GLEX-Alltoall: Overlapped Numa-aware Multi-leader Multi-port alltoall algorithm on Multi-core Supercomputer

Jintao Peng[1], Jie Liu[2,*], Min Xie[3]

Changsha, China

Abstract

All-to-all communication is commonly used in parallel applications like FFT. In mordern supercomputers, there are multiple cores in a node. To optimize all-to-all communication, a typical way is gather-scatter-based alltoall which aggregate messages on each pair of nodes. Multiple cores, NUMAs and network endpoints bring much parallelism. However, there is no method which makes uses these parallelism to improve the all-to-all communication. In this paper, we introduce Overlapped Numa-aware Multi-leader Multi-port alltoall (ONMPML) collectives which explore the parallelism on network, CPU cores and overlap the intra- and inter-node communication. The results show that, compared to MPI, our library achieves up to 20x speedup. For application, our method achieves up to 17x speedup on peak performance for 16384 cores.

Keywords: Collective Communication, Multi-core processor, MPI all-to-all, RDMA, Shared Heap
2010 MSC: 00-01, 99-00

1. Introduction

Many parallel applications may suffer from global communication. Especially for communication-intensive applications, their time-to-solution and scalablily may be affected by global communication. Message Passing Interface (MPI) provides a set of commonly used collective communication. MPI_Alltoall is one of the collective communication where each process will send a different message to all processes. It is broadly used in some parallel applications like Fast Fourier Transform (FFT) [1] and some graph algorithms like MapReduce [2] and Breadth-first search (BFS) [3]. However, each time we double the processes, the communication workload of all-to-all is quadrupled. On mordern supercomputers, network throughput has a linear relationship with the number of nodes. This brings great challenges to design large-scale all-to-all collectives.

For multiple-core supercomputers, an effective way is node-aware all-to-all method [4]. A p (p=N times M, M is #processes on a node) processes all-to-all is replaeced by a N node inter-node all-to-all. Which is: N-1 times intra-node gather + local transpose + inter-node transpose + N-1 times intra-node gather. This method is very effective for small messages. Because, compared to original method, a node-aware all-to-all reduce the number of inter-node messages from $(M^N)^2$ to N^2 times. The size of the message is increased by M^2 times, which can effective use of the network bandwidth. On the current supercomputer, there may be 20-64 CPU cores in a node. The single message size can be scaled 400-4096 times. In production environment, node-aware all-to-all may more efficient than Bruck's algorithm.

However, the gathering/scattering overhead on node-aware all-to-all make it hard to be scaled to larger messages. Multiple CPU cores, NUMA and network endpoints brings 4 kinds of parallism to optimize a node-aware all-to-all method:

(1) Multiple network endpoints can simultaneously process multiple communication requests.
(2) Processes in different NUMA can simultaneously access it local memory without contention.
(3) Multiple processes can simultaneously gather/scatter data and compose communication requests.
(4) Inter-node communication can be overlapped with intra-node communication.

As we known, no methods combine these parallism togather to improve a node-aware all-to-all collective communication.

In this paper, we proposed Overlapped NUMA-aware Multi-port Multi-leader (ONMPML) method to scale the node-aware all-to-all collective. ONMPML using multiple leaders on different NUMA to gather/scatter data, compose communication requests, and transpose local matrix and open the different network endpoints to do inter-node all-to-all. ONMPML explore the parallelism existing in mordern multi-core processor with NUMA memory architecture and multi-port network. For intra-node gather/sactter, we implemented a shared-heap-based remote accessible memory to support multiple leader to gather/scatter simutaneously. Inter-node communication is implemented on Remote Direct Memory Access (RDMA) which provide high throughput and low latency. The results show that, compared to MPI_alltoall, our implementation achieves up to 18x speedup and 4x speedup on average.

*Corresponding author
[1]JintaoPengCS@gmail.com
[2]liujie@nudt.edu.cn
[3]xiemin@nudt.edu.cn

Table 1: Overall design-space for all-to-all collective on mulit-core processers

Approach/Metric	SH-NUMA-CO	SA-orig	elan_alltoall	PAIRWISE-SLR and BRUCK-SLR	ONMPML (ours)
inter-node implementation	MVAPICH2	MVAPICH	RDMA	MPI-P2P	RDMA
intra-node implementation	Shared Heap	Shared Memory	Shared Memory	MPI-P2P	Shared Heap
Leader-based aggregation support	✔	✔	✔	✔	✔
Multi-leader support	✖	✔	✖	✖	✔
NUMA-aware	✔	✖	✖	✖	✔
Multiple network endpoints	✖	✔	✔	✖	✔
Overlapping Inter/intra-node	✔	✖	✖	✔	✔

2. Related Work

Bruck algorithm [5] is commonly used for small message all-to-all. For mid size messages, isend-irecv algorithm is used. Linear shift exchange [6], pairwise exchange[7] apply to large messages.

When considering the multi-core processors: Cache-oblivious MPI all-to-all (SH-NUMA-CO) based on morton order is proposed to minimize the cache miss rate [8]. For Infiniband on multi-core systems, a non-leader all-to-all collective (SA-orig) which based on shared memory aggregation techniques is proposed in [9]. For multi-rail QsNet SMP clusters, a shared memory and RDMA based all-to-all collectives (elan_alltoall) is proposed in [10]. For Intel Many Integrated Core (MIC) architecture, the re-routing scheme based all-to-all collective (PAIRWISE-SLR/BRUCK-SLR) is proposed in [11]. As these works are all support leader-based aggreagation. Table 1 shows the overall design-space comparation between these methods and our method.

Besides, to improve the intra-node communication, several kernel-assistant techniques have been proposed for multi-core processor (LIMIC [12], KNEM [13], XPMEM). Compared to shared memory, these method works well for large message intra-node communication. Because they avoid one memory copy overhead. Shared heap [14] has the same performance with these kernel-assistant techniques. Shared heap do not need extra kernel module.

When considering the network topology: A bandwidth-optimal all-to-all exchange is proposed for fat-tree network [15]. For torus network, a large scale all-to-all is proposed for Blue Gene/L Supercomputer [16]. A optimal schedule for all-to-all personalized communication is proposed for multiprocessor systems [17]. For Infiniband clusters, their is a topology aware all-to-all scheduler which lower the contention by adding extra communication steps [18]. A L-level hierarchy method which based on mapping of intensively communicating processes into the same computer nodes is proposed in [19]. A method which considering dynamic performance model of the network is proposed at [20].

3. Experimental Platforms

The methods in this paper are validated on two different systems:

(1) HPC-A: Matrix-2000+ CPU on each node, Tianhe series network.
(2) HPC-B: E5-2692 v2 CPU * 2 on each node, Tianhe series network.

The memory bandwidth test (Stream) and network bandwidth test results are shown in the Table 2. In table 2, copy bandwidth is tested by STREAM. Loopback bandwidth means where sending process and receiving process is placed on a same node. Bi-loopback bandwidth is the bidirectional bandwidth of two processes on a same node.

4. Motivation

Node-aware all-to-all is a traditional method to optimize small message all-to-all. Node-aware all-to-all will aggregate messages between each pair of node to reduce message number.

As shown in Figure 1.

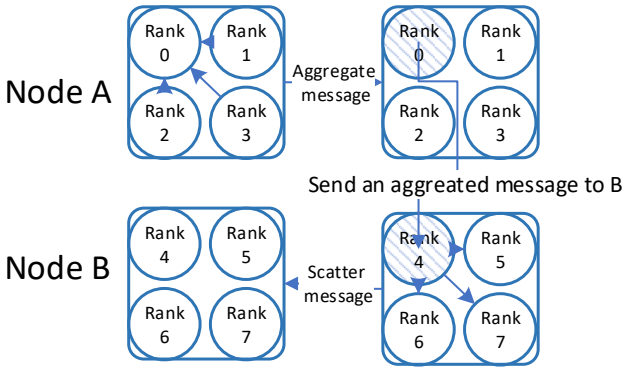
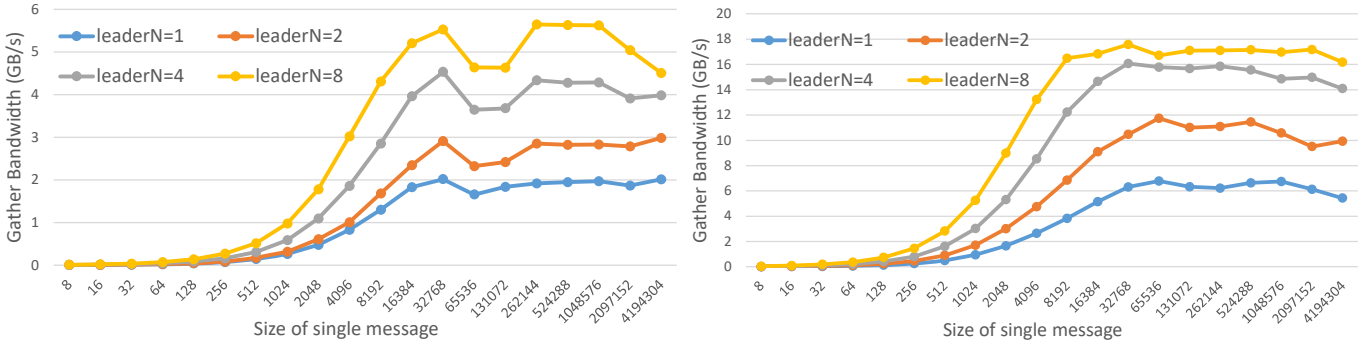


Figure 1: Multi-leader gahtering/scattering throughput based on MPI-RMA, Leader Number ∈ [1,2,4,8]

It include four steps: intra-node gather, local transpose, inter-node all-to-all, intra-node scatter. For a 32/64 cores CPU, each inter-node message in node-aware method will be 1024/4096 times larger than direct method. After aggreatation, the small messages become large messages and message number is reduced. However, when message getting larger, gatering, transpose, and scattering overhead increase with the message size. It limit the application scope of node-aware all-to-all.

Table 2: Memory Bandwidth (tested on Stream) and Point-to-Point Bandwidth (tested on RDMA) of Experimental Platforms

Bandwidth tests (MB/s)	Copy (Single-thread)	Copy (Multi-threads)	NUMA Count	loopback banwdith	bi-loopback banwdith	two node p2p banwdith	two node bip2p banwdith
HPC-A (32 cores)	6481	21351	4	1366	1368	1181	1336
HPC-B (24 cores)	11280	46252	2	11530	11568	7089	11629



(a) HPC-A (b) HPC-B

Figure 2: Multi-leader gahtering/scattering throughput based on MPI-RMA, Leader Number ∈ [1,2,4,8]

With the increase in the number of processor cores, multiple CPU cores, NUMA, network endpoints and inter-/intra-node overlapping brings huge parallelism. We notice that all traditional method do not take advantage of this parallelism to solve the problem. As the single-core memory access bandwidth may not faster than the network bandwidth. Using a single core to gather, scatter, and transpose data, and initialize communication request may causing communication hotspot on a CPU core.

Multi-leader can get faster message aggregation throughput than one-leader gathering/scattering. Results of figure 2 show the multi-leader gathering/scattering throughput compared to one-leader. Notice the result of HPC-B, its one-leader gathering/scattering bandwidth is about 6 GB/s, while it p2p bandwidth is 7.089 GB/s. As a node-aware all-to-all includes: intra-node gather, inter-node all-to-all, intra-node scatter steps. The overhead to aggregate M gigabyte messages is about M/6 second. The overhead to send it to another node is about M/7 second. The overhead to scatter it on is about M/6 second. In this case, intra-node communication become bottelneck of node-aware all-to-all. Using multiple leaders can improve the gathering and scattering thoughput on a node.

With non-uniform memory access (NUMA) systems has emerged, a CPU is composed with multi-socket. When different core try to access a same NUMA, its local memory bandwidth is shared between multiple cores. For a Multi-leader gathering/scattering on these processors, NUMA architecture has performance effect on gathering/scattering performance. Putting multiple leader on the same NUMA or lets leaders uniform distributed on the CPU may have significant performance difference. We bind each processes to corresponding core and compared the NUMA-aware and test the NUMA-aware multi-leader gahtering/scattering Vs multi-leader gahtering/scattering. The performance improvements are shown in Figure **??**. Because the CPU has multiple memory access units for different NUMAs.

Morden interconnect network on supercomputer usually have multiple endpoints for each nodes. Multiple endpoints improve the concurrent ability to process multiple communication requests. We uses two nodes, one nodes activate several processes to concurrently send to corresponding processes on another node. As each process in a nodes will activate a network endpoints, the number of activated processes number in a node is equal to the number of network endpoint in using. As Figure 4 show, we find that multiple endpoints do improve the node-to-node throughput in most cases.

Besides, as the intra-node communication can be overlappable with inter-node communication, gathering/scattering overhead can be overlapped with inter-node overhead.

Although there are a lot of parallelism, there is no method using all these parallelism to optimize the communication performance of all-to-all.

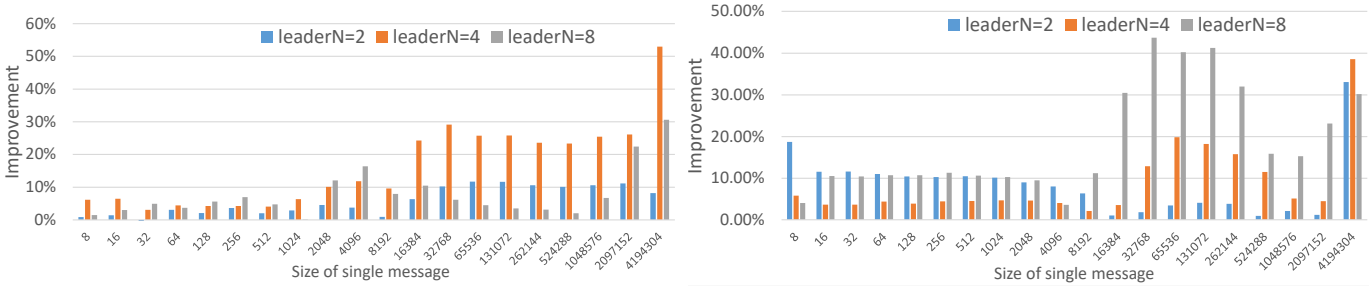
5. Gather-Scatter-based Multi-leader all-to-all algorithm

5.1. Leader-based All-to-All Collective (L-a2a)

In this section, we present the Leader-based all-to-all design. Its algorithm is shown in Algorithm **??**.

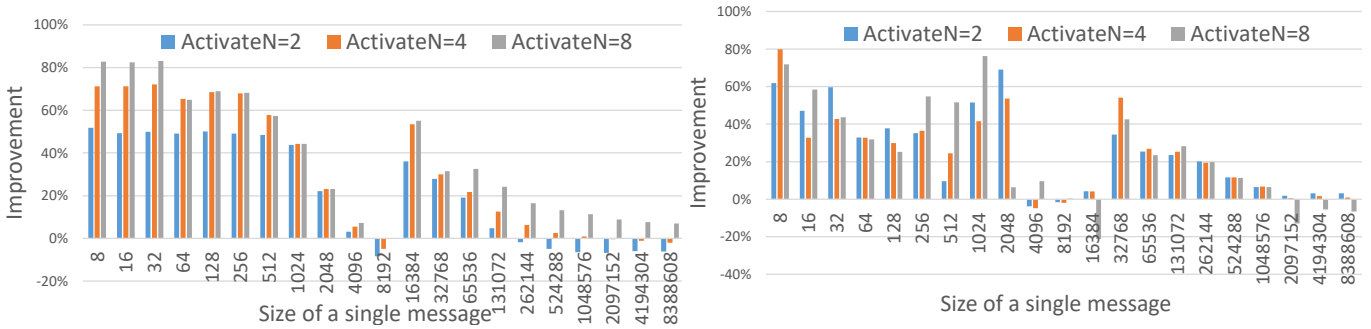
In algorithm 1, the first step is a gathering procedure. In each loop i, all message which need to be sent to node i are gathered into leader ranks of each node and transpose blocks sequentially. The second step is a inter-node all-to-all between node-leaders. In each loop i of the final step, messages which received from node i are scattered to the corresponding processes.

The advantage of leader-based all-to-all collective are:



(a) HPC-A 4 NUMAs in a node (b) HPC-B 2 NUMAs in a node

Figure 3: NUMA-aware multi-leader gahtering/scattering throughput compared to multi-leader gahtering/scattering.



(a) HPC-A 4 NUMAs in a node (b) HPC-B 2 NUMAs in a node

Figure 4: Multiple endpoints delivery throughput compared to single endpoints.

Algorithm 1: Leader-based All-to-All (L-a2a)

```
def All-to-all(SendBuf,RecvBuf,count,type,comm):
    for i in range(0,nodeN) do
        Gather(SendBuf,gather_result,intra_node);
        transpose(gather_result,BufferS)
    end
    if myrank==leader then
        Alltoall(BufferS,BufferR,inter_node)
    end
    for i in range(0,nodeN) do
        Scatter(BufferR,RecvBuf,intra_node)
    end
```

Algorithm 2: Leaders Placement of MPML

```
def am-i-leader(intra-rank):
    if intra-rank < LeaderN then
        return True
    end
    return False
def my-leader-id(intra-rank):
    return intra-rank
```

(1) Compared to direct all-to-all, leader-based all-to-all reduce the number of message by M^2 time (M is the number of processes in each node).

(2) Similar to Bruck algorithm [5], leader-based all-to-all aggregate the small message to large message. But leader-based all-to-all has less memory copy. It makes leader-based all-to-all suitable for medium-sized messages.

5.2. *Multi-port Multi-leader All-to-all Collective (MPML)*

For larger messages, L-a2a involve too many intra-node overhead. As a result, multiple network endpoints and processes can help to push the applicable boundary of gathering/scattering all-to-all futher away. Multi-leader method is using multiple processes in a node to gather/scatter and transpose data. Multi-port method is opening up multiple network endpoints by multiple processes which using multiple endpoints to process different communication requests. MPML algorithm is shown in Algorithm 2 and 3 and Figure 5.

As shown in Figure 5, there are 2 leaders in a node, different processes are responsible for communicating with different nodes in a round-robin way. Assume node A send a aggregated message to Node B where B equal to (A + i) mod NodeN. Leaders i in node A and B are responsible for send and recv the message.

Compared to L-a2a, MPML takes advantage of multi-leader and multi-port to optimize L-a2a. As disscussed in Motiva-

Algorithm 3: Multi-port Multi-leader a2a (MPML)

```
Input: PPN:processes per-node, intra-rank: rank
       within a node, inter-rank: index of nodex
def Multi-leader-gather(SendBuf,count):
    MPI_Win_fence(intra-node)
    if am-i-leader(intra-rank) then
        Lid = my-leader-id(intra-rank)
        for i in range(Lid,nodeN,LeaderN) do
            target-node ← (inter-rank + i) mod nodeN
            k ← i / LeaderN
            shift ← target-node*count*PPN
            for j in range(0,PPN) do
                source ← (intra-rank + j) mod P
                Get(gatherbuf[k][source],source,shift)
            end
        end
    end
    MPI_Win_fence(intra-node)
    Leaders transpose all blocks to BufferS
def Multi-port-a2a(BufferS,BufferR):
    if am-i-leader(intra-rank) then
        Lid = my-leader-id(intra-rank)
        for i in range(Lid,nodeN,LeaderN) do
            k ← i / LeaderN
            target ← (inter-rank + i) mod nodeN
            source ← (inter-rank + nodeN - i) mod
              nodeN
            MPI_Isend(BufferS[k],target)
            MPI_Recv(BufferR[k],source)
        end
    end
def Multi-leader-scatter(Recvbuf,count):
    MPI_Win_fence(intra-node)
    if am-i-leader(intra-rank) then
        Lid = my-leader-id(intra-rank)
        for i in range(Lid,nodeN,LeaderN) do
            source-node ← (inter-rank + nodeN - i)
              mod nodeN
            k ← i / LeaderN
            shift ← source-node*count*PPN
            for j in range(0,PPN) do
                target ← (intra-rank + j) mod P
                Put(BufferR[k][target],target,shift)
            end
        end
    end
    MPI_Win_fence(intra-node)
def all-to-all(SendBuf,RecvBuf,count,type,comm):
    Multi-leader-gather(SendBuf,count,gatherbuf)
    Multi-port-a2a(BufferS,BufferR)
    Local-Transpose(BufferR)
    Multi-leader-scatter(Recvbuf,count)
```

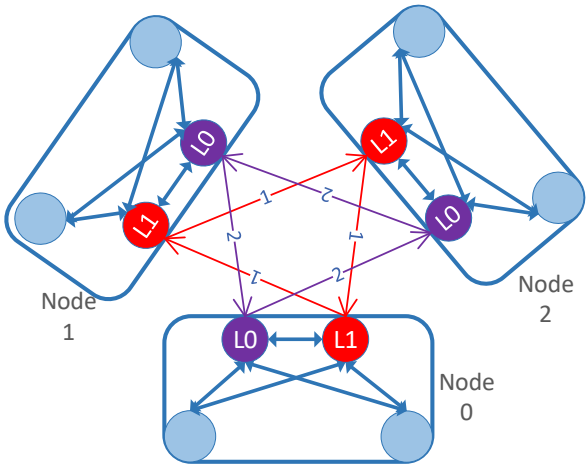
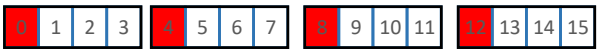


Figure 5: Example of 3 nodes 12 processes 2-leader MPML

(a) 4 leaders on a 4 NUMA 16 core processor



(b) spreaded 4 leaders on a 4 NUMA 16 core processor

Figure 6: NUMA-aware leader placement.

tion part, using multiple processes can improving the gathering and scattering throughput a lot in all message size and HPC we tested. Besides, using multiple processes to transpose the local blocks can parallise the local stransposing processes. Multiple network endpoints can parallelize the startup overhead of multiple communication requests.

5.3. NUMA-aware MPML all-to-all collective (NMPML)

There are usually NUMA architecture on mordern multi-core processes on supercomputers. As different CPU has different count of NUMA. We uniformly spread the leaders on a node to make them utilize all NUMAs. The core idea can be shown as Figure 6.

From algorithm view, the only difference between NMPML and MPML is the am-i-leader and my-leader-id functions. In NMPML, we only to replace the Algorithm 2 into Algorithm 4.

As we have discussed in Motivation section, when spreading leaders on the node, we acquire beter gathring/scattering throughput in most cases.

5.4. Overlapping Intra-node and Inter-node Communication of NMPML (ONMPML)

We notice that the overead of gathering, scattering, and transpose can be overlapped with inter-node communication. As shown in Figure 7, there are 2 leaders in a node. Each leaders send a block immediately after the block is gathered.

In Figure 7, assume there are 2 leaders and 2 endpoints in a node. First, each leaders post all non-blocking receive requests

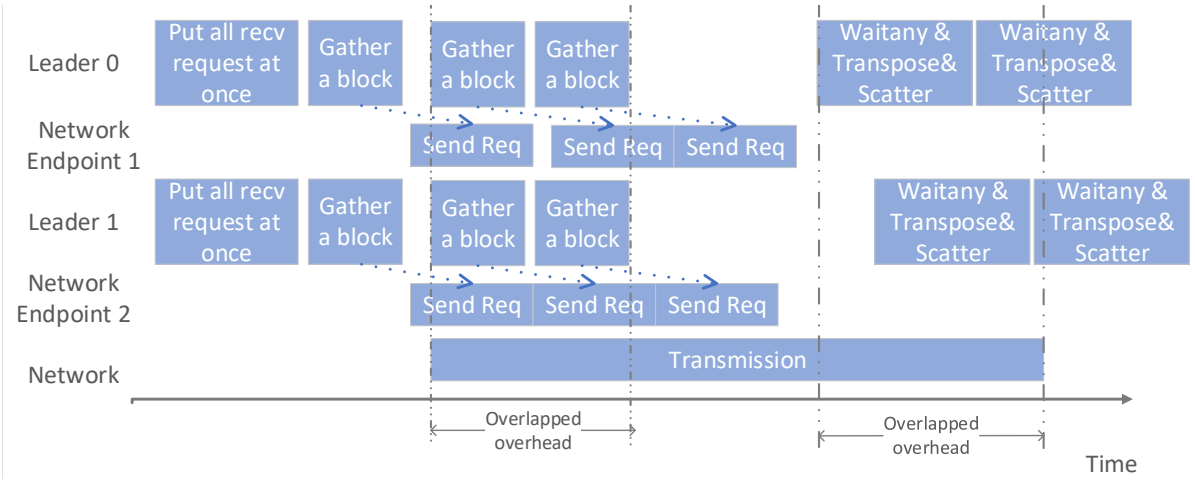


Figure 7: Overlap the intra-node communication and inter-node communication with 2 leaders

Algorithm 4: Leaders Placement of NMPML

```
def am-i-leader(intra-rank):
    dist ← max(⌊intra−procn / leaderN⌋, 1)
    t ← intra−rank / dist
    r ← intra-rank mod dist
    if r == 0 and t < leaderN then
        return True
    end
    return False
def my-leader-id(intra-rank):
    dist ← max(⌊intra−procn / leaderN⌋, 1)
    return intra−rank / dist
```

at once. Then, each leader start gathering data, once a block is aggregated, it sends out the block immediately. After the first gathering, the second gathering is overlapped with network communication. As a result, as long as the number of node is big enough, the gathering overhead can be mostly overlapped with inter-node communiation. As to the scattering overhead, receive leaders start scattering as long as it receive one block. While in NMPML, receive leaders start scattering data after all blocks arrived. The throughput of ONMPML depends on the slowest one: network bandwidth or gather-scattering throughput.

The Algorithm 5 show the ONMPML algorithm on RMA (intra-node communication) and RDMA (inter-node communication). Compared to NMPML, ONMPML using nonblock communication (MPI_Isend/Irecv or RDMA etc.) to overlap the network communication with gathering and scattering procedure. For gathering and scattering procedure, only leader processes are activated other processes do not participate gathering or scattering. Remote Memory Access (RMA) makes the leaders can access other processes space directly.

Algorithm 5: Overlapped NUMA-aware Multi-port Multi-leader a2a (ONMPML)

```
Input: PPN:processes per-node, intra-rank: rank
       within a node, inter-rank: index of nodex
def all-to-all(SendBuf,RecvBuf,count,type,comm):
    intra-node-barrier()
    if am-i-leader(intra-rank) then
        Lid = my-leader-id(intra-rank)
        for i in range(Lid,nodeN,LeaderN) do
            target-node ← (inter-rank + i) mod nodeN
            k ← i / LeaderN
            shift ← target-node*count*PPN
            for j in range(0,PPN) do
                source ← (intra-rank + j) mod P
                Get(gatherbuf[k][source],source,shift)
            end
            RDMA_req.target ← target
            RDMA_req.event ← [inter_rank, k]
            RDMA_Put(gatherbuf[k],RDMA_req)
        end
        for i in range(Lid,nodeN,LeaderN) do
            WaitAnyEvent(&event)
            source-node ← event[0]
            k ← event[1]
            LocalTranspose(BufferR[k])
            shift ← source-node*count*PPN
            for j in range(0,PPN) do
                target ← (intra-rank + j) mod P
                Put(BufferR[k][target],target,shift)
            end
        end
    end
    intra-node-barrier()
```

6. Modeling the performance of all-to-all algorithms on multi-core supercomputer

In this section, we analyze the performance of traditional algorithms and our algorithms. We need a model which considering the effect of multiple core, NUMA, and network endpoints. For inter-node communication, we just using the classical logP model [21]. Classical intra-node communication model cannot meet our needs to model and anylyze our algorithms. We use the τ-Lop [22] to model the intra-node communication.

6.1. The parameters of the model

In τ-Lop [22], there are several definition we need to use:

(1) $L(m, \tau)$ is cost to concurrently transfer τ messages with size m. There is a equation in τ-Lop:

$$L(m, 1) \leq L(m, A) \leq A \times L(m, 1) \tag{1}$$

It means that sending a message with size m is faster than concurrent sending A messages with size m. Concurrent sending A messages with size m is faster than seperate sending A messages with size m.

(2) n is the number of data transfers time needeeds to move from send buffer to the receive buffer. For kernel-assistant or shared-heap-based intra-node communication, n equals to 1. Because a process can directly access other processes memory space, a process only need one copy to sending or receiving a message. For POSIX shared memory, n equals to 2. Because data on send buf has to be copied onto a shared buffer, then the receiving process copy the data from shared buffer to receive buffer.

(3) k, the number of segments the message is split into.

(4) o(m) is the time elapsed since the invoking of a message transmission operation until the beginning of data injection into the channel.

The time to send a intra-node message with size m is:

$$T(m) = o(m) + \sum_{j=0}^{k+n-2} L_j(\frac{m}{k}, \tau_j) \tag{2}$$

In POSIX shared memory, where n = 2, equation 2 becomes:

$$T(m) = o(m) + 2L(S, 1) + (k-1)L(S, 2) \tag{3}$$

When concurrently transer A messages, its overhead defined as $A||T(m)$, which is:

$$A||T(m) = o(m) + \sum_{j=0}^{k+n-2} L_j(m, A\tau_j) \tag{4}$$

Through equation 1 and 4, we have:

$$A||T(m) \leq A \times T(m) \tag{5}$$

For inter-node communication, we using logP model. Some other definition we need are:

(1) M, the number of nodes.
(2) p, the total number of processes.
(3) ppn, the number of processes in each nodes.
(4) nn, the number of NUMAs in each nodes.
(5) n1, the number of processes in a NUMA.
(6) ln, the number of leaders and network endpoints used in each nodes. We usually have $ln \leq nn$.
(7) sz, the all-to-all message size between a pair of processes.
(8) b', the thoughput of transposing a buffer. $b' * m$ is the overhead to transpose a buffer where its length is m.

Beside, in order to ensure the consistency of the algorithms, assuming all intra-node gatering/scattering operations in L-a2a, MLMP, NMLMP, ONMLMP are based on direct algorithm. The inter-node alltoall are based on pairwise algorithm.

6.2. Overhead of direct pairwise all-to-all

Direct all-to-all takes p step to finish the communication. In step i, rank j send a message to rank (j+i) mod p and receive a message from rank (j+p-i) mod p. We consider the MPI rank is uniformly spreaded on the node. The overhead of first step where each processes send a message to themself is ignored. For step 1, there is one outgoing message from a node. For step ppn, there are ppn messages outgoing from a node. When p larger than ppn, there are still ppn messages outgoing from a node. If we consider the inter-node bandwidth is smaller than intra-node bandwidth, we have: The overhead of step i ($i < ppn$) is: $i \times L + o + i(sz-1) \times G + g$. The overhead of step i ($i \geq ppn$) is: $ppn \times L + o + ppn(sz-1) \times G + g$. As a result, the total overhead of direct all-to-all is:

$$\begin{aligned} T_{d-a2a}(sz) &= 2 * \sum_{i=1}^{ppn-1} (i \times L + o + i(sz-1) \times G + g) \\ &\quad + \sum_{i=1}^{p+1-2ppn} (ppn \times L + o + ppn(sz-1) \times G + g) \\ &= ppn \times (p - ppn) \times (L + G \times (sz-1)) \\ &\quad + (p-1)(o+g) \\ &\approx ppn \times (p - ppn) \times (L + G \times sz) + p \times (o+g) \end{aligned} \tag{6}$$

6.3. Overhead of leader-based all-to-all (L-a2a)

As we are considering direct Gather/Scatter algorithm, in τ-lop, the overhead to gather/scatter one block is: $ppn||T(sz \times ppn)$. This means that, there are ppn concurent messages, the size of each message is $sz \times ppn$. As there are M blocks in a node, overhead to gather/scatter M blocks is: $M \times (ppn||T(sz \times ppn))$. The LogP overhead of inter-node pariwise all-to-all is: $(M-1) \times (L + o + (size-1) \times G + g) \approx (M-1) \times (L + o + size \times G + g)$. Overall, the overhead of L-a2a is O(gather) + O(transpose) + O(inter-node alltoall) + O(scatter) which is :

$$\begin{aligned} T_{L-a2a}(sz) &= M \times (ppn||T(sz \times ppn)) \\ &\quad + M \times b' \times sz \times ppn \times ppn \\ &\quad + (M-1)(L + o + (sz \times ppn \times ppn - 1) \times G + g) \\ &\quad + M \times (ppn||T(sz \times ppn)) \end{aligned} \tag{7}$$

Compared to T_{d-a2a}, T_{L-a2a} has same coeffient of G, which is the overhead on the network. L-a2a has a far less message number than d-a2a. But L-a2a incurs a lot gathering/scatering overhead. So, for small message and medium message, L-a2a may have performance improvement than d-a2a.

6.4. Overhead of Multi-port Multi-leader a2a (MPML)

Compared to L-a2a, MPML using multiple process to do gatering, scattering, and transposing operation and using multiple enpoint to concurrently process communication request. For ppn processes in a node, the first ln processes are chosed to be leader. For local transpose overhead, the local transpose overhead are evently amortised on multiple pocesses. For gathering and scattering overhead, as these leaders are placed on a same NUMA, all leaders gathring procedure are concurrently processed. As for inter-node communication, MPML has lower overhead of o and g than L-a2a because of using multiple network endpoint.

So the overhead of MPML is:

$$\begin{aligned} T_{MPML}(sz) = 2 \times \frac{M}{ln} \times ((ppn \times ln) \| T(sz \times ppn)) \\ + \frac{M}{ln} \times b' \times sz \times ppn \times ppn \\ + (M-1)(L + \frac{o}{ln} + (sz \times ppn \times ppn - 1) \times G + \frac{g}{ln}) \end{aligned} \tag{8}$$

For gathering/scattering overhead, according to Equation 5, we have: $\frac{M}{ln} \times ((ppn \times ln) \| T(sz \times ppn)) \leq M \times (ppn \| T(sz \times ppn))$. So, gathering/scattering overhead on MPML must be less than L-a2a according to τ-lop model. The NUMA which the leaders placed on have the most memory access pressure than other NUMA. This lead to uneven memory access among different NUMA.

6.5. Overhead of NUMA-aware MPML all-to-all (NMPML)

In MPML, multiple leaders are contigously placed on the node. NMPML spread the leaders on the node. In MPML, it causing the NUMA which have most leaders placed on become the hostspot. When leaders are spreaded on the node, the gathering/scattering between different NUMA has less contention. The overhead of NMPML is:

$$\begin{aligned} T_{NMPML}(sz) = 2 \times \frac{M}{ln} \times ((ppn + (ln-1) \times n1) \| T(sz \times ppn)) \\ + \frac{M}{ln} \times b' \times sz \times ppn \times ppn \\ + (M-1)(L + \frac{o}{ln} + (sz \times ppn \times ppn - 1) \times G + \frac{g}{ln}) \end{aligned} \tag{9}$$

The only difference between MPML and NMPML is coeffient of $T(sz \times ppn))$. There are $ppn \times ln$ concurent message in a multi-eader gathering step of MPML, because all messages are placed on a same NUMA. For NMPML, in a multi-eader gathering step, there are $ppn + ln \times n1$ concurent messages on a NUMA. As $(ppn \times ln) - (ppn + (ln-1) \times n1) = ppn(ln - 1 - \frac{ln-1}{nn}) >= 0$, we have $T_{NMPML} \leq T_{MPML}$.

6.6. Overhead of ONMPML

The ONMPML overlaps intra-node and inter-node overhead. As long as a leader finished a gathering procedure, it immediately sends out the data and prepares next block. The first gathering and last transposing and scattering cannot be overlapped. So the overhead of ONMPML is:

$$\begin{aligned} T_{start} &= ((ppn + (ln-1) \times n1) \| T(sz \times ppn)) \\ T_{end} &= T_{start} + b' \times sz \times ppn \times ppn \\ T_{inter} &= (M-1)(L + \frac{o}{ln} + (sz \times ppn \times ppn - 1) \times G + \frac{g}{ln}) \\ T_{intra} &= (2 \times \frac{M}{ln} - 1) \times ((ppn + (ln-1) \times n1) \| T(sz \times ppn)) \\ &+ (\frac{M}{ln} - 1) \times b' \times sz \times ppn \times ppn \\ T_{ONMPML}(sz) &= T_{start} + max(T_{inter}, T_{intra}) + T_{end} \end{aligned}$$

Notice the $T_{NMPML}(sz) = T_{start} + T_{inter} + T_{intra} + T_{end}$. So $T_{NMPML}(sz) \geq T_{ONMPML}(sz)$.

7. Implementation with POSIX shared memory and RDMA

As shown in Figure 8, the dash box include our impementation parts. Our implemtation is based on shared memory and Galaxy Express (GLEX) communication layer (to support RDMA) for Tianhe series network.

7.1. Impementation of Intra-node Communication

Shared Heap [23] is a large free shared memory created by POSIX interface. Then, each processes in a node associative the shared memory into its own addres space. After that, a reimplented malloc (glex-malloc) which allocated the memory on this shared memory. The memory allocated by glex-malloc are accessable by all ranks in a same node. User's application must using glex-malloc to allocate sendbuf and recvbuf of all-to-all communication. As a result, each processes in the node, can directly access the memory allocate on shared heap.

There is a set of intra-node operations RMA-fence, RMA-Put,and RMA-Get which based on shared heap. They are similar to MPI-Win-fence, MPI-Put and MPI-Get. RMA-fence is a intra-node fence. Between two RMA-fence, all ranks in a node is free to access other ranks buffer with RMD-Put and RMA-Get. Difference is that a rank call RMA-Put or RMA-Get is blocking communication. When RMA-Put or RMA-Get return, the data is actually moved from one rank to another rank.

7.2. Impementation of Inter-node Communication

In gather-scatter-based all-to-all, the messages are aggregated. For most cases, they are large message. In MPI, there are eager protocol for small message and rendezvous protocol for large message. Rendenzvous protocal have a "rendenzvous start" and "rendenzvous fin" overhead for each large message sending. To avoid these protocal overhead, we directly use Remote Direct Memory Access (RDMA) to do inter-node communication. RDMA communication must based on preregisted buffer. This paper using two pre-registered buffer (BufferSend,

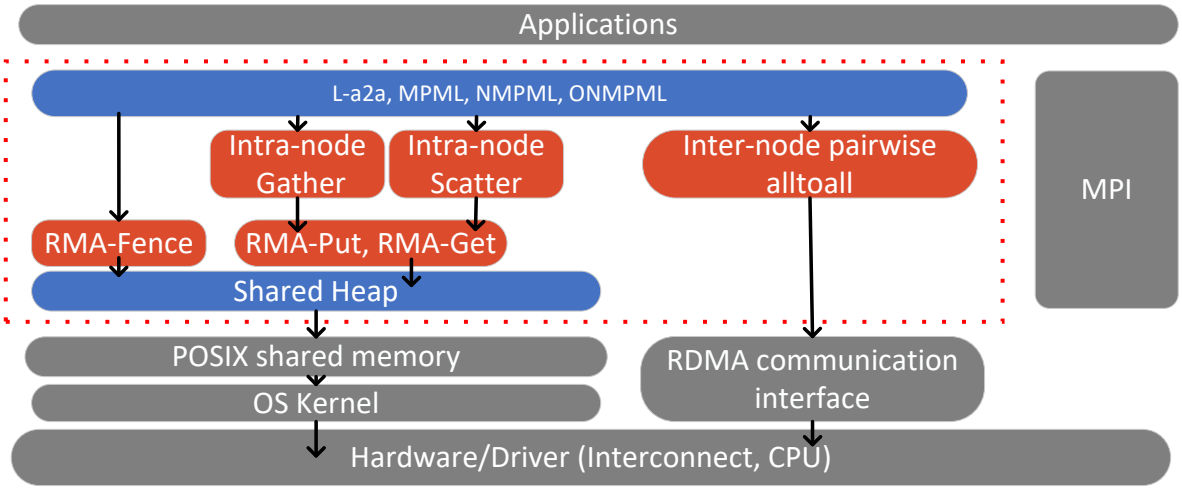


Figure 8: Implementation overview.

BufferRecv) on each node to do inter-node RDMA. Gathering operation direct gather the data on to BufferSend, than start RDMA communication. After receiver rank detect a receive event, it transpose the data and directly scatter the data on to receive buffer. RDMA-based communication do not need "rendenzvous start" and "rendenzvous fin" overhead for each large message transmission. This avoids the overhead caused by the rendenzvous protocol.

8. Evaluation

8.1. Microbenchmark Evalutation

This paper evaluate performance of different based on following microbenchmark 6.

Algorithm 6: all-to-all microbenchmark

```
for i in range(0,loopN) do
    MPI_Barrier(comm)
    startT = MPI_Wtime()
    Alltoall()
    endT = MPI_Wtime()
    totalT += (endT-start)
end
MPI_Reduce(totalT,MaxT)
MaxT ← MaxT/loopN  It compares the MPI_Alltoall to our RDMA-based Alltoall (L-a2a, MPML, NMPML, ONMPML) on different HPC systems.
```

8.1.1. Microbenchmark Result on HPC-A

The results of microbenchmark on four HPC systems are shown in Figure 9. We can see that multi-endpoint, multi-leader alltoall (MPML) has a better performance than one leader alltoall (L-a2a). When we using 4 leaders, its faster than 2 leader MPML. In subfigure a of Figure 9, the improvement going down as the message getting larger. As the HPC-A has the 1.3 GB/s bidirectional network bandwidth among these systems, which is far slower than the gatering/scattering bandwidth. The whole alltoall is limited by the network bandwidth for large message. As a result, MPML has a lesser performance improvement on a low-bandiwidth, large message alltoall.

Subfigure b of Figure 9 show the performance of NUMA-aware MPML compared to MPML. 2-leader NMPML is compared to 2-leader MPML. 4-leader NMPML is the same the 2-leader NMPML. For small message NUMA-aware MPML has a little performance improvement compared to MPML. The large message has nearly no improvement. This is same with the Figure **??**. The NUMA-aware gathering-scattering has batter throughput on small messages.

Subfigure c of Figure 9 show that the ONMPML compared to corresponding NMPML. ONMPML has a significant performance improvement. As ONMPML overlaped the inter-node and intra-node communication. Subfigure d show the ONMPML compared to MPI_Alltoall on HPC-A. 4-leader ONMPML has a 3.53x speedup on average compared to MPI_Alltoall.

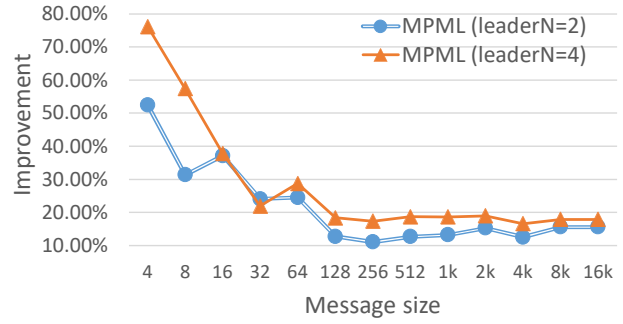
8.1.2. Microbenchmark Result on HPC-B

For subfigure a of Figure 10, MPML show a 45% or 65% improvement compared to L-a2a. This is because its bisectional network bandwidth is 11 GB/s. one leader's gathering/scattering throughput are 6 GB/s. The intra-node communication may take up a major part of communication overhead. As a result, MPML can obliviously improve the performance compare to L-a2a under the big messages.

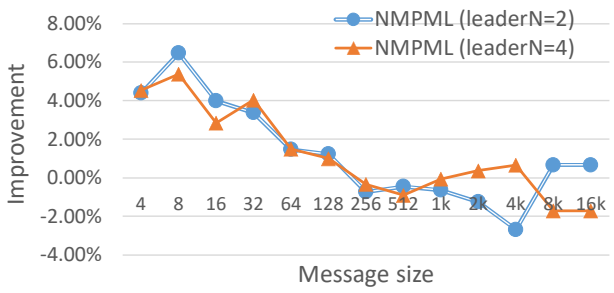
For subfigure b of Figure 10, NMPML still show a little performance improvement for small messages. Subfigure c show that 5%(2-leader ONMPML) and 12.5%(4-leader ONMPML) performance improvement compared to NMPML. Compare to MPI_Alltoall, 4-leader ONMPML acheives 3.7x speedup on average.

8.2. HPCC-FFT Evaluation

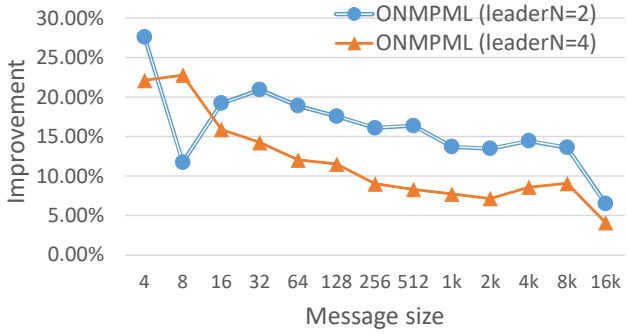
HPC Challenge is a benchmark suite which combines several benchmarks to test the performance of high-performance



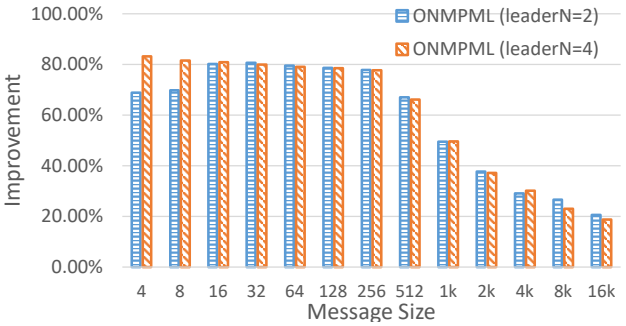
(a) HPC-A:MPML compared to L-a2a



(b) HPC-A:NMPML compared to MPML

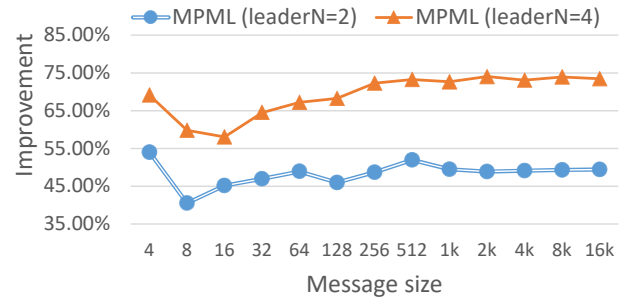


(c) HPC-A:ONMPML compared to NMPML

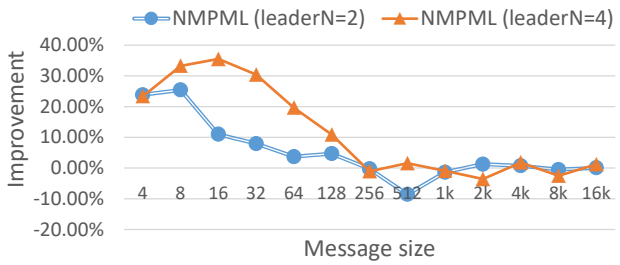


(d) HPC-A:ONMPML compared to MPI

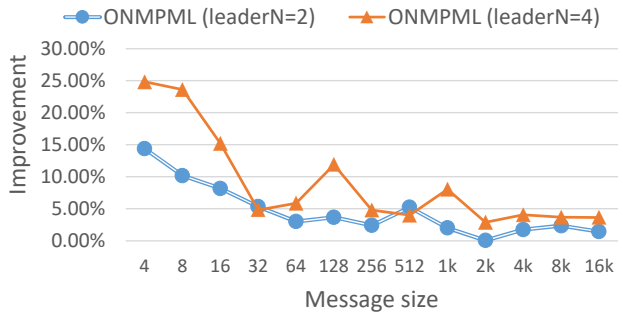
Figure 9: Microbenchmark results of 256 nodes, 8192 processes alltoall collective on HPC-A.



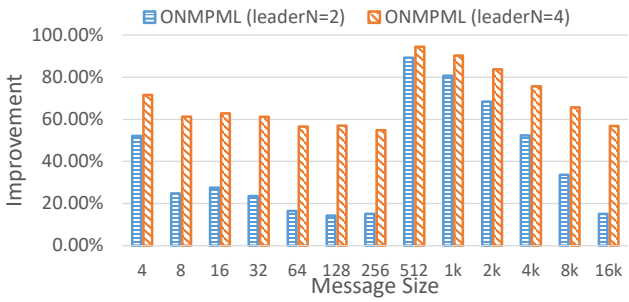
(a) HPC-B:MPML compared to L-a2a



(b) HPC-B:NMPML compared to MPML

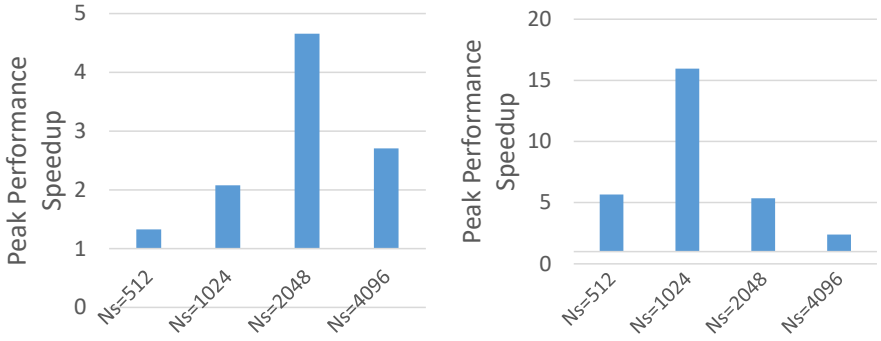


(c) HPC-B:ONMPML compared to NMPML



(d) HPC-B:ONMPML compared to MPI

Figure 10: Microbenchmark results of 50 nodes, 1200 processes alltoall collective on HPC-B.



(a) HPC-A: 256 nodes (b) HPC-B: 200 nodes

Figure 11: Application evaluations.

computer (HPC) systems [24]. Their is a global Fast Fourier Transforms (FFT) benchmark in HPCC. FFT is very useful in applications such as spectral methods, signal processing and climate modeling. HPCC global FFT doing a 1D FFT Z=FFT(X). MPI_Alltoall is used to transpose matrix in global FFT. We replace the MPI_Alltoall with our 4-leader ONMPML based on GLEX RDMA (GLEX_Alltoall). We stested different vector size Ns. For HPC-A we test Ns={256,512,1024,2048}. For HPC-B, we test Ns={512,1024,2048,4096}. The result shown as Figure ře fApp-EXPR.

References

[1] C. Mehta, A. Karthi, V. Jetly, B. Chaudhury, Parallel fast multipole method accelerated fft on hpc clusters, Parallel Computing (2021) 102783.

[2] S. J. Plimpton, K. D. Devine, Mapreduce in mpi for large-scale graph algorithms, Parallel Computing 37 (9) (2011) 610–632.

[3] K. Ueno, T. Suzumura, Highly scalable graph search for the graph500 benchmark, in: Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing, 2012, pp. 149–160.

[4] S. Sistare, R. Vandevaart, E. Loh, Optimization of mpi collectives on clusters of large-scale smp's, in: Proceedings of the 1999 ACM/IEEE conference on Supercomputing, 1999, pp. 23–es.

[5] J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, D. Weathersby, Efficient algorithms for all-to-all communications in multiport message-passing systems, IEEE Transactions on parallel and distributed systems 8 (11) (1997) 1143–1156.

[6] S. Ranka, J.-C. Wang, G. C. Fox, Static and run-time algorithms for all-to-many personalized communication on permutation networks, IEEE Transactions on Parallel and Distributed Systems 5 (12) (1994) 1266–1274.

[7] R. Thakur, R. Rabenseifner, W. Gropp, Optimization of collective communication operations in mpich, The International Journal of High Performance Computing Applications 19 (1) (2005) 49–66.

[8] S. Li, Y. Zhang, T. Hoefler, Cache-oblivious mpi all-to-all communications based on morton order, IEEE Transactions on Parallel and Distributed Systems 29 (3) (2017) 542–555.

[9] R. Kumar, A. Mamidala, D. K. Panda, Scaling alltoall collective on multi-core systems, in: 2008 IEEE International Symposium on Parallel and Distributed Processing, IEEE, 2008, pp. 1–8.

[10] Y. Qian, A. Afsahi, Efficient shared memory and rdma based collectives on multi-rail qsnet ii smp clusters, Cluster Computing 11 (4) (2008) 341–354.

[11] A. Venkatesh, S. Potluri, R. Rajachandrasekar, M. Luo, K. Hamidouche, D. K. Panda, High performance alltoall and allgather designs for infiniband mic clusters, in: 2014 IEEE 28th International Parallel and Distributed Processing Symposium, IEEE, 2014, pp. 637–646.

[12] H.-W. Jin, S. Sur, L. Chai, D. K. Panda, Lightweight kernel-level primitives for high-performance mpi intra-node communication over multi-core systems, in: 2007 IEEE International Conference on Cluster Computing, IEEE, 2007, pp. 446–451.

[13] B. Goglin, S. Moreaud, Knem: A generic and scalable kernel-assisted intra-node mpi communication framework, Journal of Parallel and Distributed Computing 73 (2) (2013) 176–188.

[14] S. Li, T. Hoefler, C. Hu, M. Snir, Improved mpi collectives for mpi processes in shared address spaces, Cluster computing 17 (4) (2014) 1139–1155.

[15] B. Prisacari, G. Rodriguez, C. Minkenberg, T. Hoefler, Bandwidth-optimal all-to-all exchanges in fat tree networks, in: Proceedings of the 27th international ACM conference on International conference on supercomputing, 2013, pp. 139–148.

[16] S. Kumar, Y. Sabharwal, R. Garg, P. Heidelberger, Optimization of all-to-all communication on the blue gene/l supercomputer, in: 2008 37th International Conference on Parallel Processing, IEEE, 2008, pp. 320–329.

[17] D. Saha, K. Sinha, Optimal schedule for all-to-all personalized communication in multiprocessor systems, ACM Transactions on Parallel Computing (TOPC) 6 (1) (2019) 1–29.

[18] H. Subramoni, K. Kandalla, J. Jose, K. Tomko, K. Schulz, D. Pekurovsky, D. K. Panda, Designing topology-aware communication schedules for alltoall operations in large infiniband clusters, in: 2014 43rd International Conference on Parallel Processing, IEEE, 2014, pp. 231–240.

[19] M. G. Kurnosov, Dynamic mapping of all-to-all collective operations into hierarchical computer clusters, in: 2016 13th International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE), Vol. 2, IEEE, 2016, pp. 475–478.

[20] X. Cui, X. Li, B. Wang, Communication optimization technology based on network dynamic performance model, Mathematical Problems in Engineering 2020.

[21] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, T. Von Eicken, Logp: Towards a realistic model of parallel computation, in: Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming, 1993, pp. 1–12.

[22] J.-A. Rico-Gallego, J.-C. Díaz-Martín, τ-lop: Modeling performance of shared memory mpi, Parallel Computing 46 (2015) 14–31.

[23] A. Friedley, G. Bronevetsky, T. Hoefler, A. Lumsdaine, Hybrid mpi: efficient message passing for multi-core systems, in: SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, IEEE, 2013, pp. 1–11.

[24] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, D. Takahashi, The hpc challenge (hpcc) benchmark suite, in: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, Vol. 213, 2006, pp. 1188455–1188677.