP=64)

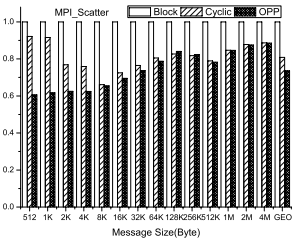

Fig. 13. Scatter (NP=64)

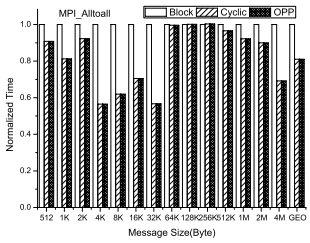

Fig. 14. Alltoall (NP=64)

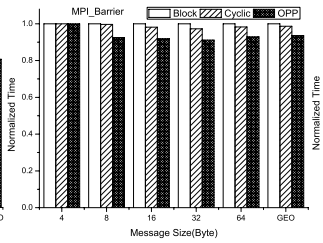

Fig. 15. Barrier (NP=64)

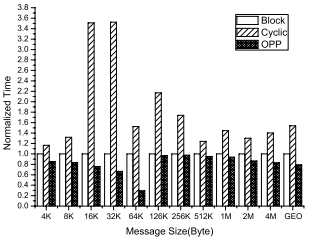

Fig. 16. Allreduce (NP=33)

Non-Power-of-Two MPI Ranks. We take a further investigation on process placement for non-power-of-two MPI ranks. Some collective communication algorithms favor more on the situation of the power-of-two MPI ranks because they are symmetric in design, so their performance may degrade when the number of ranks is not power-of-two.

We show how OPP can improve performance in these situations. Due to space limitation , we only take $MPI\_Allreduce$ as an example. Figure 16 exhibits the result of $MPI\_Allreduce$ with 33 MPI ranks.

Figure 16 shows that *OPP* performs significantly better than the $block$ and $cyclic$ placement schemes for all tested message size. OPP is 20.4% better than the block placement and 53.6% better than the cyclic placement on average.

The result indicates that OPP has more benefits for applications with collective communications which require running with non-power-of-two MPI ranks.

6.2 NPB and Other Applications

In this section, we perform experiments with NPB and three parallel applications, ASP, GE and PAPSM as described in Section 5.2. The results are obtained with 64 MPI ranks except for PAPSM which only supports up to 20 MPI ranks. The results are shown in Table 1. For applications which only contain collective communications, such as ft, PAPSM and ASP, performance of MPIPP is not applicable since it can only find optimized placement for point-to-point communications.

By examining the results in Table 1, we can see that:

- For applications dominated by point-to-point communications, such as *bt, cg, sp* and *mg*, both MPIPP and OPP can get better performance than *block* and *cyclic* schemes. MPI and OPP are equally good for this category of applications.
- For applications that only contain collective communications, such as *ft, PAPSM* and *ASP*, MPIPP can not get optimized rank placement because it does not deal with collective communications. So we can only compare OPP with *block* and *cyclic* schemes. We see that OPP shows its capability to find optimized MPI rank placement scheme for this class of applications which can get up to 26% performance gain over the best of block or cyclic schemes.
- *IS* and *GE* are applications that have both point-to-point and collective communications, but are dominated by collective communications. MPIPP decides the MPI rank placement based on the point-to-point communication patterns and get suboptimal placement schemes. MPIPP is 6.0% and 5.1% worse than the best of $block$ or $cyclic$ scheme for IS and GE respectively. On the contrary, OPP can find optimized layout for both point-to-point and collective communications, which is 0.1% and 19% better in these two applications.

In summary, OPP shows that it can find optimized MPI process placement for all three classes of parallel applications.

Table 1. The execution time (in seconds) of NPB suite and three parallel programs with different placement schemes

Name	Block(s)	Cyclic(s)	MPIPP(s)	OPP(s)	Speedup of OPP vs. Block
bt.C.64	95.47	103.76	90.79	90.66	1.05
cg.C.64	64.14	89.03	62.3	62.3	1.03
ep.C.64	11.12	11.12	11.12	11.09	1.00
lu.C.64	69.32	74.39	68.72	68.72	1.01
sp.C.64	143.97	151.40	132.12	132.12	1.09
mg.C.64	9.56	9.12	9.11	9.10	1.06
is.C.64	12.59	12.14	12.87	12.13	1.04
ft.C.64	31.52	23.20	N.A.	22.89	1.38
PAPSM.20	15.78	19.45	N.A.	12.54	1.26
GE.64	20.83	25.14	21.89	17.48	1.19
ASP.64	55.93	50.46	N.A.	50.16	1.11

7 Discussion

An interesting issue is the interaction between MPI process placement and optimized collective communication implementation. In our current scheme, we first fix the collective communication implementation according to the MPI library, then perform process placement optimization based on this implementation. This is a reasonable choice if we wish our MPI process placement approach to be compatible with existing MPI libraries.

An alternative approach to deal with this problem is to fix the process placement based on the point-to-point communication pattern of a parallel application first, then determine the optimized collective communication implementation for the given process placement scheme. This approach has the potential of achieving better performance, but loses the compatibility because it only works with MPI libraries which are process placement aware. Nevertheless, we believe this is a promising way to go.

8 Conclusions

In this paper, we argue that it is an important problem to map virtual parallel processes to physical processors (or cores) in an optimized way to get scalable performance due to non-uniform communication cost in modern parallel computers. Existing work either determines optimized process mapping based on point-to-point communication patterns or optimizes collective communications only without awareness of point-to-point communication patterns in parallel applications. Thus they may all fall into sub-optimal placement results.

To solve the problem, we propose a method which first decomposes a given collective communication into a series of point-to-point communications based on its implementation in the MPI library that are used in the target machine. Then we generate the communication patterns of the whole application by aggregating all collective and point-to-point communications in it. We then use a graph partition algorithm to find optimized process mapping schemes.

We perform extensive experiments on each single MPI collective communications and 11 parallel applications with different communication characteristics. Results show that our method (OPP) can get best results in all cases, and perform significantly better than previous work for applications with both point-to-point and collective communications.

References

[1] Colwell, R.R.: From terabytes to insights. Commun. ACM 46(7), 25–27 (2003)
[2] Pant, A., Jafri, H.: Communicating efficiently on cluster based grids with MPICH-VMI. In: CLUSTER, pp. 23–33 (2004)
[3] Chen, H., Chen, W., Huang, J., Robert, B., Kuhn, H.: MPIPP: an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters. In: ICS, pp. 353–360 (2006)
[4] NASA Ames Research Center. NAS parallel benchmark NPB, `http://www.nas.nasa.gov/Resources/Software/npb.html`
[5] Phinjaroenphan, P., Bevinakoppa, S., Zeephongsekul, P.: A heuristic algorithm for mapping parallel applications on computational grids. In: EGC, pp. 1086–1096 (2005)

[6] Sanyal, S., Jain, A., Das, S.K., Biswas, R.: A hierarchical and distributed approach for mapping large applications to heterogeneous grids using genetic algorithms. In: CLUSTER, pp. 496–499 (2003)
[7] Bhanot, G., Gara, A., Heidelberger, P., Lawless, E., Sexton, J., Walkup, R.: Optimizing task layout on the Blue Gene/L supercomputer. IBM Journal of Research and Development 49(2-3), 489–500 (2005)
[8] Yu, H., Chung, I., Moreira, J.: Topology mapping for Blue Gene/L supercomputer. In: SC, pp. 52–64 (2006)
[9] Kielmann, T., Hofman, R.F.H., Bal, H.E., Plaat, A., Bhoedjang, R.: MagPIe: MPI's collective communication operations for clustered wide area systems. In: PPOPP (1999)
[10] Sanders, P., Traff, J.L.: The hierarchical factor algorithm for all-to-all communication (research note). In: Monien, B., Feldmann, R.L. (eds.) Euro-Par 2002. LNCS, vol. 2400, pp. 799–804. Springer, Heidelberg (2002)
[11] Sistare, S., vande Vaart, R., Loh, E.: Optimization of MPI collectives on clusters of large-scale SMP's. In: SC, pp. 23–36 (1999)
[12] Tipparaju, V., Nieplocha, J., Panda, D.K.: Fast collective operations using shared and remote memory access protocols on clusters. In: IPDPS, pp. 84–93 (2003)
[13] Barnett, M., Gupta, S., Payne, D.G., Shuler, L., van de Geijn, R., Watts, J.: Interprocessor collective communication library (InterCom). In: SHPCC, pp. 357–364 (1994)
[14] Kalé, L.V., Kumar, S., Varadarajan, K.: A framework for collective personalized communication. In: IPDPS, pp. 69–77 (2003)
[15] Ohio State University. MVAPICH: MPI over infiniband and iWARP, http://mvapich.cse.ohio-state.edu
[16] Bruck, J., Ho, C., Upfal, E., Kipnis, S., Weathersby, D.: Efficient algorithms for all-to-all communications in multiport message-passing systems. IEEE Trans. Parallel Distrib. 8(11), 1143–1156 (1997)
[17] Rabenseifner, R.: New optimized MPI reduce algorithm, http://www.hlrs.de/organization/par/services/models/mpi/myreduce.html
[18] Argonne National Laboratory. MPICH1, http://www-unix.mcs.anl.gov/mpi/mpich1
[19] Intel Ltd. Intel IMB benchmark, http://www.intel.com/cd/software/products/asmo-na/eng/219848.htm
[20] Huang, Z., Purvis, M.K., Werstein, P.: Performance evaluation of view-oriented parallel programming. In: ICPP, pp. 251–258 (2005)
[21] Xue, W., Shu, J., Wu, Y., Zheng, W.: Parallel algorithm and implementation for realtime dynamic simulation of power system. In: ICPP, pp. 137–144 (2005)
[22] Hewlett-Packard Development Company. HP-MPI user's guide, http://docs.hp.com/en/B6060-96024/ch03s12.html