Encrypted All-reduce on Multi-core Clusters

Mohsen Gavahi
Department of Computer Science
Florida State University
Tallahassee, FL, USA
gavahi@cs.fsu.edu

Abu Naser
Department of Computer Science
Florida State University
Tallahassee, FL, USA
naser@cs.fsu.edu

Cong Wu
Department of Computer Science
Florida State University
Tallahassee, FL, USA
wu@cs.fsu.edu

Mehran Sadeghi Lahijani
Department of Computer Science
Florida State University
Tallahassee, FL, USA
sadeghil@cs.fsu.edu

Zhi Wang
Department of Computer Science
Florida State University
Tallahassee, FL, USA
zwang@cs.fsu.edu

Xin Yuan
Department of Computer Science
Florida State University
Tallahassee, FL, USA
xyuan@cs.fsu.edu

***Abstract*—We consider the encrypted all-reduce operation on multi-core clusters. We derive performance bounds for the encrypted all-reduce operation and develop efficient algorithms that are theoretically optimal in that they asymptotically achieve the performance bounds. We empirically evaluate our encrypted all-reduce algorithms on production clusters. The results show that with the right algorithm, encryption can be incorporated in the all-reduce operation on large messages without significant overheads on modern multi-core clusters whose compute node has a large number of cores.**

***Keywords*—Message Passing Interface (MPI), encryption, all-reduce**

I. Introduction

There is an ongoing trend to move High Performance Computing (HPC) applications to execute in the public cloud. The security of these applications is becoming a rising concern. Since a large number of HPC applications are developed using Message Passing Interface (MPI), efforts have been made to add encryption to MPI [1], [2], [3], [4], [5] to protect communications.

MPI_Allreduce performs the all-reduce operation among a group of processes and is one of the most widely used MPI collective routines in HPC applications. Research shows that over 40% of the time spent in MPI operations is spent in the reduction operation [6]. This work considers efficient algorithms for encrypted MPI_Allreduce on contemporary multi-core clusters where each compute node has many cores for running MPI processes. We assume that the underlying network does not support secure communication and develop efficient algorithms to incorporate encryption to this operation at the MPI library level to protect inter-node communications.

In a modern multi-core cluster, the encryption and decryption speed is often slower than the network communication speed [4]. As such, developing an efficient algorithm for encrypted all-reduce must take into consideration communication, encryption, and decryption together. We derive the performance bounds for encrypted all-reduce. We then leverage the existing (unencrypted) all-reduce algorithms and design encrypted all-reduce algorithms that theoretically achieve the performance bounds. We empirically evaluate the algorithms on production systems. The results indicate that, with our encrypted all-reduce algorithms, security can be supported with negligible overhead for large messages, which confirms our theoretical analysis.

The rest of the paper is organized as follows. Section II discusses the background and related work. In Section III, we analyze the performance bounds and discuss encrypted all-reduce algorithms. In Section IV, we report the results of the performance evaluation. Finally, we conclude our work in Section V.

978-1-6654-4331-9/21/$31.00 ©2021 IEEE

II. Background

A. MPI_Allreduce

Message Passing Interface (MPI) [7] is the de facto standard for developing portable message passing programs. MPI defines a set of point-to-point and collective communication routines. MPI_Allreduce is an MPI collective operation that involves a group of processes and combines the values from all processes in the group through a reduction operation.

B. Threat model

This study assumes that compute nodes are secure, and the adversary can observe and interfere with network packets. Hence, only inter-node communication needs to be encrypted and the intra-node communication is not observable for adversary. There are other threats such as replay attacks where the adversary replaces a ciphertext by an old one, and side-channel information such as computational timing, are not addressed in this study. Since we assume that compute nodes are secure, cache side-channel attacks are not considered.

C. Encryption

Following Naser *et al.* [4], we use the Galois-Counter Mode (GCM) encryption scheme [8] that *provably* offers both privacy and integrity. To encrypt a plaintext P by GCM,

reserving a *nonce* N, i.e., a public value that must appear at most once per key, is required. Since receiver needs the same nonce N for decryption, the sender should send both the nonce N and the ciphertext C. In our implementation, we pick nonces at random, which is standard-compliant.

We assume that the keys used in GCM have been distributed (this is usually done in the initialization phase such as MPI_Init), and focus on the performance of encrypted all-reduce operations where all inter-node communications are encrypted.

D. Related work

Since the standardization of MPI, many studies have been done to improve MPI collective performance in different situations: some focus on architecture-oblivious techniques [9], [10], [11]; some work on the network topology [12], [13], [14], [15], [16]; some take advantage of SMP and multi-core features [17], [18], [19]; some consider special network features [20]. All-reduce is among the operations considered in these studies. An overview of the earlier all-reduce algorithms is given in [21]. The algorithms for all-reduce can be classified into flat algorithms where all processes are treated the same and hierarchical algorithms where processes within each node are treated as a group. Common flat algorithms include the recursive doubling algorithm [9], Rabenseifner's algorithm [21], and the ring algorithm [22]. Hierarchical all-reduce algorithms are designed for multi-core clusters to capture the performance difference between intra-node communication and inter-node communication. In a hierarchical algorithm, MPI processes within each node are grouped with one or more processes in the group being designated as leaders that are in charge of inter-node communications. The hierarchical all-reduce algorithms include single-leader algorithms [23] and multi-leader algorithms [24]. The recently developed node-aware algorithms are all hierarchical algorithms [25], [26], [27]. Our encrypted all-reduce schemes are built over these algorithms by incorporating encryption and decryption in the algorithms.

Encrypted MPI libraries have also been developed [28], [29], [30], [4]. Earlier encrypted MPI libraries [28], [29], [30] suffer from various issues, including using outdated security algorithms and insufficient privacy and integrity support (a discussion of these issues can be found in [4]). A recent library, CryptMPI [4], [5] has fixed those issues. However, CryptMPI uses a naive method to realize collective routines [4]. Such a method cannot be applied to the all-reduce operation, and thus CryptMPI does not support encrypted MPI_Allreduce. Efficient algorithms for all-gather have been developed by Lahijani [31]. All-reduce is very different from other collectives that do not contain computation (reduction operation) in the middle of the collective operation.

III. Encrypted All-reduce

We consider encrypted all-reduce of m bytes data on p processes and N nodes, where each node has $\ell = p/N$ processes. For simplicity, we will assume that the unit for the reduction operation is byte and that p and N are powers of two. The algorithms can be slightly modified to work for any choice of p and N with the same asymptotic complexity. The experiments in Section IV are based on an implementation of the general version of the algorithms that handle any p and N values. With the GCM mode of encryption, 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. We assume that $N \geq 2$ since a single-node all-reduce does not need encryption.

A. Performance bounds

An encrypted all-reduce operation has four components: encryption, decryption, communication, and reduction computation. Following [27], we use the postal model for communication and reduction computation:

$$T = \alpha_c t + \beta_c s + \gamma c \tag{1}$$

where α_c is the per-message start-up cost, β_c is the per-byte communication cost, γ is the flop rate for the reduction operation, and t, s and c are the number of communication rounds, bytes, and reduction operations, respectively. This model implicitly uses the Hockney's model [32] where communicating an m bytes message takes $\alpha_c + \beta_c m$ time.

We will also use the Hockney's model for encryption and decryption. Encrypting an m-byte message takes $\alpha_e + \beta_e m$ time units and decrypting an m-byte message takes $\alpha_d + \beta_d m$ time units. We establish the lower bounds of seven key performance metrics for encrypted all-reduce:

- r_c: the number of communication rounds to complete the operation,
- s_c: the total size of data that at least one process must send or receive,
- r_e: the number of encryption rounds,
- s_e: the amount of data that at least one process must encrypt,
- r_d: the number of decryption rounds, and
- s_d: the amount of data that at least one process must decrypt,
- s_r: the amount of reduction operations that at least one process must perform.

With these metrics, an encrypted all-reduce algorithm will have at least $t_c = r_c \cdot \alpha_c + s_c \cdot \beta_c$ communication time, $t_e = r_e \cdot \alpha_e + s_e \cdot \beta_e$ encryption time, $t_d = r_d \cdot \alpha_d + s_d \cdot \beta_d$ decryption time, and $t_r = \gamma \cdot s_r$ reduction computation time. For small messages, r_c, r_e, and r_d are the dominating terms, whereas for large messages, s_c, s_e, s_d, and s_r will determine the performance. Depending on how communication and computation overlap, the total time will be between $\max\{t_c, t_e, t_d, t_r\}$ and $t_c + t_e + t_d + t_r$. Since each of these terms may dominate the performance, the algorithm design must consider all of them.

s

Table I shows the bounds for the encrypted all-reduce operation. The lower bounds r_c and s_c for communication cost are well known [9], [33], we list them here for completeness and for comparison to the encryption and decryption costs. For s_r, to compute the results for the reduction of m items (bytes) on p processes requires a total of $(p-1)\cdot m$ reduction operations. Since there are p processes, at least one process will need to perform $\frac{(p-1)\cdot m}{p} \approx m$ operations.

To derive lower bounds for encryption and decryption cost, without loss of generality, we will assume that a process needs exactly one round to encrypt a plaintext or decrypt a ciphertext of any size. Additionally, information within a node can be shared by all processes in the node without extra cost. Hence, in one round, q plaintexts can be encrypted by q processes, and likewise q ciphertexts can be decrypted by q processes.

Let us now consider the number of rounds for decryption (r_d). 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 $T_0 = 1$ 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 its own data), and each ciphertext in the next round 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. By repeating this argument, if the protocol terminates in r_d rounds then at the end of the encrypted gather operation, each node can obtain unencrypted data from at most $(\ell+1)^{r_d}$ nodes. Since at the end of the operation, each node needs to have data from all N nodes, $N \leq (\ell+1)^{r_d}$, and thus $r_d \geq \lceil \lg_{\ell+1}(N) \rceil = \left\lceil \frac{\lg(N)}{\lg(\ell+1)} \right\rceil$. If we assume ℓ is a constant then this bound means that $r_d \in \Omega\big(\lg(p)\big)$ since $N = \frac{p}{l}$. The bound for r_e is same.

For s_e, to complete the operation, each node must send its m bytes information to other nodes (its own data or reductions results with data from other nodes). At least one process in the node must encrypt $\frac{m}{\ell}$ data. Similarly, each node must receive at least m bytes of data to complete the operation and at least one process in the node must decrypt $\frac{m}{\ell}$ bytes data.

B. Encrypted all-reduce algorithms

We use a simple approach to incorporate encryption and decryption that works with any underlying all-reduce algorithm:

TABLE I: Lower bounds for encrypted all-reduce (m-byte message, p processes, N nodes, $\ell = p/N$)

Metric	Lower bound
r_c	$lg(p)$
s_c	$\Omega(m)$
s_r	$\frac{(p-1)m}{p} \approx m$
r_e	$\lceil \frac{\lg(N)}{lg(\ell+1)} \rceil$
s_e	$\frac{m}{\ell}$
r_d	$\lceil \frac{\lg(N)}{lg(\ell+1)} \rceil$
s_d	$\frac{m}{\ell}$

for every inter-node communication, our scheme encrypts the message before communication and decrypts the message after the communication. *The research lies in finding and designing the underlying all-reduce algorithm such that the bounds for r_c, r_e, and r_d can be reached for small messages and that the bounds for s_c, s_e, s_d, and s_r can be reached for large messages.*

ALGORITHMS FOR SMALL MESSAGES

Algorithms for small messages optimize for r_c, r_e, and r_d. With our scheme to incorporate encryption and decryption, clearly, $r_e \leq r_c$ and $r_d \leq r_c$ (if multiple messages can be communicated concurrently in one phase, the messages can also be encrypted and decrypted concurrently in that phase). Hence, conventional all-reduce algorithms that are designed for small messages, which have a small r_c, will yield good results for r_e and r_d.

A common all-reduce algorithm for small messages is the *recursive doubling* (RD) algorithm [9]. RD has $lg(p)$ steps. In step k ($0 \leq k < lg(p)$), each process i exchanges m-byte data with process $i \oplus 2^k$ and performs the reduction. Hence, $r_c = \lg(p)$, $s_c = \lg(p)m$, and $s_r = \lg(p)m$. Depending on the process mapping, the rounds for encryption and decryption may be different (intra-node communication does not require encryption and decryption). In the worst case when each round contains inter-node communication, $r_e = r_c = \lg(p)$ and $r_d = r_c = \lg(p)$. In each round, each process with inter-node communication encrypts and decrypts m-byte messages. Hence, $s_e = m\lg(p)$ and $s_d = m\lg(p)$. Theoretically, if we assume that ℓ is a constant, N and p are in the same order, and r_e and r_d are asymptotically optimum.

The hierarchical single-leader algorithm with RD for the inter-node communication (SL-RD) [23] is also effective for small messages. In SL-RD, each node has one leader that will perform the inter-node operations. The all-reduce operation is performed in three steps. In the first step, an intra-node reduce operation is performed on each node to gather the reduction results for each node in the leader process. In the second step, the leaders in different nodes perform an inter-node all-reduce operation using the RD algorithm to obtain the all-reduce results in all of leaders. In the third step, processes within each node perform an intra-node broadcast operation to distribute the all-reduce results to all processes. If one ignores intra-node communication costs, we have $r_c = r_e = r_d = \lg(N)$, $s_c = s_e = s_d = m\lg(N)$, $s_r = m(\lg(N) + \lg(\ell))$ (assume intra-node reduction is done with a tree).

Another effective hierarchical single-leader algorithm, which we name SL-A, replaces the RD algorithm in the second step in SL-RD with an encrypted all-gather operation to collect all data to all leaders. After that, the leaders perform the reduction operations on all data. This algorithm separates the communication from reduction operations, which can be effective in practice. The cost of this algorithm depends of the encrypted all-gather algorithm. Using the O-RD2 encrypted all-gather algorithm [31] and ignoring intra-node communication costs, we have $r_c = r_e = r_d = \lg(N)$, $s_c = s_e = s_d = m(N-1)$, $s_r = m(N - 1 + \lg(\ell))$.

TABLE II: Performance of encrypted all-reduce algorithms (m-byte message, p processes, N nodes, ℓ processes per node)

	r_c	s_c	s_r	r_e	s_e	r_d	s_d
RD	$\lg(p)$	$m\lg(p)$	$m\lg(p)$	$\lg(p)$	$m\lg(p)$	$\lg(p)$	$m\lg(p)$
SL-RD	$\lg(N)$	$m\lg(N)$	$m(\lg(N)+lg(\ell))$	$\lg(N)$	$m\lg(N)$	$\lg(N)$	$m\lg(N)$
SL-A	$\lg(N)$	$m(N-1)$	$m(N-1+lg(\ell))$	$\lg(N)$	$m\lg(N)$	$\lg(N)$	$m(N-1)$
RS	$2\lg(p)$	$2m$	$\frac{(p-1)m}{p} \approx m$	$2\lg(p)$	$2m$	$2\lg(p)$	$2m$
RING	$2(p-1)$	$2m$	$\frac{(p-1)m}{p} \approx m$	$2(p-1)$	$2m$	$2(p-1)$	$2m$
SL-RS	$2\lg(N)$	$2m$	$m(\frac{N-1}{N}+\frac{\ell-1}{\ell}) \approx 2m$	$2\lg(N)$	$2m$	$2\lg(N)$	$2m$
ML-RD	$\lg(N)$	$(m/\ell)\lg(N)$	$\frac{m}{\ell}(\lg(N)+(\ell-1))$	$\lg(N)$	$(m/\ell)\lg(N)$	$\lg(N)$	$(m/\ell)\lg(N)$
ML-RS	$2\lg(N)$	$2m/\ell$	$\frac{m}{\ell}(\frac{N-1}{N}+(\ell-1)) \approx m$	$2\lg(N)$	$2m/\ell$	$2\lg(N)$	$2m/\ell$
ML-RING	$2(N-1)$	$2m/\ell$	$\frac{m}{\ell}(\frac{N-1}{N}+(\ell-1)) \approx m$	$2(N-1)$	$2m/\ell$	$2(N-1)$	$2m/\ell$

ALGORITHMS FOR LARGE MESSAGES

Algorithms for large messages optimize for s_c, s_e, s_d, and s_r. A common algorithm is the Rabenseifner's algorithm (RS) [21]. In this algorithm, the all-reduce operation is performed by a reduce-scatter followed by an all-gather. The reduce-scatter is done by recursive halving and the all-gather is done by recursive doubling [21]. With RS, $r_c = 2\lg(p)$, $r_s = 2\frac{p-1}{p}m \approx 2m$ and $s_r = \frac{p-1}{p}m$. When adding encryption, the rounds and amount of encryption and decryption depend on how the processes are mapped. In the worst case, $r_e = r_c = 2\lg(p)$, $s_e = s_c = 2\frac{p-1}{p}m \approx 2m$, $r_d = r_c = 2\lg(p)$, and $s_d = s_c = 2\frac{p-1}{p}m \approx 2m$.

Another algorithm for large messages is the Ring algorithm (RING) [22]. Like RS, RING also performs a reduce-scatter followed by an all-gather. However, both reduce-scatter and all-gather are performed with a ring pattern [22]. With RING, $r_c = 2(p-1)$, $r_s = 2\frac{p-1}{p}m \approx 2m$ and $s_r = \frac{p-1}{p}m$. When adding encryption, the rounds and amount of encryption and decryption depend on how the processes are mapped. In the worst case, $r_e = r_c = 2(p-1)$, $s_e = s_c = 2\frac{p-1}{p}m \approx 2m$, $r_d = r_c = 2(p-1)$, and $s_d = s_c \approx 2m$.

Single leader hierarchical algorithms can use RS or RING in the second step. As table II shows, for RS, RING, and SL-RS, s_e and s_d are $2m$, which is significantly larger than the lower bound $\frac{m}{\ell}$, especially when ℓ is large. Thus, none of these are optimal for encrypted all-reduce on large messages. Next, we will introduce algorithms, which are multi-leader hierarchical algorithms [24], that achieve the lower bounds for s_c, s_r, s_e, and s_d.

In a multi-leader hierarchical algorithm, each node has multiple leader processes that will collectively perform inter-node operations. The number of leaders can be any value from 1 to ℓ in theory. However, for encrypted all-reduce, we found that making all processes in a node leaders (ℓ leaders) yields the best performance when m is sufficiently large. In this approach, the all-reduce operation is performed in three steps. In the first step, an intra-node reduce-scatter operation is performed on each node to (1) produce the local reduction results in the node and (2) to distribute the results evenly among leaders in the node. This is done by copying data to shared memory, and then having each of the ℓ processes calculate the reduction results of $\frac{m}{\ell}$-byte data. Hence, each node perform $\frac{(\ell-1)m}{\ell}$ reduction operations in this step. After this step, Leader 0 in each node has local reduction results for the first $\frac{m}{\ell}$ bytes of the reduction results in the node; Leader 1 has the reduction results for the second $\frac{m}{\ell}$ bytes of local reduction results in the node; and so on.

In the second step, Leaders i, $0 \leq i \leq \ell-1$, across all nodes form the i-th sub-group. There are ℓ sub-groups and ℓ concurrent all-reduce operations are performed, one for each sub-group, on $\frac{m}{\ell}$ bytes local reduction results to obtain final reduction results across all processes. After the second step, Leader i-th in every node will have final reduction results for the i-th $\frac{m}{l}$ bytes data. All final reduction results now exist in each node. In the third step, processes in each node perform a local all-gather operation to distribute final reduction results to all processes. Using shared memory, this can be done by each process copying the results from shared memory to its local memory.

Our implementation uses shared memory. Different flat all-reduce algorithms can be used to perform the concurrent sub-group all-reduce's in the second step. We will use ML-RD, ML-RS, and ML-RING to denote the algorithms with RD, RS, and RING in the second step, respectively. Hence, ignoring the intra-node communication costs, with RD, we have $r_c = r_d = r_e = \lg(N)$, $s_c = s_d = s_e = \frac{m}{l}\lg(N)$, $s_r = \frac{m}{\ell}(\lg(N)+\ell-1)$; with RS, we have $r_c = r_d = r_e = 2\lg(N)$, $s_c = s_d = s_e = \frac{2m}{l}$, $s_r = \frac{m}{\ell}(\frac{N-1}{N}+\ell-1) \approx m$; and with RING, we have $r_c = r_d = r_e = 2(N-1)$, $s_c = s_d = s_e = \frac{2m}{l}$, $s_r = \frac{m}{\ell}(\frac{N-1}{N}+\ell-1) \approx m$.

Table II summarizes the performance of the algorithms. For large messages, ML-RS and ML-Ring reach the bounds for s_c, s_e, s_d, and s_r. For large messages, each process encrypts and decrypts $O(\frac{m}{\ell})$ data while communicating $O(m)$ data and performing reduction computation on $O(m)$ data. This indicates that on modern multi-core clusters where ℓ is large, encryption can be supported in the all-reduce operation without significant overheads. This is confirmed in our experiments.

IV. PERFORMANCE STUDY

A. Experiment setup

We implemented all of the encrypted all-reduce algorithms listed in Table II in MVAPICH2-2.3.3. We compiled the library with the default MVAPICH compilation flags and optimization level O2. We used the AES-GCM-128 encryption scheme in the BoringSSL cryptographic library [34]; this library was compiled under default settings and linked with MPI during the complication of MVAPICH2-2.3.3.

TABLE III: Performance of MPI_Allreduce on Noleland ($p = 128$ and $N = 8$).

Size	Unencryp. MVAPICH Latency (us)	Encrypted MVAPICH Latency (us)	Best Encryp. Latency (us)	Best Over-head	Best Encryp. Methods
4B	7.63	14.44	11.52	51.0	SL-A
64B	7.72	14.15	12.18	57.8	SL-A
512B	11.46	18.83	18.59	62.2	SL-A
1KB	13.86	21.87	19.58	41.4	ML-RD
8KB	25.31	43.13	28.38	12.3	ML-RD
32KB	44.42	66.62	45.09	1.7	ML-RD
64KB	67.27	78.08	72.73	7.2	ML-RS
256KB	254.21	341.96	274.53	8.0	ML-RING
1MB	1318.48	1342.49	954.84	-27.6	ML-RS
4MB	5555.11	6720.13	4608.25	-17.0	ML-RING

The experiments were performed on two systems: a local $Noleland$ cluster and the Bridges-2 supercomputer at Pittsburgh Supercomputing Center (PSC) [35]. Noleland is equipped with Intel Xeon Gold 6130 CPUs with 2.10 GHz frequency. Each node has 16 cores, 32 threads, and 187GB DDR4-2666 RAM. This cluster runs CentOS-7, and the underlying network is a 100 Gbps Mellanox MT28908 Infiniband. We allocated nodes manually, and the same nodes were chosen for all measurements on this cluster. All experiments reported on Noleland have $p = 128$, $N = 8$, and $\ell = 16$.

The second system is the Bridges-2 supercomputer at Pittsburgh Supercomputing Center (PSC) [35]. We used the Regular Memory partition that has 504 nodes, each equipped with 2 AMD EPYC 7742 CPUs and 64 cores per CPU. We used the nodes with 256GB of RAM for our experiments. This system has 200Gbps Mellanox ConnectX-6-HDR Infiniband and runs CentOS-8. All results reported on Bridges-2 have $p = 1024$, $N = 16$, and $\ell = 64$. The mapping of the MPI ranks is controlled by the system and unknown to us.

We used the *OSU_Allreduce* benchmark from the OSU benchmark suite [36] and MiniAMR [37] to measure the latency of all-reduce operation with different algorithms and message sizes. In the both benchmarks, the main reduction operation is MPI_SUM and the latencies reported in this article are an average of at least 10 runs for each experiment.

B. Results on Noleland

Table III shows the performance for different sizes on the Noleland cluster. In this experiment, $p = 128$, $N = 8$, $\ell = 16$. The table shows the performance of the baseline MVAPICH, encrypted MVAPICH (the encryption scheme directly applied to MVAPICH), the best performing encrypted algorithm, and the overhead of the best performing algorithm over the baseline MVAPICH. We will use *unencrypted* to denote the unencrypted MVAPICH, *encrypted* to denote the encrypted MVAPICH, and *best* to denote the best performing algorithm. MVAPICH uses SL-RD for messages in range of 4B to 2KB, SR in range of 4KB to 1MB, and reduce-scatter (uses ring pattern) followed by an all-gather for 2MB and larger sizes. The table also gives the best performing algorithms.

As shown in the table III, while *encrypted* exposes significant overheads to the operation across all message sizes, *best* improves fairly significantly over *encrypted*. This shows the effectiveness of the new algorithms. For this configuration, SL-A is the best for small messages (4B to 512B), and ML-RD for medium sized messages (1KB to 32KB). For large messages (64KB and larger), ML-RS and ML-RING have very similar performance that is much better than the rest. For 1MB and 4MB cases, ML-RS and ML-RING actually perform better than the *unencrypted* baseline. This is because (1) the underlying communication algorithm is more efficient, and (2) the encryption overhead is very small for large messages with ML-RS and ML-RING: as shown in our analysis, each process must communicate $O(m)$ messages while only encrypting and decrypting $O(\frac{m}{l})$ data.

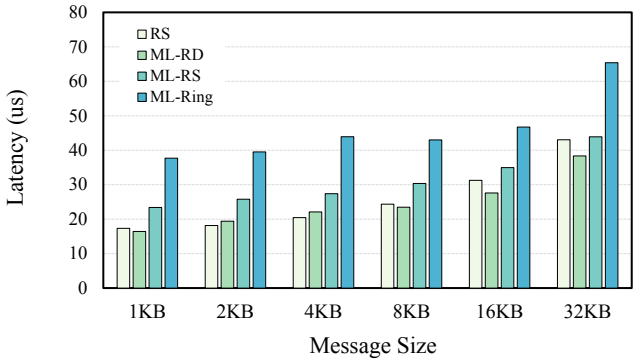
To further understand the results in Table III, Figures 1 and 2 show the performance of the unencrypted and encrypted algorithms, respectively. Figure 1a shown the unencrypted performance for medium sized messages (1KB to 32KB), where both the number of rounds and the total size have impacts. From Table II, we can see that RS, ML-RD, and ML-RS are good candidates since they have logarithmic rounds of communication, encryption, and decryption and relatively small total sizes. For this message size range and without encryption, RS is slightly better than ML-RD for 2KB and 4KB messages and slightly worse for other message sizes; ML-RS is slightly worse than ML-RD across all sizes. When encryption is incorporated, ML-RD introduces much less overheads than RS: s_e and s_d for ML-RD are $\frac{m}{l}\lg(N)$, which is less than $2m$ for RS. As a results, encrypted ML-RD is much better than encrypted RS. The encryption overhead for ML-RD is only slightly more than ML-RS and since the unencrypted ML-RD is faster than unencrypted ML-RS, encrypted ML-RD performs better than ML-RS.

For large messages (64KB to 4M), unencrypted ML-RS and ML-RING perform noticeably better than other algorithms as shown in Figure 1b. They are also the algorithms that add the least overhead when encryption is incorporated: s_e and s_d are the asymptotically optimal $\frac{2m}{l}$ for these two algorithms. They achieve the best performance for large messages as shown in Figure 2b. Additionally, the encryption overhead in ML-RS and ML-RING is very small: the encrypted and unencrypted ML-RS and ML-RING have very similar latencies, which confirms the analysis in Section III.

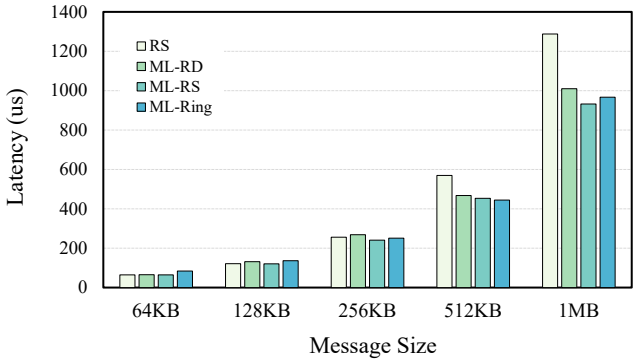
We also evaluate the all-reduce performance in MiniAMR [37], which is a 3D-stencil based adaptive mesh refinement kernel that uses all-reduce extensively. The frequency of Mesh Refinement is set to 1000. We evaluated MiniAMR for the number of refinement steps from 4K to 128K. As Table IV shows, *best* noticeably improves the *encrypted* MVAPICH in this benchmark.

C. Results on PSC Bridges-2

The results from larger scale experiments on PSC Bridges-2 yield similar trends as those from Noleland. Tables V shows the results on Bridges-2 with $p = 1024$ and $N = 16$. Again,

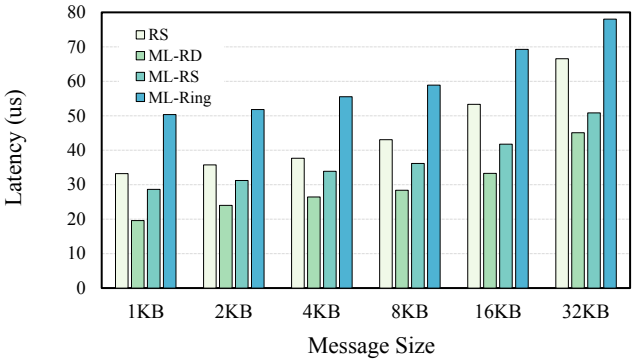


(a) Medium Message

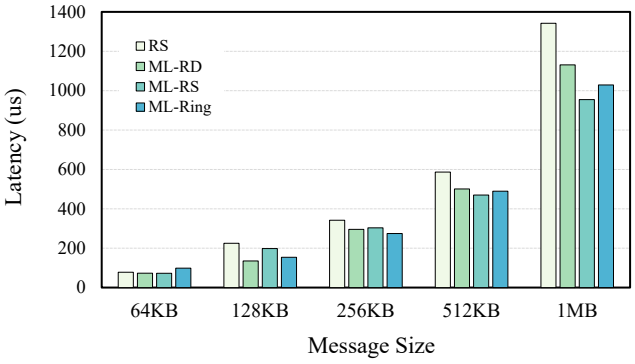


(b) Large Message

Fig. 1: Performance of unencrypted all-reduce algorithms



(a) Medium Message



(b) Large Message

Fig. 2: Performance of encrypted all-reduce algorithms

TABLE IV: Performance of miniAMR on Noleland ($p = 128$ and $N = 8$).

Number of Refin.	Unencryp. MVAPICH Latency (sec)	Encrypted MVAPICH Latency (sec)	Best Encryp. Latency (sec)	Best Over-head	Best Encryp. Methods
16K	7.99	9.88	8.76	9.6	ML-RD
32K	18.16	21.11	19.94	9.7	ML-RD
64K	55.77	60.32	53.68	-3.7	ML-RD
128K	213.72	228.05	189.13	-11.5	ML-RING

TABLE V: Performance of MPI_Allreduce on PSC Bridges-2 ($p = 1024$ and $N = 16$).

Size	Unencryp. MVAPICH Latency (us)	Encrypted MVAPICH Latency (us)	Best Encryp. Latency (us)	Best Over-head	Best Encryp. Methods
4B	25.57	44.65	43.75	71.2	SL-A
16B	19.82	46.03	36.68	85.07	SL-A
128B	21.78	50.11	43.98	101.93	SL-A
2KB	39.25	72.79	72.79	85.45	SL-RD
4KB	68.07	130.31	119.50	75.55	SL-RD
8KB	92.57	196.75	135.10	45.9	ML-RD
64KB	402.28	571.14	402.21	0.0	ML-RD
128KB	742.12	1045.34	735.77	-0.9	ML-RD
256KB	1428.10	1625.31	1057.95	-25.9	ML-RS
2MB	12687.91	14015.21	8716.61	-31.3	ML-RS
4MB	27541.10	31761.25	12863.05	-53.3	ML-RS

encrypted MVAPICH introduces large overheads across all message sizes. *Best* improves the performance significantly. For message sizes of 16KB or more, *best* out-performs the *unencrypted* baseline. Same explanation as that for Noleland applies to Bridges-2. Similar to Noleland, SL-A is the best for small messages in the range of 4B to 128B, but for messages in the range of 256B to 4KB, SL-RD is winner. Besides this, ML-RD is the best for the medium sizes in the range from 8KB to 128KB and ML-RS is the best when the message size is larger than 128KB. This trend is observed in the results from Noleland. The collected results for miniAMR on the Bridges-2 show that the trend is the same as the results on Noleland.

V. Conclusion

In this study, the lower bounds on seven key performance metrics for encrypted all-reduce are derived. We identify and develop encrypted all-reduce algorithms that achieve these bounds. The empirical evaluation on two production systems shows that our encrypted all-reduce algorithms achieve significantly improvement over naively incorporating encryption in the algorithms used in the current production MPI library. Our encrypted algorithms introduce minor overheads for large messages and out-perform the unencrypted baseline in many cases. This indicates that the algorithms used in the current MPI library are not most efficient for modern multi-core clusters and an update is necessary.

VI. Acknowledgment

We thank Prof. Viet Tung Hoang at Florida State University for the discussion and valuable inputs to this research. This material is based upon work supported by the National Science

Foundation under Grants CICI-1738912, CRI-1822737, and SHF-2007827. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) 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. This work used the XSEDE Bridges resource at the Pittsburgh Supercomputing Center (PSC) through allocations ECS190004 and CCR200042.

REFERENCES

[1] 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.

[2] 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.

[3] Xiaojun Ruan, Qing Yang, Mohammed I. Alghamdi, Shu Yin, and Xiao Qin. ES-MPICH2: A Message Passing Interface with enhanced security. *IEEE Trans. Dependable Secur. Comput.*, 9(3):361–374, May 2012.

[4] 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.

[5] Abu Naser, Cong Wu, Mehran Sadeghi Lahijani, Mohsen Gavahi, Viet Tung Hoang, Zhi Wang, and Xin Yuan. CryptMPI: A fast encrypted MPI library, 2020.

[6] Rolf Rabenseifner. Optimization of collective reduction operations. In *International Conference on Computational Science*, pages 1–9. Springer, 2004.

[7] Message Passing Interface Forum. *MPI: A Message-passing Interface Standard, Version 3.1 ; June 4, 2015*. High-Performance Computing Center Stuttgart, University of Stuttgart, 2015.

[8] Morris J Dworkin. *Sp 800-38d. recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac*. National Institute of Standards & Technology, 2007.

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

[10] Ahmad Faraj and Xin 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.

[11] Ahmad Faraj, Xin Yuan, and David 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.

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

[13] 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.

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

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

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

[17] Jesper Larsson Träff. Efficient allgather for regular SMP-clusters. In Bernd Mohr, Jesper Larsson Träff, Joachim Worringen, and Jack Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 58–65, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[18] 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.

[19] 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.

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

[21] Rolf Rabenseifner. A new optimized mpi reduce algorithm. *http://www. hlrs. de/mpi/myreduce. html*, 1997.

[22] Pitch Patarasuk and Xin Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel Distrib. Comput.*, 69(2):117–124, February 2009.

[23] K Kandalla, U Yang, Jeff Keasler, T Kolev, Adam Moody, Hari Subramoni, Karen Tomko, Jérôme Vienne, Bronis R de Supinski, and Dhabaleswar K Panda. Designing non-blocking allreduce with collective offload on infiniband clusters: A case study with conjugate gradient solvers. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, pages 1156–1167. IEEE, 2012.

[24] Mohammadreza Bayatpour, Sourav Chakraborty, Hari Subramoni, Xiaoyi Lu, and Dhabaleswar K Panda. Scalable reduction collectives with data partitioning-based multi-leader design. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2017.

[25] Surabhi Jain, Rashid Kaleem, Marc Gamell Balmana, Akhil Langer, Dmitry Durnov, Alexander Sannikov, and Maria Garzaran. Framework for scalable intra-node collective operations using shared memory. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 374–385. IEEE, 2018.

[26] Jahanzeb Maqbool Hashmi, Sourav Chakraborty, Mohammadreza Bayatpour, Hari Subramoni, and Dhabaleswar K Panda. Designing efficient shared address space reduction collectives for multi-/many-cores. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1020–1029. IEEE, 2018.

[27] Amanda Bienz, Luke Olson, and William Gropp. Node-aware improvements to allreduce. In *2019 IEEE/ACM Workshop on Exascale MPI (ExaMPI)*, pages 19–28. IEEE, 2019.

[28] MA Maffina and RS 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)*, pages 329–334. IEEE, 2013.

[29] Xiaojun Ruan, Qing Yang, Mohammed I Alghamdi, Shu Yin, and Xiao Qin. Es-mpich2: A message passing interface with enhanced security. *IEEE Transactions on Dependable and Secure Computing*, 9(3):361–374, 2012.

[30] Shridhar Shivaramakrishnan and Sachin D Babar. Rolling curve ecc for centralized key management system used in ecc-mpich2. In *2014 IEEE Global Conference on Wireless Computing & Networking (GCWCN)*, pages 169–173. IEEE, 2014.

[31] Mehran Sadeghi Lahijani, Abu Naser, Cong Wu, Mohsen Gavahi, Viet Tung Hoang, Zhi Wang, and Xin Yuan. Efficient algorithms for encrypted all-gather operation. In *the 35th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, May 2021.

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

[33] Pitch Patarasuk and Xin Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2):117–124, 2009.

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

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

[36] DK Panda. OSU micro-benchmark suite, 2011.

[37] Aparna Sasidharan and Marc Snir. Miniamr-a miniapp for adaptive mesh refinement. Technical report, 2016.