MPIBlib: MPI Benchmark library

Version 1.0.0 (Revision 105)

Generated by Doxygen 1.5.6

Wed Sep 3 09:21:02 2008

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 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

Heterogeneous Computing Laboratory
School of Computer Science and Informatics, University College Dublin
Belfield, Dublin 4, Ireland
http://hcl.ucd.ie

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

1.2 Changes

```
1.0.0 (105)
-----------

Initial release
```

2 Installation

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

Required software:
1. any MPI implementation
2. GSL (GNU Scientific Library)
3. Gnuplot - optional

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

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

Required software:
1. Subversion
2. GNU autotools

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

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 --prefix=?
$ make install

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

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

3 Usage

The package consists of a library, executables and gnuplot scripts.

Benchmarking executables and gnuplot scripts that visualize the results are described in:

- collective/main.c
- p2p/main.c

Typical parameters of executables are as follows:

- **-h** help
- **-O** *S* collective operation (required):
  MPI_Scatter, MPIB_Scatter_linear, MPIB_Scatter_binomial
  MPI_Gather, MPIB_Gather_linear, MPIB_Gather_binomial
  MPI_Scatterv
  MPI_Gatherv
- **-t** *S* timing: max, root, global (default: max)
- **-s** *I* message size stride (default: 1024)
- **-m** *I* maximum message size (default: 102400)
- **-p** 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* error: $0 < D < 1$ (default: 0.025)

where:

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

Using the gnuplot

```
$ gnuplot script_name.plot
```

The gnuplot data files should have names script_name.out

4 Module Documentation

4.1 Implementations and tests for 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)

  Scatter.

- 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 void(* MPIB_tree)(int size, int root, int rank, int *block, int *parent, int *count, int **children, int **blocks)

  Builds a segment of a tree for the process rank.

- typedef int(* MPIB_Bcast)(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

  Bcast.

Functions

- void MPIB_test_scatter (MPIB_Scatter scatter, MPI_Comm comm, int root, int *res)

  Checks a scatter implementation.

- void MPIB_test_gather (MPIB_Gather gather, MPI_Comm comm, int root, int *res)

  Checks a gather implementation.

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

  Linear MPI_Scatter.

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

  Linear MPI_Gather.

- int MPIB_Scatter_tree (MPIB_tree tree, void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

  Tree-based MPI_Scatter.

- int MPIB_Gather_tree (MPIB_tree tree, void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

  Tree-based MPI_Gather.

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

  Binomial MPI_Scatter.

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

  Binomial MPI_Gather.

4.1.1 Typedef Documentation

4.1.1.1 typedef void(* MPIB_tree)(int size, int root, int rank, int *block, int *parent, int *count, int **children, int **blocks)

Builds a segment of a tree for the process rank.

The processes' ranks are arranged in ascending order, starting with 0. If root is not equal to 0, the root and 0 processes are interchanged in the tree.

Parameters:

size communicatot size

root root process

rank current process

block message size to receive from parent process (size at root)

parent parent process (-1 at root)

count number of processes to send messages

children array of processes to send messages

blocks array of message sizes to send

4.1.2 Function Documentation

4.1.2.1 void MPIB_test_scatter (MPIB_Scatter *scatter*, MPI_Comm *comm*, int *root*, int * *res*)

Checks a scatter implementation.

Parameters:

scatter scatter implementation

comm communicator

root root process

res result (0 - ok)

4.1.2.2 void MPIB_test_gather (MPIB_Gather *gather*, MPI_Comm *comm*, int *root*, int * *res*)

Checks a gather implementation.

Parameters:

gather gather implementation

comm communicator

root root process

res result (0 - ok)

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

Binomial MPI_Scatter.

```
return MPIB_Scatter_tree(MPIB_binomial_tree, sendbuf, sendcount, sendtype,
    recvbuf, recvcount, recvtype, root, comm);
```

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

Binomial MPI_Gather.

```
return MPIB_Gather_tree(MPIB_binomial_tree, sendbuf, sendcount, sendtype,
    recvbuf, recvcount, recvtype, root, comm);
```

4.2 Containers for the communication operations to be measured

Base data structures and some implemented containers.

Data Structures

- struct MPIB_p2p_container

  Container for a point-to-point communication operation to be measured by MPIB_measure_p2p.

- struct MPIB_coll_container

  Container for a collective communication operation to be measured by: MPIB_measure_max, MPIB_measure_root, MPIB_measure_global.

Functions

- MPIB_p2p_container * MPIB_Send_Recv_container_alloc ()

  Allocates container for send/recv.

- MPIB_coll_container * MPIB_coll_container_alloc (const char *operation)

  Allocates container for a collective operation.

- MPIB_coll_container * MPIB_Scatter_container_alloc (MPIB_Scatter scatter)

  Allocates container for scatter.

- MPIB_coll_container * MPIB_Gather_container_alloc (MPIB_Gather gather)

  Allocates container for gather.

- MPIB_coll_container * MPIB_Scatterv_container_alloc (int stride)

  Allocates container for scatterv.

- MPIB_coll_container * MPIB_Gatherv_container_alloc (int stride)

  Allocates container for gatherv.

4.2.1 Detailed Description

Base data structures and some implemented containers.

How to use:

- Create the instance of the container.
- Pass the instance of the container to the measurement function.
- Destroy the instance of the container.

```
MPIB_p2p_container* container = MPIB_x_container_alloc();
MPIB_measure_p2p(container, comm, M, parallel, precision, result);
container->free(container);
```

How to extend:

- Create a data structure with the first field MPIB_p2p_container or MPIB_coll_container.

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

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

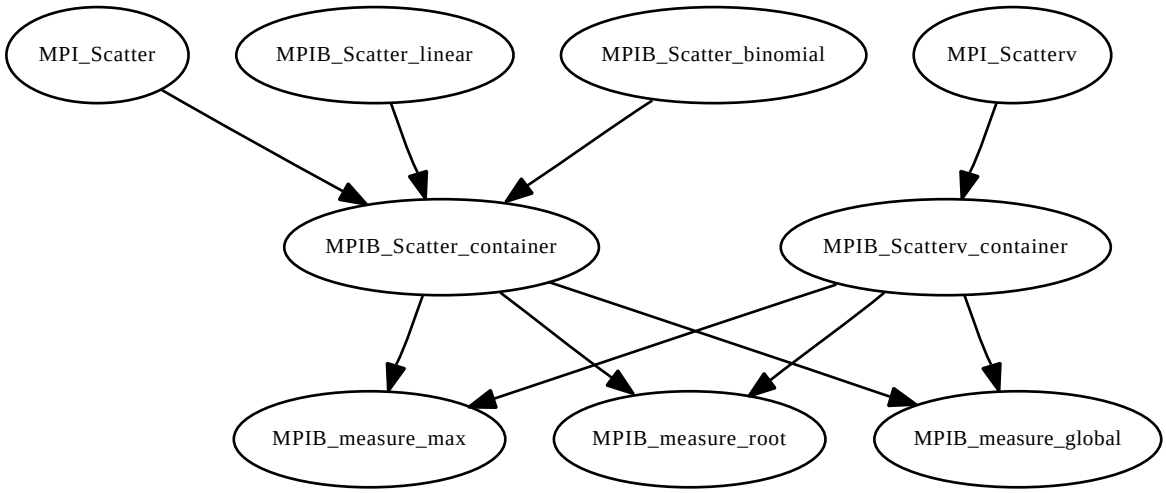
```
void MPIB_Send_Recv_initialize(void* this, MPI_Comm comm, int M)
{
    MPIB_Send_Recv_container* container = (MPIB_Send_Recv_container*)this;
    container->buffer = (char*)malloc(M);
}

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;
    MPIB_Send(container->buffer, M, MPI_CHAR, mirror, 0, comm);
    MPIB_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;
    MPIB_Recv(container->buffer, M, MPI_CHAR, measure, 0, comm, MPI_STATUS_IGNORE);
    MPIB_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.

```
void MPIB_container_free(void* this)
{
    free(this);
}
```

```
MPIB_p2p_container* MPIB_Send_Recv_container_alloc()
{
    MPIB_p2p_container* container =
        (MPIB_p2p_container*)malloc(sizeof(MPIB_Send_Recv_container));
    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;
    container->free = MPIB_container_free;
    return container;
}
```

Container allocators can have arguments, for example MPIB_Scatter_container_alloc has an argument `MPIB_Scatter`, pointer to a scatter implementation. This provides two-level extension of measurement:



4.2.2 Function Documentation

4.2.2.1 MPIB_coll_container* MPIB_coll_container_alloc (const char * *operation*)

Allocates container for a collective operation.

Parameters:

operation collective operation:

- MPI_Scatter, MPIB_Scatter_linear, MPIB_Scatter_binomial
- MPI_Gather, MPIB_Gather_linear, MPIB_Gather_binomial
- MPI_Scatterv
- MPI_Gatherv

Returns:

container (NULL if none of above)

4.2.2.2 MPIB_coll_container* MPIB_Scatter_container_alloc (MPIB_Scatter *scatter*)

Allocates container for scatter.

Parameters:

scatter scatter implementation

Returns:

container

4.2.2.3 MPIB_coll_container* MPIB_Gather_container_alloc (MPIB_Gather *gather*)

Allocates container for gather.

Parameters:

gather gather implementaiton

Returns:

container

4.3 Emulated heterogeneity

Functions

- void MPIB_hetero_procs_init (MPI_Comm comm)

  Initializes the parameters of emulated heterogeneity of processors.

- void MPIB_hetero_procs_free ()

  Frees the parameters of emulated heterogeneity of processors.

- int MPIB_Send (void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

  Slow MPI_Send based on emulated heterogeneity of processors.

- int MPIB_Recv (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

  Slow MPI_Recv based on emulated heterogeneity of processors.

4.3.1 Function Documentation

4.3.1.1 int MPIB_Send (void * *buf*, int *count*, MPI_Datatype *datatype*, int *dest*, int *tag*, MPI_Comm *comm*)

Slow MPI_Send based on emulated heterogeneity of processors.

Requires MPIB_hetero_procs_init

4.3.1.2 int MPIB_Recv (void * *buf*, int *count*, MPI_Datatype *datatype*, int *source*, int *tag*, MPI_Comm *comm*, MPI_Status * *status*)

Slow MPI_Recv based on emulated heterogeneity of processors.

Requires MPIB_hetero_procs_init

4.4 Measurement

Base structures and functions for measurements.

Data Structures

- struct MPIB_precision

  Measurement presicion.

- struct MPIB_result

  Result of the measurement defined by MPIB_precision.

Functions

- void MPIB_max_wtick (MPI_Comm comm, double *wtick)

  Returns a resolution of MPI_Wtime, maximum in the communicator.

4.4.1 Detailed Description

Base structures and functions for measurements.

4.4.2 Function Documentation

4.4.2.1 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

4.5 Measurement of point-to-point communications

Measures the execution time of point-to-point communications.

Functions

- void MPIB_measure_p2p (MPIB_p2p_container *container, MPI_Comm comm, int M, int parallel, MPIB_precision precision, MPIB_result *results)

  Measures the point-to-point execution time.

4.5.1 Detailed Description

Measures the execution time of point-to-point communications.

4.5.2 Function Documentation

4.5.2.1 void MPIB_measure_p2p (MPIB_p2p_container * *container*, MPI_Comm *comm*, int *M*, int *parallel*, MPIB_precision *precision*, MPIB_result * *results*)

Measures the point-to-point execution time.

Measures the execution time of each $i \xleftrightarrow[M]{M} j$ roundtrip in the communicator, $i < j$. Performes series of roundtrips to obtain reliable results.

Parameters:

container communication operaion container

comm communicator, number of nodes should be ≥ 2

M message size

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

precision measurement precision

results array of C_n^2 measurement results

4.6 Measurement of collective communications (universal methods)

Measures the execution time of collective communications by universal methods.

Typedefs

- typedef void(* MPIB_measure_coll)(MPIB_coll_container *container, MPI_Comm comm, int root, int M, MPIB_precision precision, MPIB_result *result)

  Measures the execution time of collective operation.

Functions

- void 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.

- 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.

- void 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.

- 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.

- void 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.

4.6.1 Detailed Description

Measures the execution time of collective communications by universal methods.

4.6.2 Typedef Documentation

4.6.2.1 typedef void(* MPIB_measure_coll)(MPIB_coll_container *container, MPI_Comm comm, int root, int M, MPIB_precision precision, MPIB_result *result)

Measures the execution time of collective operation.

Parameters:

container communication operaion container

comm communicator

root root process

M message size

precision measurement precision

result measurement result

4.6.3 Function Documentation

4.6.3.1 void 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.

4.6.3.2 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)

4.6.3.3 void 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.

4.6.3.4 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)

4.6.3.5 void 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.

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.

4.7 Measurement of collective communications (operation-specific methods)

Measures the execution time of collective communications by operation-specific methods.

Functions

- 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.

- 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.

4.7.1 Detailed Description

Measures the execution time of collective communications by operation-specific methods.

4.7.2 Function Documentation

4.7.2.1 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

4.7.2.2 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.

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

4.8 Utilities

Data Structures

- struct MPIB_pair

  List of pairs.

- struct MPIB_pairs

  List of lists of pairs.

Defines

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

  C_n^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, $i \neq j < n$.

Functions

- double MPIB_ci (double cl, int reps, double *T)

  Returns a confidence interval that contains the average execution time with a certain probability.

- MPIB_pairs * MPIB_build_pairs (int n)

  Builds all combinations of non-overlapped pairs of communicating nodes.

- void MPIB_free_pairs (MPIB_pairs *pairs)

  Frees the pairs recursively.

- void MPIB_print_precision (MPIB_precision precision)

  Prints a precision.

- void MPIB_print_p2p_table (int M, MPIB_precision precision, int n, MPIB_result *results)

  Prints the per-p2p table of the p2p benchmark results.

- void MPIB_print_p2p_th (MPIB_precision precision, int n)

  Prints the table header of the p2p benchmark results.

- void MPIB_print_p2p_tr (int M, int n, MPIB_result *results)

  Prints the table row of the p2p benchmark results.

- void MPIB_print_coll_th (const char *operation, int n, int root, MPIB_precision precision)

  Prints the table header of the collective benchmark result.

- void MPIB_print_coll_tr (int M, MPIB_result result)

  Prints the table row of the collective benchmark result.

4.8.1 Define Documentation

4.8.1.1 #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, $i \neq j < n$.

$\frac{(n-1)+(n-I)}{2} I + (J - I - 1), I = min(i, j), J = max(i, j)$

4.8.2 Function Documentation

4.8.2.1 double MPIB_ci (double *cl*, int *reps*, double * *T*)

Returns a confidence interval that contains the average execution time with a certain probability.

$Pr(|\bar{T} - \mu| < ci) = cl$

Parameters:

cl confidence level

reps number of measurements (should be > 1)

T array of reps measurement results

Returns:

confidence interval

4.8.2.2 void MPIB_print_p2p_table (int *M*, MPIB_precision *precision*, int *n*, MPIB_result * *results*)

Prints the per-p2p table of the p2p benchmark results.

Parameters:

precision precision

M message size

n number of nodes

results array of C_n^2 results

4.8.2.3 void MPIB_print_p2p_th (MPIB_precision *precision*, int *n*)

Prints the table header of the p2p benchmark results.

Parameters:

precision precision

n number of nodes

4.8.2.4 void MPIB_print_p2p_tr (int *M*, int *n*, MPIB_result * *results*)

Prints the table row of the p2p benchmark results.

Parameters:

M message size

n number of nodes

results array of C_n^2 results

4.8.2.5 void MPIB_print_coll_th (const char * *operation*, int *n*, int *root*, MPIB_precision *precision*)

Prints the table header of the collective benchmark result.

If operation == NULL, header is commented, with the time coulumn title "time". Otherwise, header is uncommented (for gnuplot autoheader), with the time column title operation.

Parameters:

operation collective operation

n number of nodes

root root process

precision precision

4.8.2.6 void MPIB_print_coll_tr (int *M*, MPIB_result *result*)

Prints the table row of the collective benchmark result.

Parameters:

M message size

result measurement result

5 Data Structure Documentation

5.1 MPIB_coll_container Struct Reference

Container for a collective communication operation to be measured by: MPIB_measure_max, MPIB_measure_root, MPIB_measure_global.

`#include <mpib_container.h>`

Data Fields

- void(∗ initialize)(void ∗this, MPI_Comm comm, int root, int M)
  Initializion of data required for the communication operation.
- void(∗ execute)(void ∗this, MPI_Comm comm, int root, int M)
  Communication operation.
- void(∗ finalize)(void ∗this, MPI_Comm comm, int root)
  Finalization of data required for the communication operation.
- void(∗ free)(void ∗this)
  Frees container.

5.1.1 Detailed Description

Container for a collective communication operation to be measured by: MPIB_measure_max, MPIB_measure_root, MPIB_measure_global.

Parameters:

this pointer to itself

comm MPI communicator over which the communication operation will be performed

root root process

M message size

5.1.2 Field Documentation

5.1.2.1 void(∗ MPIB_coll_container::initialize)(void ∗this, MPI_Comm comm, int root, int M)

Initializion of data required for the communication operation.

Used by MPIB_measure_coll

5.1.2.2 void(∗ MPIB_coll_container::execute)(void ∗this, MPI_Comm comm, int root, int M)

Communication operation.

Used by MPIB_measure_coll

5.1.2.3 void(∗ MPIB_coll_container::finalize)(void ∗this, MPI_Comm comm, int root)

Finalization of data required for the communication operation.

Used by MPIB_measure_coll

5.1.2.4 void(∗ MPIB_coll_container::free)(void ∗this)

Frees container.

Usage:

```
MPIB_p2p_container* container = ...;
...
container->free(container);
```

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

- src/mpib_container.h

5.2 MPIB_p2p_container Struct Reference

Container for a point-to-point communication operation to be measured by MPIB_measure_p2p.

`#include <mpib_container.h>`

Data Fields

- void(∗ initialize)(void ∗this, MPI_Comm comm, int M)
  Initializion of data required for the communication operation.
- void(∗ execute_measure)(void ∗this, MPI_Comm comm, int M, int mirror)
  Part of communication at the measure side.
- void(∗ execute_mirror)(void ∗this, MPI_Comm comm, int M, int measure)
  Part of communication at the mirror side.
- void(∗ finalize)(void ∗this, MPI_Comm comm)
  Finalization of data required for the communication operation.

- void(∗ free)(void ∗this)

  Frees container.

5.2.1 Detailed Description

Container for a point-to-point communication operation to be measured by MPIB_measure_p2p.

Parameters:

this pointer to itself

comm MPI communicator over which the communication operation will be performed

M message size

mirror mirror process

measure measure process

5.2.2 Field Documentation

5.2.2.1 void(∗ MPIB_p2p_container::initialize)(void ∗this, MPI_Comm comm, int M)

Initializion of data required for the communication operation.

Used by MPIB_measure_p2p

5.2.2.2 void(∗ MPIB_p2p_container::execute_measure)(void ∗this, MPI_Comm comm, int M, int mirror)

Part of communication at the measure side.

Used by MPIB_measure_p2p

5.2.2.3 void(∗ MPIB_p2p_container::execute_mirror)(void ∗this, MPI_Comm comm, int M, int measure)

Part of communication at the mirror side.

Used by MPIB_measure_p2p

5.2.2.4 void(∗ MPIB_p2p_container::finalize)(void ∗this, MPI_Comm comm)

Finalization of data required for the communication operation.

Used by MPIB_measure_p2p

5.2.2.5 void(∗ MPIB_p2p_container::free)(void ∗this)

Frees container.

Usage:

```
MPIB_p2p_container* container = ...;
...
container->free(container);
```

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

- src/mpib_container.h

5.3 MPIB_pair Struct Reference

List of pairs.

`#include <mpib_utilities.h>`

Data Fields

- int values [2]
  values
- struct MPIB_pair * prev
  previous pair
- struct MPIB_pair * next
  next pair

5.3.1 Detailed Description

List of pairs.

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

- src/mpib_utilities.h

5.4 MPIB_pairs Struct Reference

List of lists of pairs.

`#include <mpib_utilities.h>`

Data Fields

- MPIB_pair * list
  items
- struct MPIB_pairs * prev
  previous list
- struct MPIB_pairs * next
  next list

5.4.1 Detailed Description

List of lists of pairs.

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

- src/mpib_utilities.h

5.5 MPIB_precision Struct Reference

Measurement presicion.

`#include <mpib_measurement.h>`

Data Fields

- int min_reps

  minimum number of repetitions

- int max_reps

  maximum number of repetitions

- double cl

  confidence level $\in [0, 1]$

- double eps

  error $\in [0, 1]$

5.5.1 Detailed Description

Measurement presicion.

To provide reliable results, the communication experiments in each benchmark are repeated either fixed or variable number of times. This allows the user to control the accuracy of the obtained estimation of the execution time.

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

  Note:

  As communication operations in a series are isolated from each other, we suppose that the execution times are an independent sample from a normally distributed population.

5.5.2 Field Documentation

5.5.2.1 double MPIB_precision::cl

confidence level $\in [0, 1]$

$cl = Pr(|\bar{T} - \mu| < ci) = Pr(\frac{|\bar{T}-\mu|}{|\bar{T}|} < \epsilon)$

5.5.2.2 double MPIB_precision::eps

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:

- src/mpib_measurement.h

5.6 MPIB_result Struct Reference

Result of the measurement defined by MPIB_precision.

`#include <mpib_measurement.h>`

Data Fields

- double T
  execution time
- double wtick
  resolution of MPI_Wtime
- int reps
  number of repetitions the benchmark has actually taken
- double ci
  confidence interval, $|\bar{T} - \mu| < ci$.

5.6.1 Detailed Description

Result of the measurement defined by MPIB_precision.

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

- src/mpib_measurement.h

6 File Documentation

6.1 collective/main.c File Reference

Collective benchmark.

```
#include "mpib.h"
#include <getopt.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

6.1.1 Detailed Description

Collective benchmark.

Checks and benchmarks collective operations

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

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

6.2 p2p/main.c File Reference

p2p benchmark

```
#include "mpib.h"
#include <getopt.h>
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
```

6.2.1 Detailed Description

p2p benchmark

Benchmarks p2p communications between all pairs

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

- input: p2p.out
- output: 0-1.eps (0-1 with error bars), 0-1-2.eps (comparison of 0-1, 0-2, 1-2)

Index