Pallas MPI Benchmarks - PMB, Part MPI-1

Pallas GmbH
Hermülheimer Str. 10
D-50321 Brühl
Phone: +49-(0)2232-1896-0
Fax: +49-(0)2232-1896-29
http://www.pallas.com

1 INTRODUCTION 3

2 INSTALLATION AND QUICK START OF PMB-MPI1 4

2.1 Download 4

2.2 Installation 4

2.3 Running PMB-MPI1 6

3 OVERVIEW OF PMB-MPI1 6

3.1 General 6

3.2 The Benchmarks 6

3.3 Version changes 7
3.3.1 Version 2.1 vs. 2.0 7
3.3.2 Version 2.2 vs. 2.1 7

3.4 PMB-MPI1 vs. PMB1.x Definitions 7
3.4.1 Changed Definitions 8
3.4.2 Throughput Calculations 8
3.4.3 Corrected Methodology 8

4 PMB-MPI1 BENCHMARK DEFINITIONS 9

4.1 Benchmark Classification 9
4.1.1 Single Transfer Benchmarks 10
4.1.2 Parallel Transfer Benchmarks 10
4.1.3 Collective Benchmarks 11

4.2 Definition of Single Transfer Benchmarks 11
4.2.1 PingPong 12
4.2.2 PingPing 13

4.3 Definition of Parallel Transfer Benchmarks 14
4.3.1 Sendrecv 15
4.3.2 Exchange 16

4.4 Definition of Collective Benchmarks 16
4.4.1 Reduce 17
4.4.2 Reduce_scatter 17
4.4.3 Allreduce 17
4.4.4 Allgather 18
4.4.5 Allgatherv 18
4.4.6 Alltoall 18
4.4.7 Bcast 18
4.4.8 Barrier 19

5 BENCHMARK METHODOLOGY 19

5.1 Running PMB, Command Line Control 20

5.1.1 Default Case 20
5.1.2 Command Line Control 20

5.2 PMB Parameters and Hard Coded Settings 22
5.2.1 Parameters Controlling PMB 22
5.2.2 Communicators, Active Processes 24
5.2.3 Message Lengths 24
5.2.4 Buffer Initialization 24
5.2.5 Warm-up Phase 25
5.2.6 Synchronization 25
5.2.7 The Actual Benchmark 25

6 OUTPUT 26

6.1 Sample 1 27

6.2 Sample 2 29

6.3 Sample 3 31

7 FURTHER DETAILS 33

7.1 Memory Requirements 33

7.2 SRC Directory 33

7.3 Results Checking 33

7.4 Use of MPI 34

8 REVISION HISTORY 34

REFERENCES 34

1 Introduction

In an effort to define a standard API for message-passing programming, a forum of HPC vendors, researchers and users has developed the Message Passing Interface. MPI-1 [1] and MPI-2 [2] are now firmly established as the premier message-passing API, with implementations available for a wide range of platforms in the high-performance and general computing area, and a growing number of applications and libraries using MPI. To help compare the performance of various computing platforms and/or MPI implementations, the need for a set of well-defined MPI benchmarks arises.

This document presents the Pallas MPI Benchmarks (PMB) suite. Its objectives are:

- provide a concise set of benchmarks targeted at measuring the most important MPI functions.
- set forth a precise benchmark methodology.
- don’t impose much of an interpretation on the measured results: report bare timings instead. Show throughput values, if and only if these are well defined.

This document accompanies the version 2.2 of PMB. The code is written in *ANSI C plus standard MPI* (about 10 000 lines of code, 100 functions in 48 source files).

The PMB 2.2 package consists of 3 separate parts:

- PMB-MPI1 (the focus of this document)
- PMB-MPI2 (see [3]), subdivided into
  PMB-EXT (Onesided Communications benchmarks),
  PMB-IO (I/O benchmarks).

For each part, a separate executable can be built. Users who don’t have the MPI-2 extensions available, can install and use just PMB-MPI1. Only standard MPI-1 functions [1] are used, no dummy library is needed.

This document is dedicated to PMB-MPI1.

Section 2 is a brief installation guide, in section 3 an overview of the suite is given.

Section 4 defines the single benchmarks in detail. PMB introduces a classification of its benchmarks. *Single Transfer, Parallel Transfer, Collective* are the classes. Roughly speaking, Single transfers run *dedicated*, without obstructions from other transfers, undisturbed results are to be expected (`PingPong` being the most well known example). Parallel transfers test the system under global load, with concurrent actions going on. Finally, Collective is a proper MPI classification, these benchmarks test the quality of the implementation for the higher level collective functions.

Section 5 defines the methodology and rules of PMB, section 6 shows the output tables format. In section 7, further important details are explained, in particular a results checking mode for PMB.

2 Installation and Quick Start of PMB-MPI1

In order to run PMB-MPI1, one needs:

- `cpp`, ANSI C compiler, `make`.
- Full MPI-1 installation, including startup mechanism for parallel MPI programs.

See 7.1 for the memory requirements of PMB-MPI1.

2.1 Download

Get `PMB.tar.gz` at `http://www.pallas.de/pages/pmbd.htm`

2.2 Installation

After unpacking, on the `current directory is created:`

`PMB2` (directory)

`PMB2/SRC` (subdirectory containing sources and `Makefile`, see 7.2)

`PMB2/RESULTS` (subdirectory with sample results from various machines)

`PMB2/DOC` (subdirectory containing this document in postscript format)

The installation is performed in the SRC subdirectory to keep the structure easy. Here, a generic `Makefile` can be found. All rules and dependencies are defined there. Only a few (7, precisely) machine dependent variables have to be set, whereafter an easy

```
make
```

will perform the installation.

For defining the machine dependent settings, the section

```
##### User configurable options #####
#include make_i86
#include make_solaris
#include make_dec
#include make_sp2
#include make_sr2201
#include make_vpp
#include make_t3e
#include make_sgi
#include make_sx4
### End User configurable options ###
```

is provided in the Makefile header. The listed make_* files are on the directory and have been used successfully on certain systems. *First check whether one of these can be used for your purpose (with no or marginal changes).*

However, usually the user will have to edit an own `make_mydefs` file. This has to contain 7 variable assignments used by `Makefile`:

```
CC = <name of the ANSI C compiler>
CLINKER  = <name of the ANSI C linker>
OPTFLAGS = <flags for $(CC) compilation step>
CPPFLAGS = <cpp flags>
/*
Allowed values: -DnoCHECK, -DCHECK (see 7.3;
check that proper benchmarks are always with the cpp
flag -DnoCHECK!)
*/
LIB_PATH = <path of libraries (libmpi.a,...)>,
/* in the form ″-L<path1> -L<path2> ..″ */
LIBS = <lib flags for link step>, e.g. -lmpi -l....
MPI_INCLUDE = <directory containing MPI include file
               mpi.h>
```

Activate these flags by editing the Makefile header:

```
##### User configurable options #####
include make_mydefs
#include make_i86
#include make_solaris
#include make_dec
#include make_sp2
#include make_sr2201
#include make_vpp
#include make_t3e
#include make_sgi
#include make_sx4
### End User configurable options ###
```

User flags will be used in the following way:

```
$(CC) -I$(MPI_INCLUDE) $(CPPFLAGS) $(OPTFLAGS) -c
```

(for compilation)

```
$(CLINKER) -o <exe-name> [.o’s]  $(LIB_PATH) $(LIBS)
```

(for linking).

Of course, the user may define own auxiliary variables in `make_mydefs`. Now, type

```
make [PMB-xxx]
```

Compilation should be quite short, and executable `PMB-xxx` will be generated.

2.3 Running PMB-MPI1

Check the right way of running parallel MPI programs on your system. Usually, a startup procedure has to be invoked, like

```
mpirun -np P PMB-MPI1
```

(`P` being the number of processes to load; `P=1` is allowed!). This will run all of PMB on a varying number of processes `(2,4,8,..,2`^{x}`<P,P)` and output results on stdout. Also is possible

```
mpirun -np P PMB-MPI1 [Benchmark names]
```

where the names are one or more of (`PingPong, PingPing, Sendrecv, Exchange, Reduce, Reduce_scatter, Allreduce, Bcast, Allgather, Allgatherv, Alltoall, Barrier)`. The selected benchmarks will run, their meaning should be clear to MPI experts.

For the details, see 5.1.

3 Overview of PMB-MPI1

3.1 General

The idea of PMB is to provide a concise set of elementary MPI benchmark kernels. With one executable, all of the supported benchmarks, or a subset specified by the command line, can be run. The rules, such as time measurement (including a repetitive call of the kernels for better clock synchronization), message lengths, selection of communicators to run a particular benchmark (inside the group of all started processes) are program parameters.

PMB has a *standard* and an *optional* configuration. In the standard case, all parameters mentioned above are fixed and must not be changed.

For certain systems, it may be interesting to extend the results tables (in particular, run larger message sizes than provided in the standard case). For this, the user can set certain parameters at own choice. See 5.2.1.

The minimum `P_min` and maximum number `P` of processes can be selected by the user via command line, the benchmarks run on `P_min, 2P_min, 4P_min, ... 2`^{x}`P_min<P` and `P` processes. See chapter 5 for the details.

3.2 The Benchmarks

The current version of PMB-MPI1 contains the benchmarks

- `PingPong`
- `PingPing`
- `Sendrecv`
- `Exchange`
- `Bcast`
- `Allgather`
- `Allgatherv`
- `Alltoall`
- `Reduce`
- `Reduce_scatter`

- Allreduce
- Barrier

The exact definitions will be given in section 4. Section 5 describes the benchmark methodology.

PMB-MPI1 allows for running all benchmarks in more than one process group. E.g., when running PingPong on N≥4 processes, on user request (see 5.1.2.3) N/2 disjoint groups of 2 processes each will be formed, all and simultaneously running PingPong.

Note that these multiple versions have to be carefully distinguished from their standard equivalents. They will be called

- Multi-PingPong
- Multi-PingPing
- Multi-Sendrecv
- Multi-Exchange
- Multi-Bcast
- Multi-Allgather
- Multi-Allgatherv
- Multi-Alltoall
- Multi-Reduce
- Multi-Reduce_scatter
- Multi-Allreduce
- Multi-Barrier

For a distinction, sometimes we will refer to the standard (non Multi) benchmarks as *primary* benchmarks.

The way of interpreting the timings of the Multi-benchmarks is quite easy, given a definition for the primary cases: per group, this is as in the standard case. Finally, the max timing (min throughput) over all groups is displayed. On request, all per group information can be reported, see 5.1.2.3, 6.3.

3.3 Version changes

3.3.1 Version 2.1 vs. 2.0

- Alltoall added (see 4.4.6)
- Optional settings mode included (see 5.2.1.)

3.3.2 Version 2.2 vs. 2.1

Default variable initializations (function set_default) were added (March 2000).

3.4 PMB-MPI1 vs. PMB1.x Definitions

Compared to older PMB1.x releases, all primary benchmark names except Sendrecv and Alltoall are the same. Cshift and Xover of PMB1.x have been removed. PMB1.x did not support the Multi versions.

Most important is that certain *definitions have changed*. Please check carefully.

Table 2 shows an overview of the changes.

3.4.1 Changed Definitions

The main changes are in `PingPing` and `Exchange`. `PingPing` even uses a different pattern (elementary messages rather than `Sendrecv`, see 4.3.1). Moreover, the scaling of timings and throughput data has changed.

	Timings		Throughputs	
	PMB-MPI1	PMB1.x	PMB-MPI1	PMB1.x
`PingPing`	1	0.5	1	2
`Exchange`	1	0.25	4	4

Table 1: Scaling factors PMB-MPI1 vs. PMB1.x

Thus, the corresponding tables show different values when comparing PMB1.x and PMB-MPI1 on a particular system.

The PMB1.x scaling factors for the timings gives confusing answers for the startup components (small message sizes). The scaling of `PingPing` throughputs by 2 is reasonable (bi-directional throughput), however PMB imposes a different interpretation, leaving the bi-directional throughput measurement to the `Sendrecv` benchmark.

`Sendrecv` and `Alltoall` are new benchmarks. Functionally, `Sendrecv` exactly corresponds to the `Cshift` benchmark of PMB1.x, however, displays timings with a different scaling. The `Xover` benchmark of PMB1.x has been removed, as it has shown no significant information on any tested system.

3.4.2 Throughput Calculations

Throughput results are based on *real* MBytes (1048576 bytes) in PMB-MPI1, in contrast to PMB1.x, which used 1 MByte = 1000000 bytes.

In contrast to PMB1.x, PMB-MPI1 does not display throughput values for the global operations `Bcast, Allgather and Allgatherv`.

3.4.3 Corrected Methodology

PMB1.x was not cleanly defined in the case that a certain benchmark was run in a process group strictly smaller than the group of all started MPI processes.

In PMB-MPI1, all non active processes will wait for the active ones in an `MPI_Barrier(MPI_COMM_WORLD)`.

In PMB1.x, non active processes immediately went through the output-collecting phase (`MPI_Gather`) and then, eventually, switched to the following benchmark(s). This may induce unpredictable obstructions of the active processes. The `MPI_Barrier` may also, but now the way is well defined and reasonable.

See 5 for precise definitions of the methodology.

PMB-MPI1 Benchmark name	Contained in releases 1.x	Compared to PMB-1.x, in PMB-MPI1 there is
`PingPong`	×	slight change in throughput data due to re-definition of 1MByte = 1048576 bytes
`PingPing`	×	other pattern, no scaling expectation: timings doubled, throughputs halved
`Sendrecv`		
`Exchange`	×	4 fold timings, equal throughputs
`Bcast`	×	no output of throughput data
`Allgather`	×	no output of throughput data
`Allgatherv`	×	no output of throughput data
`Alltoall`		
`Reduce`	×	no change
`Reduce_scatter`	×	no change
`Allreduce`	×	no change
`Barrier`	×	no change
PMB1.x benchmarks that are no longer in PMB-MPI1		
`Cshift`	`Sendrecv` benchmark is a full substitute	
`Xover`	Has shown no significant information	

Table 2: PMB-MPI1 vs. PMB1.x benchmarks

4 PMB-MPI1 Benchmark Definitions

In this chapter, the single benchmarks are described. Here we focus on the elementary *patterns* of the benchmarks. The methodology of measuring these patterns (message lengths, sample repetition counts, timer, synchronization, number of processes and communicator management, display of results) are defined in chapters 5 and 6.

4.1 Benchmark Classification

For a clear structuring of the set of benchmarks, PMB now introduces classes of benchmarks: *Single Transfer, Parallel Transfer, and Collective*. This classification refers to different ways of interpreting results, and to a structuring of the code itself. It does not actually influence the way of using PMB. Also holds this classification hold for the PMB-MPI2 part [3].

PMB-MPI1		
Single Transfer	Parallel Transfer	Collective
PingPong	Sendrecv	Bcast
PingPing	Exchange	Allgather
		Allgatherv
	Multi-PingPong	Alltoall
	Multi-PingPing	Reduce
	Multi-Sendrecv	Reduce_scatter
	Multi-Exchange	Allreduce
		Barrier
		Multi-versions of these

4.1.1 Single Transfer Benchmarks

The benchmarks in this class are to focus on a *single* message transferred between two processes. As to `PingPong`, this is the usual way of looking at. In PMB interpretation, `PingPing` measures the same as `PingPong`, under the particular circumstance that a message is obstructed by an oncoming one (sent simultaneously by the same process that receives the own one).

Single transfer benchmarks, roughly speaking, are *local mode*. The particular pattern is purely local to the participating processes, there is no concurrency with other message passing activity. Best case message passing results are to be expected. Important for this is that *single transfer benchmarks only run with 2 active processes* (see 3.4.3, 5.2.2 for the definition of *active*).

For `PingPing`, and this is in contrast to PMB1.x and other code systems containing this benchmark, pure timings will be reported, and the throughput is related to a *single* message. Expected numbers, very likely, are between half and full `PingPong` throughput. With this, `PingPing` determines the throughput of messages under non optimal conditions (namely, oncoming traffic).

See 4.2.1 and 4.2.2 for exact definitions.

4.1.2 Parallel Transfer Benchmarks

Benchmarks focusing on *global mode*, say, patterns. The activity at a certain process is in concurrency with other processes, the benchmark measures message passing efficiency under global load.

For the interpretation of `Sendrecv` and `Exchange`, more than 1 message (per sample) counts. As to the throughput numbers, the *total turnover* (the number of *sent plus the number of received bytes*) at a certain process is taken into account. E.g., for the case of 2 processes, `Sendrecv` becomes the *bi-directional* test: perfectly bi-directional systems are rewarded by a double `PingPong` throughput here.

Thus, the throughputs are scaled by certain factors. See 4.3.1 and 4.3.2 for exact definitions. As to the timings, raw results without scaling will be reported.

The `Multi` mode secondarily introduces into this class

- `Multi-PingPong`
- `Multi-PingPing`
- `Multi-Sendrecv`

- Multi-Exchange

4.1.3 Collective Benchmarks

This class contains all benchmarks that are collective in proper MPI convention. Not only is the message passing power of the system relevant here, but also the quality of the implementation.

For simplicity, we also include the Multi versions of these benchmarks into this class.

Raw timings and no throughput are reported.

Note that certain collective benchmarks (namely the reductions) play a particular role as they are not pure message passing tests, but also depend on an efficient implementation of certain numerical operations.

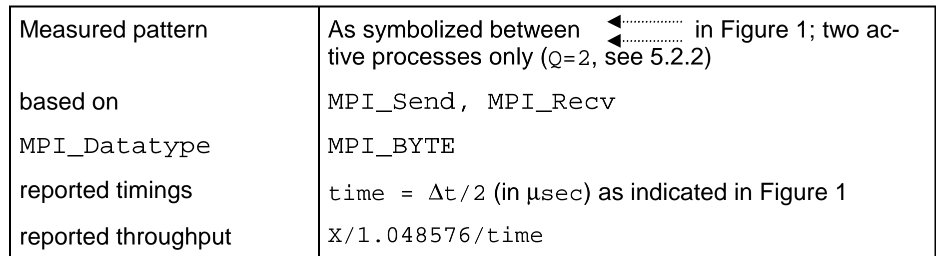
4.2 Definition of Single Transfer Benchmarks

This section describes the single transfer benchmarks in detail. Each benchmark is run with varying message lengths X bytes, and timings are averaged over multiple samples. See 5 for the description of the methodology. Here we describe the view of one single sample, with a fixed message length X bytes. Basic MPI datatype for all messages is MPI_BYTE.

Throughput values are defined in MBytes / sec = 2^{20} bytes / sec scale (i.e. throughput = X / 2^{20} * 10^{6} / time = X / 1.048576 / time, when time is in μsec).

4.2.1 PingPong

PingPong is the classical pattern used for measuring startup and throughput of a single message sent between two processes.



Measured pattern	As symbolized between in Figure 1; two active processes only (Q=2, see 5.2.2)
based on	MPI_Send, MPI_Recv
MPI_Datatype	MPI_BYTE
reported timings	time = Δt/2 (in μsec) as indicated in Figure 1
reported throughput	X/1.048576/time

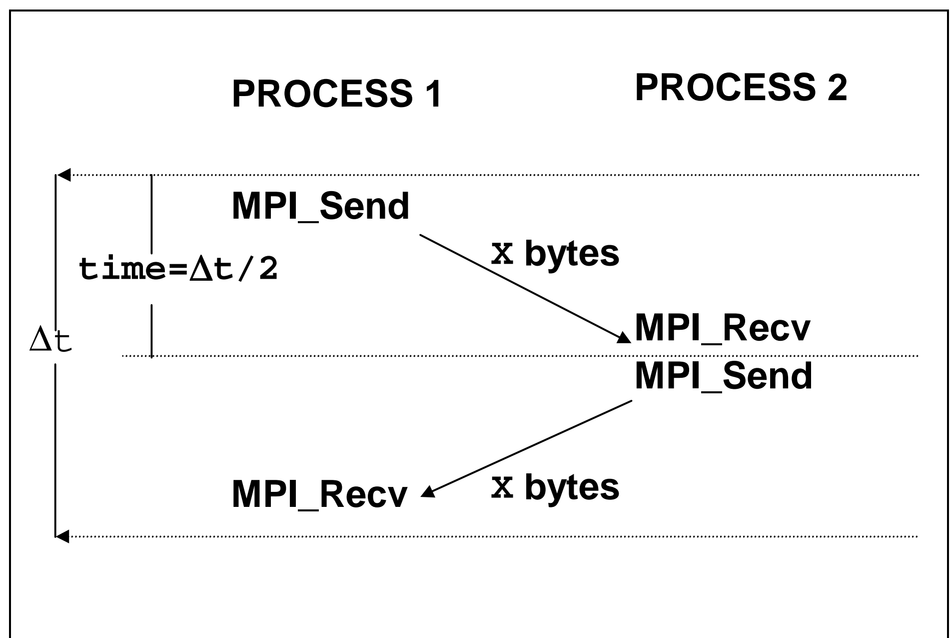
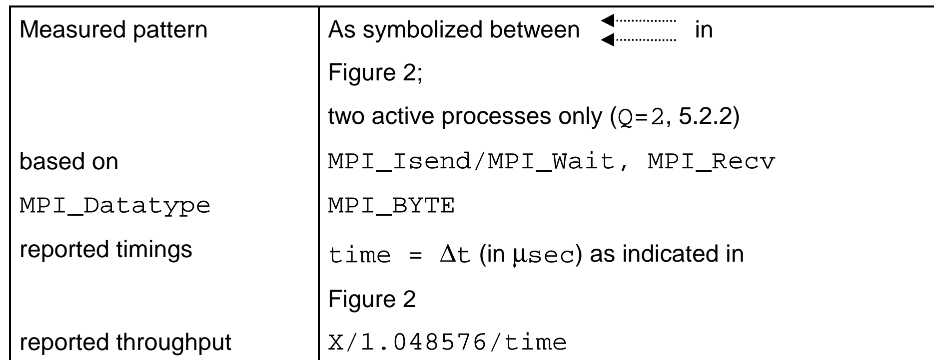


Figure 1: PingPong pattern

4.2.2 `PingPing`

As `PingPong`, `PingPing` measures startup and throughput of single messages, with the crucial difference that messages are obstructed by oncoming messages. For this, two processes communicate (`MPI_Isend/MPI_Recv/MPI_Wait`) with each other, with the `MPI_Isend`'s issued simultaneously.



Measured pattern	As symbolized between ◄·········· ◄·········· in Figure 2; two active processes only (`Q=2`, 5.2.2)
based on	`MPI_Isend/MPI_Wait, MPI_Recv`
`MPI_Datatype`	`MPI_BYTE`
reported timings	`time` = Δ`t` (in μ`sec`) as indicated in Figure 2
reported throughput	`X/1.048576/time`

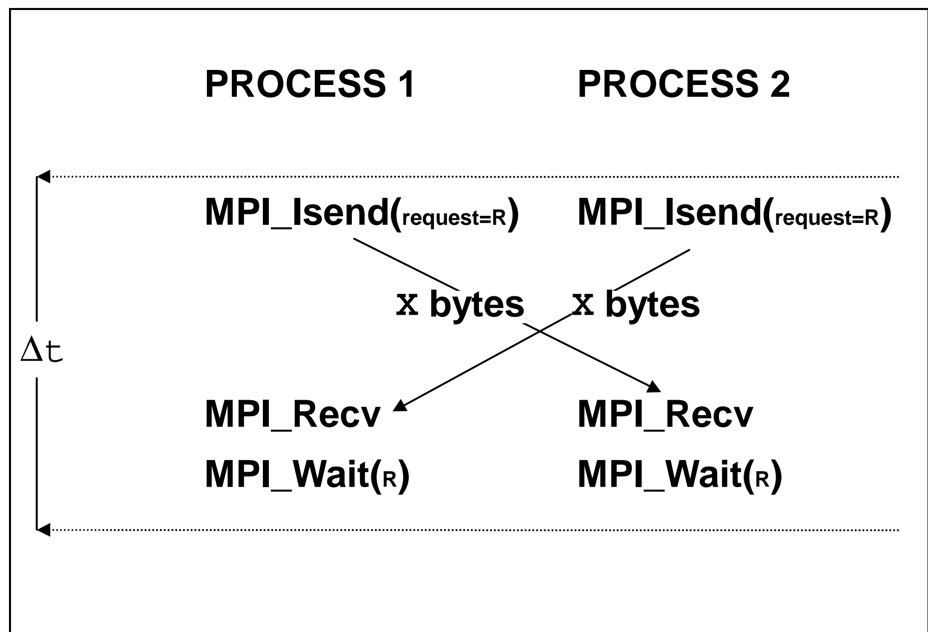


Figure 2: `PingPing` pattern

4.3 Definition of Parallel Transfer Benchmarks

This section describes the parallel transfer benchmarks in detail. Each benchmark is run with varying message lengths `X` bytes, and timings are averaged over multiple samples. See 5 for the description of the methodology. Here we describe the view of one single sample, with a fixed message length `X` bytes. Basic MPI datatype for all messages is `MPI_BYTE`.

The throughput calculations of the benchmarks described here take into account the (per sample) multiplicity `nmsg` of messages outgoing from or incoming at a particular process. In the `Sendrecv` benchmark, a particular process sends and receives `X` bytes, the turnover is `2X bytes`, `nmsg=2`. In the `Exchange` case, we have `4X bytes` turnover, `nmsg=4`.

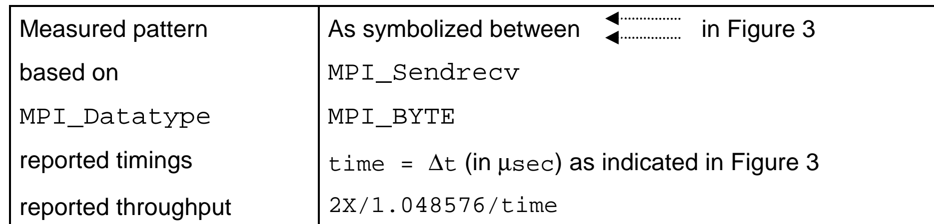
Throughput values are defined in `MBytes/sec` $= 2^{20}$ `bytes / sec` scale (i.e.
`throughput = nmsg*X/`2^{20} `*` 10^6`/time = nmsg*X / 1.048576 / time`, when `time` is in `μsec`).

4.3.1 Sendrecv

Based on MPI_Sendrecv, the processes form a periodic communication chain. Each process sends to the right and receives from the left neighbor in the chain.

The turnover count is 2 messages per sample (1 in, 1 out) for each process.

Sendrecv is equivalent with the Cshift benchmark and, in case of 2 processes, the PingPing benchmark of PMB1.x. For 2 processes, it will report the bi-directional bandwidth of the system, as obtained by the (optimized) MPI_Sendrecv function.



Measured pattern	As symbolized between ◄····· ◄····· in Figure 3
based on	MPI_Sendrecv
MPI_Datatype	MPI_BYTE
reported timings	time = Δt (in μsec) as indicated in Figure 3
reported throughput	2X/1.048576/time

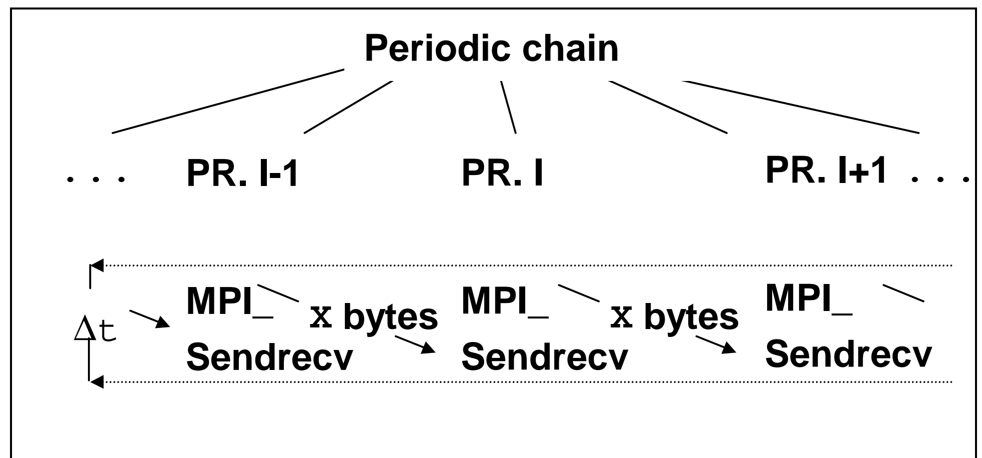
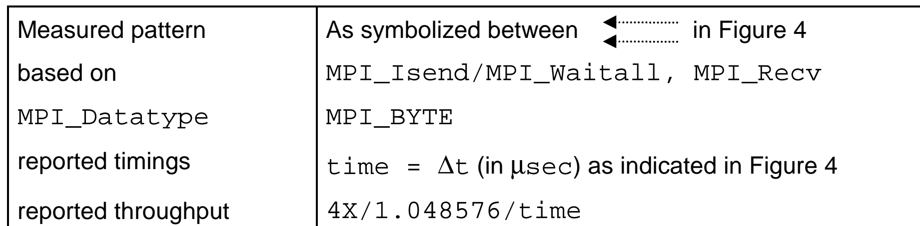


Figure 3: Sendrecv pattern

4.3.2 `Exchange`

`Exchange` is a communications pattern that often occurs in grid splitting algorithms (boundary exchanges). The group of processes is seen as a periodic chain, and each process exchanges data with both left and right neighbor in the chain.

The turnover count is 4 messages per sample (2 in, 2 out) for each process.



Measured pattern	As symbolized between ⇇ in Figure 4
based on	`MPI_Isend/MPI_Waitall, MPI_Recv`
`MPI_Datatype`	`MPI_BYTE`
reported timings	`time` = Δt (in μsec) as indicated in Figure 4
reported throughput	`4X/1.048576/time`

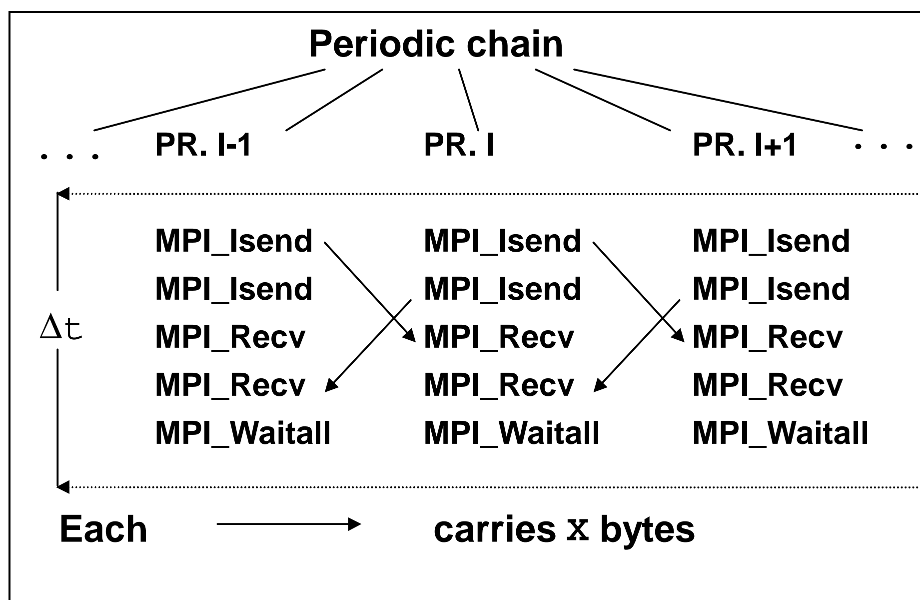


Figure 4: `Exchange` pattern

4.4 Definition of Collective Benchmarks

This section describes the Collective benchmarks in detail. Each benchmark is run with varying message lengths `X` bytes, and timings are averaged over multiple samples. See 5 for the description of the methodology. Here we describe the view of one single sample, with a fixed message length `X` bytes. Basic MPI datatype for all messages is `MPI_BYTE` for the pure data movement functions, and `MPI_FLOAT` for the reductions.

For all Collective benchmarks, only bare timings and no throughput data is displayed.

4.4.1 Reduce

Benchmark of the MPI_Reduce function. Reduces a vector of length L = X/sizeof(float) float items. The MPI datatype is MPI_FLOAT, the MPI operation is MPI_SUM.

The root of the operation is changed cyclically, see 5.2.7

See also the remark in the end of 4.1.3.

measured pattern	MPI_Reduce
MPI_Datatype	MPI_FLOAT
MPI_Op	MPI_SUM
root	changing
reported timings	bare time
reported throughput	none

4.4.2 Reduce_scatter

Benchmark of the MPI_Reduce_scatter function. Reduces a vector of length
L = X/sizeof(float)float items. The MPI datatype is MPI_FLOAT, the MPI operation is MPI_SUM. In the scatter phase, the L items are split as evenly as possible. Exactly, when

np = #processes, L = r*np+s (s = L mod np),

then process with rank i gets r+1 items when i<s, and r items when i≥s.

See also the remark in the end of 4.1.3.

measured pattern	MPI_Reduce_scatter
MPI_Datatype	MPI_FLOAT
MPI_Op	MPI_SUM
reported timings	bare time
reported throughput	none

4.4.3 Allreduce

Benchmark of the MPI_Allreduce function. Reduces a vector of length L = X/sizeof(float) float items. The MPI datatype is MPI_FLOAT, the MPI operation is MPI_SUM.

See also the remark in the end of 4.1.3.

measured pattern	MPI_Allreduce
MPI_Datatype	MPI_FLOAT
MPI_Op	MPI_SUM
reported timings	bare time
reported throughput	none

4.4.4 Allgather

Benchmark of the MPI_Allgather function. Every process inputs X bytes and receives the gathered X*(#processes) bytes.

Measured pattern	MPI_Allgather
MPI_Datatype	MPI_BYTE
reported timings	bare time
reported throughput	none

4.4.5 Allgatherv

Functionally the same as Allgather, however with the MPI_Allgatherv function. Shows whether MPI produces overhead due to the more complicated situation as compared to MPI_Allgather.

Measured pattern	MPI_Allgatherv
MPI_Datatype	MPI_BYTE
reported timings	bare time
reported throughput	none

4.4.6 Alltoall

Benchmark of the MPI_Alltoall function. Every process inputs X*(#processes) bytes (X for each process) and receives X*(#processes) bytes (X from each process).

Measured pattern	MPI_Alltoall
MPI_Datatype	MPI_BYTE
reported timings	bare time
reported throughput	none

4.4.7 Bcast

Benchmark of MPI_Bcast. A root process broadcasts X bytes to all.

The root of the operation is changed cyclically, see 5.2.7.

measured pattern	MPI_Bcast
MPI_Datatype	MPI_BYTE
root	changing
reported timings	bare time
reported throughput	none

4.4.8 **Barrier**

measured pattern	`MPI_Barrier`
reported timings	bare time
reported throughput	none

5 Benchmark Methodology

Recall that in chapter 4 only the underlying patterns of each benchmark have been defined. In this section, the measuring method for those patterns is explained.

Some control mechanisms are hard coded (like the selection of process numbers to run the benchmarks on), some are set by preprocessor parameters in a central include file. Important is that (in contrast to the previous release 2.0) there is a *standard* and an *optional* mode to control PMB. In standard mode, all configurable sizes are predefined and should not be changed. This assures comparability for a result tables in standard mode. In optional mode, the user can set those parameters at own choice. For instance, this mode can be used to extend the results tables as to larger message size.

The following graph shows an overview of the flow of control inside PMB. All *emphasized* items will be explained in more detail.

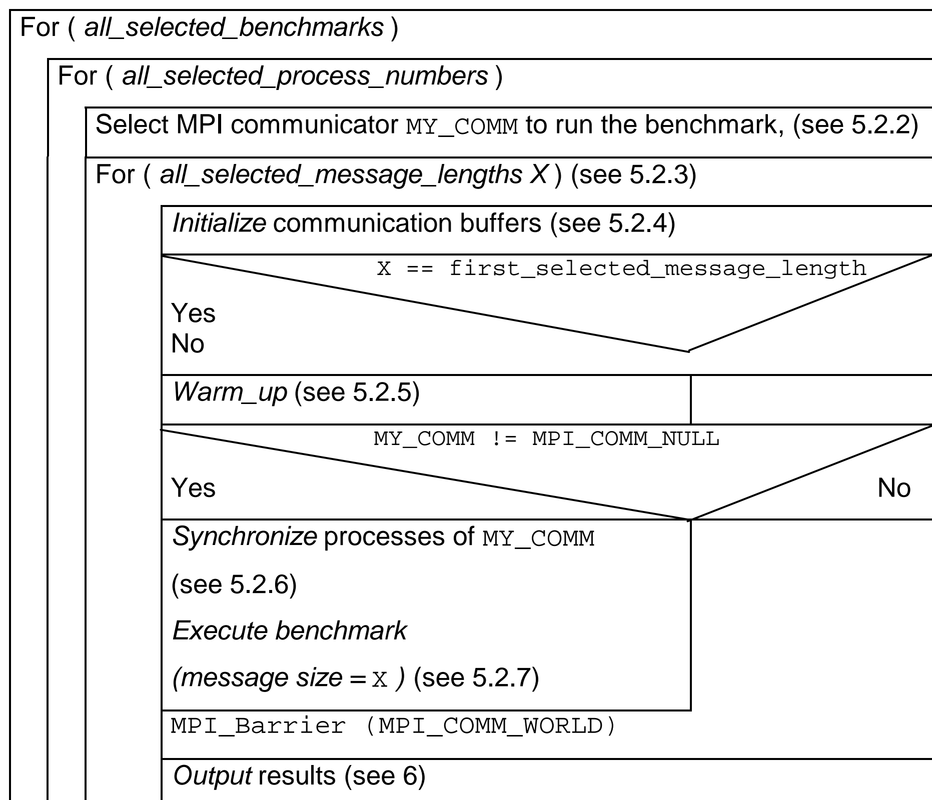


Figure 5: Control flow in PMB

The control parameters obviously necessary are either *command line arguments* (see 5.1) or parameter selections inside the PMB include file `settings.h` (see 5.2).

5.1 Running PMB, Command Line Control

After installation, see 2.2, an executable PMB-MPI1 should exist.

Given `P`, the (normally user selected) number of MPI processes to run PMB-MPI1, a startup procedure has to load parallel PMB-MPI1. Lets assume, for sake of simplicity, that this done by

```
mpirun -np P PMB-MPI1 [arguments]
```

`P=1` is allowed, will be ignored only by Single Transfer benchmarks. Control arguments (in addition to `P`) can be passed to PMB-MPI1 via `(argc,argv)` which will be read by PMB-MPI1 process 0 (in `MPI_COMM_WORLD` ranking) and then distributed to all processes.

5.1.1 Default Case

Just invoke

```
mpirun -np P PMB-MPI1
```

All primary (non `Multi`) benchmarks will run on

`Q=2, 4, 8, ..., largest` 2^x`<P, P processes`

(E.g `P=11`, then `2,4,8,11` processes will be selected). The `Q<=P` processes running the benchmark are called active processes. A communicator is formed out of a group of `Q` processes, see 5.2.2., which is used as communicator argument to the MPI functions crucial for the benchmark.

5.1.2 Command Line Control

The general syntax is

```
mpirun -np P PMB-MPI1
      [Benchmark1 [Benchmark2 [ ... ] ] ]
      [-npmin P_min]
      [-multi Outflag]
      [-input <File>]
```

(where the 4 major [ ] may appear in any order).

Examples:

```
mpirun -np  8 PMB-MPI1

mpirun -np 10 PMB-MPI1 PingPing Reduce

mpirun -np 11 PMB-MPI1 -npmin 5

mpirun -np  4 PMB-MPI1 -npmin 4 -input PMB_SELECT_MPI1

mpirun -np 14 PMB-MPI1 -multi 0 PingPong Barrier
                       -npmin 7
```

5.1.2.1 Benchmark Selection Arguments

A set of blank-separated strings, each being the name of one primary (non `Multi`) PMB benchmark (in exact spelling, case insensitive).

Default (no benchmark selection): select all primary benchmark names.

Given a name selection, either

`-multi` flag is missing,

in which case all selected primary benchmarks are run, or

`-multi` flag is selected,

and then the `Multi-` versions of all selected benchmarks are executed.

5.1.2.2 –npmin Selection

The argument after `-npmin` has to be an integer `P_min`, specifying the minimum number of processes to run all selected benchmarks.

- `P_min` may be 1
- `P_min > P` is handled as `P_min = P`
- *Default* (no `-npmin` selection): as `P_min = 2`

Given P_min, the selected process numbers are

`Q=P_min, 2P_min, 4P_min, ..., largest` 2^x`P_min <P, P.`

Exception: Single Transfer benchmarks will only run on `Q=2` *and ignore* `P_min` *when* `P_min≠2`*.*

Now, running on a subset of `Q<=P` processes means that a communicator with a group of `Q` *active* processes is formed (or eventually several such communicators in the `Multi` cases), see 5.2.2. This communicator is used as argument to the MPI functions crucial for the benchmark.

5.1.2.3 –multi Outflag Selection

`-multi` activates the `Multi` versions of the benchmarks. The argument after `-multi` has to be an integer `Outflag`, either 0 or 1. This flag just controls the way of displaying results.

- `Outflag = 0`: only display max timings (min throughputs) over all active groups
- `Outflag = 1`: report on all groups separately (may become longish)
- *Default* (no `-multi` selection): run primary (non `Multi`) versions.

See also 6.2, 6.3.

5.1.2.4 `-input` <File> Selection

An ASCII input file is used to select the benchmarks to run, e.g. a file `PMB_SELECT_MPI1` looking as follows:

```
#
# PMB benchmark selection file
#
# every line must be a comment (beginning with #), or it
# must contain exactly 1 PMB benchmark name
#
# PingPong
PingPing
Allreduce
# Alltoall
```

```
mpirun .... PMB-MPI1 -input PMB_SELECT_MPI1
```

would run benchmarks `PingPing` and `Allreduce`.

5.2 PMB Parameters and Hard Coded Settings

5.2.1 Parameters Controlling PMB

There are 9 parameters (set by preprocessor definition) controlling PMB. The definition is the files `settings.h` (PMB-MPI1, PMB-EXT) and `settings_io.h` (PMB-IO).

A complete list and explanation of the parameters is in Figure 6 below.

Only `settings.h` is relevant here. It is important that (in contrast to PMB 2.0) PMB 2.2 allows for two sets of parameters: *standard* and *optional.*

Parameter (standard mode value)	Meaning
PMB_OPTIONAL (not set)	has to be set when user optional settings are to be activated
MINMSGLOG (0)	second smallest data transfer size is max(unit,$2^{MINMSGLOG}$) (the smallest always being 0), where unit = sizeof(float) for reductions, unit = 1 else
MAXMSGLOG (22)	largest message size is $2^{MAXMSGLOG}$ Sizes 0, 2^i (i=MINMSGLOG,..,MAXMSGLOG) are used
MSGSPERSAMPLE (1000)	max. repetition count for all PMB-MPI1 benchmarks
MSGS_NONAGGR (100)	max. repetition count for non aggregate benchmarks (cf. [3], irrelevant for PMB-MPI1)
OVERALL_VOL (40 MBytes)	for all sizes < OVERALL_VOL, the repetition count is eventually reduced so that not more than OVERALL_VOL bytes overall are processed. This avoids unnecessary repetitions for large message sizes. Finally, the real repetition count for message size X is MSGSPERSAMPLE (X=0), min(MSGSPERSAMPLE,max(1,OVERALL_VOL/X)) (X>0) NOTE: OVERALL_VOL does *not* restrict the size of the max. data transfer. $2^{MAXMSGLOG}$ is the largest size, independent of OVERALL_VOL
N_WARMUP (2)	Number of *Warmup* sweeps (see 5.2.5)
N_BARR (2)	Number of MPI_Barrier for synchronization (see 5.2.6)
TARGET_CPU_SECS (0.01)	CPU seconds (as float) to run concurrent with nonblocking benchmarks (currently irrelevant for PMB-MPI1)

Figure 6: PMB parameters

Figure 7 below shows a sample of file settings.h. Here, PMB_OPTIONAL is set, so that user defined parameters are used. Message sizes 8 and 16 MBytes are selected, extending the standard mode tables.

If PMB_OPTIONAL is deactivated, the obvious standard mode values are taken.

Note:

PMB has to be re-compiled after a change of settings.h*.*

```
#define PMB_OPTIONAL
#ifdef PMB_OPTIONAL

#define MINMSGLOG 23
#define MAXMSGLOG 24
/* etc as below */
#else
/*
DON'T change anything below here !!
*/
#define MINMSGLOG 0
#define MAXMSGLOG 22
#define MSGSPERSAMPLE 1000
#define MSGS_NONAGGR   100
#define OVERALL_VOL 40*1048576
#define N_WARMUP 2
#define N_BARR   2
#define TARGET_CPU_SECS 0.01

#endif
```

Figure 7: file settings.h

5.2.2 Communicators, Active Processes

Communicator management is repeated in every select `MY_COMM` step in Figure 5. If exists, the previous communicator is freed.

Given `Q≤P` as in 5.1.2.2, subcommunicators are formed out of the groups consisting of the `MPI_COMM_WORLD` ranks

`{0,..,Q-1}` (non `Multi` case),

`{0,..,Q-1}, {Q,..,2Q-1} ...` (`Multi` case).

All processes belonging to such a group are called *active* processes, the corresponding communicator is called `MY_COMM` in Figure 5. It is used as communicator argument to the MPI functions defining the pattern.

All non active processes get `MY_COMM=MPI_COMM_NULL`.

5.2.3 Message Lengths

Set in `settings.h`, see 5.2.1

5.2.4 Buffer Initialization

Communication buffers are dynamically allocated as `void`* and used as `MPI_BYTE` buffers for all (non reduction) benchmarks. See 7.1 for an estimate of the memory requirement. To assign the buffer contents, a cast to an assignment type is performed. On the one hand, a sensible datatype is mandatory for reduction benchmarks. On the other hand, this facilitates results checking which may become necessary eventually (see 7.3).

PMB sets the buffer assignment type by `typedef assign_type` in `settings.h`. Currently, `float` is selected for PMB-MPI1 (as this is sensible for reductions). The values are set by a CPP macro, currently

```
#define BUF_VALUE(rank,i) (0.1*((rank)+1)+(float)(i)
```

In each initialization, communication buffers are seen as typed arrays and initialized as to

```
((assign_type*)buffer)[i] = BUF_VALUE(rank,i);
```

where rank is the MPI rank of the calling process.

5.2.5 Warm-up Phase

Before starting the actual benchmark measurement, the selected benchmark is executed `N_WARMUP` (defined in `settings.h`, see 5.2.1) times with the maximum message length. This is to hide eventual initialization overheads of the message passing system.

5.2.6 Synchronization

Before the actual benchmark, `N_BARR` (defined in `settings.h`, see 5.2.1) many `MPI_Barrier(MY_COMM)` (ref. Figure 5) are used for process synchronization.

5.2.7 The Actual Benchmark

In order to reduce inaccuracies due to insufficient clock resolutions, every benchmark is run repeatedly. The repetition count is `MSGSPERSAMPLE` (constant defined in `settings.h`, see 5.2.1). In order to avoid an excessive run time in case of large message lengths `X`, an upper bound is set to `OVERALL_VOL / X` (`OVERALL_VOL` defined in `settings. H`). Finally,

```
n_sample = MSGSPERSAMPLE (X=0)

n_sample = max(1,min(MSGSPERSAMPLE,OVERALL_VOL/X)) (X>0)
```

is the repetition count for all benchmarks, given message size `X`. Now, the key measurement is performed according to

```
for ( i=0; i<N_BARR; i++ ) MPI_Barrier(MY_COMM)
time = MPI_Wtime()
for ( i=0; i<n_sample; i++ )
      execute MPI pattern
time = (MPI_Wtime()-time)/n_sample
```

Important to stress is that *execute MPI pattern* really means the pure pattern as specified in 4, without any further function call. (`Bcast` and `Reduce` need a root process for their operation. In both cases, the root process is changed so that in iteration `i` the root rank is `i%(#processes in group)`. This is an additional integer operation inside the loop, looked upon as negligible.)

The communicator argument to the `MPI_XX` functions constituting the pattern is as defined in 5.2.2.

6 Output

Most easily, output is explained by sample outputs, see the tables below (generated with the previous version 2.1, looking identical with 2.2). What one sees is the following.

- *General information*
  Machine, System, Release, Version are obtained by the code `g_info.c`:

```
#include <sys/utsname.h>
void   make_sys_info()
{
struct utsname info;
int err;
err = uname( &info );

fprintf(unit, "# Machine    : %s"  ,info.machine);
fprintf(unit, "# System     : %s\n",info.sysname);
fprintf(unit, "# Release    : %s\n",info.release);
fprintf(unit, "# Version    : %s\n",info.version);
}
```

- *Non multi case numbers*
  After a benchmark, 3 time values are available: `Tmax, Tmin, Tavg`, the maximum, minimum and average time, resp., extended over the group of active processes. Time unit is `μsec`.
  Single Transfer Benchmarks:
  Display `X = message size [bytes], T=Tmax[μsec], bandwidth = X / 1.048576 / T`
  Parallel Transfer Benchmarks:
  Display `X = message size, Tmax, Tmin and Tavg`, `bandwidth` based on `time = Tmax`
  Collective Benchmarks:
  Display `X = message size` (except for `Barrier`), `Tmax, Tmin` and `Tavg`
- *Multi case numbers*
  `-multi 0`: the same as above, with `max, min, avg` over all groups.
  `-multi 1`: the same for all groups, `max, min, avg` over single groups.

6.1 Sample 1

```
mpirun -np 2 PMB-MPI1 PingPong Allreduce
#---------------------------------------------------
#    PALLAS MPI Benchmark Suite V2.1, MPI-1 part
#---------------------------------------------------
# Date       : Thu Sep 10 10:22:58 1998
# Machine    : alpha# System     : OSF1
# Release    : V4.0
# Version    : 564

#
# Minimum message length in bytes:   0
# Maximum message length in bytes:   4194304
#
# MPI_Datatype                    :   MPI_BYTE
# MPI_Datatype for reductions     :   MPI_FLOAT
# MPI_Op                          :   MPI_SUM
#
#
# List of Benchmarks to run:

# PingPong
# Allreduce

#---------------------------------------------------
# Benchmarking PingPong
# ( #processes = 2 )
#---------------------------------------------------
       #bytes #repetitions      t[usec]   Mbytes/sec
            0         1000         4.51         0.00
            1         1000         5.41         0.18
            2         1000         5.41         0.35
            4         1000         5.41         0.70
            8         1000         5.41         1.41
           16         1000         5.00         3.05
           32         1000         5.84         5.23
           64         1000         8.34         7.32
          128         1000         8.76        13.94
          256         1000         9.59        25.46
          512         1000        12.51        39.03
         1024         1000        18.35        53.22
         2048         1000        30.44        64.16
         4096         1000        57.55        67.88
```

```
       8192         1000       105.03        74.38
      16384         1000       226.09        69.11
      32768         1000       407.32        76.72
      65536          640       793.45        78.77
     131072          320      1564.79        79.88
     262144          160      3089.89        80.91
     524288           80      6224.01        80.33
    1048576           40     13533.05        73.89
    2097152           20     29867.15        66.96
    4194304           10     64889.35        61.64

#-----------------------------------------------------
# Benchmarking Allreduce
# ( #processes = 2 )
#-----------------------------------------------------
 #bytes #repetitions  t_min[usec]  t_max[usec]   t_avg[usec]
      0         1000         1.43         1.46         1.45
      4         1000        25.85        25.85        25.85
      8         1000        26.69        26.69        26.69
     16         1000        26.69        26.69        26.69
     32         1000        27.52        27.52        27.52
     64         1000        31.69        31.69        31.69
    128         1000        32.53        32.53        32.53
    256         1000        35.86        35.86        35.86
    512         1000        42.53        42.53        42.53
   1024         1000        62.55        62.55        62.55
   2048         1000        98.25        98.25        98.25
   4096         1000       180.54       180.54       180.54
   8192         1000       309.07       309.07       309.07
  16384         1000       630.30       630.30       630.30
  32768         1000      1180.77      1180.77      1180.77
  65536          640      2571.14      2571.14      2571.14
 131072          320      4929.39      4929.39      4929.39
 262144          160      9909.67      9909.67      9909.67
 524288           80     20586.26     20596.66     20591.46
1048576           40     46326.08     46326.08     46326.08
2097152           20    105728.35    105728.35    105728.35
4194304           10    235253.11    235253.11    235253.11
```

6.2 Sample 2

`mpirun -np 7 PMB-MPI1 reduce -npmin 3 -multi 0`

(`PMB_OPTIONAL` mode)

```
#---------------------------------------------------
#    PALLAS MPI Benchmark Suite V2.1, MPI-1 part
#---------------------------------------------------
# Date       : Thu Sep 10 10:30:49 1998
# Machine    : alpha# System     : OSF1
# Release    : V4.0
# Version    : 564

#
# Minimum message length in bytes:   0
# Maximum message length in bytes:   1024
#
# MPI_Datatype                   :   MPI_BYTE
# MPI_Datatype for reductions    :   MPI_FLOAT
# MPI_Op                         :   MPI_SUM
#
#
# !! Attention: results have been achieved in
# !! PMB_OPTIONAL mode.
# !! Results may differ from standard case.
#
#

# List of Benchmarks to run:

# (Multi-)Reduce

#-------------------------------------------------------------
# Benchmarking Multi-Reduce
# ( 2 groups of 3 processes each running simultaneous )
# Group    0                     : 0 1 2
# Group    1                     : 3 4 5
# ( 1 additional process waiting in MPI_Barrier)
#-------------------------------------------------------------
 #bytes #repetitions  t_min[usec]  t_max[usec]  t_avg[usec]
      0         1000         0.64         0.83         0.77
      4         1000       142.84       145.34       143.54
      8         1000       141.61       142.44       141.75
     16         1000       141.61       142.44       142.03
     32         1000       142.44       143.28       142.72
     64         1000       161.66       161.66       161.66
```

```
     128         1000       164.30       164.30       164.30
     256         1000       173.47       174.31       174.03
     512         1000       197.22       198.89       198.06
    1024         1000       246.03       246.03       246.03

#-------------------------------------------------------------
# Benchmarking Reduce
# ( #processes = 6 )
# ( 1 additional process waiting in MPI_Barrier)
#-------------------------------------------------------------
 #bytes #repetitions  t_min[usec]  t_max[usec]  t_avg[usec]
       0         1000         0.64         0.83         0.78
       4         1000       246.64       247.47       246.92
       8         1000       232.96       233.79       233.65
      16         1000       188.43       189.26       188.57
      32         1000       169.73       169.73       169.73
      64         1000       195.52       196.35       195.66
     128         1000       197.30       198.14       197.72
     256         1000       204.92       205.75       205.61
     512         1000       218.65       220.31       219.62
    1024         1000       266.56       266.56       266.56

#-------------------------------------------------------------
# Benchmarking Reduce
# ( #processes = 7 )
#-------------------------------------------------------------
 #bytes #repetitions  t_min[usec]  t_max[usec]  t_avg[usec]
       0         1000         0.64         0.83         0.78
       4         1000       223.25       224.08       223.61
       8         1000       209.92       210.75       210.39
      16         1000       208.65       209.48       209.13
      32         1000       211.58       212.41       211.82
      64         1000       235.74       235.74       235.74
     128         1000       238.39       239.23       238.51
     256         1000       250.20       251.03       250.68
     512         1000       279.79       279.79       279.79
    1024         1000       331.10       332.77       331.69
```

6.3 Sample 3

`mpirun -np 5 PMB-MPI1 pingping -multi 1`

(`PMB_OPTIONAL` mode)

```
#---------------------------------------------------
#    PALLAS MPI Benchmark Suite V2.1, MPI-1 part
#---------------------------------------------------
# Date        : Thu Sep 10 10:37:46 1998
# Machine     : alpha# System     : OSF1
# Release     : V4.0
# Version     : 564

#
# Minimum message length in bytes:   0
# Maximum message length in bytes:   256
#
# MPI_Datatype                    :   MPI_BYTE
# MPI_Datatype for reductions     :   MPI_FLOAT
# MPI_Op                          :   MPI_SUM
#
#

# !! Attention: results have been achieved in
# !! PMB_OPTIONAL mode.
# !! Results may differ from standard case.
#
#

# List of Benchmarks to run:

# (Multi-)PingPing
```

```
#-----------------------------------------------------------
# Benchmarking Multi-PingPing
# ( 2 groups of 2 processes each running simultaneous )
# Group    0                    : 0 1
# Group    1                    : 2 3
# ( 1 additional process waiting in MPI_Barrier)

#-----------------------------------------------------------
  Group        #bytes #repetitions      t[usec]   Mbytes/sec
      0             0         1000         9.98         0.00
      1             0         1000        78.21         0.00

      0             1         1000         9.98         0.10
      1             1         1000        70.59         0.01

      0             2         1000         9.15         0.21
      1             2         1000        71.40         0.03

      0             4         1000         9.15         0.42
      1             4         1000        70.74         0.05

      0             8         1000         9.17         0.83
      1             8         1000        70.69         0.11

      0            16         1000         9.16         1.67
      1            16         1000        70.80         0.22

      0            32         1000        10.00         3.05
      1            32         1000        71.53         0.43

      0            64         1000        17.49         3.49
      1            64         1000        86.63         0.70

      0           128         1000        19.99         6.11
      1           128         1000        89.02         1.37

      0           256         1000        22.06        11.07
      1           256         1000        92.74         2.63
```

7 Further details

7.1 Memory Requirements

Benchmarks	Standard mode memory demand per process (Q active processes)	Optional mode memory demand per process (X = $2^{MAXMSGLOG}$)
Alltoall	Q × 8 MBytes	Q × 2X bytes
Allgather, Allgatherv	(Q+1) × 4 MBytes	(Q+1) × X bytes
All other benchmarks	8 MBytes	2X bytes

Table 3 : Memory Requirements

7.2 SRC Directory

The following source files are on the directory:

PMB-MPI1 benchmark kernels:

PingPing.c, PingPong.c, Exchange.c, Sendrecv.c, Allgather.c, Allgatherv.c, Allreduce.c, Alltoall.c, Bcast.c, Reduce.c, Reduce_scatter.c, Barrier.c

PMB-MPI2 benchmark kernels (irrelevant here):

Window.c, OneS_accu, OneS_bidir.c, OneS_unidir.c
Write.c, Read.c, Open_Close.c

Driver routines:

pmb.c, pmb_init.c, Output.c, BenchList.c, Warm_Up.c, declare.c, g_info.c, Err_Handler.c, strgs.c, Mem_Manager.c, chk_diff.c, Parse_Name_EXT.c, Parse_Name_IO.c, Parse_Name_MPI1.c

Init_File.c, Init_Transfer.c, User_Set_Info.c, CPU_Exploit.c

Include files:

Benchmark.h, Comments.h, appl_errors.h, comm_info.h, declare.h, err_check.h, settings.h, settings_io.h, Bnames_EXT.h, Bnames_IO.h, Bnames_MPI1.h

7.3 Results Checking

By activating the `cpp` flag `-DCHECK` through the `CPPFLAGS` variable (see 2.2), and recompiling, at PMB runtime every message passing result will be checked against the expected outcome (note that the contents of each buffer is well defined, see 5.2.4). Output tables will contain an additional column displaying the diffs as floats (named *defects*).

Attention: `-DCHECK` results are not valid as real benchmark data! Don't forget to deactivate `DCHECK` and recompile in order to get proper results.

7.4 Use of MPI

Except ist documented use in the benchmark kernels, MPI is used to the following extent in PMB-MPI1:

`MPI_Init`

`MPI_Bcast, MPI_Recv, MPI_Get_count, MPI_Send, MPI_Gather`

`MPI_Comm_size, MPI_Comm_rank, MPI_Comm_group,`

`MPI_Group_translate_ranks, MPI_Comm_split, MPI_Comm_free`

`MPI_Error_string, MPI_Errhandler_create, MPI_Errhandler_set,`

`MPI_Errhandler_free, MPI_Abort`

`MPI_Finalize`

8 Revision History

Release No.	Date	Content	Related Software Releases
1.0	1997/06	draft documentation	PMB 1.0, 1.1, 1.2, 1.3
2.0	1998/06	complete definition	PMB 2.0
2.1	1998/09	4.4.6 added 5.2.1 added Minor textual changes	PMB 2.1
2.2	2000/03	3.3 updated	PMB 2.2

9 References

1 MPI: A Message-Passing Interface Standard. Message Passing Interface Forum, 1995

2 MPI-2: Extensions to the Message-Passing Interface. Message Passing Interface Forum, 1997

3 Pallas MPI Benchmarks - PMB, part MPI-2