Guide to the **pbdPROF** Package

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

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

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

[3]Remote Data Analysis and Visualization Center
University of Tennessee,
Knoxville, TN, USA

July 26, 2013

Contents

This publication was typeset using LaTeX.

Acknowledgement

Sehrawat was generously supported by Google for Google Summer of Code 2013.

Chen and Ostrouchov were supported in part by the project "Visual Data Exploration and Analysis of Ultra-large Climate Data" funded by U.S. DOE Office of Science under Contract No. DE-AC05-00OR22725. Ostrouchov, Schmidt, and Patel were supported in part by the project "NICS Remote Data Analysis and Visualization Center" funded by the Office of Cyberinfrastructure of the U.S. National Science Foundation under Award No. ARRA-NSF-OCI-0906324 for NICS-RDAV center.

Warning: 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 document is written to explain the main functions of **pbdPROF** (Sehrawat *et al.*, 2013), version 0.1-0. 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, such as **fpmpi** (Gropp, 2000), **mpiP** (Vetter and McCracken, 2001), or **TAU** (Shende and Malony, 2006), to profile parallel R code and understand hidden MPI communications between processors. Numbers of communications, sizes of messages, times and types of functions calls all affect program performance and design of algorithm. The MPI profiling libraries are able to high-jack MPI functions at run time that intercept some of MPI function calls, then provide MPI information without disturbing original programs or algorithms.

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

1. providing linking information to pbdR (Ostrouchov *et al.*, 2012),
2. output profiling information associated with MPI calls,
3. parsing and summarizing profiling information, and
4. support three MPI profiling libraries.

1.1 System Requirements

pbdPROF requires an MPI installation and an MPI-using 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/`.

2 Installation

The **pbdPROF** currently is by default using **fpmpi** library internally, i.e., a source copy of **fpmpi** is located at **pbdPROF/src/fpmpi** and built in a static library at **pbdPROF/lib/libfpmpi.a**. However, external profiler libraries such as **fpmpi**, **mpiP**, and **TAU** can be also linked by **pbdPROF** via suitable `--configure-args` to `R CMD INSTALL`. We explain the whole procedure in Section 2.1 using **fpmpi** as an example and leave some keys steps for **mpiP** and **TAU** in Sections 2.2 and 2.3.

No matter using **fpmpi**, **mpiP**, or **TAU**, we strongly recommend to add `CPPFLAGS="-fPIC"` at the `configure` step.

2.1 fpmpi

Using internal **fpmpi** library, via

Shell Command

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

By default, this compiles `src/fpmpi/*`, generates a static library `libfpmpi.a`, and installs the library to `pbdPROF/lib/`. No shared library is generated or needed, so the directory `pbdPROF/libs/` is empty (no need to build `pbdPROF.so`.) The linking argument is saved in `Makeconf` and installed to `pbdPROF/etc/` for further linking such as **pbdMPI** is reinstalled with `--enable-pbdPROF`.

Linking with external **fpmpi** library, via

Shell Command

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

or

Shell Command

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

Since **fpmpi** only builds a static library `libfpmpi.a`, there is no difference of 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 further linking such as **pbdMPI** is reinstalled with `--enable-pbdPROF`.

2.1.1 Reinstall pbdMPI

Reinstall **pbdMPI** via

Shell Command

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

Note that the `pbdMPI/R/get_conf.r` and `pbdMPI/R/get_lib.r` are used in `pbdMPI/configure.ac` or `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 the `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 the external or internal library is used, the `PROF_LDFLAGS` in `pbdMPI/etc/Makefile` provides the linking information to the profiler library. It is also used in `PKG_LIBS` which will be export to other pbdR packages at installation via the flag `SPMD_LDFLAGS`, therefore, no need to add further flags to `R CMD INSTALL` when reinstall packages for further profiling.

2.1.2 Reinstall pbdBASE

For further profiling, such as **pbdBASE** (Schmidt *et al.*, 2012), one may reinstall both packages, 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 C functions involved, it is necessary to link with profiler library in order to profile communications evoked by both packages.

2.1.3 Reinstall Rmpi

Reinstall **Rmpi** via

Shell Command

```
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 available at https://github.com/snoweye/Rmpi_PROF.

2.2 mpiP

Users may consider to install the **mpiP** library on their own. Note that some of dependent libraries are prerequisites of **mpiP**, such as **libunwind**, but some of them can be disable at **mpiP** configuration time.

After **mpiP** is installed correctly, one may install **pbdPROF** by

Shell Command

```
R CMD INSTALL pbdPROF_0.1-0.tar.gz \
  --configure-args="--with-mpiP='/path_to_mpiP/lib/libmpiP.a'"
```

or

Shell Command

```
R CMD INSTALL pbdPROF_0.1-0.tar.gz \
  --configure-args="--with-mpiP='-L/path_to_mpiP/lib -lmpiP'"
```

will work for **pbdPROF** installation.

There may have some loading problems for the dependent shared libraries if `LD_PRELOAD` is not set, since neither R nor **pbdPROF** is not responsible to know where the shared libraries are. We strongly recommend to use the static library to avoid dynamic loading problems, since pre-loading shared libraries are also necessary for profiling code.

The same as Sections 2.1.1, 2.1.2, and 2.1.3, the re-installation of **pbdMPI**, **pbdBASE**, and **Rmpi** is required for profiling code.

2.3 TAU

<< TBD >>

3 Test Scripts

We provide two short R scripts for **pbdMPI** and **Rmpi** to test the installation of profiling libraries and **pbdPROF**. If installation is correct, one may profile the following scripts to obtain correcponding outputs.

3.1 Test with pbdMPI

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

Test script for pbdMPI

```
### 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
```

A successful output of **fpmpi** in the profiling 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] 05000000050000000000000000000
        Time       :   3.61e-05     3.72e-05 [   0] 07000000030000000000000000000
        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

```
### 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
```

A successful output of **fpmpi** in the profiling file `fpmpi_profile.txt` could be

```
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.

4 Profiling with fpmpi

4.1 Demo of pbdMPI

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

R Script

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

which will provide an output file `fpmpi_profile.txt`. We can read this file back into R and analyze it as next

```
> library(pbdPROF)
> da <- read.prof("fpmpi_profile.txt", lib.type = "fpmpi")
> (da.out <- parse.prof(da))
        Routine Calls     Time Data.Sent SyncTime
1 MPI_Allreduce    10 0.000121       188 0.000327
2   MPI_Barrier    21  0.00203      NULL     NULL
```

Two MPI C functions `MPI_Allreduce` and `MPI_Barrier` are evoked inside this R code. The `MPI_Allreduce` is called 10 times, span 0.000121 seconds, and 188 bytes are sent. The `MPI_Barrier` is called 21 times and span 0.00203 seconds.

4.2 Demo of pbdDMAT

`pbdDMAT/demo/svd.r` in **pbdDMAT** (Schmidt *et al.*, 2012)

<< TBD >>

4.3 Demo of Rmpi

`Rmpi/demo/masterslavePI.R` in **Rmpi**

<< TBD >>

5 References

Chen WC, Ostrouchov G, Schmidt D, Patel P, Yu H (2012a). "pbdMPI: Programming with Big Data – Interface to MPI." R Package, URL http://cran.r-project.org/package=pbdMPI.

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

Gropp W (2000). "FPMPI-2: Fast Profiling Library for MPI." URL http://www.mcs.anl.gov/research/projects/fpmpi/WWW/.

Ostrouchov G, Chen WC, Schmidt D, Patel P (2012). "Programming with Big Data in R." URL http://r-pbd.org/.

Schmidt D, Chen WC, Ostrouchov G, Patel P (2012). "pbdBASE: Programming with Big Data – Core pbd Classes and Methods." R Package, URL http://cran.r-project.org/package=pbdBASE.

Sehrawat G, Chen W-C Schmidt D, Patel P, Ostrouchov G (2013). "pbdPROF: Programming with Big Data – MPI Profiling Tools." R Package, URL http://cran.r-project.org/package=pbdPROF.

Shende SS, Malony AD (2006). "The Tau Parallel Performance System." *Int. J. High Perform. Comput. Appl.*, **20**(2), 287–311. ISSN 1094-3420. doi:10.1177/1094342006064482. URL http://dx.doi.org/10.1177/1094342006064482.

Vetter JS, McCracken MO (2001). "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, pp. 123–132. ACM, New York, NY, USA. ISBN 1-58113-346-4. doi:10.1145/379539.379590. URL http://doi.acm.org/10.1145/379539.379590.

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