An Overlap study for Cluster Computing

Eduardo Colmenares[1]
Department of Computer Science
Midwestern State University
Wichita Falls, Texas
eduardo.colmenares@mwsu.edu

Per Andersen[2]
High Performance Computing Center (HPCC)
Texas Tech University
Lubbock, Texas
per.andersen@ttu.edu

Bingyang Wei[3]
Department of Computer Science
Midwestern State University
Wichita Falls, Texas
bingyang.wei@mwsu.edu

***Abstract*—Distributed memory systems (DMS) (clusters) are one of the tools being used by researchers to solve a wide spectrum of computational intensive problems in a fraction of the time of a sequential approach. The nature of a DMS does not enforce intense data sharing among computational nodes, this will occur if the problem under analysis happens to be data dependent in nature. The latency associated with dynamic data sharing in a DMS is well known to increase the total execution time. One of the possible techniques that can be used to reduce the negative effects associated with this latency is overlapping. In this paper we show why a characterization of the overlapping capabilities of a cluster is important to justify results.**

Keywords—cluster; communication-computation overlapping; synchronization-computation overlapping; latency; data-dependent

I. Introduction

A very important and commonly used strategy in scenarios where dynamic data sharing among multiple participant processes is required is overlapping of communication and computation. The purpose of overlapping is to contribute to the reduction of the negative effects associated with the latency generated by intense data sharing in a multicore environment by allowing the processes to engage in useful computation while some additional activities such as communication take place [13]. The authors in this paper present a study of overlapping in two different clusters, by using two variants of overlapping, overlapping of synchronization and computation, and overlapping of communication and computation. The first cluster is a multi-user research oriented cluster (MUC), while the second is a one-user cluster with older technology (OUC).

II. Testing Environments

Initially we used two different clusters as our testing environments. The first cluster corresponds to a multi-user cluster which provides scientific computational capabilities to a research community. The second testing environment is a personal cluster with only one user and a maximum of 9 nodes.

A. Multi User Cluster (MUC)

This cluster has 12TB of public shared lustre storage and three groups of public and private nodes, all connected by SDR InfiniBand and Gigabit Ethernet. The quad-core nodes have Infinihost III Lx (PCI-e) cards, and the older nodes have Infinihost (PCI-X) cards.

1) Public quad-core (512 cpu, 4.77 TF).
64 nodes with dual quad-core Intel 5345 processors (2.33 GHz) and 12GB of memory each. Designated compute-1-x, 2-x.

2) Public single-core (128 cpu, 0.82 TF).
64 nodes with dual single-core Intel "Irwindale" processors (3.2 GHz) and 4 GB of memory each. Designated compute-3-x, 4-x, 5-x.

3) Public AMD dual-core (8 cpu, .04 TF).
1 node with quad dual-core AMD 8218 processors (2.60 GHz) and 64GB of memory. Designated compute-8-1

B. One User Cluster (OUC)

Each one of the computational nodes in this cluster has the following hardware characteristics: one Intel(R) Pentium(R) 4 CPU at 1.70GHz, one 3Com PCI 3c905C Tornado network card. All the nodes in this cluster are interconnected via a 3Com® Super Stack® 3 Switch 3300 12-Port. Table 1 summarizes the major hardware differences between nodes.

Table 1: Hardware Differences among Nodes for OUC

Node	Memory (MB)	Hard Disk
0, 3	511.46	40020 MB-T340016A, ATA
1	1023.4	40020 MB-T340016A, ATA
2,4, 5	1023.4	20547 MB-MAXTOR 6L020J1, ATA
6	1023.4	40020 MB-WDC WD400BB-75DEA0, ATA
7, 8	1023.4	40027 MB-MAXTOR 6L040J2, ATA

III. Overlap tests

Two overlap tests were implemented to characterize both clusters, MUC and OUC. The first of these tests examines the support of synchronization with computation, while the second evaluates the support of overlap of communication with data transfer.

A. Overlap of Synchronization with Computation

According to [12], overlap of synchronization with computation in a MPI implementation is supported if a message can be sent from process A to process B without requiring a previous synchronization step between A and B.

For this test two processes are considered. Processes 0 and 1, executed in workstations 0 and 1 respectively. The test is

978-1-4673-9795-7/15 $31.00 © 2015 IEEE
DOI 10.1109/CSCI.2015.103

structures as follows; process 0 will send a message to process 1 using a non-blocking send instruction, and process 1 will receive the message using a blocking receive. The synchronization between both processes is enforced with a barrier, as shown in figure 1.

Process 0 returns almost immediately after the MPI_Isend [7, 8, 10], and proceeds to process its workload before issuing an MPI_Wait. When process 0 reaches MPI_Wait, the execution of further instruction at process 0 will be blocked until the message is totally sent. At the same time, right after the barrier, process 1 will compute a quarter of the workload computed by process 0, as soon as process 1 has finished its quarter of workload, it will block further instructions until the reception of the message is completed [12]. Figure 1 is based on a figure shown in [12].

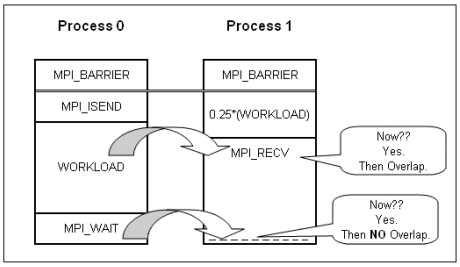

Figure 1: Test for Overlap of Synchronization with Computation.

The completion time of the MPI_Recv right after the barrier on process 1, will be considered the outcome of this test [12]. It is possible to infer that the time needed by process 0 to complete its workload will be more than the time needed by process 1 to complete the processing of a quarter of the same workload.

The results will confirm if overlap has been supported or not. If the completion time of delivery at process 0 is almost the same time needed by process 1 to compute its quarter of workload, then overlap has been supported. Different conclusions are achieved from those other cases where the completion time is almost equal to the time required by process 0 to process its workload. In cases like this, it is possible to conclude that overlap has not been supported.

This procedure which we refer to as case-1, will be executed 1000 times, and the average values of these 1000 executions will be used to generate our conclusions. The code executed by process 0 has been structured in such a way that it makes an attempt for overlapping of computation with data transfer "if supported".

In order to conclude whether overlap of communication and computation exists in process 0, the following steps need to be followed. Step 1 (case-2), remove the workload from process 0, and send the message 1000 times using MPI_Isend. The time needed to complete a sending will be saved, this means that by the end of the 1000 iterations, 1000 execution-time-samples will be available, and an accurate average value can be estimated.

Step two (case-3), consider the workload once again, execute the routine, record the time needed to complete the processing of a single workload, repeat until you have collected all 1000 samples, next compute the average time.

Step three, add the average values obtained in case-2 and case-3. The summation of these two values implies the non-existence of overlapping, next, compare this no-overlap value against the value computed where both MPI_Irecv and workload worked together in the same application (case-1). Thus, if a difference exists, it is because overlap of computation and data transfer was supported.

B. Overlap of Data Transfer with Computation

According to [2, 4, 12], a MPI implementation capable of performing useful computation while data is in transit between processes is said to support overlap of data transfer with computation. The percentage of improvement achieved by the overlap can be computed by using equation (1) [12].

$$improvement(\%) = \left\{ \frac{T_{overlap} - T_{nonoverlap}}{T_{nonoverlap}} \right\} \times 100 \quad (1)$$

This approach is based on a test referenced by [12]. Although based on ideas presented in [12], our approach does not make use of matrix multiplication; instead the authors replaced matrix multiplication with a different workload of reduced programming complexity. In a similar way as explained in [12], this second overlap procedure consists of two tests. The first test handles an scenario where the program has been written in such a way that it will get no benefit from overlap, it is shown in figure 2, and is called the non-overlap test.

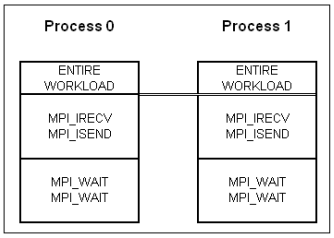

Figure 2: Non-overlap Test.

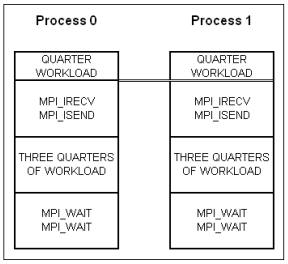

Figure 3: Overlap Test.

The second test, covers the same scenario, except it will make an attempt for overlapping of computation with communication. In this case, the whole workload has been divided into two parts, one quarter and three quarters. This procedure is called the overlap test, and is shown in figure 3.

IV. RESULTS

A. Synchronization and Computation Overlap

1) Using Delays as Workload: The size of the message sent from process 0 to process 1 was approximately 39.06 KB or 10000 integers. This technique was carried out 100 times in both clusters, a script was used to automatize the executions [9], and its nature is shown in figure 4, which has been taken from [12] .

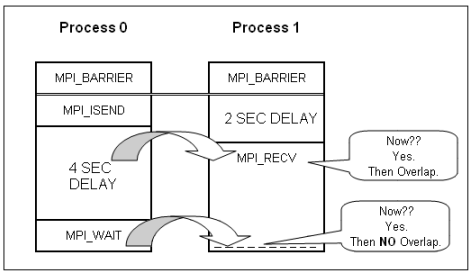

Figure 4: Test for Overlap of Synchronization with Computation.

The results are summarized in tables 2 and 3. From these two tables, it is clear that no overlap of synchronization with computation is supported by any of these two clusters.

Table 2: Synchronization and Computation Overlap for MUC

Multi User Cluster (MUC)				
Collected Samples	**Process**	**Min (secs)**	**Max (secs)**	**Average (secs)**
106	0	3.99E+00	4.06E+00	4.00E+00
	1	3.99E+00	4.06E+00	4.00E+00

Table 3: Synchronization and Computation Overlap for OUC

One User Cluster (OUC)				
Collected Samples	**Process**	**Min (secs)**	**Max (secs)**	**Average (secs)**
100	0	3.99E+00	4.01E+00	4.00E+00
	1	3.99E+00	4.02E+00	4.00E+00

It is meaningful to mention that both systems, MUC and OUC have a MPICH2 compiler. In order to find out if the overlap of synchronization with computation was potentially hardware related, the same test was executed on a third cluster, a multi-user Solaris cluster, which we refer to as MSC. This third cluster uses LAM and a MPICH1 compiler.

Table 4. Synchronization and Computation Overlap for "Solaris-Cluster"

Multi User Solaris Cluster (MSC)				
Collected Samples	**Process**	**Min (secs)**	**Max (secs)**	**Average (secs)**
100	0	3.99E+00	4.47E+00	4.03E+00
	1	1.99E+00	2.05E+00	2.00E+00

Comparing Tables 2, 3, and 4, it is possible to conclude that support for overlapping of synchronization and computation existed in the cluster that uses MPICH1, the Solaris cluster.

It is essential to find a reason to justify this behavior, for such purpose a new test is proposed and implemented. It is called "Message size sweep".

2) Message Size Sweep: This test is based on recommendations and ideas mentioned in [10]. This technique has some similarities with the previously introduced synchronization and computation overlap approach, however, some important modifications are considered.

The first modification is that both processes, process 0 and process 1 will only use non-blocking instructions, MPI_Isend and MPI_Irecv respectively. The second consideration is that process 0 will be the only process with a computational workload that does not use a delay. Finally, process 1 just needs to process the MPI_Irecv and then wait for its completion.

Figure 5 presents the nature of the procedure to be repeated several times, for different message sizes. The initial size was 10 KB, with increments of 10 KB, until a final size of 85 KB was reached. The repetition of this core procedure is what the authors of this paper have named "message size sweep" and constitutes an original contribution. This contribution was developed based on comments and ideas provided by [10], and the nature of the frequency sweep test in electronics [3].

For process 0, the elapsed time from MPI_Isend to MPI_Wait has been called T_{P0}. For process 1, the time from MPI_Irecv to MPI_wait has been called T_{P1}, both are given in seconds [3]. This test will help to determine if the size of the message plays a critical role in the support for overlapping of synchronization with computation.

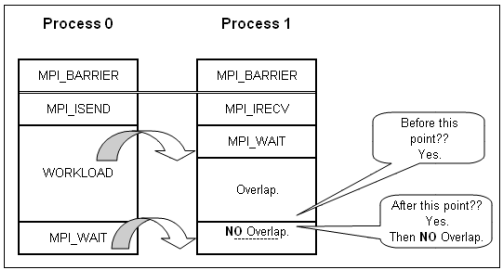

Figure 5: Core Procedure for the Message Size Sweep Test.

Table 5 shows the results of the proposed "message size sweep" method in two different clusters. The code used in both clusters is the same, and the differences in processing time are consequence of the different computational capabilities between clusters.

A comparison between both halves of table 5, will lead to the conclusion that the size of the message plays a relevant role for the achievement of overlapping of synchronization and computation.

It is clear that in each system, the size of the buffer used to send and receive messages has a different limit. If the size of the message to be sent, is smaller than the limit, the system is

able to support overlap of synchronization with computation, otherwise the support will not be available [3].

Table 5: Message Size Sweep Results on (MSC) and (OUC)

SOLARIS-CLUSTER - "MPICH1"			OUC - "MPICH2"	
Message Size (KB)	**TP0 (secs)**	**TP1 (secs)**	**TP0 (secs)**	**TP1 (secs)**
10	61.0	1.08E-03	3.8	1.24E-03
20	61.0	1.54E-03	3.8	3.8
30	61.0	1.92E-03	3.8	3.9
40	61.0	2.69E-03	3.8	3.8
50	97.6	3.62E-03	3.8	3.8
60	61.0	2.82E-03	3.8	3.8
70	61.0	61.0	3.8	3.8
80	61.0	61.1	3.8	3.8

The first half of table 5 shows that for the multi-user Solaris Cluster (MSC), the limit is a value between 60KB and 70KB. The second half of table 6 shows a limit between 10KB and 20KB for OUC.

According to [7, 10], when non-blocking instructions are being used to establish communication between processes and the size of the message is larger than the socket buffer size; non-blocking instruction will no longer behave as non-blocking and the support of overlap of synchronization with computation is no longer available.

The previous finding explains why during the synchronization and computation overlap test that uses a two, and four seconds delays, overlap was supported by MSC. This is shown in table 4, section IV-A1. It is also significant to mention that the size of the message used for this procedure was 39.06 KB; and 39.06 KB is greater than the 10 KB limit supported by OUC, but less than the 70 KB limit supported by MSC [3].

The synchronization and computation overlap using the 2 seconds and 4 seconds delays was repeated in OUC for a message size of 10 KB using a script to automatize the executions [9]. The results are presented in table 6.

Table 6: Synchronization and Computation Overlap for "My Cluster-10 KB"

One User Cluster OUC				
Collected Samples	**Process**	**Min (secs)**	**Max (secs)**	**Average (secs)**
10	0	3.99	4.01	4.00
	1	1.99	2.00	1.99

B. *Communication and Computation Overlap*

1) Using Computation as Workload and Based on Synchronization and Computation Overlap Test: According to [3], the work performed by process 0 during the synchronization and computation overlap test can also be used to estimate if some percentage of computation and communication overlap is achieved by process 0 during such test. In order to find this out, two actions need to be completed. The first one, is to partition the task performed by process 0 into two sub-tasks. The first of these sub-tasks provides the time needed to send a message from process 0 to process 1 without considering the workload to be processed by process 0, this time has been called $T_{message\text{-}P0}$. The second sub-task contribute with the time needed by process 0 to process its workload, this time has been called $T_{workload\text{-}P0}$.

Once $T_{message\text{-}P0}$ and $T_{workload\text{-}P0}$ have been computed we add them up, and the result will be the time needed by process 0 to complete its task when no overlap of computation and communication has been supported. This time has been called $T_{non\text{-}overlap}$.

For the completion of the second step, it is necessary to compute the time required by process 0 to complete its task considering the possibility of overlap. This time has been called $T_{overlap}$ and it is the time to go from MPI_Isend to MPI_Wait on process 0.

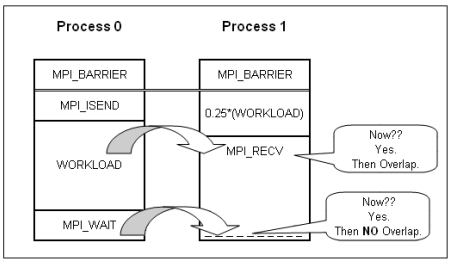

Figure 6: Synchronization and Computation Overlap Test Using Workloads.

The workloads used for this test are shown on figure 7. The one called work_load( ) has been assigned to process 0, while work_load_quater( ) has been assigned to process 1. The value assigned to "maximum" is 100000.

```
void work_load()
{
        int i;
        for(i=0; i<maximum; i++)
        {
                my_summ=my_summ+0.001;
        }
}
```

```
void work_load_quarter()
{
        int i;
        for(i=0; i<maximum/4; i++)
        {
                my_summ=my_summ+0.001;
        }
}
```

Figure 7: Entire Workload and a Quarter of Workload.

Table 7 and table 8 show the results for the previous described test. These tests were conducted on OUC, for two different message sizes, 10 KB and 65K respectively.

Each one of the sub-tasks was executed 1000 times, all executions were hardcoded and no script was used. A total of 1000 samples were collected per parameter, and no single outlier was removed.

The percentage of improvement was computed by using equation (1) [12], and corresponds to the rightmost column in tables 8 and 9. A positive value means that no improvement was achieved. Careful observation of tables 8 and 9 will show that $T_{overlap}$ is greater than $T_{nonoverlap}$ for two different message

sizes. This means that the effectiveness of the overlap version is not as good as the effectiveness of the non-overlap version.

It should be possible to observe, that independent of the size of the message, the overlap of communication and computation is not supported by OUC, when this procedure is followed.

One additional conclusion that can be derived, is that support for overlap of synchronization and computation does not imply support for overlap of communication and computation.

Table 7: Communication-Computation Overlap results for a Message of 10KB.

Statistics	NO OVERLAP			OVERLAP	Eq (1)
1000 samples	$T_{message-P0}$ (secs)	$T_{workload-P0}$ (secs)	$T_{non-overlap}$ (secs)	$T_{overlap}$ (secs)	%
Minimum	4.60E-05	3.77E-04	4.23E-04	4.23E-04	-
Maximum	3.03E-04	9.33E-03	9.63E-03	7.11E-03	-
Average	8.10E-05	4.15E-04	4.96E-04	5.14E-04	**3.61**
Stdev	1.10E-05	2.99E-04	3.10E-04	2.40E-04	-

Table 8: Communication-Computation Overlap results for a Message of 65KB.

Statistics	NO OVERLAP			OVERLAP	Eq (1)
1000 samples	$T_{message-P0}$ (secs)	$T_{workload-P0}$ (secs)	%	$T_{overlap}$ (secs)	%
Min	3.31E-04	3.77E-04	7.08E-04	7.77E-04	-
Max	3.03E-03	9.33E-03	1.24E-02	3.04E-03	-
Avg	3.90E-04	4.15E-04	8.05E-04	8.32E-04	**3.29**
Stdev	1.17E-04	2.99E-04	4.16E-04	9.37E-05	-

2) A Better Test to Evaluate the Overlap of Communication with Computation: There are two major differences between this test and the one presented in section IV (A2). The first major difference is that now both processes are using non-blocking instructions. The second and probably the most important difference is that now it is possible to check if overlap of communication and computation has been supported not only in process 0 but also process 1.

This approach compares two versions of the same program. The first version named non-overlap, shown in see figure 8, does not attempt to achieve overlapping of communication and computation. The second and complementary version shown in figure 9, was structured to take advantage of overlap of computation and communication. This method is based on ideas provided by [12], but important differences are implemented. The authors discarded the use of matrix multiplication, instead an equally effective workload of reduced programming complexity was used.

In this test, each process will complete its task in a time equal to T_{total}. The comparison between the non-overlap version and the overlap-version will be used to determine the percentage of overlap achieved by the overlap version. The percentage of improvement was computed by using equation (1) as suggested by [12].

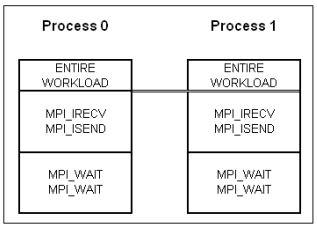

Figure 8: Non-Overlap Version.

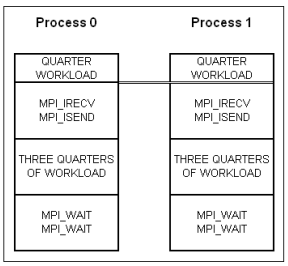

Figure 9: Overlap Version.

This approach was executed a total of 1000 times in OUC, for a message size of 7KB. No script was used to automatize the test, instead its execution was hardcoded in the program. This means that T_{total} was recorded 1000 times for process 0 and a 1000 more for process 1. The statistics associated with this test are shown in table 9 and table 10.

Table 9: Statistics for Process 0–Communication and Computation Overlap - 7 KB

Statistics	T_{total} for P0 (secs)		Overlap
1000 Samples	Non-Overlap version	Overlap version	(%)
Minimum	5.28E-04	5.23E-04	-
Maximum	4.66E-02	7.01E-03	-
Average	1.08E-03	7.95E-04	**-26.27**
Stdev	1.68E-03	3.12E-04	-

Table 10: Statistics for Process 1–Communication and Computation Overlap - 7 KB

Statistics	T_{total} for P1 (secs)		Overlap
1000 Samples	Non-Overlap version	Overlap version	(%)
Minimum	1.24E-03	9.60E-04	-
Maximum	9.89E-02	9.27E-03	-
Average	1.80E-03	1.22E-03	**-32.23**
Stdev	3.71E-03	4.11E-04	-

The rightmost column in tables 9 and 10, show a negative percentage of improvement for both processes. This means that the system has achieved overlap of computation and communication [12].

It is relevant to recall that the main criteria to compute the percentage of overlap, is the average value, and no outliers have been removed from the set of collected samples. It is also valuable to consider that the acquired numerical values are small (in the order of milliseconds) and that the slightest difference might be interpreted as a reasonable percentage of improvement.

This test was repeated once again on OUC, but this time for a message size of 70 KB. The statistics associated with this test are shown in table 11 and table 12.

Table 11: Statistics for Process 0–Communication and Computation Overlap–70 KB

Statistics	**T_{total} for P0 (secs)**		**Overlap**
1000 Samples	**Non-Overlap version**	**Overlap version**	**(%)**
Minimum	1.25E-03	1.23E-03	-
Maximum	4.86E-02	2.50E-02	-
Average	6.92E-03	6.65E-03	**-3.97**
Stdev	2.69E-03	1.20E-03	-

Table 12: Statistics for Process 1–Communication and Computation Overlap-70 KB

Statistics	**T_{total} for P1 (secs)**		**Overlap**
1000 Samples	**Non-Overlap version**	**Overlap version**	**(%)**
Minimum	7.07E-03	6.79E-03	-
Maximum	2.21E-01	6.95E-02	-
Average	8.45E-03	7.77E-03	**-8.07**
Stdev	9.69E-03	2.32E-03	-

For a message size of 70 KB, the percentage of overlap achieved decreases drastically if compared to the percentage achieved when a message which size is smaller than the system socket buffer [3]. It is essential to highlight that only two processes were used for this test, and because of that, the sequentialization introduced by the network switch is minimum.

CONCLUSIONS

According to results presented in section IV-B2, MPICH-2 provides some support for the overlapping of communication with computation. However, section IV-B2 presents an overlap test where only two processes are used, and the exchange of information between two processes does not present the same amount of traffic required by a system with a larger number of processes.

It is clear that as the number of participant processes grow, the highest the level of sequentialization required to coordinate the interchange of information, and it is possible that this might reduce the level of overlap [3, 7]. As concluded by [3], overlap of communication and computation provided by MPICH-2 might have poor scalability and its support is also hardware dependent.

Non-blocking instructions in MPICH-2 and non-blocking instructions on MPICH-1 provide limited support for overlapping of computation and communication. This support is directly related to the size of the message and/or the cumulative size of all messages to be sent. If the message size is larger than the system socket buffer, then non-blocking instructions no longer will behave as expected and the support for overlap will be negatively impacted [3]. For these reasons, choosing the right type of data for an application becomes more difficult.

REFERENCES

[1] B. Barret, G. Shipman, and A. Lumsdaine. "Analysis of Implementation for MPI-2 One-Sided". *In EuroPVM/MPI,* 2007, pp. 242-250.

[2] C. Bell, D. Bonachea, R. Nishtala, and K. Yelick. "Optimizing Bandwith Limited Problems Using One-sided Communication and Overlap". *Parallel and Distributed Processing Symposium*, Apr. 29, 2006.

[3] E. Colmenares, "Overlapping Communication and Computation with MPI-2 for Floyd's Algorithm," master's thesis, Dept. Computer Science, Texas Tech University, 2008.

[4] A. Danalis, K. Kim, L. Polloc, and M. Swany. "Transformations to Parallel Codes for Communication Overlap". *Proc. of the 2005 ACM/IEEE SC|05 Conference.* (SC 05), IEEE CS Press, pp 58-58.

[5] W. Gropp, E. Lusk, and R. Thakur, *Using MPI-2: Advanced Features of the Message-Passing Interface*, MIT Press, 1999.

[6] J. Hein, S. Booth and M. Bull, *Exchanging Multiple Messages via MPI*, HPCx Consortium, EPCC University of Edinburgh, 2003.

[7] T. Hoefler, A. Lumsdaine, W. Rehm. "Implementation and performance Analysis of Non-Blocking Collective Operations for MPI". *International Conference for High Performance Computing, Networking, Storage and Analysis*. Reno, Nevada, 2007.

[8] T. Hoefler, J. Squyres, G. Bosilca, G. Fagg, A. Lumsdaine, and W. Rehm. "Non-Blocking Collective Operations for MPI-2". *Presentation at the High Performance Computing Center Stuttgart (HLRS)*, Stuttgart, Germany, Dec. 2007.

[9] Rice University - Information Technology. "Advanced Unix Scripts", Sep. 2003. Available: http://www-teaching.physics.ox.ac.uk/Unix+Prog/rice/pdf/unix18.pdf

[10] T. Saif, and M. Parashar. "Understanding The Behavior and Performance of Non-blocking Communications in MPI". *10th International Euro-Par Conference,* Italy, 2004, pp.173-182.

[11] L. Schneidenbach, and B. Schnor. "Design Issues in the Implementation of MPI2 One Sided Communication in Ethernet Based Networks". *In IASTED International Multi-conference of Parallel and Distributed Computing and Networks*, 2007, pp. 277-284.

[12] J. White III and S. Bova. (1999). Where is the Overlap? an Analysis of Popular MPI Implementations. Available: http://citeseer.ist.psu.edu/297838.html

[13] K. Tomko, H. Subramoni, A. Awan, K. Hamidouche, D. Pekurouvski, A. Venkatesh, S. Chakrabort, D. Panda. Designing Non-Blocking Personalized Collectives with Near Perfect Overlap for RDMA-Enabled Clusters. *High Performance Computing*, 2015, pp. 434-453.