A Quick Guide for the **pbdPROF** Package

Wei-Chen Chen[1], Drew Schmidt[2], Gaurav Sehrawat[3], Pragneshkumar Patel[4],
George Ostrouchov[4,5]

[1]pbdR Core Team

[2]Business Analytics and Statistics,
University of Tennessee,
Knoxville, TN, USA

[3]Jaypee Institute of Information Technology
Uttar Pradesh, India

[4]National Institute for Computational Sciences,
University of Tennessee,
Knoxville, TN, USA

[5]Computer Science and Mathematics Division,
Oak Ridge National Laboratory,
Oak Ridge, TN, USA

September 21, 2016

Contents

i

The findings and conclusions in this article have not been formally disseminated by the U.S. Department of Energy and should not be construed to represent any determination or policy of University, Agency, and National Laboratory.

This publication was typeset using LaTeX.

Acknowledgement

Part I

Installation

This document is written to explain the main functions of **pbdPROF** (Chen et al., 2013), version 0.2-0. Every effort will be made to ensure future versions are consistent with these instructions, but features in later versions may not be explained in this document.

Information about the functionality of this package, and any changes in future versions can be found on website: "Programming with Big Data in R" at http://r-pbd.org/.

1 Introduction

The goal of **pbdPROF** is to utilize external MPI profiling libraries to profile parallel R code and understand hidden MPI communications between processors. The number of communications, sizes of messages, times, and types of functions calls all affect program performance, and so having these measurements can greatly aid in debugging and algorithm design.

An MPI profiling libraries is able to hijack calls to MPI functions and then capture the profiling information (such as that described above), all without disturbing the execution of the original program.

The current main features of **pbdPROF** include:

1. the support of several profiling libraries

2. provide linking information to pbdR (Ostrouchov et al., 2012) and other MPI-using R packages

3. output profiling information associated with MPI calls

4. parse and summarize profiling information

1.1 Supported MPI Profilers

As of version 0.2-0 of **pbdPROF**, the officially supported MPI profilers are

- **fpmpi** (Gropp, 2000), and

- **mpiP** (Vetter and McCracken, 2001)

with plans to eventually support additional profilers, including **TAU** (Shende and Malony, 2006).

1.2 Choice of Profiler

The **pbdPROF** package currently uses the **fpmpi** library by default. More explicitly, a source copy of **fpmpi** is located at pbdPROF/src/fpmpi of the **pbdPROF** source. Although we bundle **pbdPROF** with **fpmpi**, it is not the best MPI profiler (though it may be sufficient for your needs). The results from other libraries, such as **mpiP**, are much more thorough and may lead to much deeper insights. Additionally, **fpmpi** does not handle profiler output file naming nearly as well as the others (see Section 3). However, **fpmpi** is the easiest to install.

If **fpmpi** is used, a static library will be built and placed in `pbdPROF/lib/libfpmpi.a` of the **pbdPROF** install directory. However, external profiling libraries such as **mpiP**, **TAU**, or even **fpmpi** can be also linked with **pbdPROF** by passing a suitable `--configure-args` argument during an installation via R `CMD INSTALL`. We will explain this procedure in depth in Section 2.3 using an external **fpmpi** and **mpiP** as an example, **TAU** will be added in next release.

While it is possible to link with other profiling libraries, at the time of writing (for version 0.2-0), we only support **fpmpi** and **mpiP**. We anticipate full of **TAU** for the next version of this package.

2   Installation

In this section, we will describe the various ways that one can build **pbdPROF** and link it with MPI-using R packages. For installation troubleshooting, see Appendix A.

2.1   System Requirements

The **pbdPROF** package requires an MPI installation, such as OpenMPI or MS-MPI. Additionally, the package is basically useless without some kind of MPI-using R package, such as **pbdMPI** (Chen et al., 2012a) or **Rmpi** (Yu, 2002). For information regarding how to install MPI or **pbdMPI**, please see the **pbdMPI** vignette (Chen et al., 2012b) or the pbdR website `http://r-pbd.org/install`.

2.2   The Big Picture

Before pressing on, let us stop to take a moment and understand the "big picture" here. The following sections will contain *more than sufficient* detail, to the point where it would be easy to lose sight of the proverbial forest for the trees.

For the remainder of this document, we will be providing information for two fairly distinct groups of people: R-level MPI package developers, and C/Fortran-level MPI package developers. If you are in the former category, then the use of this package is a bit simpler for you. All you need to do is get **pbdPROF** installed and reinstall your MPI-using package of choice (**pbdMPI**, **Rmpi**, etc. …). Each package that directly uses MPI (packages produced by developers in the latter category) will have to explicitly support **pbdPROF** (or the reader will have to get his/her hands dirty in another developer's makefiles — an unpleasant business). It is worth nothing here that there are instructions in this document for how a developer of the second kind could explicitly add **pbdPROF** support to his/her package.

So why the need to reinstall things? It boils down to how the profilers actually work. Under normal circumstances, a user writes some R code from an MPI-using package (e.g., `allreduce(x)` from **pbdMPI**, `mpi.allreduce(x, type=2)` from **Rmpi**, etc. …).

This then makes a call to some C or Fortran code which directly interfaces with MPI. You can see this pictures in Figure 1a. When you use a profiler, you instead hijack the calls to MPI from the C/Fortran code so that some metadata can be stored about MPI usage.

This process is represented in Figure 1b. Hopefully it should be clear what, and when, something should be reinstalled. For the sake of completion, we summarize the possibilities below:

To *enable* MPI profiling:

1. install **pbdPROF**

(a) Without a Profiler   (b) With the Profiler
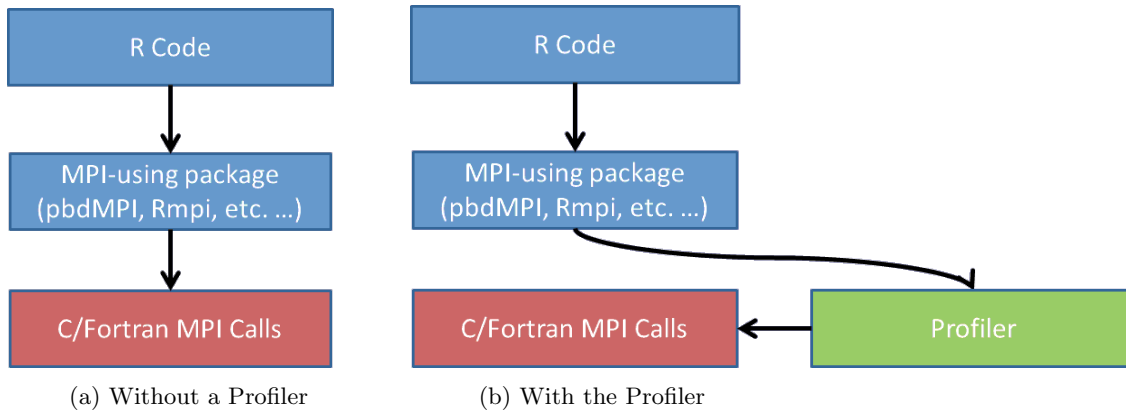
2. reinstall an MPI-using package and link it with **pbdPROF**

3. write and execute your MPI-using R code as normal

4. use the **pbdPROF** utilities `read.prof()`, etc. for interpreting profiling results

To *disable* MPI profiling:

1. reinstall any MPI-using package that was linked it with **pbdPROF**, and this time *do not* link with **pbdPROF**

2.3   Installing pbdPROF with fpmpi

We can install **pbdPROF** using the internal **fpmpi** library via

<div align="center">Shell Command</div>

```
R CMD INSTALL pbdPROF_0.1-0.tar.gz
```

By default, this compiles `pbdPROF/src/fpmpi/*` of the **pbdPROF** source, generates a static library `libfpmpi.a`, and installs the library to `pbdPROF/lib/` of the **pbdPROF** install. No shared library is generated or needed, so the directory `pbdPROF/libs/` is empty, i.e., there is no need to build `pbdPROF.so`. The linking argument is saved in `Makeconf` and installed to `pbdPROF/etc/` for later use by other packages, such as **pbdMPI** or **Rmpi**.

However, if we choose, we can link with an external **fpmpi** library, via

<div align="center">Shell Command</div>

```
R CMD INSTALL pbdPROF_0.1-0.tar.gz \
  --configure-args="--with-fpmpi='/path_to_fpmpi/lib/libfpmpi.a'"
```

or

<div align="center">Shell Command</div>

```
R CMD INSTALL pbdPROF_0.1-0.tar.gz \
  --configure-args="--with-fpmpi='-L/path_to_fpmpi/lib -lfpmpi'"
```

Or the conventional method in R console

<div align="center">Shell Command</div>

```
install.packages("pbdPROF",
    configure.args=c("--with-fpmpi=/path/to/your/fpmpi/lib/libfpmpi.a"))
```

Or

Shell Command

```
install.packages("pbdPROF",
    configure.args=c("--with-fpmpi=-L/path/to/your/fpmpi/lib -lfpmpi"))
```

Since **fpmpi** only builds a static library `libfpmpi.a`, there is no difference between these two installations of **pbdPROF**. This only provides the linking arguments, either `/path_to_fpmpi/lib/libfpmpi.a` or `-L/path_to_fpmpi/lib -lfpmpi`, which is saved in `Makeconf` and installed to `pbdPROF/etc/` for later use by other packages, such as **pbdMPI** or **Rmpi**.

2.3.1 Linking pbdMPI with pbdPROF

Reinstall **pbdMPI** via

Shell Command

```
R CMD INSTALL pbdMPI_1.0-0.tar.gz --configure-args="--enable-pbdPROF"
```

Package developers who are directly interfacing with MPI functions (via C or Fortran) should note that `pbdMPI/R/get_conf.r` and `pbdMPI/R/get_lib.r` are utilized in `pbdMPI/configure.ac` (used to generate `pbdMPI/configure`) to determine an appropriate linking flag `PROF_LDFLAGS` based on preset flags in `pbdPROF/etc/Makeconf`.

If the internal library is used in **pbdPROF**, then the path to `pbdPROF/lib/libfpmpi.a` is set in the flag `PKG_LIBS` of `pbdMPI/src/Makevars.in`. If the external library is used in **pbdPROF**, then the linking arguments either `/path_to_fpmpi/lib/libfpmpi.a` or `-L/path_to_fpmpi/lib -lfpmpi` is set in the flag `PKG_LIBS` of `pbdMPI/src/Makevars.in`. Therefore, the **pbdMPI** can be intercepted by the **fpmpi** library when MPI function calls are evoked.

No mater which library is used, internal or external, the `PROF_LDFLAGS` in `pbdMPI/etc/Makefile` provides the linking information to the profiling library. It is also used in `PKG_LIBS`, which will be exported to other **pbdR** packages at installation via the flag `SPMD_LDFLAGS`. Therefore there is no need for additional flags in `R CMD INSTALL` when reinstalling packages for profiling.

2.3.2 Linking pbdBASE with pbdPROF

For further profiling, such as **pbdBASE** (Schmidt et al., 2012), one may reinstall the package, via

Shell Command

```
R CMD INSTALL pbdBASE_0.2-2.tar.gz
```

There is no need to provide any flag since **pbdMPI/etc/Makefile** has the information and installation of **pbdBASE** already considers it. Note that since both packages (**pbdMPI** and **pbdBASE**) have MPI-using C/Fortran functions involved, it is necessary to link with **pbdPROF** in order to profile communications evoked by the package.

2.3.3   Linking Rmpi with pbdPROF

Reinstall **Rmpi** via

<div align="center">Shell Command</div>

```
wget https://github.com/snoweye/Rmpi_PROF/archive/master.zip
unzip master.zip
mv Rmpi_PROF-master Rmpi
find ./Rmpi -type f -perm 777 -print -exec chmod 644 {} \;
find ./Rmpi -type d -perm 777 -print -exec chmod 755 {} \;
chmod 755 ./Rmpi/configure
chmod 755 ./Rmpi/cleanup
chmod 755 ./Rmpi/inst/*.sh
R CMD build --no-resave-data Rmpi
R CMD INSTALL Rmpi_0.6-6.tar.gz --configure-args="--enable-pbdPROF"
```

Note that 0.6-6 is not an official release of **Rmpi**. It is a modified version of 0.6-3 and it is currently available at https://github.com/snoweye/Rmpi_PROF. The authors of **Rmpi** have plans to eventually incorporate these changes, but this can be used as a temporary measure.

2.4   Installing pbdPROF with mpiP

We have to install **mpiP** externally from its source code to use it in **pbdPROF**. We can install **pbdPROF** using the external **mpiP** library via

<div align="center">Shell Command</div>

```
R CMD INSTALL pbdPROF_0.2-0.tar.gz
    --configure-args="--with-mpiP='/path/to/your/mpiP/lib/libmpiP.a' "
```

Or

<div align="center">Shell Command</div>

```
R CMD INSTALL pbdPROF_0.2-0.tar.gz
    --configure-args="--with-mpiP='-L/path/to/your/mpiP/lib lmpiP' "
```

Or the conventional method in R console

<div align="center">Shell Command</div>

```
install.packages("pbdPROF",
    configure.args=c("--with-mpiP=/path/to/your/mpiP/lib/libmpiP.a"))
```

Or

<div align="center">Shell Command</div>

```
install.packages("pbdPROF",
    configure.args=c("--with-mpiP=-L/path/to/your/mpiP/lib -lmpiP"))
```

`pbdPROF/libs/` is empty, i.e., there is no need to build `pbdPROF.so`. The linking argument is saved in `Makeconf` and installed to `pbdPROF/etc/` for later use by other packages, such as **pbdMPI** or **Rmpi**. Since **mpiP** has external dependency `libfpmpi.a` on libunwind so while installing **mpiP** you are suggested to use the below command while configuring **mpiP**. This only provides the linking arguments, either

<div align="center">R Script</div>

```
./ configure --disable - libunwind CPPFLAGS ="-fPIC -I/usr/lib/openmpi/include"
    LDFLAGS ="-L/usr/lib/openmpi/lib -lmpi"
```

since one has changed the linking so need to reinstall packages depend on **Code**pbdPROF

2.4.1 Linking pbdMPI with pbdPROF

Reinstall **pbdMPI** via

Shell Command

```
R CMD INSTALL pbdMPI_1.0-0.tar.gz --configure - args ="-- enable - pbdPROF "
```

Package developers who are directly interfacing with MPI functions (via C or Fortran) should note that `pbdMPI/R/get_conf.r` and `pbdMPI/R/get_lib.r` are utilized in `pbdMPI/configure.ac` (used to generate `pbdMPI/configure`) to determine an appropriate linking flag `PROF_LDFLAGS` based on preset flags in `pbdPROF/etc/Makeconf`.

If your pbdMPI is correctly installed with all correct linking you will have the screenshot just similar to below output during installation of **pbdMPI** or else you might get error

```
******************* Results of pbdMPI package configure *****************

>> TMP_INC = /usr/local/include
>> TMP_LIB = /usr/local/lib
>> MPI_ROOT =
>> MPITYPE = OPENMPI
>> MPI_INCLUDE_PATH = /usr/local/include
>> MPI_LIBPATH = /usr/local/lib
>> MPI_LIBS =  -lutil -lpthread
>> MPI_DEFS = -DMPI2
>> MPI_INCL2 =
>> PKG_CPPFLAGS = -I/usr/local/include  -DMPI2 -DOPENMPI
>> PKG_LIBS = /home/g/Documents/new_life/lib/libmpiP.a -L/usr/local/lib -lmpi
    -lutil -lpthread
>> PROF_LDFLAGS = /home/g/Documents/new_life/lib/libmpiP.a

*************************************************************************
```

No mater which library is used, internal or external, the `PROF_LDFLAGS` in `pbdMPI/etc/Makefile` provides the linking information to the profiling library. It is also used in `PKG_LIBS`, which will be exported to other **pbdR** packages at installation via the flag `SPMD_LDFLAGS`. Therefore there is no need for additional flags in `R CMD INSTALL` when reinstalling packages for profiling.

2.4.2 Linking pbdBASE with pbdPROF

For further profiling, such as **pbdBASE** (Schmidt et al., 2012), one may reinstall the package, via

Shell Command

```
R CMD INSTALL pbdBASE_0.2-2.tar.gz
```

There is no need to provide any flag since **pbdMPI/etc/Makefile** has the information and installation of **pbdBASE** already considers it. Note that since both packages (**pbdMPI** and **pbdBASE**) have MPI-using

C/Fortran functions involved, it is necessary to link with **pbdPROF** in order to profile communications evoked by the package.

2.4.3   Linking Rmpi with pbdPROF

Reinstall **Rmpi** via

<div align="center">Shell Command</div>

```
wget https://github.com/snoweye/Rmpi_PROF/archive/master.zip
unzip master.zip
mv Rmpi_PROF-master Rmpi
find ./Rmpi -type f -perm 777 -print -exec chmod 644 {} \;
find ./Rmpi -type d -perm 777 -print -exec chmod 755 {} \;
chmod 755 ./Rmpi/configure
chmod 755 ./Rmpi/cleanup
chmod 755 ./Rmpi/inst/*.sh
R CMD build --no-resave-data Rmpi
R CMD INSTALL Rmpi_0.6-4.tar.gz --configure-args="--enable-pbdPROF"
```

Note that 0.6-4 is not an official release of **Rmpi**. It is a modified version of 0.6-3 and it is currently available at https://github.com/snoweye/Rmpi_PROF. The authors of **Rmpi** have plans to eventually incorporate these changes, but this can be used as a temporary measure.

3   Testing pbdPROF Installation

Here, we provide two simple R scripts, one for **pbdMPI** and one for **Rmpi**, to test the installation and profiling capabilities of **pbdPROF**. Assuming all went well, then a profiler output file will be produced (in the directory where you executed the above command). The name of the file depends on how **pbdPROF** was built:

- **fpmpi**: the profiler output file will always be called `fpmpi_profile.txt`.

- **mpiP**: the profiler output file will be named according to the scheme `R.ncores.PID.1.mpiP`, where `ncores` is the actual number of cores used, and `PID` is the job PID that was used.

Here again, **mpiP** has several advantages over **fpmpi**. For one, **fpmpi** will always overwrite old profiler output in the same directory. Additionally, **fpmpi** profiler outputs give no context to the calling command, whereas **mpiP** gives the calling command (and whence, which R script was used to generate the profiler output) on the second line of the profiler output.

If you followed the instructions found in Section 2, but no profiler output is produced, then please see the troubleshooting guide, Appendix A.

For the remainder, we will be using **fpmpi** in examples.

3.1   Test with pbdMPI

Below we provide sample scripts to test that the installation of **pbdPROF** was successful. For **pbdMPI**, use:

<div align="center">Test script for pbdMPI</div>

```r
### Save this in a file: prof_pbdMPI.r
library(pbdMPI, quiet = TRUE)
init()

set.seed(comm.rank())
x <- allreduce(rnorm(100), op = "sum")

finalize()
```

and run this code by

R Script

```
mpiexec -np 2 Rscript prof_pbdMPI.r
```

The **fpmpi** profiling output from the file `fpmpi_profile.txt` may contain:

```
Details for each MPI routine
                 Average of sums over all processes
                                                 % by message length
                               (max over          0.........1........1........
                                processes [rank])          K          M
MPI_Allreduce:
        Calls    :            2             2 [    0] 05000000005000000000000000000
        Time     :     3.61e-05      3.72e-05 [    0] 07000000003000000000000000000
        Data Sent :        804           804 [    0]
        SyncTime :      0.00149       0.00287 [    0] 0*0000000.000000000000000000
        By bin   : 1-4 [1,1]      [    2.5e-05,  2.72e-05] [   4.1e-05,    0.00286]
                 : 513-1024     [1,1]    [      1e-05,      1e-05] [   1.1e-05,
                     7.61e-05]
```

In this R script, one MPI C function `MPI_Allreduce` is called twice and 804 bytes are sent that a hundred of double precision (8 bytes) for 100 normal random variables, and one integer (4 bytes) for checking data type to call the corresponding S4 method.

3.2   Test with Rmpi

For **Rmpi**, use:

Test script for pbdMPI

```r
### Save this in a file: prof_Rmpi.r
library(Rmpi, quiet = TRUE)
mpi.comm.dup(0, 1)

set.seed(mpi.comm.rank())
x <- mpi.allreduce(rnorm(100), type = 2, op = "sum")

mpi.quit()
```

and run this code by

R Script

```
mpiexec -np 2 Rscript prof_Rmpi.r
```

The **fpmpi** profiling output from the file `fpmpi_profile.txt` may contain:

```
Details for each MPI routine
                Average of sums over all processes
                                           % by message length
                          (max over          0.........1.........1........
                          processes [rank])        K         M
MPI_Allreduce:
       Calls     :          1          1 [   0] 000000000*000000000000000000
       Time      :   4.01e-05    4.41e-05 [   1] 000000000*000000000000000000
       Data Sent :        800         800 [   0]
       SyncTime  :    0.00103     0.00204 [   1] 000000000*000000000000000000
       By bin    : 513-1024    [1,1]   [   3.6e-05,   4.41e-05] [  2.79e-05,    0
.00204]
MPI_Comm_dup:
       Calls     :          1
       Time      :    5.81e-05
       SyncTime  :   0.000211
```

Two MPI C functions `MPI_Allreduce` and `MPI_Comm_dup` are called one time for each.

Part II

Profiling

In this part, we will profile some much more substantive examples. This assumes that **pbdPROF** has been correctly configured and installed. Make sure you can produce profiler outputs as described in Section 3 before proceeding. If not, please see Appendix A.

4   Profiling with fpmpi

4.1   Demo of pbdMPI

The `allreduce.r` script is originally in **pbdMPI/demo/** and can be profiled by

<div align="center">R Script</div>

```
mpiexec -np 2 Rscript -e "demo(allreduce,'pbdMPI',ask=F,echo=F)"
```

which will provide an output file `fpmpi_profile.txt`. Part of output is listed in the next as

```
Processes:     2
Execute time:   1.176
Timing Stats: [seconds] [min/max]        [min rank/max rank]
wall-clock: 1.176 sec 1.171488 / 1.180277     0 / 1
user: 0.378 sec 0.360000 / 0.396000     0 / 1
sys: 0.07 sec  0.040000 / 0.100000    1 / 0

Average of sums over all processes
Routine                 Calls      Time Msg Length    %Time by message length
```

```
0.........1........1........
K       M
MPI_Allreduce      :      10   0.000118       188 061003000000000000000000000000
MPI_Barrier        :      21      0.0054

Details for each MPI routine
Average of sums over all processes
% by message length
(max over          0.........1........1........
 processes [rank])        K         M
MPI_Allreduce:
Calls    :         10            10 [    0] 051004000000000000000000000000
Time     :    0.000118      0.000119 [    0] 061003000000000000000000000000
Data Sent :         188           188 [    0]
SyncTime :    0.000312      0.000453 [    0] 07.002000000000000000000000000
By bin   : 1-4 [5,5]    [  7.01e-05,  7.01e-05] [  0.000117,  0.000343]
: 5-8 [1,1]    [  7.87e-06,  9.06e-06] [  9.06e-06,  9.06e-06]
: 33-64        [4,4]    [  3.91e-05,  4.03e-05] [  4.51e-05,     0.0001]
MPI_Barrier:
Calls    :         21
Time     :      0.0054
```

Two MPI C functions `MPI_Allreduce` and `MPI_Barrier` are evoked inside this R code. The `MPI_Allreduce` is called 10 times, span 0.000118 seconds, and 188 bytes are sent. The `MPI_Barrier` is called 21 times and span 0.0054 seconds.

4.2   Demo of pbdDMAT

The `svd.r` is originally in **pbdDMA/demo/** (Schmidt et al., 2012) and can be profiled by

R Script

```
mpiexec -np 2 Rscript -e "demo(svd,'pbdDMAT',ask=F,echo=F)"
```

which will provide an output file `fpmpi_profile.txt`. Part of output is listed in the next as

```
Processes:  2
Execute time:  1.774
Timing Stats: [seconds] [min/max]        [min rank/max rank]
wall-clock: 1.774 sec   1.766181 / 1.781962     1 / 0
user: 0.962 sec 0.956000 / 0.968000      1 / 0
sys: 0.046 sec  0.044000 / 0.048000      0 / 1

Average of sums over all processes
Routine                 Calls        Time Msg Length    %Time by message length
0.........1........1........
K       M
MPI_Allreduce      :      12   0.000108        72 064000000000000000000000000000
MPI_Barrier        :       8   0.000784

Details for each MPI routine
Average of sums over all processes
% by message length
(max over          0.........1........1........
 processes [rank])        K         M
MPI_Allreduce:
```

```
Calls      :          12             12 [   0] 0550000000000000000000000000000
Time       :     0.000108       0.000113 [   0] 0640000000000000000000000000000
Data Sent :          72             72 [   0]
SyncTime  :     0.000143        0.00016 [   1] 0640000000000000000000000000000
By bin    : 1-4 [6,6]   [  5.44e-05,  6.91e-05] [  6.91e-05,  8.89e-05]
: 5-8   [6,6]    [  4.36e-05,  4.79e-05] [  5.72e-05,  7.08e-05]
MPI_Barrier:
Calls      :           8
Time       :     0.000784
```

Two MPI C functions `MPI_Allreduce` and `MPI_Barrier` are evoked inside this R code. The `MPI_Allreduce` is called 12 times, span 0.000108 seconds, and 72 bytes are sent. The `MPI_Barrier` is called 8 times and span 0.000784 seconds.

4.3 Demo of Rmpi

The `masterSlavePI.r` is originally in **Rmpi/demo/** and can be profiled by

R Script

```
mpiexec -np 4 Rscript -e "demo(masterslavePI,'Rmpi',ask=F,echo=F)"
```

which will provide an output file `fpmpi_profile.txt`. Part of output is listed in the next as

```
Processes:      1
Execute time:    0.05362
Timing Stats: [seconds] [min/max]         [min rank/max rank]
wall-clock: 0.05362 sec 0.053622 / 0.053622      0 / 0
user: 0.236 sec 0.236000 / 0.236000     0 / 0
sys: 0.052 sec  0.052000 / 0.052000     0 / 0

Average of sums over all processes
Routine                 Calls       Time Msg Length    %Time by message length
0.........1........1........
K        M
MPI_Reduce            :      1   6.51e-05          8 00*0000000000000000000000000000

Details for each MPI routine
Average of sums over all processes
% by message length
(max over           0.........1........1........
 processes [rank])          K        M
MPI_Reduce:
Calls      :           1              1 [   0] 00*0000000000000000000000000000
Time       :    6.51e-05       6.51e-05 [   0] 00*0000000000000000000000000000
Data Sent :           8              8 [   0]
By bin    : 5-8 [1,1]   [  6.51e-05,  6.51e-05]
```

One MPI C function `MPI_Reduce` is evoked inside this R code. The `MPI_Reduce` is called only 1 time, span $6.51e - 05$ seconds, and 8 bytes are sent. Note that there is only one processor (master in `comm=0`) profiled by **fpmpi**, and the other three processors (slaves in `comm=1`) are not.

5   Profiling with mpiP

5.1   Demo of pbdMPI

The `allreduce.r` is originally in **pbMPI/demo** and can be profiled by

R Script

```
mpiexec -np 2 Rscript -e "demo(allreduce,'pbdMPI',ask=F,echo=F)"
```

which will produce an output file `allreduce.r.mpiP` part of file is listed below

```
@ Collector Rank          : 0
@ Collector PID           : 24033
@ Final Output Dir        : .
@ Report generation       : Single collector task
@ MPI Task Assignment     : 0
@ MPI Task Assignment     : 1


-------------------------------------------------------------------------------
@--- MPI Time (seconds) -------------------------------------------------------
-------------------------------------------------------------------------------
Task   AppTime    MPITime    MPI%
0        0.153    0.00207    1.35
1        0.155    0.0284     18.35
*        0.308    0.0305     9.90
-------------------------------------------------------------------------------
@--- Callsites: 6 -------------------------------------------------------------
-------------------------------------------------------------------------------
ID Lev File/Address        Line Parent_Funct          MPI_Call
1   0 0x7f335d1108c3            [unknown]             Allreduce
2   0 0x7f335d110acb            [unknown]             Barrier
3   0 0x7f335d1107f3            [unknown]             Allreduce
4   0 0x7f2ded6f68c3            [unknown]             Allreduce
5   0 0x7f2ded6f6acb            [unknown]             Barrier
6   0 0x7f2ded6f67f3            [unknown]             Allreduce
-------------------------------------------------------------------------------
@--- Aggregate Time (top twenty, descending, milliseconds) ----------------
-------------------------------------------------------------------------------
Call              Site      Time     App%     MPI%      COV
Barrier              5      28.1     9.13    92.21     0.00
Barrier              2      1.63     0.53     5.36     0.00
Allreduce            3     0.322     0.10     1.06     0.00
Allreduce            6     0.217     0.07     0.71     0.00
Allreduce            1     0.117     0.04     0.38     0.00
Allreduce            4     0.083     0.03     0.27     0.00
-------------------------------------------------------------------------------
@--- Aggregate Sent Message Size (top twenty, descending, bytes) ----------
-------------------------------------------------------------------------------
Call              Site     Count      Total      Avrg   Sent%
Allreduce            1         4        160        40   42.55
Allreduce            4         4        160        40   42.55
Allreduce            3         6         28      4.67    7.45
Allreduce            6         6         28      4.67    7.45
```

The above statistics shows various criteria for the program run. The "MPI Time" shows running time

per process while executing the `allreduce.r`. There are four columns:

- `Task` which is the rank of the processor,

- `AppTime` which is the application level runtime having values 0.153 and 0.155 seconds for the first (0) and second (1) ranks, respectively,

- `MPITime` which is the MPI level runtime of code having values 0.00207 seconds for the first rank and 0.0284 seconds for the second rank, and

- `MPI%` which is the percentage of `MPITime` in `AppTime` having values 1.35% and 18.35% for rank 0 processor and rank 1, respectively.

The * shows the sums of total ranks in respective columns.

The "Callsites" division shows 6 MPI calls in these two processors are evoked. One `Barrier` and two types of `Allreduce` (one for `integer` and one for `double`) for each processor. The general `allreduce()` function in **pbdMPI** is a S4 method which checks data type first (`matrix`, `array`, `integer`, or `double`) using `MPI_Allreduce`, then bases on the data type to evoke the corresponding S3 function using the other call to `MPI_Allreduce`. The `Barrier` is mainly evoked from `comm.cat()` and `comm.print()` in **pbdMPI**.

Furthermore, the **mpiP** library provides deeper analyses of each `MPI Calls` like "Aggregate Time" and "Aggregate Sent Message Size". In "Aggregate Time" division, the `Call` column shows information of MPI calls, `Barrier` called twice and `Allreduce` called four times. Note that for longer runs, only top twenty records are reported. The `Barrier` calls at the site 5 (ID 5 in the "Callsites" division) ran for 28.1 milliseconds of which 9.13% is application level aggregate time percentage and 92.21% is MPI level aggregate time percentage. Similarly, in "Aggregate Sent Message Size" division, per bytes information of each MPI call is elaborated. For example, for `Allreduce` at the site 1 has the count value of 4 while total message size is 160 bytes, on average 40 bytes are there. Also, the sent percentage is 42.55% for `Allreduce` at the site 1.

5.2   Demo of pbdDMAT

The `svd.r` is originally in **pbdDMA/demo/** (Schmidt et al., 2012) and can be profiled by

R Script

```
mpiexec -np 2 Rscript -e "demo(svd,'pbdDMAT',ask=F,echo=F)"
```

which will provide an output file `svd.r.mpiP`. Part of output is listed in the next as

```
@ Collector Rank            : 0
@ Collector PID             : 25363
@ Final Output Dir          : .
@ Report generation         : Single collector task
@ MPI Task Assignment       : 0
@ MPI Task Assignment       : 1


--------------------------------------------------------------------------
@--- MPI Time (seconds) ---------------------------------------------------
--------------------------------------------------------------------------
Task    AppTime   MPITime     MPI%
0       0.768     0.000527    0.07
1       0.784     0.00195     0.25
*       1.55      0.00248     0.16
--------------------------------------------------------------------------
```

```
@--- Callsites: 6 -----------------------------------------------------------
------------------------------------------------------------------------------
ID Lev File/Address       Line Parent_Funct              MPI_Call
1   0 0x7f676ef298c3           [unknown]                Allreduce
2   0 0x7f676ef29acb           [unknown]                Barrier
3   0 0x7f676ef297f3           [unknown]                Allreduce
4   0 0x7fa461caf8c3           [unknown]                Allreduce
5   0 0x7fa461cafacb           [unknown]                Barrier
6   0 0x7fa461caf7f3           [unknown]                Allreduce
------------------------------------------------------------------------------
@--- Aggregate Time (top twenty, descending, milliseconds) ----------------
------------------------------------------------------------------------------
Call               Site       Time     App%    MPI%    COV
Barrier               5       1.55     0.10   62.40    0.00
Allreduce             6      0.295     0.02   11.90    0.00
Barrier               2      0.256     0.02   10.33    0.00
Allreduce             3      0.177     0.01    7.14    0.00
Allreduce             4       0.11     0.01    4.44    0.00
Allreduce             1      0.094     0.01    3.79    0.00
------------------------------------------------------------------------------
@--- Aggregate Sent Message Size (top twenty, descending, bytes) ----------
------------------------------------------------------------------------------
Call               Site      Count      Total      Avrg   Sent%
Allreduce             1          6         48         8   33.33
Allreduce             4          6         48         8   33.33
Allreduce             3          6         24         4   16.67
Allreduce             6          6         24         4   16.67
```

The above statistics shows various criteria the code has been profiled for the program `svd.r`. The interpretation is similar to that of `allreduce.r` above. However, these `MPI_Allreduce` functions are mainly called by functions of **ScaLAPACK** (Blackford et al., 1997) via **pbdBASE** (Schmidt et al., 2012) and **pbdSLAP** (Chen et al., 2012c).

5.3   Demo of Rmpi

The `masterSlavePI.r` is originally in **Rmpi/demo/** and can be profiled by

R Script

```
mpiexec -np 4 Rscript -e "demo(masterslavePI,'Rmpi',ask=F,echo=F)"
```

which will provide an output file `masterSlavePI.r.mpiP`. Part of output is listed in the next as

```
@ Collector Rank          : 0
@ Collector PID           : 25839
@ Final Output Dir        : .
@ Report generation       : Single collector task
@ MPI Task Assignment     : 0


------------------------------------------------------------------------------
@--- MPI Time (seconds) ------------------------------------------------------
------------------------------------------------------------------------------
Task   AppTime    MPITime     MPI%
0       0.0303    0.00125     4.12
*       0.0303    0.00125     4.12
------------------------------------------------------------------------------
```

```
@--- Callsites: 4 -----------------------------------------------------------
-----------------------------------------------------------------------------
ID Lev File/Address          Line Parent_Funct           MPI_Call
1   0 0x7f8cdbc03628              [unknown]              Comm_free
2   0 0x7f8cdbc03a2e              [unknown]              Intercomm_merge
3   0 0x7f8cdbc02ce6              [unknown]              Reduce
4   0 0x7f8cdbc0398b              [unknown]              Comm_free
-----------------------------------------------------------------------------
@--- Aggregate Time (top twenty, descending, milliseconds) ----------------
-----------------------------------------------------------------------------
Call               Site       Time     App%    MPI%     COV
Intercomm_merge       2       1.06     3.52    85.47    0.00
Reduce                3       0.102    0.34     8.19    0.00
Comm_free             4       0.053    0.18     4.25    0.00
Comm_free             1       0.026    0.09     2.09    0.00
-----------------------------------------------------------------------------
@--- Aggregate Sent Message Size (top twenty, descending, bytes) ----------
-----------------------------------------------------------------------------
Call               Site      Count      Total        Avrg   Sent%
Reduce                3         1           8           8  100.00
-----------------------------------------------------------------------------
@--- Callsite Time statistics (all, milliseconds): 4 --------------------
-----------------------------------------------------------------------------
Name            Site Rank   Count      Max      Mean      Min     App%    MPI%
Comm_free          1    0       1     0.026    0.026    0.026    0.09    2.09
Comm_free          1    *       1     0.026    0.026    0.026    0.09    2.09

Comm_free          4    0       1     0.053    0.053    0.053    0.18    4.25
Comm_free          4    *       1     0.053    0.053    0.053    0.18    4.25

Intercomm_merge    2    0       1      1.06     1.06     1.06    3.52   85.47
Intercomm_merge    2    *       1      1.06     1.06     1.06    3.52   85.47

Reduce             3    0       1     0.102    0.102    0.102    0.34    8.19
Reduce             3    *       1     0.102    0.102    0.102    0.34    8.19
-----------------------------------------------------------------------------
@--- Callsite Message Sent statistics (all, sent bytes) -------------------
-----------------------------------------------------------------------------
Name            Site Rank   Count      Max      Mean      Min             Sum
Reduce             3    0       1         8         8        8               8
Reduce             3    *       1         8         8        8               8
```

The above statistics shows various criteria the code has been profiled for the program `masterSlavePI.r`. Three main MPI calls are used in this program: `MPI_Intercomm_merge`, `MPI_Reduce` and `MPI_Comm_free` since **Rmpi** uses the master/workers framework.

6   Plotting

The plotting utilities of **pbdPROF** have been moved to the **pbdSCRIBE** package.

Part III

Appendix

A   pbdPROF Troubleshooting

A.1   Installation

Problem 1: If you have downloaded the package from github and tried to using R CMD INSTALL **pbdPROF** and you see an error similar to this

```
ERROR: 'configure' exists but is not executable -- see the 'R Installation and
    Administration Manual'
```

Solution: You have to make the configure executable which means giving it permission , which can done by

```
chmod +x configure
```

after changing the folder to package's main directory.

Problem 2: If you are using **fpmpi** (Gropp, 2000) externally and during it's installation you get an error similar to this

```
error :checking for library containing MPI_Init... (cached) no configure:
error: Could not find MPI library
```

Solution: You probably need to specify the path to MPI library using this in command line in the fpmpi main directory

```
./configure CPPFLAGS="-fPIC -I/usr/lib/openmpi/include"
    LDFLAGS="-L/usr/lib/openmpi/lib -lmpi"
```

Problem 3: If you are using **mpiP** externally and during it's installation you get an error similar to this

```
libmpiP.a(wrappers.o): relocation R_X86_64_32 against '.rodata.str1.1' can not
    be used when making a shared object; recompile with -fPIC
libmpiP.a: could not read symbols: Bad value collect2: error: ld returned 1
    exit status
```

Solution: You probably need to specify the path to MPI library using this in command line when installing **mpiP**

```
./configure CPPFLAGS="-fPIC -I/usr/lib/openmpi/include"
    LDFLAGS="-L/usr/lib/openmpi/lib -lmpi"
```

Problem 4: If you are using **mpiP** externally and during **pbdMPI** installation you get an error similar to this

```
Error : .onLoad failed in loadNamespace () for 'pbdMPI', details:
  call: dyn.load(file, DLLpath = DLLpath, ...)
  error: unable to load shared object 'pbdMPI.so':
  pbdMPI/libs/pbdMPI.so: undefined symbol: _Ux86_64_getcontext
```

Solution: You probably need to disable some external library prerequisite by **mpiP**, using this in command line when installing **mpiP**

<div align="center">R Script</div>

```
./configure --disable-libunwind CPPFLAGS="-fPIC -I/usr/lib/openmpi/include"
    LDFLAGS="-L/usr/lib/openmpi/lib -lmpi"
```

A.2   Running

Problem 5: No profiler output is produced.

Solution: If no profiler output is produced, then it is almost certainly the case that **pbdPROF** and/or the MPI-using R package (e.g., **pbdMPI**, **Rmpi**, etc.) was/were not set up and installed correctly. Please refer to Section 2 and the relevant package's installation documentation and reinstall.

Problem 6: While running **Rmpi** code for profiling, if you encounter the error below:

```
error: mpiexec was unable to launch the specified application as it could not
    access
or execute an executable:
Executable: /path/to/R/package_installation_directory/2.15/Rmpi/Rslaves.sh
Node: "Your_node"
while attempting to start process rank 0.
```

Solution: You need to make executable of the shell scripts in the `inst/` directory of **Rmpi** main directory using the following command from command line in `inst/` directory:

<div align="center">R Script</div>

```
chmod +x *.sh
```

Problem 7: While running **Rmpi** code for profiling, if you encounter the error below:

```
[G:12221] [[39704,0],0] ORTE_ERROR_LOG: Not found in file
    ../../../../../orte/mca/plm/base/plm_base_launch_support.c at line 758
--------------------------------------------------------------------------
mpiexec was unable to start the specified application as it encountered an
    error.
More information may be available above.
--------------------------------------------------------------------------
```

Solution:

1. You need to check whether your **Rmpi** is working without the **pbdPROF**. If yes try running your **Rmpi** code on single process only.

2. If above does not help, then you may need `.Rprofile` in `Rmpi/inst/` to run your code from `inst/` directory.

3. If still your code does not run, you need to update your OpenMPI version to the latest one. You can check your OpenMpi version http://www.open-mpi.org/software/ompi/ through

```
ompi_info
```

4. If further you came to this far and luck is not with you somehow (pun intended), there might some configuration problem in your machine.

B   References

L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997. ISBN 0-89871-397-8 (paperback). URL http://netlib.org/scalapack/slug/scalapack_slug.html/.

D. Chen, W.-C. Schmidt, G. Sehrawat, P. Patel, and G Ostrouchov. pbdPROF: Programming with big data – MPI profiling tools, 2013. URL http://cran.r-project.org/package=pbdPROF. R Package.

W.-C. Chen, G. Ostrouchov, D. Schmidt, P. Patel, and H. Yu. pbdMPI: Programming with big data – interface to MPI, 2012a. URL http://cran.r-project.org/package=pbdMPI. R Package.

W.-C. Chen, G. Ostrouchov, D. Schmidt, P. Patel, and H. Yu. *A Quick Guide for the pbdMPI package*, 2012b. URL http://cran.r-project.org/package=pbdMPI. R Vignette.

W.-C. Chen, D. Schmidt, G. Ostrouchov, and P. Patel. pbdSLAP: Programming with big data – scalable linear algebra packages, 2012c. URL http://cran.r-project.org/package=pbdSLAP. R Package.

W.D. Gropp. Fpmpi-2: Fast profiling library for mpi, 2000. URL http://www.mcs.anl.gov/research/projects/fpmpi/WWW/.

G. Ostrouchov, W.-C. Chen, D. Schmidt, and P. Patel. Programming with big data in R, 2012. URL http://r-pbd.org/.

D. Schmidt, W.-C. Chen, G. Ostrouchov, and P. Patel. pbdBASE: Programming with big data – core pbd classes and methods, 2012. URL http://cran.r-project.org/package=pbdBASE. R Package.

Sameer S. Shende and Allen D. Malony. The tau parallel performance system. *Int. J. High Perform. Comput. Appl.*, 20(2):287–311, May 2006. ISSN 1094-3420. doi: 10.1177/1094342006064482. URL http://dx.doi.org/10.1177/1094342006064482.

Jeffrey S. Vetter and Michael O. McCracken. Statistical scalability analysis of communication operations in distributed applications. In *Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*, PPoPP '01, pages 123–132, New York, NY, USA, 2001. ACM. ISBN 1-58113-346-4. doi: 10.1145/379539.379590. URL http://doi.acm.org/10.1145/379539.379590.

Hao Yu. Rmpi: Parallel statistical computing in r. *R News*, 2(2):10–14, 2002. URL http://cran.r-project.org/doc/Rnews/Rnews_2002-2.pdf.