MPIBlib: Benchmarking MPI Communications for Parallel Computing on Homogeneous and Heterogeneous Clusters

Alexey Lastovetsky, Vladimir Rychkov, and Maureen O'Flynn

School of Computer Science and Informatics, University College Dublin, Belfield, Dublin 4, Ireland
{alexey.lastovetsky,vladimir.rychkov,maureen.oflynn}@ucd.ie
http://hcl.ucd.ie

Abstract. In this paper, we analyze existing MPI benchmarking suites, focusing on two restrictions that prevent them from a wider use in applications and programming systems. The first is a single method of measurement of the execution time of MPI communications implemented by each of the suites. The second one is the design of the suites in the form of a standalone executable program that cannot be easily integrated into applications or programming systems. We present a more flexible benchmarking package, MPIBlib, that provides multiple methods of measurement, both operation-independent and operation-specific. This package can be used not only for benchmarking but also as a library in applications and programming systems for communication performance modeling and optimization of MPI operations.

Keywords: MPI, benchmark, parallel computing, computational cluster, communication performance model.

1 Introduction

Accurate estimation of the execution time of MPI communication operations plays an important role in optimization of parallel applications. *A priori* information about the performance of each MPI operation allows a software developer to design a parallel application in such a way that it will have maximum performance. This data can also be useful for tuning collective communication operations and for the evaluation of different available implementations. The choice of collective algorithms becomes even more important in heterogeneous environments. In addition to general timing methods that are universally applicable to all communication operations, MPIBlib includes methods that can only be used for measurement of some particular operations. Where applicable, these operation-specific methods work faster than their universal counterparts and can be used as time-efficient alternatives. The efficiency of timing methods will be particularly important in self-adaptable parallel applications using run-time benchmarking of communication operations to optimize their performance on the executing platform.

A typical MPI benchmarking suite uses only one timing method to estimate the execution time of the MPI communications. The method provides a certain accuracy and efficiency. The efficiency of the timing method is particularly important in self-adaptable parallel applications using runtime benchmarking of communication operations to optimize their performance on the executing platform. In this case, less accurate results can be acceptable in favor of a rapid response from the benchmark. In this paper, we analyze different timing methods used in the benchmarking suites and compare their accuracy and efficiency on homogeneous and heterogeneous clusters. Based on this analysis, we design a new MPI benchmarking suite called MPIBlib that provides a variety of timing methods. This suite supports both fast measurement of collective operations and exhaustive benchmarking.

In addition to general timing methods that are universally applicable to all communication operations, MPIBlib includes methods that can only be used for measurement of one or more specific operations. Where applicable, these operation-specific methods work faster than their universal counterparts and can be used as their time-efficient alternatives.

Most of the MPI benchmarking suites are designed in the form of a standalone executable program that takes the parameters of communication experiments and produce a lot of output data for further analysis. As such, they cannot be integrated easily and efficiently into application-level software. Therefore, there is a need for a benchmarking *library* that can be used in parallel applications or programming systems for communication performance modeling and tuning communication operations. MPIBlib is such a library that can be linked to other applications and used at runtime.

The rest of the paper is structured as follows. Section 2 outlines existing benchmarking suites. We analyze different methods of measuring MPI communication operations, which are implemented in the suites or described in work on MPI benchmarking. Section 3 describes main features of MPIBlib, the benchmarking library that provides a variety of operation-independent and operation-specific methods of measurement. In Section 4, we discuss application of the library. The results of experiments on homogeneous and heterogeneous clusters are presented in Section 5. We compare the results and costs of different methods of measurement and focus on measuring point-to-point, scatter and gather communication operations as their results are used in the estimation of parameters of advanced heterogeneous communication performance models.

2 Related Work

There are several commonly used MPI benchmarking suites [1]-[5]. The aim of all these suites is to estimate the execution time of MPI communication operations as accurate as possible. In order to evaluate the accuracy of the estimation given by different suites, we need a unified definition of the execution time. As not all of the suites explicitly define their understanding of the execution time, we suggest the following as a natural definition. The execution time of a communication operation is defined as the real (wall clock) time elapsed from the start of the operation, given all the participating processors have started the operation simultaneously, until the successful completion of the operation by the last participating processor. Mathematically, this time can be defined as the minimum execution time of the operation, given that the participating processors do not synchronize their start and are not participating in any other communication operation. It is important to note that the definition assumes that we estimate the execution time for a single isolated operation.

Practically, the execution time of the communication operation is estimated from the results of an experiment that, in addition to the operation, includes other communications and computations. As parallelism introduces an element of non-determinism, there is a problem of reproducibility of such experiments. The methodology of designing reproducible communication experiments is described in [1]. It includes:

- Repeating the communication operation multiple times to obtain the reliable estimation of its execution time,
- Selecting message sizes adaptively to eliminate artifacts in a graph of the output of the communication operation, and
- Testing the communication operation in different conditions: cache effects, communication and computation overlap, communication patterns, non-blocking communication etc.

In the **mpptest** suite [1], these ideas were implemented and applied to benchmarking point-to-point communications.

The execution time of communication operations depends on the MPI library, native software, and hardware configurations. NetPIPE [2] provides benchmarks for different layers in the communication stack. It is based on the ping-pong communication experiments that are implemented over **memcpy**, TCP, MPI etc. In addition to evaluation of communication performance, this suite helps us identify where inefficiencies lie.

Regarding both the reproducibility of communication experiments and the dependency on communication layers, we focus on benchmarking not only point-to-point operations but also collective ones. We analyzed several MPI benchmarking suites that include tests for collective operations. Despite the different approaches to what and how to measure, they have several common features:

- computing an average, minimum, maximum execution time of a series of the same communication experiments to get accurate results;
- measuring the communication time for different message sizes – the number of measurements can be fixed or adaptively increased for messages when time is fluctuating rapidly;
- performing simple statistical analysis by finding averages, variations, and errors.

The MPI benchmarking suites are also very similar in terms of the software design. Usually, they provide a single executable that takes a description of communication experiments to be measured and produces an output for plotting utilities to obtain graphs.

As more than two processors are involved in collective communications and connected in different ways (communication trees), there are two main issues concerned with the estimation of execution time of MPI collective operations:

– measuring the execution time, and
– scheduling the communication experiments.

2.1 Measuring the Execution Time of MPI Collective Operations

Estimation of the execution time of the communication operation includes the selection of two events marking the start and the end of the operation respectively and measuring the time between these events. First of all, the benchmarking suites differ in what they measure, which can be:

– The time between two events on a single designated processor,
– For each participating processor, the time between two events on the processor, or
– The time between two events but on different processors.

The first two approaches are natural for clusters as there is no global time in these environments where each processor has its own clock showing its own local hour. The local clocks are not synchronized and can have different clock rates, especially in heterogeneous clusters. The only way to measure the time between two events on two different processors is to synchronize their local clocks before performing the measurement. Therefore, the third approach assumes the local clocks to be regularly synchronized. Unlike the first two, this approach introduces a measurement error as it is impossible to keep the independent clocks synchronized all the time with absolute accuracy.

In order to measure time, most of the packages rely on the MPI_Wtime function. This function is used to measure the time between two events on the same processor (the local time). For example, the execution time of a roundtrip can be measured on one process and used as an indication of the point-to-point communication execution time [3], [5]. The execution time of a collective communication operation can also be measured at a designated process. For collective operations with a root, the root can be selected for the measurement. As for many collective operations the completion of the operation by the root does not mean its completion by all participating processes, short or empty messages can be sent by the processors to the root to confirm the completion. A barrier, reduce, or empty point-to-point communications can be used for this purpose. The result must be corrected by the average time of the confirmation. The drawback of this approach is that the confirmation can be overlapped with the collective operation and hence it cannot simply be subtracted. As a result, this technique may give negative values of the execution time for very small messages.

The accuracy of this approach (***root timing***) is strongly dependent on whether all processes have started the execution of the operation simultaneously. To ensure the more or less accurate synchronization of the start, a barrier, reduce, or empty point-to-point communications can be used. They can be overlapped with the collective operation to be measured and previous communications as well. To achieve even better synchronization, multiple barriers are used in the benchmarking suites [3]-[5].

The local times can be measured on all processes involved in the communication and the maximum can be taken as the communication execution time. This approach (***maximum timing***) is also dependent on synchronization of the processes before communication, e.g. with a barrier.

To measure the time between two events on different processors, the local clocks of the processors have to be synchronized. Such synchronization can be provided by the MPI global timer if the MPI_WTIME_IS_GLOBAL attribute is defined and true. Alternatively, local clocks of two processors A and B can be synchronized by the following simple algorithm implemented in MPIBench [4]. Processor A sends a message to processor B, which contains the current time plus a half of the previously observed minimum roundtrip time. Processor B receives the message and returns it to A, which calculates the total time that the roundtrip took to complete. If the roundtrip time is the fastest observed so far, then the estimated time of arrival of the initial message is the most accurate yet. If so, processor B calculates the current approximation of the time offset as the message's value received in the next iteration. The processors repeat this procedure until a new minimum roundtrip time has not been observed for a prearranged number of repetitions. Given A being a base processor, this synchronization procedure is performed sequentially for all pairs (A, B_i). A similar procedure is implemented in SKaMPI [5] to find offsets between local times of the root and the other processes.

As local clocks can run at different speeds, especially in heterogeneous environments, the synchronization has to be regularly repeated. The synchronization procedures are quite costly and introduce a significant overhead in benchmarking when used. As soon as the global time has been set up, the time between two events on different processors can be measured [4], [5]. The accuracy of this approach will depend on the accuracy of the clock synchronization and on whether processors start the communication simultaneously. The ***global timing*** usually gives a more accurate estimate because its design is closer to the natural definition of the communication execution time given in the beginning of this section. However, while being more time-efficient, the methods based on local clocks can also provide quite accurate results for many popular platforms and MPI implementations. Therefore, it makes sense to allow a choice of different methods so the user may choose the most efficient for benchmarking with a required accuracy. This is especially important if the benchmarks are to be used in the software that requires the runtime results of the benchmarking.

2.2 Scheduling the Communication Experiments

To obtain a statistically reliable estimate of the execution time, a series of the same experiments are typically performed in the benchmarking suites. If the communications are not separated from each other in this series, the successive

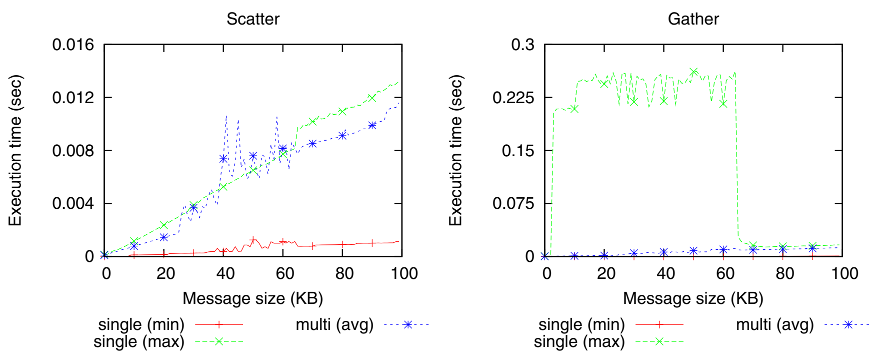

Fig. 1. IMB benchmark on a 16-node heterogeneous cluster: single/multiple scatter/gather measurements

executions may overlap, resulting in a so-called pipeline effect [6], when some processes finish the current repetition earlier and start the next repetition of the operation before the other processes have completed the previous operation. The pipeline affects the overall performance of the series of the operations, resulting in inaccurate averaged execution time. This is the case for the IMB (former PMB) benchmark [3], where the repetitions in a series are not isolated in the attempt to prevent participation of the processes in third-party communications. The IMB measures the communication execution times locally on each process, and the minimum, maximum, and average times are then returned. Fig. 1 shows the results returned by the IMB on a 16-node heterogeneous cluster for scatter and gather operations when single and multiple repetitions are used in the experiments. One can see that for the scatter experiments with a single repetition, the minimum time represents the execution time of a non-blocking send on the root and is therefore relatively small. In the gather experiments with a single repetition, the maximum time is observed on the root, reflecting the communication congestion. The difference between the minimum and maximum times decreases with an increase in the number of repetitions. In both cases, we observe a clear impact of the pipeline effect on the measured execution time of the operation:

– **Scatter:** For small and large messages, the execution time of a repetition in the series is smaller than that measured in a single experiment. For medium-sized messages, escalations of the execution time are observed that do not happen in single experiments.
– **Gather:** Escalations of the execution time for medium-sized messages, observed for single experiments, disappear with the increase of the number of repetitions due to the pipelining.

Thus, the pipeline effect can significantly distort the actual behavior of the communication operation, given that we are interested in accurate estimation of the time of its single and isolated execution.

In order to find the execution time of a communication operation that is not distorted, it should be measured in isolation from other communications. A barrier, reduce, or point-to-point communications with short or empty messages can be used between successive operations in the series. The approach with isolation gives results that are more accurate.

Some particular collective operations and their implementations allow for the use of more accurate and efficient methods that cannot be applied to other collective operations. One example is the method of measurement of linear and binomial implementations of the MPI broadcast on heterogeneous platforms proposed in [7]. It is based on measuring individual tasks rather than the entire broadcast and therefore it does not need the global time. An individual task is a part of the broadcast communication between the root and the i-th process. In each individual task, the pipelining effect is eliminated by sending an acknowledgement message from the i-th process to the root. The execution time of the task is then corrected by the value of the point-to-point execution time.

The acquisition of detailed knowledge of the implementation of collective operations can prove useful towards improving the efficiency of measurement methodologies. This becomes particularly important for benchmarking performed at runtime with on-the-fly optimization of communication operations.

3 MPIBlib Benchmarking Suite

This work is motivated by the absence of an MPI benchmarking suite that would satisfy the following requirements:

- The suite is implemented in the form of library allowing its integration into application-level software.
- The suite provides a wide range of timing methods, both universal and operation/implementation specific, allowing for the choice of the optimal (in terms of accuracy and efficiency) method for different applications and executing platforms.

We have developed such a benchmarking library, MPIBlib, the main goal of which is to support accurate and efficient benchmarking of MPI communication operations in parallel applications at runtime. The main features of MPIBlib can be summarized as follows.

MPIBlib is implemented in the form of library and includes the benchmarks for point-to-point and collective communication operations. The MPIBlib design was influenced by our work on development of the software tool for automated estimation of parameters of the heterogeneous communication performance model proposed in [8]. The software tool widely uses MPIBlib at runtime for accurate and efficient estimation of the execution time of point-to-point, scatter, and gather communications, which is required to find the parameters of the model.

To provide reliable results, the communication experiments in each benchmark are repeated either fixed or variable number of times. The latter allows the user to control the accuracy of the obtained estimation of the execution time. Namely, the definition of each benchmarking function includes the following arguments:

– Input: the minimum and maximum numbers of repetitions, min_reps and max_reps ($min\_reps \leq max\_reps$), and a maximum error, $alpha$ ($0 < alpha < 1$);
– Output: the actual number of repetitions, $reps$, and error, a.

Assigning to min_reps and max_reps the same values results in the fixed number of repetitions of the communication operation, with the error arguments being ignored. As communication operations in a series are isolated from each other, we suppose that the measurement errors are distributed according to the normal law, which enables estimating within a confidence interval, $(1 - alpha)$. If $min\_reps < max\_reps$, the experiments are repeated until the sample satisfies the Student's t-test or the number of repetitions reaches its maximum. In this case, the number of repetitions the benchmark has actually taken, reps, and the final error, a, are returned. For statistical analysis, the GNU Scientific Library [9] is used.

The point-to-point benchmarks can be run either sequentially or in parallel on the communicator consisting of more than two processors. The point-to-point benchmark estimates the execution time of the roundtrips between all pairs of processes in the MPI communicator, $i \xleftrightarrow[M_2]{M_1} j$, $i < j$. It returns three arrays, each of which contains C_n^2 values corresponding to each pair: estimations of execution time, numbers of repetitions and errors. Several point-to-point communications as well as statistical analysis can be performed in parallel, with each process being involved in no more than one communication. This allows us to significantly reduce the overall execution time of the point-to-point benchmark code and gives us quite accurate results on the clusters based on switched networks. Network switches are capable of inspecting data packets as they are received, determining the source and destination device and forwarding it appropriately. By delivering each message only to the original intended device, a network switch conserves network bandwidth and offers a generally better performance for simultaneous point-to-point communications.

The use of the results of the point-to-point benchmarks can be various. They can be used for the estimation of parameters of the analytical communication performance models, such as Hockney, LogGP. For example, the parameters of the Hockney model can be found from the execution times of two roundtrips with empty and non-empty messages. In practice, due to noises in measurements, they are found from the execution times averaged in two series of such roundtrips. MPIBlib point-to-point benchmark provides this accurate estimation.

The set of communication operations that can be benchmarked by MPIBlib is open for extensions. The definition of operation-independent benchmark functions includes a data structure argument that includes the function pointer referencing to an MPI collective operation. MPIBlib provides a choice of different implementations of MPI collective operations (for example, linear and binomial MPI_Scatter/MPI_Gather). Any of those functions as well as user-defined versions of MPI collective operations can be passed as an argument to the benchmarking subroutines.

Three timing methods that are universally applicable to all MPI communication operations are provided; these are global, maximum, and root timings. MPIBlib provides API to all timing methods described in Section 2. The user is responsible for building the MPI communicator and mapping the processes to processors. To use benchmarks on SMP/multicore processors, an accurate MPI_Wtime implementation is required, as intra-processor communications may take very short time.

MPIBlib provides both operation-specific and implementation-specific methods of measurement. One example is a method of measuring the linear and binomial scatter, which is based on the method of measuring broadcast proposed in [7].

4 Application

The results of benchmarking the collective operations can be used for evaluation of their different implementations, for building of the communication performance models, and for optimization of collective operations.

With help of MPIBlib, we managed to observe the escalations of the execution time of linear scatter/gather on the clusters based on Ethernet switch [10]. It was possible due to the isolation of collective operations and the use of the maximum timing method for scatter and the root timing method for gather.

The library was also integrated into the software tool that automates the estimation of parameters of an advanced heterogeneous communication performance model [8]. The software tool calls the MPIBlib functions for estimation of the execution time of the $i \xleftrightarrow[0]{0} j$ and $i \xleftrightarrow[0]{M} j$ roundtrips, scatter and gather communications. We used this tool on a 16-node heterogeneous cluster with a single switch, with parallel point-to-point benchmarking and the root timing of collective operations, which proved efficient and quite accurate on heterogeneous clusters with a single switch. To estimate the parameters of the heterogeneous communication performance model, we carried out additional (neither point-to-point, nor scatter/gather) communication experiments, namely, point-to-two communications $i \xleftrightarrow[0]{M} j, k$ [11]. The function measuring the execution time of this communication experiment was implemented on the top of the MPIBlib library. In this function, the communication experiments between non-overlapping triplets of processors were performed in parallel on the cluster.

The MPIBlib benchmarking library can also be used to tune MPI communications either upon installation of an application (or a programming system) or at runtime. For example, the results of the scatter and gather benchmarks carried out upon installation of HeteroMPI are used for optimization of collective operations [10].

The following fragment shows an example of the use of MPIBlib for finding the fastest scatter implementation. In the beginning of the program, MPIB_measure_scatter_root function is used to find estimates of the execution time of different scatter implementations for different message sizes. Then these results are used in the optimized scatter, Opt_Scatter, in order to pick the fastest implementation for each particular message size. Opt_Scatter calculates the message size in bytes, compares the estimated execution times of all implementations for this message size and invokes the fastest implementation.

```
//initialization, in the beginning of main()
for (i=0; i<N; i++)
  MPIB_measure_scatter_root(comm, algs[i], n, M,
    min_reps, max_reps, alpha, T[i], &reps, &a);

//globals
MPIB_Scatter* impls[N];//N scatter implementations
int M[n];//n message sizes
double T[N][n];//estimated times for each impl/msg

//optimized scatter, to be used instead of MPI_Scatter
int Opt_Scatter(list of MPI_Scatter arguments){
  //calculate message size m
  //find i such that M[i]<=m and M[i+1]>m
  //find j such that T[j][i]=min(T[0..N-1][i])
  return impls[j](list of MPI_Scatter arguments);
}
```

5 Experiments

In addition to the library, the MPIBlib suite provides a standalone application for benchmarking point-to-point and collective MPI communications, and a set of **gnuplot** scripts for visualization of the results of measurements. We performed experiments with point-to-point, scatter and gather benchmarks on homogeneous and heterogeneous clusters with different MPI implementations. In this paper, we present the results for a heterogeneous 16-node cluster: 11 x Xeon 2.8/3.4/3.6, 2 x P4 3.2/3.4, 1 x Celeron 2.9, 2 x AMD Opteron 1.8, Gigabit Ethernet, LAM 7.1.3. They demonstrate the effects of pipelining (Section 2) and the importance of benchmarking collective operations for different message sizes (in [10], we reported on the escalations of the execution time of gather caused by the use of TCP/IP layer in the communication stack with switched networks, see Fig. 1).

Table 1. The execution time of scatter and gather benchmarks with different timing methods on 16 node heterogeneous cluster

Timing method	Scatter, 0..100KB, 1KB stride, 1 rep (sec)	Gather, 0..100KB, 1KB stride, 1 rep (sec)
Global	28.7	44.7
Maximum	0.8	15.6
Root	0.8	15.7

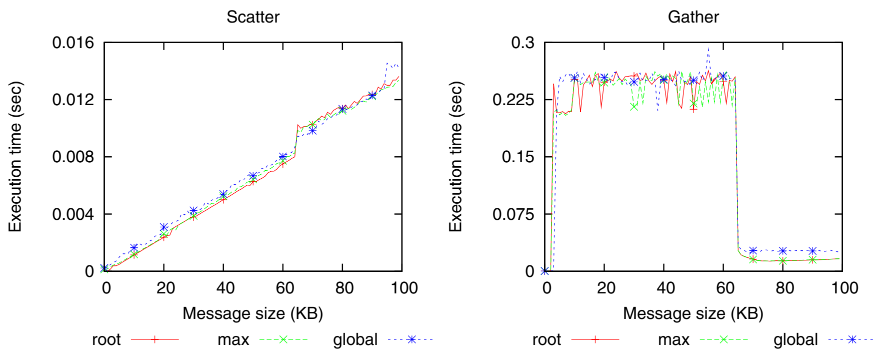

Fig. 2. Comparison of different timing methods for native (linear) LAM scatter and gather on 16 node heterogeneous cluster

We compared the results of sequential and parallel point-to-point benchmarks. In the sequential mode, while two processes are communicating, a barrier blocks all other processes. The experiment included 100 repetitions of 4KB ping-pong and took 3.5 sec. In the case of parallel roundtrips, the benchmarking procedure took significantly less time, 0.5 sec, with the same accuracy of estimation, which was possible due to the nature of the experimental network.

In the next experiment, we use MPIBlib to compare the cost and accuracy of different methods of the measurement of native MPI scatter and gather operations on the target platform. Table 1 shows the overall execution time of the benchmarks that use different timing methods and consist of one collective communication for each message size from 0 to 100 KB, with 1 KB stride. One can see that the global-time approach is very costly. The maximum and root methods are as accurate as that with global-time (see Fig. 2) but much more efficient. The difference between overall scatter and gather execution times is caused by escalations of the execution time of gather for messages of middle sizes.

In summary, the experimental results demonstrate that the use of MPIBlib can significantly speed up the estimation of the execution time of MPI communication operations without the loss of its accuracy.

6 Conclusion

In the paper, we have analyzed the commonly used MPI benchmarking suites and the methods of measurement of communication execution time. We have presented MPIBlib, the new MPI benchmarking library, which provides various operation-independent and operation-specific methods of measurement. MPIBlib is aimed at accurate and efficient runtime benchmarking of MPI communication operations in parallel applications. We have also presented an experimental demonstration showing that the use of MPIBlib can significantly speed up the benchmarking of MPI communication operations, not compromising the accuracy of the estimation. The library is freely available at `http://hcl.ucd.ie/project/MPIBlib`.

Acknowledgments. This work is supported by the Science Foundation Ireland and in part by the IBM Dublin CAS.

References

1. Gropp, W., Lusk, E.: Reproducible Measurements of MPI Performance Characteristics. In: Margalef, T., Dongarra, J., Luque, E. (eds.) PVM/MPI 1999. LNCS, vol. 1697, pp. 11–18. Springer, Heidelberg (1999)
2. Turner, D., Oline, A., Chen, X., Benjegerdes, T.: Integrating New Capabilities into NetPIPE. In: Dongarra, J., Laforenza, D., Orlando, S. (eds.) EuroPVM/MPI 2003. LNCS, vol. 2840, pp. 37–44. Springer, Heidelberg (2003)
3. Intel MPI Benchmarks. User Guide and Methodology Description (2007)
4. Grove, D., Coddington, P.: Precise MPI performance measurement using MPIBench. In: Proceedings of HPC Asia (September 2001)
5. Worsch, T., Reussner, R., Augustin, W.: On Benchmarking Collective MPI Operations. In: Kranzlmüller, D., Kacsuk, P., Dongarra, J., Volkert, J. (eds.) PVM/MPI 2002. LNCS, vol. 2474, pp. 271–279. Springer, Heidelberg (2002)
6. Bernaschi, M., Iannello, G.: Collective communication operations: experimental results vs. theory. Concurrency: Practice and Experience 10(5), 359–386 (1998)
7. Supinski, B., de Karonis, N.: Accurately measuring MPI broadcasts in a computational grid. In: The 8th International Symposium on High Performance Distributed Computing, pp. 29–37 (1999)
8. Lastovetsky, A., Mkwawa, I., O'Flynn, M.: An Accurate Communication Model of a Heterogeneous Cluster Based on a Switch-Enabled Ethernet Network. In: Proceedings of ICPADS 2006, Minneapolis, MN, pp. 15–20 (2006)
9. GNU Scientific Library (2007), `http://www.gnu.org/software/gsl/manual/`
10. Lastovetsky, A., O'Flynn, M., Rychkov, V.: Optimization of Collective Communications in HeteroMPI. In: Cappello, F., Herault, T., Dongarra, J. (eds.) PVM/MPI 2007. LNCS, vol. 4757, pp. 135–143. Springer, Heidelberg (2007)
11. Lastovetsky, A., Rychkov, V.: Building the Communication Performance Model of Heterogeneous Clusters Based on a Switched Network. In: Proceedings of the 2007 IEEE International Conference on Cluster Computing (Cluster 2007), pp. 568–575. IEEE Computer Society, Los Alamitos (2007)