MPIBlib: MPI Benchmark library

Version 1.1.0 (Revision 226)

Generated by Doxygen 1.7.1

Sun Sep 12 2010 11:11:49

Contents

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.

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. We design a new MPI benchmarking suite called MPIBlib [1] 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.

1.1 Authors

Alexey Lastovetsky, Vladimir Rychkov, Maureen O'Flynn, Kiril Dichev

Heterogeneous Computing Laboratory

School of Computer Science and Informatics, University College Dublin

Belfield, Dublin 4, Ireland

http://hcl.ucd.ie

{alexey.lastovetsky, vladimir.rychkov}@ucd.ie

References

[1] A. Lastovetsky, V. Rychkov, and M. OFlynn. MPIBlib: Benchmarking MPI communications for parallel computing on homogeneous and heterogeneous clusters. In A. Lastovetsky, T. Kechadi, and J. Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 15th European PVM/MPI User's Group Meeting*, volume 5205, pages 227–238, Dublin, Ireland, September 7-10 2008. Springer-Verlag Berlin Heidelberg. 1, 12

[2] B.R. de Supinski and N.T. Karonis. Accurately measuring MPI broadcasts in a computational grid. In *HPDC'99: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, pages 29–37, Washington, DC, USA, 1999. IEEE Computer Society. 9

[3] J.L. Traff. Hierarchical gather/scatter algorithms with graceful degradation. In *Proceedings of IPDPS'04*, pages 80–89, 2004. 23, 37, 38

[4] T. Worsch, R Reussner, and W. Augustin. On benchmarking collective MPI operations. In *Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 271–279, London, UK, 2002. Springer-Verlag. 14

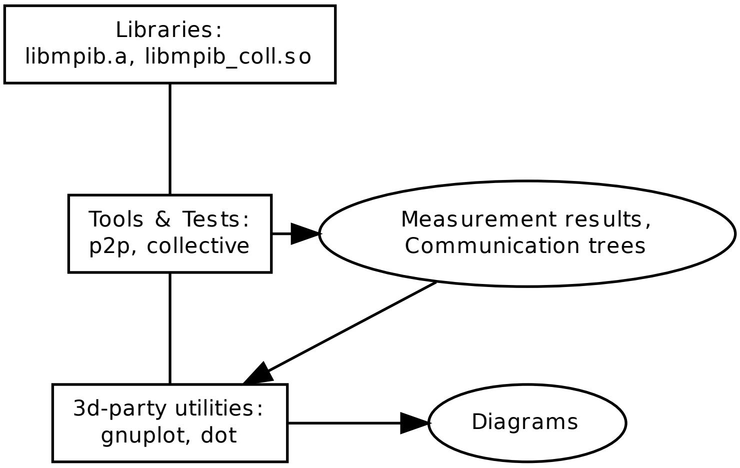
2 Installation

```
Installation
============

Required software:
1. any C/C++ and MPI (MPICH-1 does not support shared libraries)
2. GSL (GNU Scientific Library, version 1.11)
3. Boost (The Boost C++ libraries: Graph, version 1.36) -
   optional (for tree-based collectives)
4. Gnuplot (An Interactive Plotting Program) -
   optional (for performance diagrams)
5. Graphviz (Graph Visualization Software: dot) -
   optional (for tree visualization)
6. Doxygen (Source code documentation generator tool) and any TeX -
   optional (for reference manual)

GSL
If GSL is installed in a non-default directory
$ export LD_LIBRARY_PATH=DIR/lib:$LD_LIBRARY_PATH

Boost
1. Boost should be configured with at least the Graph library
   (default: all)
$ ./configure --prefix=DIR --with-libraries=graph
2. Default installation:
 - DIR/include/boost_version/boost
 - DIR/lib/libboost_library_versions.*
Create symbolic links:
$ cd DIR/include; ln -s boost_version/boost
$ cd DIR/lib; ln -s libboost_[library]_[version].[a/so] libboost_[library].[a/so]

$ export LD_LIBRARY_PATH=DIR/lib:$LD_LIBRARY_PATH

For developers
--------------

Required software:
1. Subversion
2. GNU autotools
3. Doxygen
4. Graphviz

$ svn co http://hcl.ucd.ie/repos/CPM/trunk/MPIBlib
$ cd MPIBlib
$ svn log -v > ChangeLog
$ autoreconf --install --force
$ mkdir build
$ cd build
$ ../configure --enable-debug
$ make all install check
```

```
To create a package:
$ make dist

For users
---------

Download and untar the latest package from http://hcl.ucd.ie/project/mpiblib

$ mkdir build
$ cd build
$ ../configure
$ make all install check

Configuration
-------------

Packages:
  --with-gsl-dir=DIR      GNU Scientific Library directory
  --with-boost-dir=DIR    The Boost C++ libraries directory

Check configure options:
$ ../configure -h
```

3 The software design

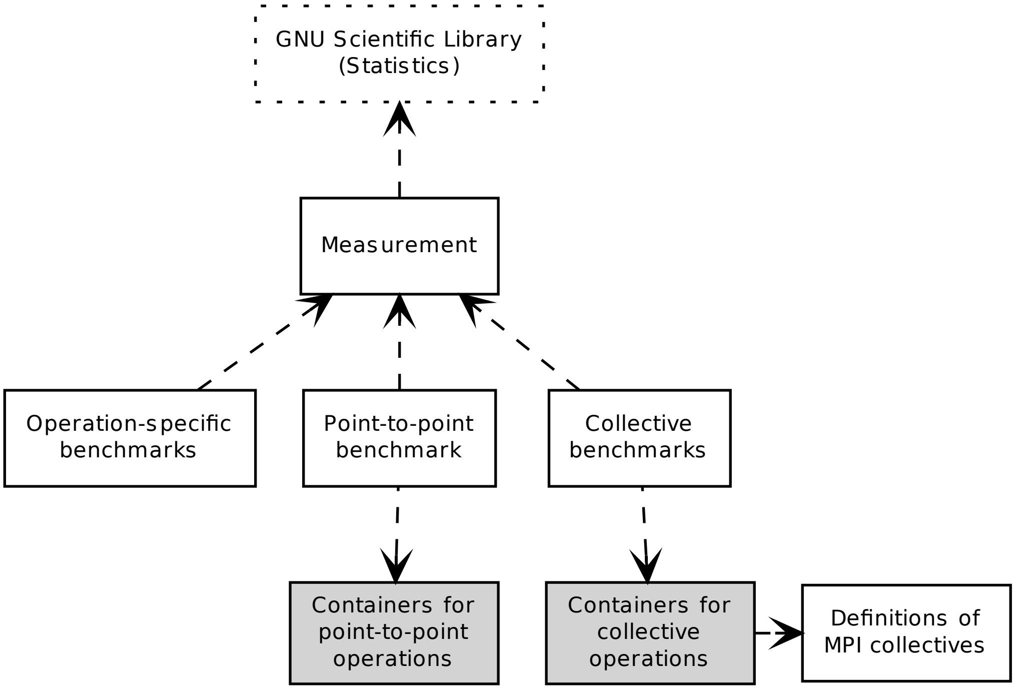
MPIBlib is implemented in C/C++ on top of MPI. The package consists of libraries, tools and tests. The libraries implement point-to-point and collective benchmarks and different algorithms of collective communication operations. The tools verify collectives and perform benchmarks. Their output includes the results of measurements and communication trees (for tree-based algorithms of collective operations). The results of measurements can be visualized by the gnuplot utility; MPIBlib provides the basic gnuplot scripts. The communication trees built in the tree-based collective algorithms can be visualized by the dot utility (a part of Graphviz).



3.1 Benchmarks library

A static library `libmpib.a` implements point-to-point and collective benchmarks. Main library modules (the modules marked grey can be extended, providing the benchmarking of user-defined point-to-point and collective operations):

- Measurement: base data structures and functions (depends on the GNU Scientific Library)
- Operation-specific benchmarks (now includes only benchmark for bcast)
- Point-to-point benchmarks (seqential/parallel point-to-point benchmark)
- Containers for point-to-point communication operations (data structures describing point-to-point communications to be measured, which are used as an argument of the point-to-point benchmark function)
- Collective benchmarks (collective benchmarks based on root, max and global timing methods)
- Containers for collective communication operations (data structures describing collective communications to be measured, which are used as an argument of collective benchmark functions)
- Definitions of MPI collective operations (typedefs of pointers to function)


GNU Scientific Library
(Statistics)
Measurement
Operation-specific benchmarks
Point-to-point benchmark
Collective benchmarks
Containers for point-to-point operations
Containers for collective operations
Definitions of MPI collectives

3.1.1 Tools

- tools/p2p.c - performs p2p benchmark between all pairs of processors in the communicator with given accuracy and efficiency.
- tools/collective.c - performs a universal or operation-specific collective benchmark with given accuracy and efficiency.
- tools/generate_factors.c - generates multiplying factors applied to message size for benchmarking scatterv/gatherv

Command-line arguments are described in Options for executables.

3.2 Collectives library

A shared library `libmpib_coll.so` implements different algorithms of collectives

- Basic algorithms of MPI collective operations (mostly linear)
- Tree-based implementations of MPI collective communication operations (depends on the Boost C++ libraries)

The command-line arguments:

- **verbose** verbose mode (default: no)
- **sgv** scatterv/gatherv mode
  - *0* propagated on duplicated communicator (default)
  - *1* broadcasted counts
  - *2* propagated with tags: not safe
  - *3* everywhere-defined counts: not standard

3.2.1 Tools

- tools/collective_test.c - verifies implementations of collective communication operations

An example showing how collective operations implemented in the library can be used in applications instead of their native counterparts tests/substitute.c.

4 Module Documentation

4.1 Measurement: base data structures and functions

Classes

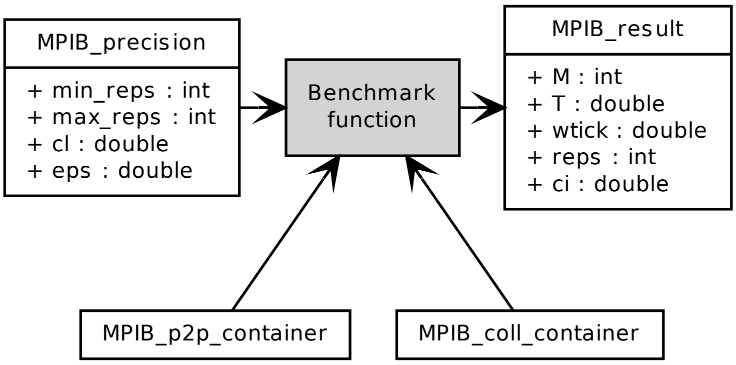
- struct MPIB_result
- struct MPIB_msgset
- struct MPIB_precision

Functions

- void MPIB_Comm (MPI_Comm comm, MPI_Comm ∗newcomm)
- double MPIB_diff (MPIB_result result, MPIB_result results[2])
- void MPIB_max_wtick (MPI_Comm comm, double ∗wtick)
- double MPIB_ci (double cl, int reps, double ∗T)

4.1.1 Detailed Description

This module provides base data structures and functions for measurement:



Benchmark input:

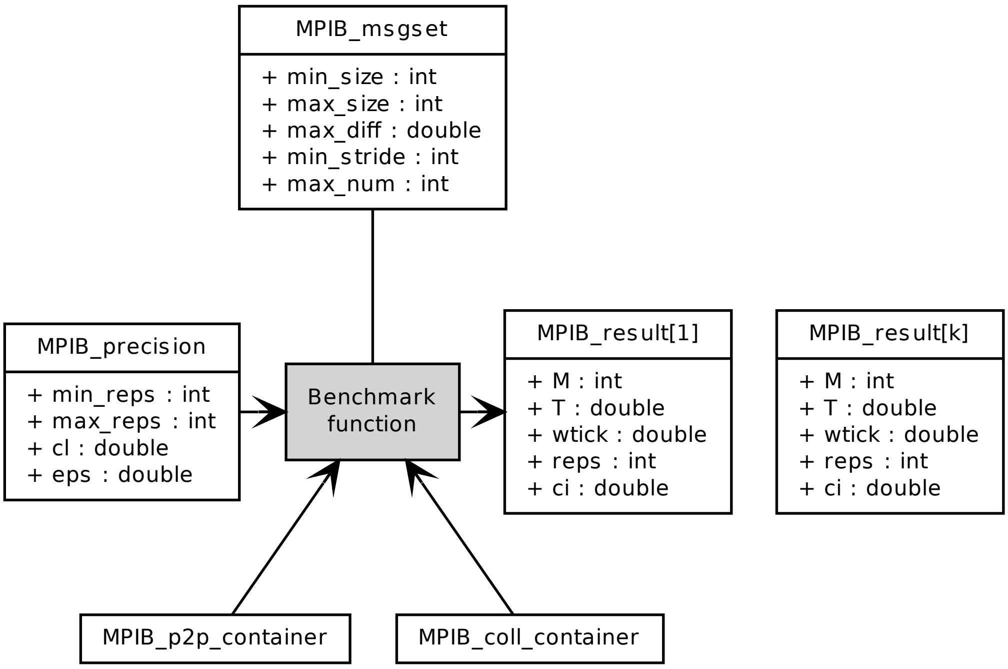
- The MPIB_precision data structure defines statistical precision of measurement. Statistical analysis is implemented with help of the GNU Scientific Library.

- The MPIB_p2p_container and MPIB_coll_container data structures encapsulate the communication operation to be measured.

Benchmark output:

- The MPIB_result data structure represents result of measurement and its reliability.

Most benchmark functions have a counterpart for measurement of execution time with a set of message sizes:



The counterpart has an extra input argument:

- The MPIB_msgset data structure defines the message sizes for which the measurements are to be performed. Message sizes are selected either regularly or adaptively.

The output of the counterpart is an array of MPIB_result.

4.1.2 Function Documentation

4.1.2.1 void MPIB_Comm (MPI_Comm *comm*, MPI_Comm ∗ *newcomm*)

Creates a copy of communicator that includes one process per processor.

4.1.2.2 double MPIB_diff (MPIB_result *result*, MPIB_result *results[2]*)

Compares the result of measurement with the linear model based on the results of two previous measurements. Required for adaptive selection of message sizes.

4.1.2.3 void MPIB_max_wtick (MPI_Comm *comm*, double ∗ *wtick*)

Returns a resolution of MPI_Wtime, maximum in the communicator. Result can be used to check the execution time measured at several processors (MPIB_measure_max, MPIB_measure_global): $T_{coll} < wtick_{max}$.

Parameters

comm MPI communicator

wtick a maximum resolution

4.1.2.4 double MPIB_ci (double *cl,* int *reps,* double ∗ *T*)

Returns a confidence interval that contains the average execution time with a certain probability: $Pr(|\bar{T} - \mu| < ci) = cl$.

Note

If communication operations in a series are isolated from each other, we can assume that the execution times form an independent sample from a normally distributed population, and use t distribution to estimate confidence interval.

Parameters

cl confidence level

reps number of measurements (should be > 1)

T array of reps measurement results

Returns

confidence interval

4.2 Operation-specific benchmarks

Functions

- void MPIB_bcast_timer_init (MPI_Comm comm, int parallel, MPIB_precision precision)
- void MPIB_measure_bcast (MPIB_Bcast bcast, MPI_Comm comm, int root, int M, int max_reps, MPIB_result ∗result)

4.2.1 Detailed Description

This module provides operation-specific benchmarks.

4.2.2 Function Documentation

4.2.2.1 void MPIB_bcast_timer_init (MPI_Comm *comm,* int *parallel,* MPIB_precision *precision*)

Performs the p2p benchmark with empty message with given precision and sets up an internal global variable, which is used by MPIB_measure_bcast. Called by MPIB_measure_bcast. Can be called directly before measurements.

Attention

As the global variable is an array, the memory should be freed by

```
MPIB_bcast_timer_init(MPI_COMM_NULL, (MPIB_precision){0, 0, 0})
```

Parameters

comm communicator

parallel several non-overlapped point-to-point communications at the same time if non-zero

precision measurement precision

4.2.2.2 void MPIB_measure_bcast (MPIB_Bcast *bcast,* MPI_Comm *comm,* int *root,* int *M,* int *max_reps,* MPIB_result ∗ *result*)

Measures the execution time of bcast between root and the rest of processes. Based on [2]. Reuses already obtained empty-roundtrip times if the previous initialization was performed on the same communicator. Otherwise, initializes the bcast timer by calling MPIB_bcast_timer_init.

In the loop over processes, except root:

In the loop over repetitions:

- bcast at all processes except root and current process in the loop
- bcast and empty recv at root and measures the execution time
- bcast and empty send at current process in the loop

Having substructed the half of empty-roundtrip times, finds maximum.

Broadcasts the result.

Note

No double barriers as no need in synchronization.
No precision as we cannot interrupt the process of measurement by broadcasting the error value after each repetition.

Parameters

bcast bcast implementation

comm communicator

root root process

M message size

max_reps maximum number of repetitions

result measurement result

4.3 Point-to-point benchmarks

Classes

- struct MPIB_p2p_container

Defines

- #define MPIB_C2(n) (n) ∗ ((n) - 1) / 2
- #define MPIB_IJ2INDEX(n, i, j) (2 ∗ (n) - ((i) < (j) ? (i) : (j)) - 1) ∗ (((i) < (j) ? (i) : (j))) / 2 + (((i) < (j) ? (j) : (i))) - (((i) < (j) ? (i) : (j))) - 1

Functions

- void MPIB_measure_p2p (MPIB_p2p_container ∗container, MPI_Comm comm, int measure, int mirror, int M, MPIB_precision precision, MPIB_result ∗result)
- void MPIB_measure_p2p_msgset (MPIB_p2p_container ∗container, MPI_Comm comm, int measure, int mirror, MPIB_msgset msgset, MPIB_precision precision, int ∗count, MPIB_result ∗∗results)
- void MPIB_measure_allp2p (MPIB_p2p_container ∗container, MPI_Comm comm, int parallel, int M, MPIB_precision precision, MPIB_result ∗results)

4.3.1 Detailed Description

This module provides the point-to-point benchmarks.

4.3.2 Define Documentation

4.3.2.1 #define MPIB_C2(*n*) (n) ∗ ((n) - 1) / 2

$$C_n^2$$

4.3.2.2 #define MPIB_IJ2INDEX(*n*, *i*, *j*) (2 ∗ (n) - ((i) < (j) ? (i) : (j)) - 1) ∗ (((i) < (j) ? (i) : (j))) / 2 + (((i) < (j) ? (j) : (i))) - (((i) < (j) ? (i) : (j))) - 1

For a symmetric square matrix stored in the array of C_n^2 elements, returns the index of the (i, j) element, $i \neq j < n$: $\frac{(n-1)+(n-I)}{2}I + (J - I - 1)$, $I = min(i, j)$, $J = max(i, j)$

4.3.3 Function Documentation

4.3.3.1 void MPIB_measure_p2p (MPIB_p2p_container ∗ *container*, MPI_Comm *comm*, int *measure*, int *mirror*, int *M*, MPIB_precision *precision*, MPIB_result ∗ *result*)

Point-to-point benchmark. Estimates the execution time of the point-to-point communications between a pair of processors in the MPI communicator. Performs series of communication experiments to obtain reliable results.

Parameters

container communication operation container

comm communicator, number of nodes should be ≥ 2

measure measure processor

mirror mirror processor

M message size

precision measurement precision

result measurement result (significant only at the measure processor)

4.3.3.2 void MPIB_measure_p2p_msgset (MPIB_p2p_container $*$ *container,* MPI_Comm *comm,* int *measure,* int *mirror,* MPIB_msgset *msgset,* MPIB_precision *precision,* int $*$ *count,* MPIB_result $**$ *results*)

Point-to-point benchmark. Estimates the execution time of the point-to-point communication between a pair of processors in the MPI communicator for different message sizes. Performs series of communication experiments to obtain reliable results.

Parameters

container communication operation container

comm communicator, number of nodes should be ≥ 2

measure measure processor

mirror mirror processor

msgset message sizes

precision measurement precision

count the number of measurements performed (significant only at the measure processor)

results array of measurement results (significant only at the measure processor, allocated by this function and must be deallocated by user)

4.3.3.3 void MPIB_measure_allp2p (MPIB_p2p_container $*$ *container,* MPI_Comm *comm,* int *parallel,* int *M,* MPIB_precision *precision,* MPIB_result $*$ *results*)

Point-to-point benchmark. Estimates the execution time of the point-to-point communications between all pairs of processors in the MPI communicator. Performs series of communication experiments to obtain reliable results.

Parameters

container communication operation container

comm communicator, number of nodes should be ≥ 2

parallel several non-overlapped point-to-point communications at the same time if non-zero

M message size

precision measurement precision

results array of C_n^2 measurement results

4.4 Containers for point-to-point communication operations

Functions

- void MPIB_p2p_container_free (MPIB_p2p_container $*$container)
- MPIB_p2p_container $*$ MPIB_Send_Recv_container_alloc ()

4.4.1 Detailed Description

Data structures describing point-to-point communications to be measured, which are used as an argument of the point-to-point benchmark function.

4.4.2 Function Documentation

4.4.2.1 void MPIB_p2p_container_free (MPIB_p2p_container ∗ *container*)

Frees point-to-point container

4.4.2.2 MPIB_p2p_container∗ MPIB_Send_Recv_container_alloc ( )

Allocates MPI_Send-MPI_Recv container

4.5 Collective benchmarks

Classes

- struct MPIB_coll_container

Typedefs

- typedef int(∗ MPIB_measure_coll)(MPIB_coll_container ∗container, MPI_Comm comm, int root, int M, MPIB_precision precision, MPIB_result ∗result)
- typedef void(∗ MPIB_measure_coll_msgset)(MPIB_coll_container ∗container, MPI_Comm comm, int root, MPIB_msgset msgset, MPIB_precision precision, int ∗count, MPIB_result ∗∗results)

Functions

- int MPIB_measure_max (MPIB_coll_container ∗container, MPI_Comm comm, int root, int M, MPIB_precision precision, MPIB_result ∗result)
- void MPIB_root_timer_init (MPI_Comm comm, int reps)
- int MPIB_measure_root (MPIB_coll_container ∗container, MPI_Comm comm, int root, int M, MPIB_precision precision, MPIB_result ∗result)
- void MPIB_global_timer_init (MPI_Comm comm, int parallel, int reps)
- int MPIB_measure_global (MPIB_coll_container ∗container, MPI_Comm comm, int root, int M, MPIB_precision precision, MPIB_result ∗result)

4.5.1 Detailed Description

This module provides collective benchmarks based on root, max and global timing methods (see [1]).

4.5.2 Typedef Documentation

4.5.2.1 typedef int(∗ MPIB_measure_coll)(MPIB_coll_container ∗container, MPI_Comm comm, int root, int M, MPIB_precision precision, MPIB_result ∗result)

Collective benchmark. Measures the execution time of collective operation for a given message size.

Parameters

container communication operaion container

comm communicator

root root process

M message size

precision measurement precision

result measurement result

4.5.2.2 typedef void($*$ MPIB_measure_coll_msgset)(MPIB_coll_container $*$container, MPI_Comm comm, int root, MPIB_msgset msgset, MPIB_precision precision, int $*$count, MPIB_result $**$results)

Collective benchmark for multiple message sizes. Measures the execution time of collective operation for different message sizes.

Parameters

container communication operaion container

comm communicator

root root process

msgset message sizes

precision measurement precision

count the number of measurements performed (significant only at the root processor)

results array of measurement results (significant only at the root processor, allocated by this function and must be deallocated by user)

4.5.3 Function Documentation

4.5.3.1 int MPIB_measure_max (MPIB_coll_container $*$ *container,* MPI_Comm *comm,* int *root,* int *M,* MPIB_precision *precision,* MPIB_result $*$ *result*)

Measures the execution time of collective operation at all processes and finds a maximum. In the loop over repetitions:

- Synchronizes the processes by double barrier.
- Measures the execution time of collective operation at all processes.
- Finds maximum by allreduce.
- Performs statistical analysis.

4.5.3.2 void MPIB_root_timer_init (MPI_Comm *comm,* int *reps*)

Measures the average execution time of barrier at all processes in the communicator and sets up an internal global variable, which is used by MPIB_measure_root. Called by MPIB_measure_root. Can be called directly before measurements.

Parameters

comm communicator

reps number of repetitions (not precision - we suppose no pipeline effect with barrier)

4.5.3.3 int MPIB_measure_root (MPIB_coll_container ∗ *container,* MPI_Comm *comm,* int *root,* int *M,* MPIB_precision *precision,* MPIB_result ∗ *result*)

Measures the execution time of collective operation at the root process. Reuses already obtained barrier time if the previous initialization was performed over the same communicator. Otherwise, initializes the root timer by calling MPIB_root_timer_init.

In the loop over repetitions:

- Synchronizes the processes by double barrier.
- Measures the execution time of collective operation and barrier confirmation at the root process.
- Subtracts the execution time of barrier.
- Performs statistical analysis at root.

Broadcasts the result.

4.5.3.4 void MPIB_global_timer_init (MPI_Comm *comm,* int *parallel,* int *reps*)

Measures the offsets between local clocks of of all processes in the communicator and sets up an internal global variable, which is used by MPIB_measure_global. If MPI_WTIME_IS_GLOBAL (MPI global timer) is defined, the offsets are set to zero. Otherwise, the offsets are measured. Called by MPIB_measure_global. Can be called directly before measurements.

Attention

As the global variable is an array, the memory should be freed by

```
MPIB_global_timer_init(MPI_COMM_NULL, 0)
```

Parameters

comm communicator

parallel several non-overlapped point-to-point communications at the same time if non-zero

reps number of repetitions (not precision because series of ping-pongs are required - we cannot interrupt them by sending/receiving the result of the statistical estimation)

4.5.3.5 int MPIB_measure_global (MPIB_coll_container ∗ *container,* MPI_Comm *comm,* int *root,* int *M,* MPIB_precision *precision,* MPIB_result ∗ *result*)

Measures the execution time of collective operation between processes using global time. Based on [4]. Reuses already obtained offsets between local clocks if the previous initialization was performed on the same communicator. Otherwise, initializes the global timer by calling MPIB_global_timer_init.

In the loop over repetitions:

- Synchronizes the processes by double barrier.
- Measures the moment of start at the root and the moment of finish at the rest of processes.
- Having substructed the offset, finds maximum by reducing to the root.
- Performs statistical analysis at root.

Broadcasts the result.

4.6 Containers for collective communication operations

Functions

- void MPIB_coll_container_free (MPIB_coll_container ∗container)
- MPIB_coll_container ∗ MPIB_Scatter_container_alloc (MPIB_Scatter scatter)
- MPIB_coll_container ∗ MPIB_Gather_container_alloc (MPIB_Gather gather)
- MPIB_coll_container ∗ MPIB_Bcast_container_alloc (MPIB_Bcast bcast)
- MPIB_coll_container ∗ MPIB_Reduce_container_alloc (MPIB_Reduce reduce)
- MPIB_coll_container ∗ MPIB_Comm_dup_free_container_alloc ()
- MPIB_coll_container ∗ MPIB_Scatterv_container_alloc (MPIB_Scatterv scatterv, const double ∗factors)
- MPIB_coll_container ∗ MPIB_Gatherv_container_alloc (MPIB_Gatherv gatherv, const double ∗factors)

4.6.1 Function Documentation

4.6.1.1 void MPIB_coll_container_free (MPIB_coll_container ∗ *container*)

Frees collective container

4.6.1.2 MPIB_coll_container∗ MPIB_Scatter_container_alloc (MPIB_Scatter *scatter*)

Allocates Scatter container

4.6.1.3 MPIB_coll_container∗ MPIB_Gather_container_alloc (MPIB_Gather *gather*)

Allocates Gather container

4.6.1.4 MPIB_coll_container∗ MPIB_Bcast_container_alloc (MPIB_Bcast *bcast*)

Allocates Bcast container

4.6.1.5 MPIB_coll_container∗ MPIB_Reduce_container_alloc (MPIB_Reduce *reduce*)

Allocates Reduce container

4.6.1.6 MPIB_coll_container∗ MPIB_Comm_dup_free_container_alloc ( )

Allocates MPI_Comm_dup-MPI_Comm_free container

4.6.1.7 MPIB_coll_container∗ MPIB_Scatterv_container_alloc (MPIB_Scatterv *scatterv,* const double ∗ *factors*)

Allocates Scatterv container

4.6.1.8 MPIB_coll_container∗ MPIB_Gatherv_container_alloc (MPIB_Gatherv *gatherv,* const double ∗ *factors*)

Allocates Gatherv container

4.7 Definitions of MPI collective operations

Typedefs

- typedef int(∗ MPIB_Scatter)(void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- typedef int(∗ MPIB_Gather)(void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- typedef int(∗ MPIB_Scatterv)(void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- typedef int(∗ MPIB_Gatherv)(void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- typedef int(∗ MPIB_Bcast)(void ∗buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
- typedef int(∗ MPIB_Reduce)(void ∗sendbuf, void ∗recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

4.7.1 Detailed Description

This module provides the definitions of types of MPI collective operations.

4.7.2 Typedef Documentation

4.7.2.1 typedef int(∗ MPIB_Scatter)(void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

Scatter typedef

4.7.2.2 typedef int(∗ MPIB_Gather)(void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

Gather typedef

4.7.2.3 typedef int(∗ MPIB_Scatterv)(void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

Scatterv typedef

4.7.2.4 typedef int(∗ MPIB_Gatherv)(void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)

Gatherv typedef

4.7.2.5 typedef int(∗ MPIB_Bcast)(void ∗buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

Bcast typedef

4.7.2.6 typedef int(∗ MPIB_Reduce)(void ∗sendbuf, void ∗recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

Reduce typedef

4.8 Options for executables

4.8.1 Detailed Description

Since getopt cannot be reused, this module provides an interface for getopt, which can be used as follows:

```
X x;
Y y;
MPIB_getopt_x_default(&x);
MPIB_getopt_y_default(&y);
if (root == 0)
{
    char options[256] = "";
    strcat(options, MPIB_getopt_x_options());
    strcat(options, MPIB_getopt_y_options());
    while ((c = getopt(argc, argv, options)) != -1)
    {
        MPIB_getopt_x_optarg(c, &x);
        MPIB_getopt_y_optarg(c, &y);
    }
}
MPIB_getopt_x_bcast(&x, 0, MPI_COMM_WORLD);
MPIB_getopt_y_bcast(&y, 0, MPI_COMM_WORLD);
```

Options are divided into sets, which are managed by the following functions:

- **MPIB_getopt_X_default** - fills the parameters by default values
- **MPIB_getopt_X_options** - returns a string of the getopt options
- **MPIB_getopt_X_optarg** - fills the parameters by the getopt argument
- **MPIB_getopt_X_bcast** - broadcasts the parameters (parallel) where X stands for a name of the set.

Typical options for executables are as follows:

- **-h** help
- **-v** verbose
- **-l** *S* collectives shared library (required: a full path or relative to LD_LIBRARY_PATH)
- **-o** *S* suboptions (comma separated options for the shared library: `subopt1`,subopt2=value2)
- **-O** *S* collective operation defined in the shared library (required: the name must contain Bcast, Scatter, etc)
- **-t** *S* timing: max, root, global (default: max)

- **-m** *I* minimum message size (default: 0)
- **-M** *I* maximum message size (default: 204800)
- **-S** *I* stride between message sizes (default: 1024)
- **-d** *D* maximum relative difference: 0 < D < 1 (default: 0.1)
- **-s** *I* minimum stride between message sizes (default: 64)
- **-n** *I* maximum number of message sizes (default: 100)
- **-p** 0/1 parallel p2p benchmarking (default: 1)
- **-r** *I* minimum number of repetitions (default: 5)
- **-R** *I* maximum number of repetitions (default: 100)
- **-c** *D* confidence level: 0 < D < 1 (default: 0.95)
- **-e** *D* relative error: 0 < D < 1 (default: 0.025)

where:

- *S* - string
- *I* - integer
- *D* - double

4.9 Basic algorithms of MPI collective operations

Functions

- int MPIB_Scatter_flat (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int MPIB_Gather_flat (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int MPIB_Scatterv_flat (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int MPIB_Gatherv_flat (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int MPIB_Scatterv_sorted_flat_asc (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int MPIB_Gatherv_sorted_flat_asc (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int MPIB_Scatterv_sorted_flat_dsc (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int MPIB_Gatherv_sorted_flat_dsc (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)

4.9.1 Detailed Description

This module provides basic, mostly flat-tree, algorithms of MPI collective operations.

4.9.2 Function Documentation

4.9.2.1 int MPIB_Scatter_flat (void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Flat-tree scatter

4.9.2.2 int MPIB_Gather_flat (void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Flat-tree gather

4.9.2.3 int MPIB_Scatterv_flat (void ∗ *sendbuf,* int ∗ *sendcounts,* int ∗ *displs,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Flat-tree scatterv

4.9.2.4 int MPIB_Gatherv_flat (void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int ∗ *recvcounts,* int ∗ *displs,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Flat-tree gatherv

4.9.2.5 int MPIB_Scatterv_sorted_flat_asc (void ∗ *sendbuf,* int ∗ *sendcounts,* int ∗ *displs,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Flat-tree scatterv with sendcounts sorted in ascending order

4.9.2.6 int MPIB_Gatherv_sorted_flat_asc (void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int ∗ *recvcounts,* int ∗ *displs,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Flat-tree gatherv with recvcounts sorted in ascending order

4.9.2.7 int MPIB_Scatterv_sorted_flat_dsc (void ∗ *sendbuf,* int ∗ *sendcounts,* int ∗ *displs,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Flat-tree scatterv with sendcounts sorted in descending order

4.9.2.8 int MPIB_Gatherv_sorted_flat_dsc (void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int ∗ *recvcounts,* int ∗ *displs,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Flat-tree gatherv with recvcounts sorted in descending order

4.10 Tree-based implementations of MPI collective communication operations

Functions

- int MPIB_Bcast_binomial (void ∗buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
- int MPIB_Reduce_binomial (void ∗sendbuf, void ∗recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
- int MPIB_Scatter_binomial (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int MPIB_Gather_binomial (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int MPIB_Scatterv_binomial (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int MPIB_Gatherv_binomial (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int MPIB_Scatterv_sorted_binomial_asc (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int MPIB_Gatherv_sorted_binomial_asc (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int MPIB_Scatterv_sorted_binomial_dsc (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int MPIB_Gatherv_sorted_binomial_dsc (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int MPIB_Scatterv_Traff (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int MPIB_Gatherv_Traff (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)

4.10.1 Detailed Description

This module provides tree algorithms of MPI collective operations. For rapid implementation, the Graph Boost C++ library is used. The sources of tree algorithms of collectives must be compiled by C++.

A tree algorithm of a collective operation `X` consists of the following components:

- **Communication tree** MPIB::comm_tree
- **Base tree algorithm of the collective operation** `MPIB_X_tree_algorithm` (described in Base tree algorithms of MPI collective communication operations)
- **Communication tree builder** `Y_builder`, a C++ class that builds a communication tree. (described in MPIB::BRSG and MPIB::SGv)

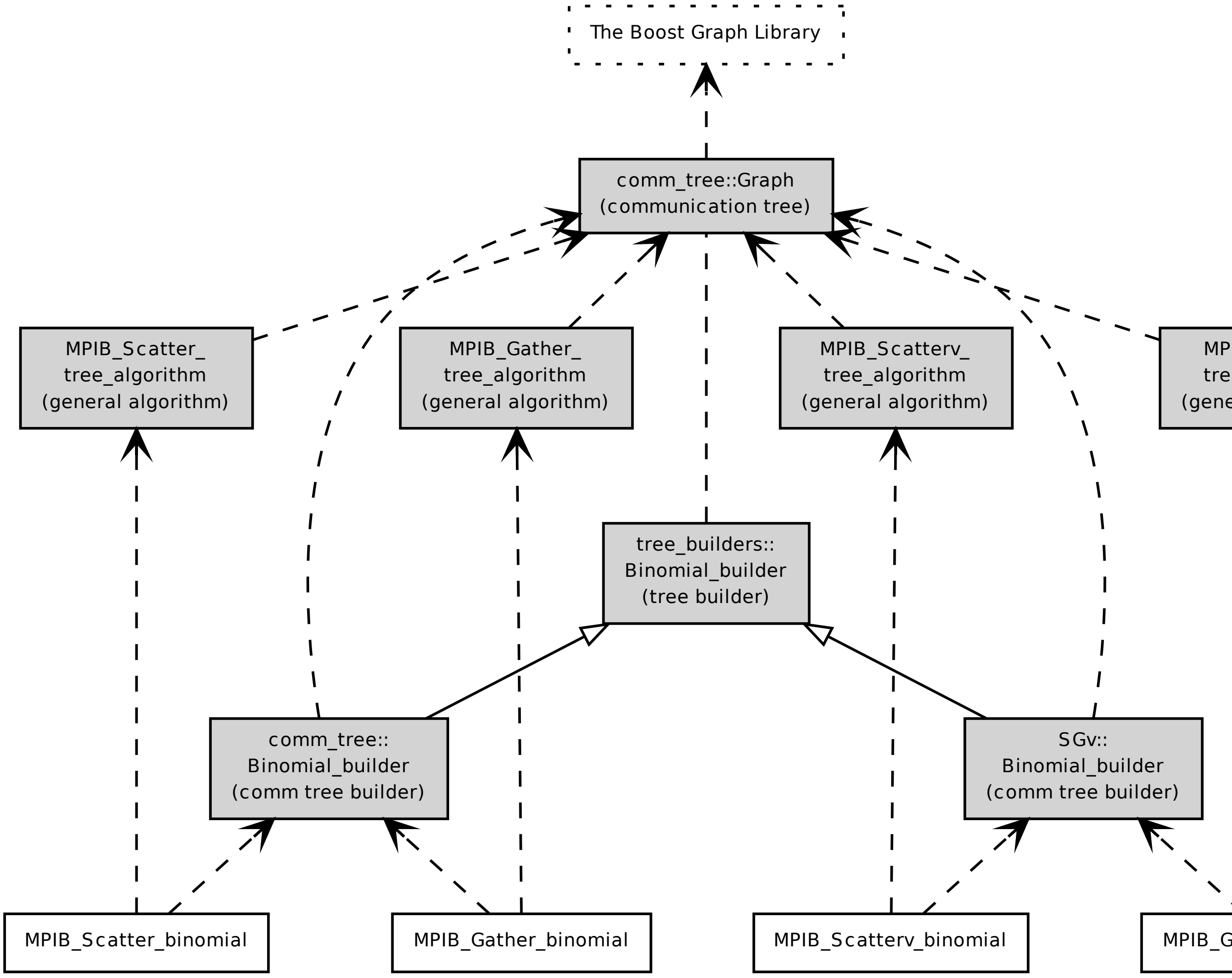
The tree-based implementation looks as follows:

```
extern "C" int MPIB_X_Y(standard args) {
    return MPIB_X_tree_algorithm(Y_builder(), standard args);
}
```

For example, MPIB_Scatter_binomial, a binomial scatter.

This approach provides flexibility:

- Tree-based algorithm of the collective operation is a basis for implementation of algorithms with communication trees of different shapes (for example, binomial and binary algorithms of scatter use the general tree-based algorithm of scatter).
- Communication tree is a universal interface between tree-based algorithms and tree builders. It can be reused for different communication operations (for example, scatter/gather and bcast/reduce use the same simple communication tree).
- Tree and communication tree builders can be reused per tree shape (for example, the same binomial communication tree builder is used in binomial scatter/gather and bcast/reduce).


The Boost Graph Library
comm_tree::Graph (communication tree)
MPIB_Scatter_ tree_algorithm (general algorithm)
MPIB_Gather_ tree_algorithm (general algorithm)
MPIB_Scatterv_ tree_algorithm (general algorithm)
MP tre (gene
tree_builders:: Binomial_builder (tree builder)
comm_tree:: Binomial_builder (comm tree builder)
SGv:: Binomial_builder (comm tree builder)
MPIB_Scatter_binomial
MPIB_Gather_binomial
MPIB_Scatterv_binomial
MPIB_G

4.10.2 Function Documentation

4.10.2.1 int MPIB_Bcast_binomial (void * *buffer,* int *count,* MPI_Datatype *datatype,* int *root,* MPI_Comm *comm*)

Binomial Bcast

4.10.2.2 int MPIB_Reduce_binomial (void * *sendbuf,* void * *recvbuf,* int *count,* MPI_Datatype *datatype,* MPI_Op *op,* int *root,* MPI_Comm *comm*)

Binomial Reduce

4.10.2.3 int MPIB_Scatter_binomial (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Binomial Scatter

4.10.2.4 int MPIB_Gather_binomial (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Binomial Gather

4.10.2.5 int MPIB_Scatterv_binomial (void * *sendbuf,* int * *sendcounts,* int * *displs,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Binomial scatterv

4.10.2.6 int MPIB_Gatherv_binomial (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int * *recvcounts,* int * *displs,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Binomial gatherv.

4.10.2.7 int MPIB_Scatterv_sorted_binomial_asc (void * *sendbuf,* int * *sendcounts,* int * *displs,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Binomial scatterv with children sorted by counts in ascending order.

4.10.2.8 int MPIB_Gatherv_sorted_binomial_asc (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int * *recvcounts,* int * *displs,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Binomial gatherv with children sorted by counts in ascending order.

4.10.2.9 int MPIB_Scatterv_sorted_binomial_dsc (void ∗ *sendbuf*, int ∗ *sendcounts*, int ∗ *displs*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Binomial scatterv with children sorted by counts in descending order.

4.10.2.10 int MPIB_Gatherv_sorted_binomial_dsc (void ∗ *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int ∗ *recvcounts*, int ∗ *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Binomial gatherv with children sorted by counts in descending order.

4.10.2.11 int MPIB_Scatterv_Traff (void ∗ *sendbuf*, int ∗ *sendcounts*, int ∗ *displs*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Traff algorithm of Scatterv. Based on [3].

4.10.2.12 int MPIB_Gatherv_Traff (void ∗ *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int ∗ *recvcounts*, int ∗ *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Traff algorithm of Gatherv. Based on [3].

4.11 Base tree algorithms of MPI collective communication operations

Functions

- template<typename Builder >
  int MPIB_Bcast_tree_algorithm (Builder builder, MPIB_child_traverse_order order, void ∗buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
- template<typename Builder >
  int MPIB_Reduce_tree_algorithm (Builder builder, MPIB_child_traverse_order order, void ∗sendbuf, void ∗recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
- template<typename Builder >
  int MPIB_Scatter_tree_algorithm (Builder builder, MPIB_child_traverse_order order, void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- template<typename Builder >
  int MPIB_Gather_tree_algorithm (Builder builder, MPIB_child_traverse_order order, void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- template<typename Builder >
  int MPIB_Scatterv_tree_algorithm (Builder builder, MPIB_child_traverse_order order, void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm _comm)
- template<typename Builder >
  int MPIB_Gatherv_tree_algorithm (Builder builder, MPIB_child_traverse_order order, void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm _comm)

4.11.1 Detailed Description

Base tree algorithm of the collective operation is a template function that, in addition to the standard arguments, has the communication tree builder and the order arguments:

```
template <typename Builder>
MPIB_X_tree_algorithm(Builder builder, bool order, args, standard args);
```

For example, MPIB_Scatter_tree_algorithm, a base tree algorithm of scatter. In base tree-based algorithm, all point-to-point communications are performed over the communication tree built by the builder in order given by the order argument.

Usually, the communication tree is built at all processors independently. If the communication tree can be built only at a designated processor, it must then be sent to other processes along with the data. The Serialization Boost C++ library is used for serialization/deserialization of the communication tree/subtrees in such tree-based algorithms:

```
#include <boost/graph/adj_list_serialize.hpp>
#include <boost/archive/binary_oarchive.hpp>
#include <boost/archive/binary_iarchive.hpp>
#include <sstream>

Graph graph;

if (rank == root) {
    ostringstream oss;
    archive::binary_oarchive ar(oss);
    ar << graph;
    int length = oss.str().length();
    MPI_Send((void*)oss.str().c_str(), length, MPI_CHAR, dest, 1, comm);
}

if (rank == dest) {
    MPI_Status status;
    MPI_Probe(root, 1, comm, &status);
    int length;
    MPI_Get_count(&status, MPI_CHAR, &length);
    buffer = (char*)malloc(sizeof(char) * length);
    MPI_Recv(buffer, length, MPI_CHAR, root, 1, comm, MPI_STATUS_IGNORE);
    istringstream iss(string(buffer, length));
    archive::binary_iarchive ar(iss);
    ar >> graph;
    free(buffer);
}
```

The internal part of the tree-based implementation includes the following auxiliaries united in namespaces in order to avoid duplicates:

- **Tree visitors** traverse communication tree, for example, in order to assemble the data buffer to send or receive:

  ```
  class Visitor {
  public:
      Visitor(args) {...}
      void preorder(Vertex vertex, Tree& tree) {...}
      void inorder(Vertex vertex, Tree& tree) {...}
      void postorder(Vertex vertex, Tree& tree) {...}
  };
  ```

  Note

  > There may be many pointer or reference arguments in the visitor's constructor because visitors are copied by value.

- **Property writers** print vertex, edge and graph properties during the output of the communication tree:

```
class Vertex_writer {
public:
    void operator()(std::ostream& out, const Vertex& v) const {
        out << "[label=\"" << ... << "\"]";
    }
};

class Edge_writer {
public:
    void operator()(std::ostream& out, const Edge& e) const {
        out << "[label=\"" << ... << "\"]";
    }
};

class Graph_writer {
public:
    void operator()(std::ostream& out) const {
        out << "graph [...]\n";
        out << "node [...]\n";
        out << "edge [...]\n";
    }
};

write_graphviz(cout, graph, Vertex_writer(), Edge_writer(), Graph_writer());
```

Note

Default writers are called when the last three arguments omitted.

4.11.2 Function Documentation

4.11.2.1 template<typename Builder > int MPIB_Bcast_tree_algorithm (Builder *builder,* MPIB_child_traverse_order *order,* void ∗ *buffer,* int *count,* MPI_Datatype *datatype,* int *root,* MPI_Comm *comm*)

Base tree algorithm of bcast

4.11.2.2 template<typename Builder > int MPIB_Reduce_tree_algorithm (Builder *builder,* MPIB_child_traverse_order *order,* void ∗ *sendbuf,* void ∗ *recvbuf,* int *count,* MPI_Datatype *datatype,* MPI_Op *op,* int *root,* MPI_Comm *comm*)

Base tree algorithm of reduce.

Note

Does not perform MPI operations but allocates memory for subproduct. MPI internals should be used. For example, Open MPI:

```
#ifdef HAVE_OPENMPI_OMPI_OP_OP_H
#include <openmpi/ompi/op/op.h>
#endif
...
int MPIB_Reduce_tree_algorithm(...) {
   ...
#ifdef HAVE_OPENMPI_OMPI_OP_OP_H
   ompi_op_reduce(op, buffer, sendbuf, count, datatype);
#endif
   ...
}
```

TODO: implement MPI operation in the reduce tree algorithm

4.11.2.3 template<typename Builder > int MPIB_Scatter_tree_algorithm (Builder *builder*, MPIB_child_traverse_order *order*, void ∗ *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Base tree algorithm of scatter

4.11.2.4 template<typename Builder > int MPIB_Gather_tree_algorithm (Builder *builder*, MPIB_child_traverse_order *order*, void ∗ *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Base tree algorithm of gather

4.11.2.5 template<typename Builder > int MPIB_Scatterv_tree_algorithm (Builder *builder*, MPIB_child_traverse_order *order*, void ∗ *sendbuf*, int ∗ *sendcounts*, int ∗ *displs*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *_comm*)

Base tree algorithm of scatterv

4.11.2.6 template<typename Builder > int MPIB_Gatherv_tree_algorithm (Builder *builder*, MPIB_child_traverse_order *order*, void ∗ *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int ∗ *recvcounts*, int ∗ *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *_comm*)

Base tree algorithm of gatherv

5 Namespace Documentation

5.1 MPIB::BRSG Namespace Reference

Classes

- class Binomial_builder

5.1.1 Detailed Description

The communication tree builders for tree-based algorithms of bcast/reduce/scatter/gather must implement a function:

```
void build(int size, int root, int rank, int count,
    Graph& g, Vertex& r, Vertex& u, Vertex& v);
```

Parameters

size communicator size

root root process

rank current process

count message size in bytes (count ∗ extent)

g graph

r root vertex (introduced to avoid search)

u parent of v (introduced to avoid search)

v vertex for the current process (introduced to avoid search)

5.2 MPIB::comm_tree Namespace Reference

Typedefs

- typedef adjacency_list< listS, listS, directedS, property< vertex_index_t, int >, no_property, no_property, listS > Graph
- typedef graph_traits< Graph >::vertex_descriptor Vertex
- typedef graph_traits< Graph >::edge_descriptor Edge
- typedef graph_traits< Graph >::vertex_iterator Vertex_iterator
- typedef graph_traits< Graph >::adjacency_iterator Adjacency_iterator
- typedef graph_as_tree< Graph, iterator_property_map< vector< Vertex >::iterator, property_map< Graph, vertex_index_t >::type > > Tree

5.2.1 Detailed Description

Communication tree is a C++ namespace that contains a set of data structures for tree-based algorithms of collectives. All data types are short aliases for the Boost Graph data structures.

5.2.2 Typedef Documentation

5.2.2.1 typedef adjacency_list<listS, listS, directedS, property<vertex_index_t, int>, no_property, no_property, listS> MPIB::comm_tree::Graph

Directed graph (vertex_index = MPI rank)

Note

The vertex index property is used for indexing and must starts from zero.
The adjacent_vertices method provides only the parent->child relation. For backward connection, use the graph as tree wrapper Tree.

5.2.2.2 typedef graph_traits<Graph>::vertex_descriptor MPIB::comm_tree::Vertex

Vertex

5.2.2.3 typedef graph_traits<Graph>::edge_descriptor MPIB::comm_tree::Edge

Edge

5.2.2.4 typedef graph_traits<Graph>::vertex_iterator MPIB::comm_tree::Vertex_iterator

Vertex iterator

5.2.2.5 typedef graph_traits<Graph>::adjacency_iterator MPIB::comm_tree::Adjacency_iterator

Adjacency iterator

5.2.2.6 typedef graph_as_tree<Graph, iterator_property_map<vector<Vertex>::iterator, property_map<Graph, vertex_index_t>::type> > MPIB::comm_tree::Tree

Graph as tree wrapper. Provides access to the root, parent and children nodes.

Note

We cannot use the `Tree` data structure directly, instead of `Graph`, because of the lack of empty/copy constructors.
Access to parent nodes requires some computation (traversing the tree) - avoid this if possible.

5.3 MPIB::SG Namespace Reference

Classes

- class Assembler
- class Indexer
- class Edge_writer

5.3.1 Detailed Description

Auxiliaries for scatter/gather tree algorithms

5.4 MPIB::SGv Namespace Reference

Classes

- class Assembler
- class Indexer
- class Vertex_writer
- class Edge_writer
- class Binomial_builder
- class Sorted_binomial_builder
- class Traff_builder

5.4.1 Detailed Description

Auxiliaries for scatterv/gatherv tree algorithms

Communication tree builders for scatterv/gatherv. Must implement a function:

```
void build(int size, int root, int rank, int* counts,
    Graph& g, Vertex& r, Vertex& u, Vertex& v);
```

Parameters

- ***size*** communicator size
- ***root*** root process
- ***rank*** current process
- ***counts*** array of message sizes in bytes (counts ∗ extent) (number of elements = communicator size)
- ***g*** graph
- ***r*** root vertex (introduced to avoid search)
- ***u*** parent of v (introduced to avoid search)
- ***v*** vertex for the current process (introduced to avoid search)

tests/sgv.c is a test for tree-based scatterv/gatherv.

6 Class Documentation

6.1 MPIB::SG::Assembler Class Reference

6.1.1 Detailed Description

Finds the data to send (scatter) or recv (gather) from the local buffer (rank2index). Default copy constructors are used. Arrays must be preallocated (max = comm size).

The documentation for this class was generated from the following file:

- collectives/sg_tree_algorithms.hpp

6.2 MPIB::SGv::Assembler Class Reference

6.2.1 Detailed Description

Finds the data to send (scatterv) or recv (gatherv) from the local buffer (rank2index, counts, displs). Default copy constructors are used. Arrays must be preallocated (max = comm size).

The documentation for this class was generated from the following file:

- collectives/sgv_tree_algorithms.hpp

6.3 MPIB::SGv::Binomial_builder Class Reference

6.3.1 Detailed Description

Binomial tree builder for scatterv/gatherv. The largest subtree on the right. If root <> 0, processes' procs will be cyclically shifted to 0.

The documentation for this class was generated from the following file:

- collectives/sgv_tree_builders.hpp

6.4 MPIB::BRSG::Binomial_builder Class Reference

6.4.1 Detailed Description

Binomial tree builder for bcast/reduce/scatter/gather. The largest subtree on the right. If root <> 0, processes' ranks will be cyclically shifted to 0.

The documentation for this class was generated from the following file:

- collectives/brsg_tree_builders.hpp

6.5 MPIB::SG::Edge_writer Class Reference

6.5.1 Detailed Description

Edge writer

The documentation for this class was generated from the following file:

- collectives/sg_tree_algorithms.hpp

6.6 MPIB::SGv::Edge_writer Class Reference

6.6.1 Detailed Description

Edge writer

The documentation for this class was generated from the following file:

- collectives/sgv_tree_algorithms.hpp

6.7 MPIB::SGv::Indexer Class Reference

Public Member Functions

- Indexer (const int ∗_rscounts, int &_index, map< int, int > &_rank2index, int ∗_counts, int ∗_displs, int &_count)

6.7.1 Detailed Description

Builds the rank2index, counts and displs for the local buffer. Default copy constructors are used. Arrays must be preallocated (max = comm size).

6.7.2 Constructor & Destructor Documentation

6.7.2.1 MPIB::SGv::Indexer::Indexer (const int ∗ *_rscounts,* int & *_index,* map< int, int > & *_rank2index,* int ∗ *_counts,* int ∗ *_displs,* int & *_count*) `[inline]`

counts in the buffer to recv/send

The documentation for this class was generated from the following file:

- collectives/sgv_tree_algorithms.hpp

6.8 MPIB::SG::Indexer Class Reference

6.8.1 Detailed Description

Builds the rank2index for the local buffer. Default copy constructors are used.

The documentation for this class was generated from the following file:

- collectives/sg_tree_algorithms.hpp

6.9 MPIB_coll_container Struct Reference

Public Attributes

- const char ∗ operation
- void(∗ initialize)(void ∗_this, MPI_Comm comm, int root, int M)
- int(∗ execute)(void ∗_this, MPI_Comm comm, int root, int M)
- void(∗ finalize)(void ∗_this, MPI_Comm comm, int root)

6.9.1 Detailed Description

Container for a collective communication operation to be measured by MPIB_measure_coll (MPIB_measure_max, MPIB_measure_root, MPIB_measure_global). How to use (example in C):

- Create a data structure with the first field MPIB_coll_container.

```
typedef struct MPIB_Scatter_container {
    MPIB_coll_container base;
    char* buffer;
    MPIB_Scatter scatter;
} MPIB_Scatter_container;
```

- Implement the functions: initialize, execute, finalize, where `_this` argument can be typecasted to the data structure.

```
void MPIB_Scatter_initialize(void* _this, MPI_Comm comm, int root, int M) {
    MPIB_Scatter_container* container = (MPIB_Scatter_container*)_this;
    int rank;
    MPI_Comm_rank(comm, &rank);
    int size;
    MPI_Comm_size(comm, &size);
    container->buffer = rank == root ?
        (char*)malloc(sizeof(char) * M * size) :
        (char*)malloc(sizeof(char) * M);
}

void MPIB_Scatter_execute(void* _this, MPI_Comm comm, int root, int M) {
    MPIB_Scatter_container* container = (MPIB_Scatter_container*)_this;
    container->scatter(container->base.buffer, M, MPI_CHAR,
        container->base.buffer, M, MPI_CHAR,
        root, comm);
}

void MPIB_Scatter_finalize(void* _this, MPI_Comm comm, int root) {
    MPIB_Scatter_container* container = (MPIB_Scatter_container*)_this;
    free(container->buffer);
}
```

- Implement the functions that allocate and free the data structure.

  ```
  MPIB_Scatter_container* MPIB_Scatter_container_alloc(MPIB_Scatter scatter) {
      MPIB_Scatter_container* container =
          (MPIB_Scatter_container*)malloc(sizeof(MPIB_Scatter_container));
      container->base.operation = "Scatter";
      container->base.initialize = MPIB_Scatter_initialize;
      container->base.execute = MPIB_Scatter_execute;
      container->base.finalize = MPIB_Scatter_finalize;
      container->scatter = scatter;
      return container;
  }

  void MPIB_Scatter_container_free(void* _this) {
      free(_this);
  }
  ```

  +

In this library, collective containers are implemented in C++.

Parameters

comm MPI communicator over which the communication operation will be performed

root root process

M message size

6.9.2 Member Data Documentation

6.9.2.1 const char∗ MPIB_coll_container::operation

Communication operation

6.9.2.2 void(∗ MPIB_coll_container::initialize)(void ∗_this, MPI_Comm comm, int root, int M)

Initializion of buffers required for the communication operation (in irregular collectives, M can be different at different processors)

6.9.2.3 int(∗ MPIB_coll_container::execute)(void ∗_this, MPI_Comm comm, int root, int M)

Communication operation (in irregular collectives, M can be different at different processors)

6.9.2.4 void(∗ MPIB_coll_container::finalize)(void ∗_this, MPI_Comm comm, int root)

Finalization of buffers required for the communication operation

The documentation for this struct was generated from the following file:

- benchmarks/mpib_coll_benchmarks.h

6.10 MPIB_msgset Struct Reference

Public Attributes

- int min_size

- int max_size
- int stride
- double max_diff
- int min_stride
- int max_num

6.10.1 Detailed Description

The message sizes for which the measurements are to be performed. Bounded by min_size and max_size. If stride > 0, message sizes are selected regularly. Otherwise, they are adaptively selected at runtime, based on the max_diff, min_stride and max_num values.

6.10.2 Member Data Documentation

6.10.2.1 int MPIB_msgset::min_size

Maximum message size in bytes

6.10.2.2 int MPIB_msgset::max_size

Maximum message size in bytes

6.10.2.3 int MPIB_msgset::stride

Stride in bytes for regular selection of message sizes

6.10.2.4 double MPIB_msgset::max_diff

Maximum relative difference between the result of measurement and the linear model based on the results of two previous measurements that requires further investigation. Must be non-negative, ≤ 1. Used in adaptive selection of message sizes.

6.10.2.5 int MPIB_msgset::min_stride

Minimum stride between message sizes. Must be positive. Used in adaptive selection of message sizes.

6.10.2.6 int MPIB_msgset::max_num

Maximum number of message sizes. Limits the number of different messages sizes the measurement is performed for. Used in adaptive selection of message sizes.

The documentation for this struct was generated from the following file:

- benchmarks/mpib_measurement.h

6.11 MPIB_p2p_container Struct Reference

Public Attributes

- const char * operation
- void(* initialize)(void *_this, MPI_Comm comm, int M)
- void(* execute_measure)(void *_this, MPI_Comm comm, int M, int mirror)
- void(* execute_mirror)(void *_this, MPI_Comm comm, int M, int measure)
- void(* finalize)(void *_this, MPI_Comm comm)

6.11.1 Detailed Description

Container for a point-to-point communication operation to be measured by MPIB_measure_allp2p. How to use (example in C):

- Create a data structure with the first field MPIB_p2p_container.

```
typedef struct MPIB_Send_Recv_container {
    MPIB_p2p_container base;
    char* buffer;
} MPIB_Send_Recv_container;
```

- Implement the functions: initialize, execute_measure, execute_mirror, finalize, where `_this` argument can be typecasted to the data structure.

```
void MPIB_Send_Recv_initialize(void* _this, MPI_Comm comm, int M) {
    MPIB_Send_Recv_container* container = (MPIB_Send_Recv_container*)_this;
    container->buffer = (char*)malloc(sizeof(char) * M);
}

void MPIB_Send_Recv_execute_measure(void* _this, MPI_Comm comm, int M, int mirror
      ) {
    MPIB_Send_Recv_container* container = (MPIB_Send_Recv_container*)_this;
    MPI_Send(container->buffer, M, MPI_CHAR, mirror, 0, comm);
    MPI_Recv(container->buffer, M, MPI_CHAR, mirror, 0, comm, MPI_STATUS_IGNORE);

}

void MPIB_Send_Recv_execute_mirror(void* _this, MPI_Comm comm, int M, int measure
      ) {
    MPIB_Send_Recv_container* container = (MPIB_Send_Recv_container*)_this;
    MPI_Recv(container->buffer, M, MPI_CHAR, measure, 0, comm, MPI_STATUS_IGNORE)
      ;
    MPI_Send(container->buffer, M, MPI_CHAR, measure, 0, comm);
}

void MPIB_Send_Recv_finalize(void* _this, MPI_Comm comm) {
    MPIB_Send_Recv_container* container = (MPIB_Send_Recv_container*)_this;
    free(container->buffer);
}
```

- Implement the functions that allocate and free the data structure.

```
MPIB_p2p_container* MPIB_Send_Recv_container_alloc() {
    MPIB_p2p_container* container =
        (MPIB_p2p_container*)malloc(sizeof(MPIB_Send_Recv_container));
    container->base.operation = "MPI_Send-MPI_Recv";
    container->base.free = MPIB_p2p_container_free;
    container->initialize = MPIB_Send_Recv_initialize;
    container->execute_measure = MPIB_Send_Recv_execute_measure;
    container->execute_mirror = MPIB_Send_Recv_execute_mirror;
    container->finalize = MPIB_Send_Recv_finalize;
```

```
        return container;
    }

    void MPIB_Send_Recv_container_free(void* _this) {
        free(_this);
    }
```

Parameters

comm MPI communicator over which the communication operation will be performed

M message size

mirror mirror process

measure measure process

6.11.2 Member Data Documentation

6.11.2.1 const char∗ MPIB_p2p_container::operation

Communication operation

6.11.2.2 void(∗ MPIB_p2p_container::initialize)(void ∗_this, MPI_Comm comm, int M)

Initializion of buffers required for the communication operation.

6.11.2.3 void(∗ MPIB_p2p_container::execute_measure)(void ∗_this, MPI_Comm comm, int M, int mirror)

Part of communication at the measure side

6.11.2.4 void(∗ MPIB_p2p_container::execute_mirror)(void ∗_this, MPI_Comm comm, int M, int measure)

Part of communication at the mirror side

6.11.2.5 void(∗ MPIB_p2p_container::finalize)(void ∗_this, MPI_Comm comm)

Finalization of buffers required for the communication operation

The documentation for this struct was generated from the following file:

- benchmarks/mpib_p2p_benchmarks.h

6.12 MPIB_precision Struct Reference

Public Attributes

- int min_reps
- int max_reps
- double cl
- double eps

6.12.1 Detailed Description

Precision of measurement. Used as an input argument of benchmark functions. To provide reliable results, the communication experiments in each benchmark are repeated either fixed or variable number of times. This data structure allows the user to control the accuracy and efficiency of benchmarking.

- Assigning to min_reps and max_reps the same values results in the fixed number of repetitions of the communication operation, with the cl and eps arguments being ignored (this allows the user to control the efficiency of benchmarking).
- If min_reps $<$ max_reps, the experiments are repeated until a confidence interval, MPIB_result::ci, found with the confidence level, cl = $Pr(|\bar{T} - \mu| < ci)$, satisfies $\frac{ci}{\bar{T}} <$ eps, or the number of repetitions reaches its maximum, max_reps (this allows the user to control the accuracy of benchmarking).

6.12.2 Member Data Documentation

6.12.2.1 int MPIB_precision::min_reps

Minimum number of repetitions

6.12.2.2 int MPIB_precision::max_reps

Maximum number of repetitions

6.12.2.3 double MPIB_precision::cl

Confidence level $\in [0, 1]$: $cl = Pr(|\bar{T} - \mu| < ci) = Pr(\frac{|\bar{T} - \mu|}{\bar{T}} < \epsilon)$.

6.12.2.4 double MPIB_precision::eps

Relative error $\in [0, 1]$: $\frac{|\bar{T} - \mu|}{\bar{T}} < \frac{ci}{\bar{T}} < \epsilon = eps$.

The documentation for this struct was generated from the following file:

- benchmarks/mpib_measurement.h

6.13 MPIB_result Struct Reference

Public Attributes

- int M
- double T
- double wtick
- int reps
- double ci

6.13.1 Detailed Description

Result of measurement and its reliability. Used as an output argument of benchmark functions.

6.13.2 Member Data Documentation

6.13.2.1 int MPIB_result::M

Message size

6.13.2.2 double MPIB_result::T

Execution time

6.13.2.3 double MPIB_result::wtick

Resolution of MPI_Wtime

6.13.2.4 int MPIB_result::reps

Number of repetitions the benchmark has actually taken

6.13.2.5 double MPIB_result::ci

Confidence interval, $|\bar{T} - \mu| < ci$. The MPIB_ci function estimates confidence interval, using t distribution.

The documentation for this struct was generated from the following file:

- benchmarks/mpib_measurement.h

6.14 MPIB::SGv::Sorted_binomial_builder Class Reference

6.14.1 Detailed Description

Sorted binomial tree builder for scatterv/gatherv. The largest subtree on the right. Processors are sorted by counts in asc/dsc order. Algorithm is described in [3].

The documentation for this class was generated from the following file:

- collectives/sgv_tree_builders.hpp

6.15 MPIB::SGv::Traff_builder Class Reference

6.15.1 Detailed Description

Traff tree builder for scatterv/gatherv. Algorithm is described in [3].

The documentation for this class was generated from the following file:

- collectives/sgv_tree_builders.hpp

6.16 MPIB::SGv::Vertex_writer Class Reference

6.16.1 Detailed Description

Vertex writer

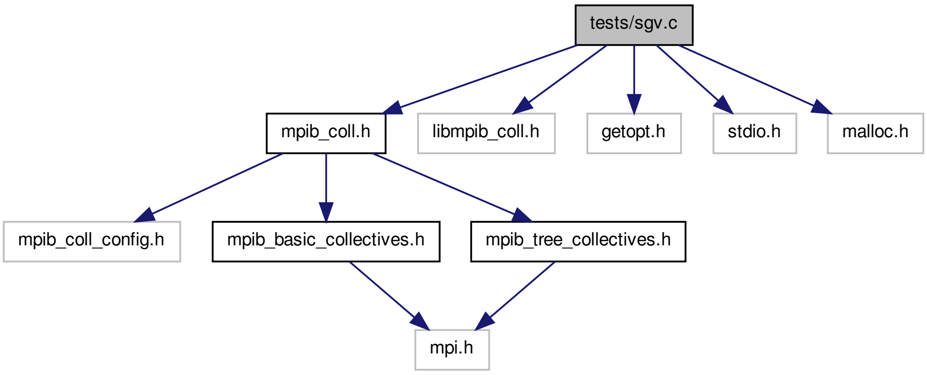
The documentation for this class was generated from the following file:

- collectives/sgv_tree_algorithms.hpp

7 File Documentation

7.1 tests/sgv.c File Reference

Include dependency graph for sgv.c:

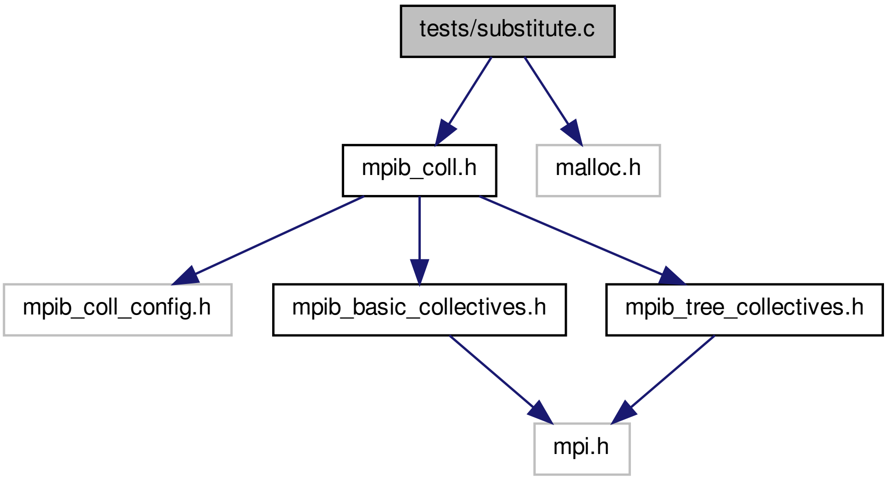


7.1.1 Detailed Description

Test for tree-based scatterv/gatherv. Based on [3].

7.2 tests/substitute.c File Reference

Include dependency graph for substitute.c:



7.2.1 Detailed Description

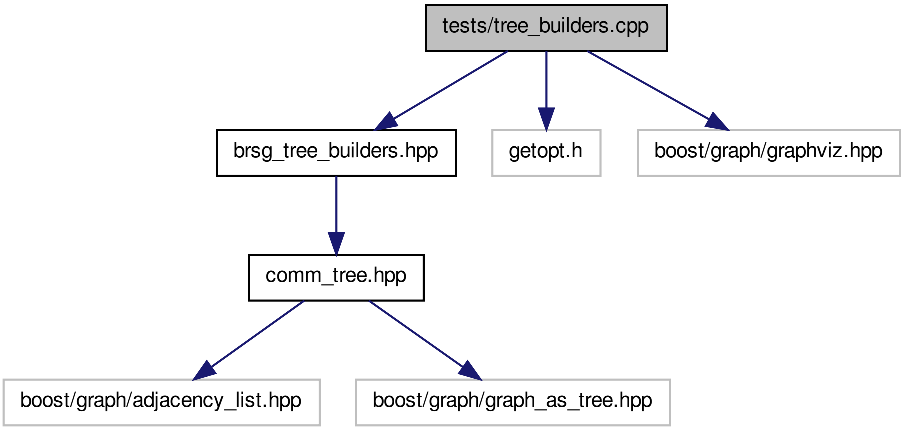
Substitute for native MPI collective operations

- add `-I$(MPIBlib-dir)/include` to CPPFLAGS
- add `-L$(MPIBlib-dir)/lib -lmpib_coll` to LDFLAGS
- replace `#include <mpi.h>` by `#include "mpib_coll.h"`
- define macro substitutes for native MPI collective operations, for example

```
#define MPI_Scatter MPIB_Scatter_flat
```

7.3 tests/tree_builders.cpp File Reference

Include dependency graph for tree_builders.cpp:



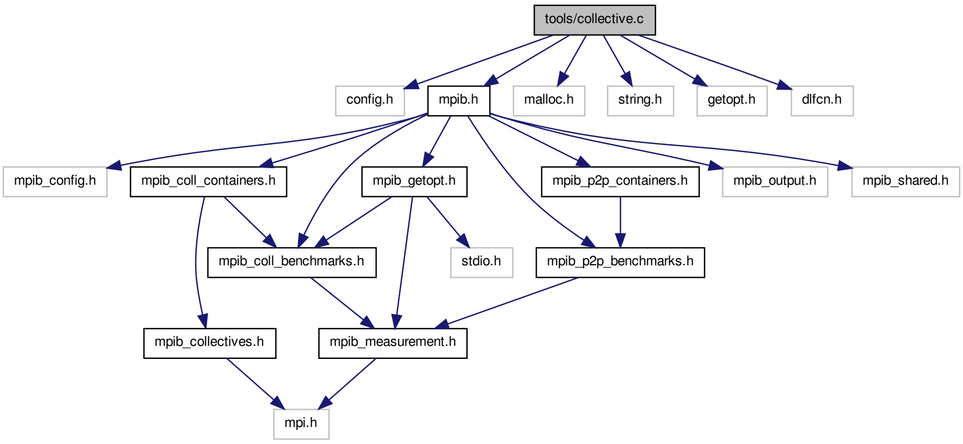
7.3.1 Detailed Description

Test for tree builders. Usage:

- include a header file with your builder class
- use your builder class in the main function

7.4 tools/collective.c File Reference

Include dependency graph for collective.c:



7.4.1 Detailed Description

Collective benchmarking executable. Performs regular universal or operation-specific collective benchmark. Basic arguments are described in Options for executables.

collective.plot draws the graph of the execution time (sec) against message size (kb) with error bars.

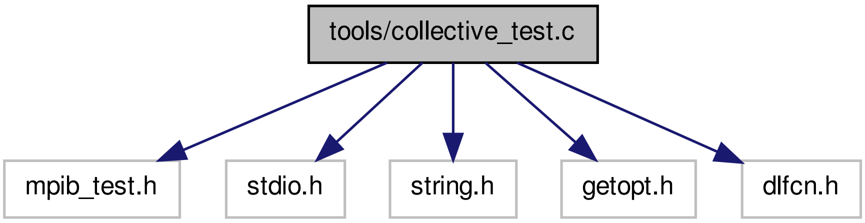
- input: collective.out
- output: collective.eps

Using the gnuplot script:

```
$ gnuplot collective.plot
```

7.5 tools/collective_test.c File Reference

Include dependency graph for collective_test.c:

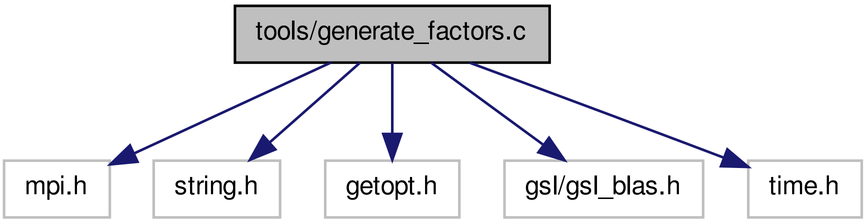


7.5.1 Detailed Description

Performs tests for the implementations of MPI collective operations with different root

7.6 tools/generate_factors.c File Reference

Include dependency graph for generate_factors.c:

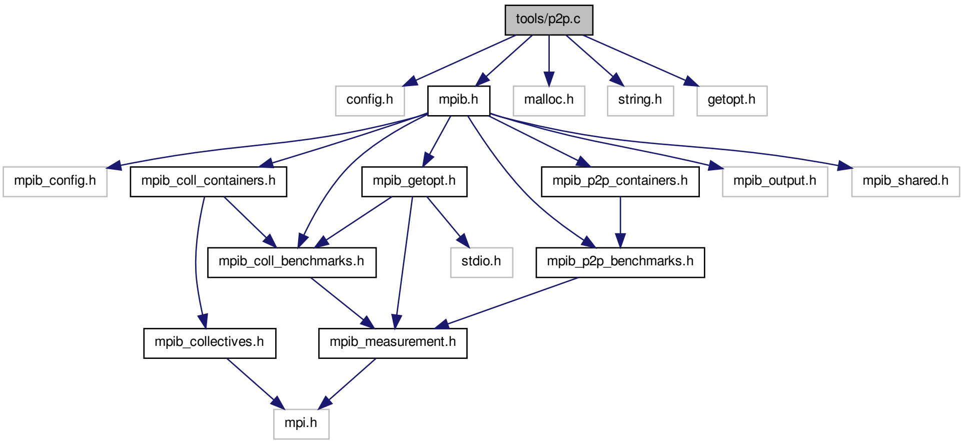


7.6.1 Detailed Description

A standalone tool for generating factors which can be used as input file for irregular scatter/gather benchmarks like in the tools/collective.c . The factors are normalized so that the average factor is 1. The only valid argument is either -r for random factors or -c for factors based on a small GSL matrix multiplication benchmark performed on every process' node

7.7 tools/p2p.c File Reference

Include dependency graph for p2p.c:



7.7.1 Detailed Description

p2p benchmarking executable. Performs adaptive p2p benchmark between a pair of processors 0-1. Arguments are described in Options for executables.

p2p.plot draws the graph of the execution time (sec) against message size (kb).

- input: p2p.out
- output: p2p.eps (0-1 with error bars)

Using the gnuplot script:

```
$ gnuplot collective.plot
```

Index