Efficient Algorithms for Encrypted All-gather Operation

Mehran Sadeghi Lahijani, Abu Naser, Cong Wu, Mohsen Gavahi, Viet Tung Hoang, Zhi Wang, Xin Yuan
Department of Computer Science, Florida State University
Tallahassee, Florida 32306, USA
{sadeghil, naser, wu, gavahi, tvhoang, zwang, xyuan}@cs.fsu.edu

***Abstract*—As more High-Performance Computing (HPC) applications that process sensitive data are moving to run on the public cloud, there is a need for the cloud infrastructure to provide privacy and integrity support. In this work, we investigate how to add encryption to all-gather to protect inter-node communication. This task is challenging since encryption is often more expensive than communication in contemporary HPC systems. We derive performance bounds for encrypted all-gather, and develop new algorithms that meet the theoretical lower bounds. Our empirical evaluation on production systems demonstrates that the new algorithms achieve substantially better performance than the naive approach.**

Keywords: Encrypted MPI_Allgather, Algorithm Design, Security

I. Introduction

With the ongoing trend of moving HPC applications to run on the public cloud infrastructures, the security of these applications has become a rising concern. In particular, most HPC systems simply send data in the clear, allowing a network adversary to eavesdrop sensitive information or even tamper with it. Using an existing network-level encryption mechanism such as IPSec is a non-solution, as it severely degrades communication speed [19]. To address this problem, researchers have considered adding encryption to the Message Passing Interface (MPI) library [16], [18], [19], [24], [25], which is a commonly used library in HPC applications. Still, they all fail to provide an efficient solution for MPI's collective routines. To partially bridge this gap, in this work, we develop efficient algorithms for encrypting inter-node communication in MPI_Allgather, one of the most widely used collective operations in scientific applications [4].

Adding encryption to MPI is challenging, because in contemporary HPC systems, encryption is usually more expensive than communication. Figure 1 compares the encryption throughput and ping-pong throughput on our local Noleland cluster—a typical HPC cluster today—whose configuration is described in Section V. In particular, encryption throughput saturates at about 5,500 MBps, whereas the maximum ping-pong throughput is twice higher. Given this situation, clever algorithms for encrypting MPI communication must be developed to achieve high performance.

Existing work on encrypting MPI [18] only adds encryption to MPI_Allgather in a naive fashion: (1) each process encrypts its local data; (2) all processes use the original MPI_Allgather to gather all ciphertexts at all processes; and (3) each process decrypts all of the received ciphertexts. Because of the large gap between encryption and communication speed, this naive approach performs poorly.

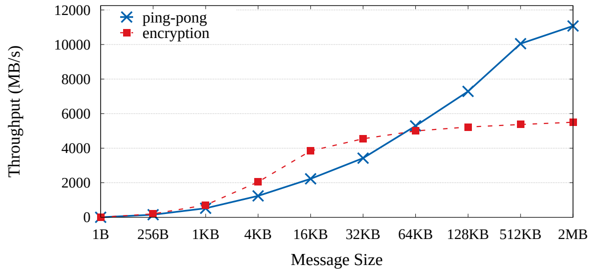


Figure 1: Encryption throughput versus ping-pong throughput on Noleland (InfiniBand).

In this work, we derive the performance bounds for the encrypted all-gather operation, develop new encrypted all-gather algorithms that meet the theoretical lower bounds, and run extensive experiments to compare their empirical performance. The results demonstrate that the new algorithms perform substantially better than the naive approach.

The rest of the paper is organized as follows. In Section II, we discuss the related work. In Section III, we present the background of our work, and in Section IV, we explain our proposed algorithms. In Section V, we present the results of the performance evaluation, and we conclude our work in Section VI.

II. Related work

Since the standardization of MPI, a very large number of algorithms have been proposed for MPI collectives [7]–[9], [14], [21]–[23], [26]. Various all-gather algorithms have been developed, optimizing the performance in different situations: some are architecture-oblivious [7], [8], [26]; some consider the network topology [6], [12], [15], [29], [30]; some focus on SMP and multicore clusters [13], [17], [28]; some use special network features [10]. Encrypted all-gather adds a new dimension for optimization that has not been considered by existing schemes.

Most prior papers on encrypting MPI communication [16], [24], [25] suffer from various security issues, such as using bad encryption methods and lack of integrity support. Recent work [18], [19] provides *provable security* by using the Galois Counter Mode (GCM) of encryption [5], but these papers either focus on point-to-point operations or only use a naive approach for collective operations. In this work, we focus on encrypted MPI_Allgather and develop faster algorithms for this operation.

III. Background

ENCRYPTION. In this work, following Naser *et al.* [18], we use the GCM encryption scheme [5] that *provably* offers both privacy (meaning that adversaries learn no additional information from the ciphertexts, even with partial knowledge of plaintexts) and integrity (meaning that adversaries cannot modify ciphertexts without being detected).[1] GCM is a *nonce-based* encryption scheme, meaning that to encrypt a plaintext P, one needs to additionally provide a *nonce* N, i.e., a public value that must appear at most once per key. The same nonce N is required for decryption, and thus the sender needs to send both the nonce N and the ciphertext C to the receiver. In our implementation, we pick nonces at random, which is standard-compliant.

ALL-GATHER. In MPI_Allgather, each process initially has a copy of its own data, and upon completion of the operation, each process has all data from all processes. Figure 2 illustrates an all-gather operation among 4 processes. This operation has been extensively investigated and is well understood. Below, we give a recap of a few algorithms for MPI_Allgather that appear in various production MPI libraries such as MPICH and MVAPICH, and perform well across different platforms.

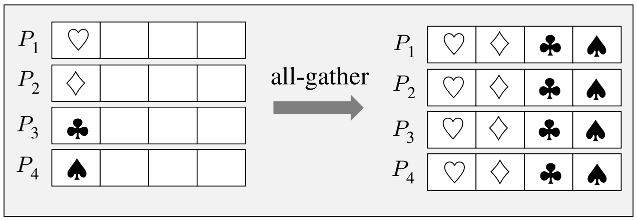


Figure 2: The all-gather operation, illustrated for 4 processes.

NOTATIONAL SETUP. We consider MPI_Allgather of m-byte messages among p processes $P_0, \dots, P_{p-1}$ over N nodes. Without loss of generality, we will assume that p is a multiple of N. Let $\ell = p/N$ denote the number of processes running on each node. We will use the Hockney's model [11] to discuss the performance of the algorithms. In this model, a transmission of m bytes takes $\alpha + \beta m$ time where α is the startup cost and β is the per-byte communication cost. In analyzing algorithms, we assume that processes begin the operation at the same time, and measure running time by estimating the time for the last process to complete the operation. For simplicity, we will assume that inter-node and intra-node communication of the same size take the same amount of time if not specified otherwise. Detailed analyses of these algorithms can be found in [26].

RING. The communication pattern in the Ring all-gather algorithm [26] is $P_0 \to P_1 \to \dots \to P_{p-1} \to P_0$. There are $p-1$ iterations of this algorithm. In the first iteration, each process P_i sends its data to P_{i+1}, where $+$ is the modular addition in mod p. In the later iterations, P_i forwards its current data to P_{i+1} and receives new data from P_{i-1}. After $p-1$ iterations, all processes have all the data. The time for this algorithm is $(p-1)\cdot\alpha + (p-1)\cdot m\beta$.

Since the logical traffic pattern is fixed, the performance of the Ring algorithm can be sensitive to the process mapping. Most MPI libraries support *block order* (namely process P_i is mapped to node $\lfloor i/N \rfloor$), and *cyclic order* (namely process P_i is mapped to node $i \bmod N$). The ring pattern can have very different characteristics (and thus performance) for different process mappings, but a rank-ordered version of this algorithm allows the performance to be consistent across different mappings [13].

RECURSIVE DOUBLING (RD). In RD [26], the distance between the sender and receiver processes and the amount of data they exchange is doubled after each iteration. In particular, in the b-th iteration, P_i exchanges its data with P_{i+B}, where $B = 2^{b-1}$, for every i such that $(i \bmod 2B) < B$. It takes $\lg(p)$ steps for RD to perform all-gather when p is a power of two, and its running time is $\lg(p)\cdot\alpha + (p-1)\cdot m\beta$. If p is not a power of two, extra steps are needed to complete the operation, but the total number of steps is still bounded by $2\cdot\lg(p)$ [26]. Unlike the Ring algorithm, RD cannot rearrange the rank order to achieve good performance across different process mappings. As a result, its performance is sensitive to process mapping.

HIERARCHICAL. In the Hierarchical algorithm [28], each node contains a leader process. The algorithm consists of three steps: (1) all processes in each node perform a local intra-node gather to collect the data to the leader of that node; (2) leaders perform an (inter-node) all-gather operation to distribute all data among leaders; and (3) all processes in each node perform a local broadcast, with the leader being the root.

The running time of the Hierarchical algorithm depends on the collective algorithms in each of the steps. If we assume that intra-node communication is much cheaper than inter-node one, then the performance of the Hierarchical algorithm is dominated by the inter-node all-gather in step (2). Assuming further than the inter-node all-gather is implemented via RD, the running time of the Hierarchical algorithm is about $\lg(N)\cdot\alpha + (p-\ell)\cdot\beta m$.

[1] Actually, the adversary can still replace a ciphertext with a prior one; this is known as *replay attacks*. Here we do not consider such attacks.

IV. Encrypted MPI_Allgather

We consider encrypted all-gather of m-byte data on p processes and N nodes, where each node has $\ell = p/N$ processes. For simplicity, here we describe the algorithms and analyze their complexity for the case where p and N are powers of two, but the algorithms can be slightly modified to work for any choice of p and N with the same asymptotic complexity. The general versions of the algorithms that work for any values of p and N have been implemented and their performance is reported in Section V. In GCM, a ciphertext is 28 bytes longer than the corresponding plaintext, but our analyses will ignore this constant overhead and assume that ciphertext and plaintext are of the same length for simplicity.

In an encrypted all-gather, inter-node communication must be encrypted while intra-node one can be sent in the clear. We assume that $N \geq 2$ since a single-node all-gather does not need encryption. In our analyses, we assume that all processes start the operation at the same time, and measure the running time by estimating the time for the last process to complete the operation.

A. Performance bounds

Setup. We assume that encryption and decryption cost also follows the Hockney's model. That is, encrypting an m-byte message takes $\alpha_e + \beta_e \cdot m$ time units and decrypting an m-byte message takes $\alpha_d + \beta_d \cdot m$ time units. Below, we establish the lower bounds of six key performance metrics for encrypted all-gather:

(i) r_c: the number of communication rounds
(ii) s_c: the total size of data to be sent and received in the critical path,
(iii) r_e: the number of encryption rounds,
(iv) s_e: the size of data to be encrypted in the critical path,
(v) r_d: the number of decryption rounds, and
(vi) s_d: the size of data to be decrypted in the critical path.

With these metrics, an encrypted all-gather algorithm will have at least $t_c = r_c \cdot \alpha + s_c \cdot \beta$ communication time, $t_e = r_e \cdot \alpha_e + s_e \cdot \beta_e$ encryption time, and $t_d = r_d \cdot \alpha_d + s_d \cdot \beta_d$ decryption time. For small messages, the terms r_c, r_e, and r_d will dominate t_c, t_e, and t_d, respectively. In contrast, for large messages, the terms s_c, s_e, and s_d will determine the performance. Depending on how communication, encryption, and decryption overlap, the time for the algorithm will be between $\max\{t_c, t_e, t_d\}$ and $t_c + t_e + t_d$. Each of these terms may dominate the performance of the operation. Thus, the algorithm design must consider all of the three terms.

The bounds. Table I shows the bounds for encrypted all-gather. The lower bounds r_c and s_c for communication cost are well known [3], [26], but we list them here for completeness.

Next, we establish lower bounds for encryption cost. Since $N \geq 2$, at least one encrypted inter-node message must be sent to complete the all-gather, and thus $r_e \geq 1$. In addition, messages in each node with a total size of $\ell \cdot m$ bytes must be encrypted in order to be sent to every other node. As there are ℓ processes in a node to collectively encrypt $\ell \cdot m$ bytes of data, there is one process that encrypts at least m bytes, and thus $s_e \geq m$.

Table I: Lower bounds for encrypted all-gather of m-byte data on p processes and N nodes, with $\ell = p/N$ processes per node.

Metric	r_c	s_c	r_e	s_e	r_d	s_d
Bound	$\lg(p)$	$(p-1)m$	1	m	$\left\lceil \frac{\lg(N)}{\lg(\ell+1)} \right\rceil$	$(N-1)m$

We now derive lower bounds for decryption cost. Since each node must decrypt all messages from other nodes, the total data size to be decrypted in each node is at least $(N-1) \cdot \ell m$. As there are ℓ processes in a node to collectively decrypt that amount of data, there is one process that decrypts at least $(N-1) \cdot m$ bytes, and thus $s_d \geq (N-1) \cdot m$.

To bound r_d, without loss of generality, we will assume the following: (1) a process needs exactly one round to decrypt a ciphertext of any size, and (2) encryption and communication time will be ignored, meaning that a process can encrypt and send data of any size with lightning speed.

With these assumptions, before the first round of decryption, each node only has unencrypted data from $T_0 = 1$ node (namely itself), and each ciphertext contains unencrypted data of just T_0 node. Since there are ℓ processes in each node, after the first round of decryption, each node can decrypt at most ℓ ciphertexts, and obtain unencrypted data from at most $T_1 = (\ell + 1) \cdot T_0 = \ell + 1$ nodes, including itself, and each ciphertext now contains unencrypted data of at most T_1 nodes. Likewise, after the second round of decryption, each node can obtain unencrypted data from at most $T_2 = (\ell+1) \cdot T_1 = (\ell+1)^2$ nodes, and each ciphertext now contains unencrypted data of at most T_2 nodes. By repeating this argument, if the protocol terminates in r_d rounds then at the end of the encrypted all-gather operation, each node can obtain unencrypted data from at most $(\ell+1)^{r_d}$ nodes. As each node must gather data from all N nodes, we must have $N \leq (\ell+1)^{r_d}$, and thus $r_d \geq \lceil \lg_{\ell+1} N \rceil = \left\lceil \frac{\lg(N)}{\lg(\ell+1)} \right\rceil$.

If we treat ℓ as a constant and N as a parameter then $r_d \in \Omega(\lg(N))$. This is a tight bound, as we will later develop an algorithm (namely O-RD2) whose r_d is $\lg(N)$. On the other hand, if ℓ can be arbitrarily large then any lower bound for r_d must involve both N and ℓ, because there is an algorithm in this paper (namely HS1) that has $r_d = \lceil N/\ell \rceil$, meaning that r_d can be as small as 1 if $\ell \geq N$.

Performance of the naive approach. Consider the approach in [18] where each process first encrypts its message, then calls MPI_Allgather to perform the all-gather operation on the ciphertexts, and finally decrypts the received ciphertexts. We call this algorithm Naive.

If we run Naive on top of the ordinary all-gather algorithms of MVAPICH, then it achieves the theoretical lower bounds for communication cost, meaning that $r_c = \lg(p)$ and $s_c = (p-1)m$. Moreover, since each process encrypts its own message, we have $r_e = 1$ and $s_e = m$, and thus Naive also meets the lower bounds on encryption cost. However, the situation for decryption cost is different. In particular, since each process must decrypt $p-1$ ciphertexts of m bytes, $r_d = p-1$ and $s_d = (p-1)m \approx (N-1)m \cdot \ell$, which are much larger than the lower bounds. Our evaluation will later show that Naive introduces very large overheads to all message sizes.

B. Faster algorithms for encrypted all-gather

OPPORTUNISTIC ALGORITHMS. The Naive algorithm performs poorly because it unnecessarily sends encrypted data for intra-node communication, leading to excessive decryption cost. The Opportunistic algorithm improves the situation by sending encrypted data only for inter-node communication. For each intra-node send, the sender is required to send the corresponding plaintext of the data, meaning that it may need to first decrypt its message to obtain the plaintext. We write O-X to refer to the Opportunistic algorithm running on top of an ordinary all-gather algorithm X $\in$ {RD, Ring}. See Figure 3 for an illustration of O-Ring.

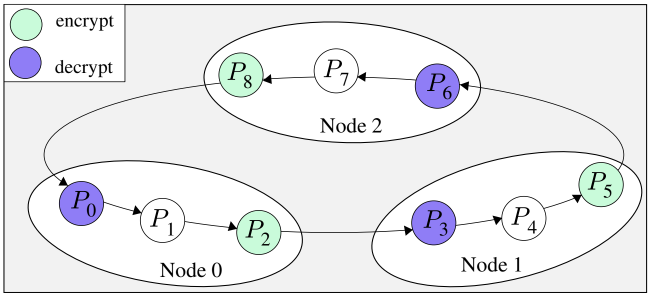


Figure 3: Illustration of O-Ring with $p = 9$ processes of block order on $N = 3$ nodes.

For block-order mapping, the metrics of O-Ring are shown in Table II. The communication terms r_c and s_c are the same as in ordinary Ring-based all-gather. Since the exit process P_i of each node must encrypt data of every process P_j with $j \neq i+1$, we have $r_e = p-1$ and $s_e = (p-1)m$. Similarly, $r_d = p-1$ and $s_d = (p-1)m$.

In O-RD with block-order mapping, in each of the last $\lg(N)$ iterations, each process P_i sends many ciphertexts to another process P_j in a different node. Alternatively, P_i can merge them into a single ciphertext via decrypting and then re-encrypting, which corresponds to a variant O-RD2 of O-RD. This variant reduces the number of decryption rounds on the receiver side, at the cost of increasing the amount of data to encrypt. Between O-RD2 and O-RD, we expect O-RD2 to be better for small messages and O-RD to be better for large ones.

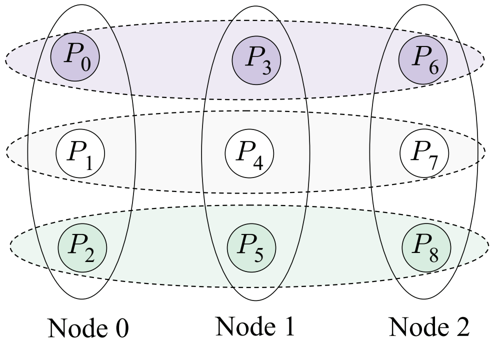


Figure 4: Illustration of the Concurrent algorithm on $N = 3$ nodes and $p = 9$ processes. Processes in the same shaded area run an encrypted all-gather on their data.

For O-RD with block-order mapping, the terms r_c and s_c are the same as in ordinary RD-based all-gather. The first $\lg(\ell) = \lg(p) - \lg(N)$ iterations only involve intra-node communication, and there is no encryption or decryption. In the next $\lg(N)$ rounds, each process only has to encrypt the data of its node, and thus $r_e = 1$ and $s_e = m\ell$. Moreover, each process only decrypts the encrypted copy of data of every other node, and thus $r_d = N-1$ and $s_d = (N-1)\ell m = (p-\ell)m$.

For O-RD2 with block-order mapping, only the last $\lg(N)$ rounds involve encryption or decryption, and in each of these rounds, each process has to decrypt a ciphertext, and then encrypt another message. Thus $r_e = r_d = \lg(N)$ and $s_e = (p-\ell)m$. The other terms are the same as O-RD.

The Opportunistic algorithms offer some advantages over the Naive algorithms, such as allowing communication-computation overlaps and reducing the total number of encryption and decryption operations. However, as shown in Table II, they all fail to substantially improve s_d, which is the main issue with the naive approach.

CONCURRENT ALGORITHMS. We now describe a family of algorithms that are specifically designed to reduce s_d, matching the theoretical lower bound $(N-1) \cdot m$. Initially, the algorithm partitions the p processes into ℓ groups such that each node has exactly one process per group. For each group, we perform an encrypted all-gather on the corresponding N processes with their m-byte data; the encrypted all-gather is implemented via the family of Opportunistic algorithms. That is, we have ℓ concurrent encrypted sub-all-gathers. This step brings all data to every node, but in each node, data are still spread across its ℓ processes. Next, each node performs a local ordinary all-gather on its ℓ processes.

Figure 4 shows an example where 9 processes run on 3 nodes. Initially, we partition the processes to three groups $\{P_0, P_3, P_6\}$, $\{P_1, P_4, P_7\}$, and $\{P_2, P_5, P_8\}$, and each group performs an encrypted sub-all-gather. Then each node runs a local ordinary all-gather on its three processes.

We write C-Ring to refer to the Concurrent algorithm

Table II: Performance of encrypted all-gather algorithms on m-byte messages, p processes on N nodes with $\ell = p/N$ processes per node. We assume that p and N are powers of two and block-order mapping is used for Opportunistic algorithms. For HS1 and HS2, we assume that the RD algorithm is used for the inter-node all-gather, and ignore the cost of intra-node communication.

	Naive	O-Ring	O-RD	O-RD2	C-Ring	C-RD	HS1	HS2
r_c	$\lg(p)$	$p-1$	$\lg(p)$	$\lg(p)$	$N+\ell-2$	$\lg(p)$	$\lg(N)$	$\lg(N)$
s_c	$(p-1)m$	$(p-1)m$	$(p-1)m$	$(p-1)m$	$(p-1)m$	$(p-1)m$	$(p-\ell)m$	$(p-\ell)m$
r_e	1	$p-1$	1	$\lg(N)$	1	1	1	1
s_e	m	$(p-1)m$	ℓm	$(p-\ell)m$	m	m	ℓm	m
r_d	$p-1$	$p-1$	$p-\ell$	$\lg(N)$	$N-1$	$N-1$	$\lceil N/\ell \rceil$	$N-1$
s_d	$(p-1)m$	$(p-1)m$	$(p-\ell)m$	$(p-\ell)m$	$(N-1)m$	$(N-1)m$	$\max\{N,\ell\} \cdot m$	$(N-1)m$

where the encrypted sub-all-gathers are implemented via the Opportunistic approach and the ranked-ordered version of the Ring algorithm in [13]. Let C-RD be the Concurrent algorithm where the encrypted sub-all-gathers are implemented via the O-RD.

For C-Ring, in the first step, the number of communication rounds is $N-1$; each process communicates $(N-1)m$ bytes; the number of encryption rounds is 1; each process encrypts m bytes; the number of decryption rounds is $N-1$; and each process decrypts $(N-1)m$ bytes. In the second step, the number of communication rounds is ℓ-1; each process communicates $(\ell-1)Nm = (p-N)m$ bytes; and there is no encryption or decryption. Summing up, we obtain the complexity of C-Ring as shown in Table II.

For C-RD, in the first step, the number of communication rounds is $\lg(N)$; each process communicates $(N-1)m$ bytes; the number of encryption round is 1; the amount of encrypted data is m bytes; the number of decryption rounds is $N-1$; and the amount of decrypted data is $(N-1)m$ bytes. In the second step, the number of rounds of communication is $\lg(\ell)$; each process communicates $(\ell-1)Nm = (p-N)m$ bytes; and there is no encryption or decryption. Summing up, we obtain the complexity of C-RD as shown in Table II.

For Naive and Opportunistic algorithms, the size of the encrypted and decrypted data is $\Theta(pm)$. In contrast, for C-Ring and C-RD, the size of the encrypted and decrypted data is only $\Theta(Nm)$, and their other metrics are comparable to those of Naive. Thus, both C-Ring and C-RD are expected to be significantly better than Naive for sufficiently large data, assuming that ℓ is fairly large, say $\ell = 8$. Moreover, on contemporary HPC systems, a single core usually does not have enough computing power to fully utilize the network link. The concurrent sub-all-gathers in the Concurrent algorithm can also better utilize the network link bandwidth.

HIERARCHICAL SHARED-MEMORY ALGORITHMS. Recall that both C-Ring and C-RD have rather large $r_d = N-1$, and thus are not competitive for small messages. Our Hierarchical Shared-memory (HS1) algorithm in contrast achieves the optimal s_d, yet reduces r_d to N/ℓ. Moreover, as shown later in the experiments, HS1 has even cheaper underlying communication cost. As a result, it would be competitive for both small and large messages. The scheme HS1 consists of the following steps:

1) Each node performs an unencrypted gather to collect the data in the node to a leader process, using a shared-memory plaintext buffer.
2) Each leader encrypts its ℓm-byte data, and runs an ordinary all-gather on ciphertexts among the N leaders, and the results are stored in a shared-memory ciphertext buffer.
3) All processes in each node jointly decrypt the $N-1$ ciphertexts from other nodes, and place the results in the shared-memory plaintext buffer.
4) All processes in each node copy the results from the shared-memory plaintext buffer to the user buffer.

Assuming that the local data transfer to a shared memory is free, we will ignore the cost in Steps 1 and 4. Assume further that RD is used for the inter-node all-gather in Step 3. Then the communication cost is only from the inter-node all-gather among N nodes on ℓm-byte messages, meaning that $r_c = \lg(N)$ and $s_c = (p-\ell)m$. Here r_c and s_c are smaller than the lower bounds in Table I because we ignore the data transfer cost to shared memory buffers. As the encryption is performed once by each leader, $r_e = 1$ and $s_e = \ell m$. Since each process decrypts up to $\lceil (N-1)/\ell \rceil$ ciphertexts of ℓm-byte messages, $r_d = \lceil (N-1)/\ell \rceil$ and $s_d = \lceil (N-1)/\ell \rceil \cdot \ell m$. As N and ℓ are powers of two, we can simplify these terms to $r_d = \lceil N/\ell \rceil$ and $s_d = \max\{N, \ell\} \cdot m$.

Although HS1 reduces r_d significantly, it has higher s_e than the Concurrent algorithms as only leaders perform encryption. We also consider the following variant HS2:

1) Each process encrypts its m-byte data and puts the ciphertext to a shared-memory ciphertext buffer.
2) Each leader runs an ordinary all-gather on ciphertexts among the N leaders, and the results are stored in a shared-memory ciphertext buffer.
3) All processes in each node jointly decrypt the $(N-1)\ell$ ciphertexts from other nodes, and place the results in a shared-memory plaintext buffer.
4) All processes in each node copy the results from the shared-memory plaintext buffer to the user buffer.

The communication cost of HS2 is the same as HS1. Each process however only encrypts its own data, and thus $r_e = 1$ and $s_e = m$. Since all processes on each node have to jointly decrypt $(N-1)\ell$ ciphertexts of m-byte data, $r_d = (N-1)$ and $s_d = (N-1)m$. Thus HS2 improves s_e at the cost of increasing r_d. We therefore expect HS2 to perform better than HS1 for large messages, but the latter is a more suitable option for small messages.

V. Performance Study

A. System setup

We implemented all of the encrypted all-gather algorithms in Table II for MVAPICH2-2.3.3, compiled with the default MVAPICH compilation flags and optimization level O2. We used the AES-GCM-128 encryption scheme in the BoringSSL cryptographic library [2]; this library was compiled under the default settings and linked with MPI during the compilation of MVAPICH2-2.3.3.

The experiments were performed on two systems: a local *Noleland* cluster and the Bridges-2 supercomputer at Pittsburgh Supercomputing Center (PSC) [1]. The Noleland system is a cluster at Florida State University of Intel Xeon Gold 6130 CPUs with 2.10 GHz frequency. Each node has 32 cores, and 192GB DDR4-2666 RAM. This cluster runs CentOS-7, and the underlying network is a 100Gbps Mellanox MT28908 Infiniband. We allocated nodes manually, and the same nodes were chosen for all measurements on this cluster.

We also used the PSC Bridges-2 supercomputer with Regular Memory partition. This system has 504 nodes, each equipped with 2 AMD EPYC 7742 CPUs (with 64 cores and 128 threads each). Among those nodes, 448 ones have 256GB RAM and 16 nodes are equipped with 512GB RAM. We use the nodes of 256GB RAM for our experiments in this work. This supercomputer uses 200Gbps Mellanox ConnectX-6-HDR Infiniband and runs CentOS-8.

We used the *OSU_Allgather* benchmark from the OSU benchmark suite [20] to measure the latency of the all-gather operation with different algorithms and message sizes. Each reported latency is an average of 10 runs. In most experiments, the standard deviation is within 10% of the reported mean. In some rare cases the standard deviation is higher, but still within 25% of the reported mean.

B. Results on Noleland

Table III shows the performance of the baseline unencrypted scheme (the MPI_Allgather routine in MVAPICH2-2.3.3), the naive encrypted version (the naive approach with the MPI_Allgather in MVAPICH2-2.3.3), and the best-performing encrypted algorithm for $p = 128$ and $N = 8$ with a block-order process mapping. The best-performing algorithms are also listed. The performance of the baseline is given as the latency in micro-seconds. The performance of Naive and the best-performing encrypted scheme is given as the overhead in percentage (%) with respect to the baseline. The negative overhead for a scheme means that the scheme runs faster than the baseline. For example, for 4KB message size, the baseline latency is 227.00 μs; the overhead for Naive is 75.93% and thus the latency for Naive is $227.00 \cdot (1 + 0.7593) = 339.36$ μs; the overhead for the best scheme is -16.02% and thus the latency for the best scheme is $227.00 \cdot (1 - 0.1602) = 190.63$ μs.

Table III: Performance of the unencrypted MPI, Naive, and the best-performing algorithm on Noleland ($p = 128$ and $N = 8$, with **block-order mapping**). The third and fourth columns show the overhead (%) of Naive and the best encrypted all-gather algorithm (listed in the last column), compared to unencrypted MPI, respectively.

Size	**Latency** (µs) **of MPI**	**Overhead of** Naive	**Overhead of best scheme**	**Best scheme**
1B	10.64	293.20	31.49	O-RD2
2B	9.26	342.86	51.49	HS1
4B	9.35	348.05	51.50	HS1
8B	9.52	364.69	55.96	O-RD
16B	9.91	309.57	53.06	O-RD
32B	10.87	301.63	50.86	O-RD
64B	12.77	265.33	39.14	O-RD
1KB	56.58	111.57	9.91	O-RD
2KB	108.43	95.54	−0.05	C-RD
4KB	227.00	75.93	−16.02	C-RD
8KB	407.83	92.21	6.25	C-Ring
16KB	1602.35	59.35	−45.89	HS2
32KB	2522.14	87.22	−33.54	HS2
256KB	15902.40	136.51	−12.42	HS2
2MB	136604.31	137.50	−13.97	HS2

As shown in Table III, for block-order mapping, compared to unencrypted MPI, Naive has very significant overheads across all message sizes. In contrast, our algorithms have much smaller overheads thanks to the minimal use of encryption and decryption, and the overlapping of communication and computation. For messages larger than 2KB (except for 8KB), they even outperform unencrypted MPI, because the underlying communication cost of HS1, C-Ring, and C-RD is cheaper than that of MPI, as shown in Figure 5.

Specifically, for small messages, O-RD2, O-RD and HS1 are generally the best, because of their small number of encryption, decryption, and communication rounds. For large messages, C-Ring, HS1, and HS2 are the best thanks to their small amount of encrypted and decrypted data. HS2 and HS1 in most cases perform better than C-Ring, as they have smaller communication cost. As expected, HS2 performs better than HS1 due to its smaller amount of encrypted data. C-RD achieves the best performance for medium sized messages.

On the other hand, MPI's default algorithms are sensitive to process mapping. For cyclic-order mapping, as shown in Table IV, for messages larger than 2KB, unencrypted MPI's performance degrades significantly compared to block-order mapping. For example, for 256KB all-gather, MPI's latency

Table IV: Performance of the unencrypted MPI, Naive, and the best-performing algorithm on Noleland ($p = 128$ and $N = 8$, with **cyclic-order mapping**).

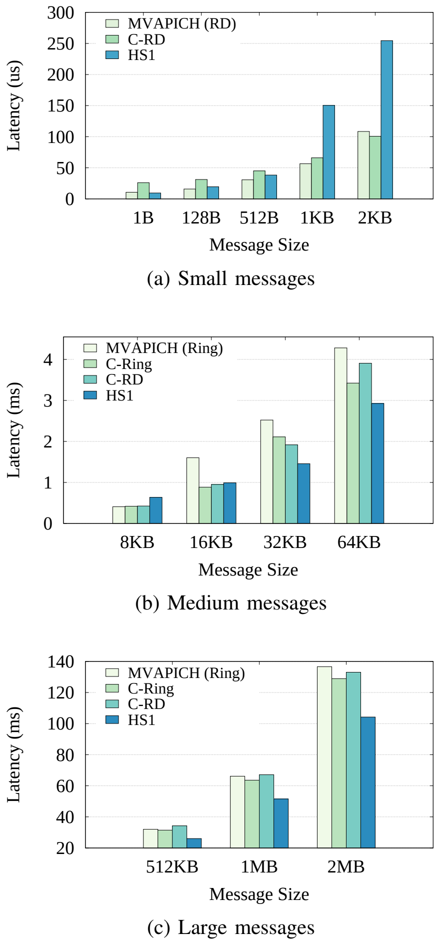
Size	**Latency** (µs) **of MPI**	**Overhead of** Naive	**Overhead of best scheme**	**Best scheme**
1B	10.27	305.67	47.70	O-RD
32B	10.18	324.35	51.21	O-RD
1KB	50.10	128.59	11.54	O-RD
2KB	93.99	104.73	7.33	O-RD
4KB	862.26	18.21	−76.50	O-RD2
8KB	1633.01	20.79	−75.16	HS2
32KB	5541.96	50.85	−63.54	HS2
64KB	10 889.97	44.12	−66.45	C-Ring
256KB	43 355.27	38.92	−61.86	C-Ring
2MB	346 830.02	39.32	−60.92	C-Ring

is 15.9 ms with the block-order mapping, but rises to 43.3 ms with the cyclic-order mapping. Therefore, Naive has better relative overheads in this case, because encryption and decryption are oblivious to process mapping. In addition, as mentioned in Section III, the RD algorithm is sensitive to process mapping, making C-RD somewhat sensitive. The performance of HS1 and HS2 also suffers, because an extra copy is needed for maintaining the correct order of messages when mapping is not in block-order. In contrast, C-Ring is oblivious to process mapping, and thus it becomes the best algorithm for large messages in this setting.

THE UNDERLYING COMMUNICATION COST. To understand the performance of encrypted MPI_Allgather algorithms, we first examine the performance of their unencrypted counterparts. We note that MVAPICH 2.3.3 on Noleland uses RD for small messages and Ring for large messages. For simplicity, we use the same name of each encrypted all-gather algorithm to refer to its unencrypted counterpart, and only consider the most competitive and relevant algorithms for each range of message sizes. Since the unencrypted versions of HS1 and HS2 are identical, we only report the results of the former scheme.

Figure 5 gives the performance of different unencrypted all-gather algorithms with $p = 128$, $N = 8$, and block-order mapping. In general, the MVAPICH implementation is still the best for small messages, but it is considerably outperformed by the best algorithms for medium and large messages. A similar trend is observed on PSC Bridges-2.

Figure 6 reports the performance of unencrypted all-gather algorithms with $p = 128$, $N = 8$, and cyclic-order mapping. As mentioned in Section III, the RD algorithm is sensitive to the process mapping, resulting in a performance drop of the MVAPICH implementation and C-RD when we move from block-order mapping to cyclic-order mapping. The performance of the HS1 algorithm also suffers, because in its Step 4, (i) for block-order mapping, each process can directly copy the entire shared-memory plaintext buffer to the user buffer, but (ii) for cyclic-order mapping, it instead needs to perform p memory copies to re-arrange the messages to proper locations in the user buffer.



(a) Small messages

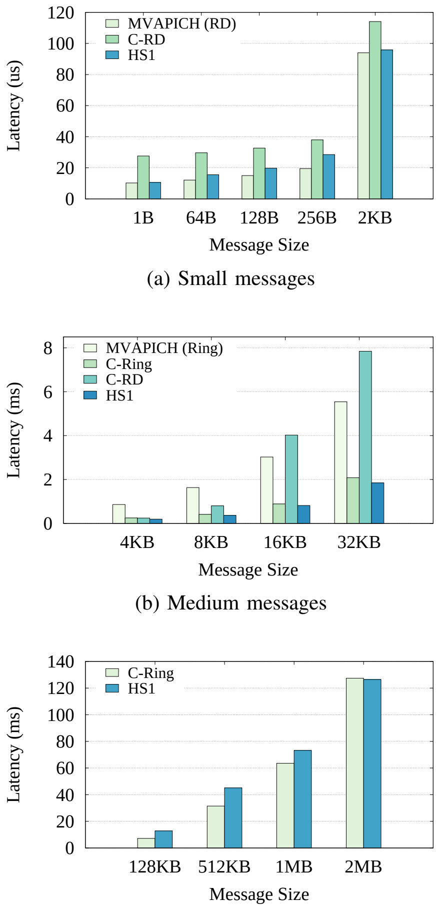
(b) Medium messages

(c) Large messages

Figure 5: Performance of **unencrypted** counterpart of all-gather algorithms on Noleland, with **block-order** mapping.

PERFORMANCE OF ENCRYPTED ALGORITHMS. Figure 7 shows the performance of encrypted all-gather algorithms with $p = 128$, $N = 8$, and block-order mapping. For small messages, O-RD has the overall best performance. This is consistent with the trend in Figure 5, as the underlying unencrypted version of O-RD is exactly the MVAPICH implementation for small messages. For medium and large messages, C-Ring, C-RD, HS1, and HS2 have comparable performance, but HS2 performs slightly better in most cases. This happens because (i) HS2 has cheaper underlying communication cost than C-Ring or C-RD, as shown in Figure 5, and the three schemes have the same s_e and s_d, and (ii) as expected, HS2 is better than HS1 for large messages thanks to its smaller s_e.

We note that C-Ring, C-RD, and HS2, with a decryption size of $(N-1)m$, have very low overheads for large messages with respect to their corresponding unencrypted algorithms. For example, for 1MB message size, latency

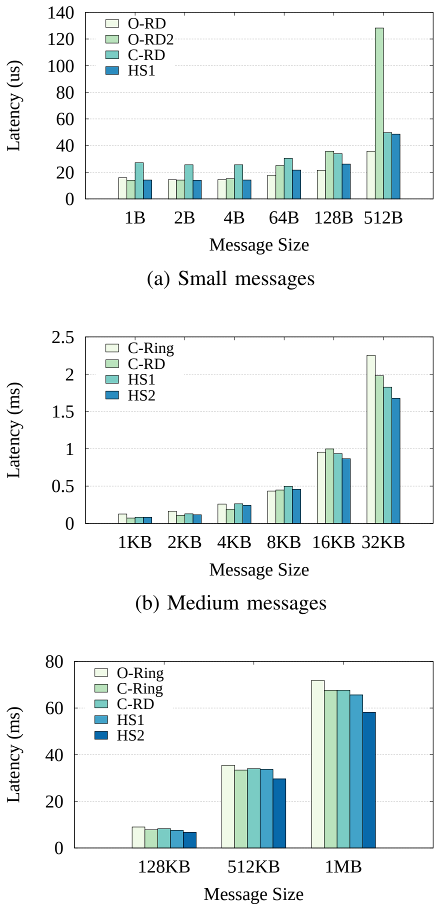


(a) Small messages

(b) Medium messages

(c) Large messages

Figure 6: Performance of **unencrypted** counterpart of all-gather algorithms on Noleland, with **cyclic-order** mapping.



(a) Small messages

(b) Medium messages

(c) Large messages

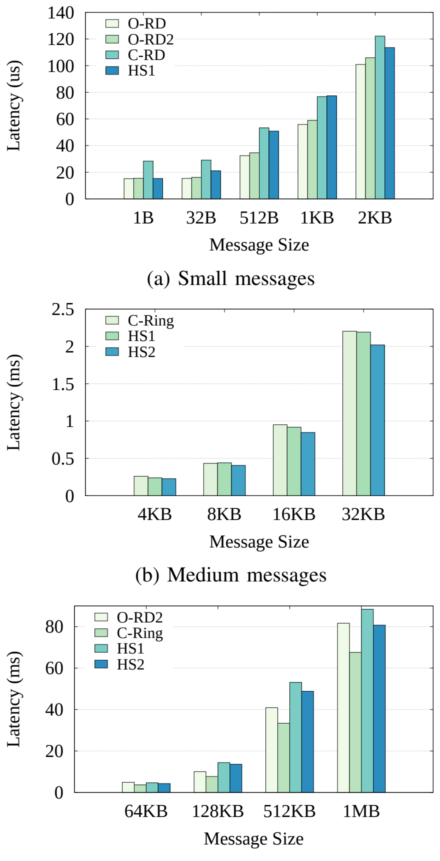
Figure 7: Performance of **encrypted** all-gather algorithms on Noleland, with **block-order** mapping.

of the unencrypted C-Ring is 63.3 ms while latency of the encrypted C-Ring is 67.6 ms, meaning 6.8% overhead; latency of the unencrypted C-RD is 67.1 ms while latency of the encrypted C-RD is 67.6 ms, meaning 0.7% overhead; latency of the unencrypted HS2 is 51.5 ms while latency of the encrypted HS2 is 58.2 ms, meaning 13.0% overhead. This is consistent with the theoretical analysis. For these algorithms, each node must receive $(p-1)m$ messages while only decrypting $(N-1)m = \frac{(p-1)m}{\ell}$ ciphertexts. In other words, the communication cost will dominate the operation when ℓ is large. In contemporary clusters, ℓ—the number of processes per node—is usually a large value. In such a system, with any of these algorithms, the overhead for encryption and decryption is reduced to almost negligible for large messages.

Table V: Performance of the unencrypted MPI, Naive, and the best-performing algorithm on Noleland ($p = 91$ and $N = 7$, with **block-order mapping**).

Size	**Latency** (µs) **of MPI**	**Overhead of** Naive	**Overhead of best scheme**	**Best scheme**
1B	15.85	166.60	−0.49	HS1
32B	18.97	135.55	−6.05	HS1
256B	47.46	65.98	−33.78	HS1
512B	76.64	48.20	−40.40	C-RD
1KB	138.91	35.45	−54.35	C-RD
4KB	154.49	74.46	5.42	C-RD
8KB	261.20	91.08	15.43	C-Ring
32KB	1586.33	77.23	−32.57	C-Ring
64KB	3056.25	74.10	−30.56	HS2
256KB	11 068.30	91.04	−19.26	HS2
2MB	92 496.05	87.95	−19.44	HS2

NON-POWER-OF-TWO SETTINGS. Table V reports the performance of encrypted all-gather algorithms for $N = 91$ and $p = 7$ and block-order mapping. As mentioned in Section III, when N and p are not powers of two, the RD algorithm has to take some extra steps, although the total cost is still bounded by $2 \cdot \lg(p)$. As a result, the performance of RD-based encrypted algorithms decreases in this setting. For messages of 8KB and more, the results are similar to those in Table III (when $N = 8$ and $p = 128$) because the dominating encrypted algorithms in both cases are Ring-based.



(a) Small messages

(b) Medium messages

(c) Large messages

Figure 8: Performance of **encrypted** all-gather algorithms on Noleland, with **cyclic-order** mapping.

C. Results on PSC Bridges-2

We ran large-scale experiments on the Bridges-2 supercomputer at PSC, and observed similar trends in the results. Table VI shows the performance of unencrypted MPI, Naive, and our best-performing algorithms on Bridges-2, with $p = 1024$ and $N = 16$. In Bridges-2, a default mapping is used; for our experiments, processes were in block-order mapping.

Again, Naive has heavy overheads across all message sizes, but the penalty on small messages is the most severe. Our algorithms significantly improve the performance of Naive, and for messages of 1KB or more, the best-performing algorithm can even beat unencrypted MPI. We observe that on Bridges-2, the shared-memory algorithms are the best algorithms overall. HS1 outperforms other algorithms for small messages up to 1KB (except for 512B where O-RD is the best algorithm). For messages of size 2KB or more, HS2 is the best algorithm. In this range of messages, the overhead of Naive over the default MVAPICH algorithm decreases, but the overhead of our best-performing algorithm also drops, which makes it significantly better than the Naive approach. Other results have also been collected, and the trends on Bridges-2 are the same as those on Noleland. We omit those results due to a lack of space.

Table VI: Performance of on Bridges-2 for $p = 1024$ and $N = 16$.

Size	Latency (μs) of MPI	Overhead of Naive	Overhead of best scheme	Best scheme
1B	118.57	344.50	−32.47	HS1
64B	167.21	201.26	16.43	HS1
128B	250.93	512.47	2.22	HS1
512B	750.43	265.85	16.20	O-RD
1KB	1438.99	191.99	−3.15	HS1
2KB	6882.52	11.18	−71.25	HS2
16KB	62 871.60	21.52	−78.10	HS2
64KB	250 752.32	20.88	−80.14	HS2
256KB	1 007 353.08	20.85	−79.41	HS2
512KB	2 007 558.81	20.75	−79.57	HS2

VI. Conclusion

We derive lower bounds on six important performance metrics for encrypted all-gather, and develop new encrypted all-gather algorithms that match these theoretical bounds. The optimizations in the algorithm design significantly reduce the encryption and decryption cost, compared to the naive approach. In particular, for clusters where each node runs many processes, the encryption and decryption overhead in our algorithms are negligible for large messages. Our algorithms even outperform the unencrypted MPI_Allgather in many cases, which indicates that the unencrypted all-gather routines need to be updated to achieve the best performance on modern HPC systems.

VII. Acknowledgment

This material is based upon work supported by the National Science Foundation under Grants CICI-1738912, CRI-1822737, SHF-2007827 and CRII-1755539. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562. Specifically, through allocations ECS190004 and CCR200042, it used the Bridges-2 system, which is supported by NSF award number ACI-1445606, at the Pittsburgh Supercomputing Center (PSC) [27].

References

[1] PSC Bridges. https://www.psc.edu/bridges.

[2] BoringSSL. https://boringssl.googlesource.com/boringssl, 2018.

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

[4] S. Chunduri, S. Parker, P. Balaji, K. Harms, and K. Kumaran. Characterization of MPI usage on a production supercomputer. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 386–400. IEEE, 2018.

[5] M. J. Dworkin. NIST SP 800-38D. Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC. 2007.

[6] A. Faraj, P. Patarasuk, and X. Yuan. Bandwidth efficient all-to-all broadcast on switched cluster. *International Journal of Parallel Programming*, 36(4):426–453, 2007.

[7] A. Faraj and X. Yuan. Automatic generation and tuning of MPI collective communication routines. In *Proceedings of the 19th Annual International Conference on Supercomputing*, ICS '05, pages 393–402, New York, NY, USA, 2005. Association for Computing Machinery.

[8] A. Faraj, X. Yuan, and D. Lowenthal. STAR-MPI: Self tuned adaptive routines for MPI collective operations. In *Proceedings of the 20th Annual International Conference on Supercomputing*, ICS '06, pages 199–208, New York, NY, USA, 2006. Association for Computing Machinery.

[9] A. Faraj, X. Yuan, and P. Patarasuk. A message scheduling scheme for all-to-all personalized communication on Ethernet switched clusters. *IEEE Transactions on Parallel and Distributed Systems*, 18(2):264–276, 2007.

[10] S. D. Girolamo, P. Jolivet, K. Underwood, and T. Hoefler. Exploiting offload enabled network interfaces. *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 26–33, 2015.

[11] R. W. Hockney. The communication challenge for MPP: Intel Paragon and Meiko CS-2. *Parallel computing*, 20(3):389–398, 1994.

[12] S. L. Johnsson and C. . Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers*, 38(9):1249–1268, 1989.

[13] K. Kandalla, H. Subramoni, G. Santhanaraman, M. Koop, and D. K. Panda. Designing multi-leader-based allgather algorithms for multi-core clusters. In *2009 IEEE International Symposium on Parallel Distributed Processing*, pages 1–8, 2009.

[14] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang. MagPIe: MPI's collective communication operations for clustered wide area systems. *SIGPLAN Not.*, 34(8):131–140, May 1999.

[15] S. Kumar and L. V. Kale. Scaling all-to-all multicast on fat-tree networks. In *Proceedings of the Tenth International Conference on Parallel and Distributed Systems, 2004. ICPADS 2004*, pages 205–214, 2004.

[16] M. A. Maffina and R. S. RamPriya. An improved and efficient message passing interface for secure communication on distributed clusters. In *2013 International Conference on Recent Trends in Information Technology (ICRTIT 2013)*, pages 329–334, July 2013.

[17] S. H. Mirsadeghi and A. Afsahi. Topology-aware rank reordering for MPI collectives. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1759–1768, Los Alamitos, CA, USA, may 2016. IEEE Computer Society.

[18] A. Naser, M. Gavahi, C. Wu, V. T. Hoang, Z. Wang, and X. Yuan. An empirical study of cryptographic libraries for MPI communications. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–11, 2019.

[19] A. Naser, C. Wu, M. S. Lahijani, M. Gavahi, V. T. Hoang, Z. Wang, and X. Yuan. CryptMPI: A fast encrypted MPI library, 2020.

[20] D. Panda. OSU micro-benchmark suite, 2011.

[21] P. Patarasuk and X. Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel Distrib. Comput.*, 69(2):117–124, Feb. 2009.

[22] P. Patarasuk, X. Yuan, and A. Faraj. Techniques for pipelined broadcast on Ethernet switched clusters. *Journal of Parallel and Distributed Computing*, 68(6):809 – 824, 2008.

[23] J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. J. Dongarra. Performance analysis of MPI collective operations. In *19th IEEE International Parallel and Distributed Processing Symposium*, 2005.

[24] X. Ruan, Q. Yang, M. I. Alghamdi, S. Yin, and X. Qin. ES-MPICH2: A Message Passing Interface with enhanced security. *IEEE Trans. Dependable Secur. Comput.*, 9(3):361–374, May 2012.

[25] S. Shivaramakrishnan and S. D. Babar. Rolling curve ECC for centralized key management system used in ECC-MPICH2. In *2014 IEEE Global Conference on Wireless Computing Networking (GCWCN 2014)*, pages 169–173, Dec 2014.

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

[27] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, et al. XSEDE: accelerating scientific discovery. *Computing in science & engineering*, 16(5):62–74, 2014.

[28] J. L. Träff. Efficient allgather for regular SMP-clusters. In B. Mohr, J. L. Träff, J. Worringen, and J. Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 58–65, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[29] E. A. Varvarigos and D. P. Bertsekas. Communication algorithms for isotropic tasks in hypercubes and wraparound meshes. *Parallel Computing*, 18(11):1233 – 1257, 1992.

[30] Yuanyuan Yang and Jianchao Wang. Efficient all-to-all broadcast in all-port mesh and torus networks. In *Proceedings Fifth International Symposium on High-Performance Computer Architecture*, pages 290–299, 1999.