Rhpc: An R Package for High Performance Computing

Ei-ji NAKAMA and Junji NAKANO

COM-ONE Ltd., Japan and The Institute of Statistical Mathematics, Japan

at SC13 on 17-22 November 2013, Denver, Colorado, USA

Introduction

R is a widely used free software environment for statistical computing and graphics. Recently, high performance computing (HPC) using **R** easily and efficiently is strongly required. To realize it in a better way, we provide a new **R** package for efficient computing using MPI.

Existing parallel environments of R for HPC

- **snow**
  The **snow** (Simple Network of Workstations) package by Tierney et al. can use PVM, MPI, NWS as well as direct networking sockets. As it is implemented mainly in **R** language, it has some inefficiency.
- **Rmpi**
  The **Rmpi** package offers access to numerous functions of MPI API, and a number of **R**-specific extensions. However, it is difficult to use for novice HPC users.
- **multicore**
  The **multicore** package provides a way of running parallel computations in **R** on just one machine with multiple cores by using operating system functions.

Objectives of Rhpc

Data has become very huge in amount and complicated in structure. To manipulate such data, parallel computing is the most useful tool at present. Although **R** has several packages for parallel computing as above, they are not well optimized for supercomputers.
We hope to improve the functionality and efficiency of the existing parallel computing functions mainly for supercomputers.

Overview of Rhpc

In **Rhpc** , we use MPI without using **Rmpi** and utilize collective communication as much as possible. Worker process is written by Embedding **R**. Main functions are:

- `Rhpc_worker_call` (∼ `snow::clusterCall`)
- `Rhpc_lapply` (∼ `snow::clusterApply`)
- `Rhpc_lapplyLB` (∼ `snow::clusterApplyLB`)

Behavior of main Rhpc functions

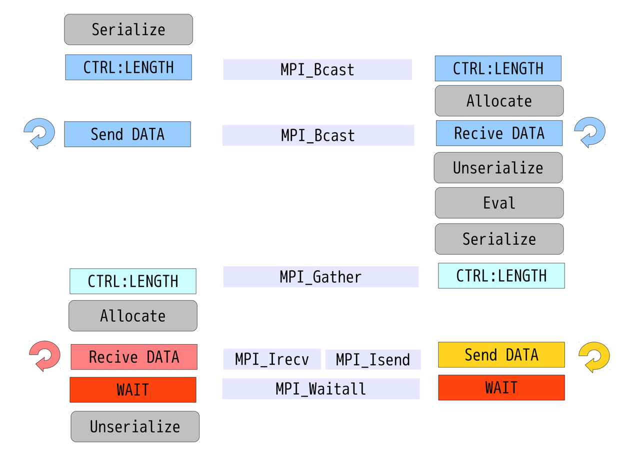


Fig 1: `Rhpc_worker_call`

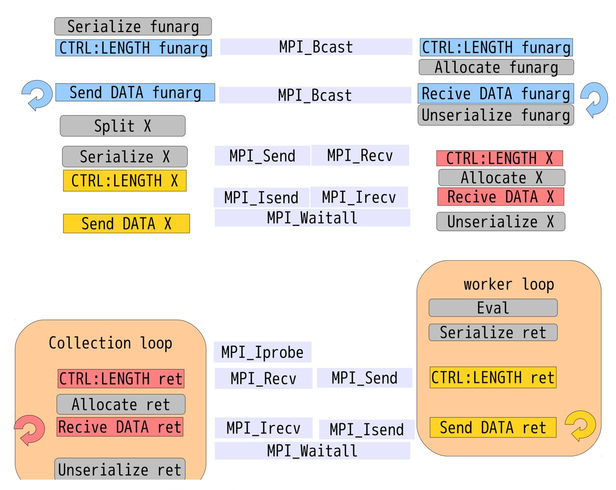


Fig 2: `Rhpc_lapply`

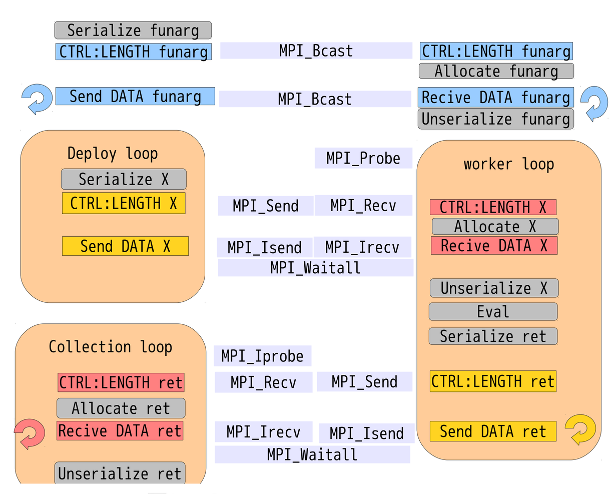


Fig 3: `Rhpc_lapplyLB`

Rhpc primitive functions

- Client initialization and finalization functions
  - `Rhpc_initialize()`
  - `Rhpc_finalize()`
- Handle function
  - `Rhpc_getHandle([NumberOfWorkers])`
- Instructions function for workers
  - `Rhpc_worker_call(handle,fun,...)`
- Parallel applying function
  - `Rhpc_lapply(handle,fun,...)`
  - `Rhpc_lapplyLB(handle,fun,...)`

Rhpc miscellaneous functions

- **Rhpc** worker call functions
  - `Rhpc_setupRNG(handle, seed)`
  - `Rhpc_Export(handle, names)`
  - `Rhpc_EvalQ(handle, expr)`

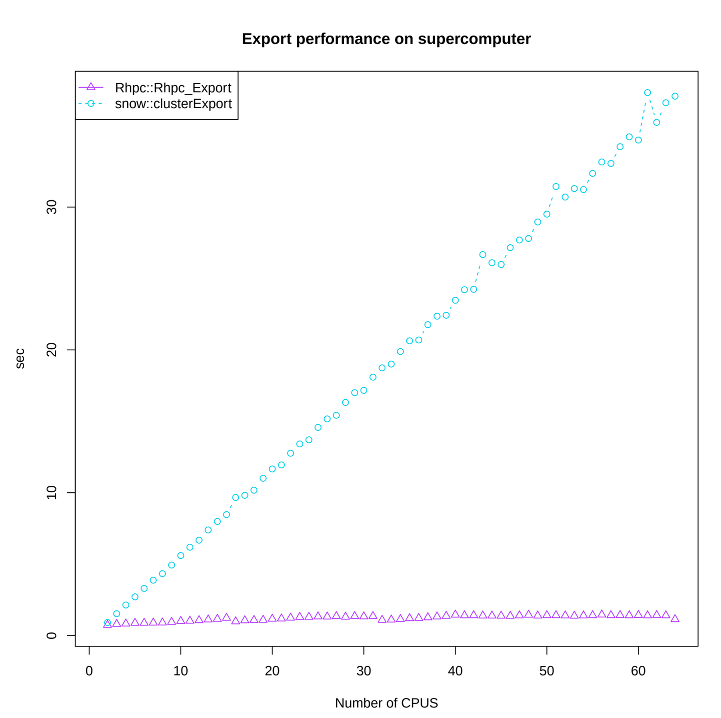
RhpcBLASctl miscellaneous functions

Control number of threads for **R**

- Control the number of threads for BLAS, MKL, ACML and GotoBLAS etc.
  - `blas_get_num_procs()`
  - `blas_set_num_threads(threads)`
- Control the number of threads for OpenMP
  - `omp_get_num_procs()`
  - `omp_get_max_threads()`
  - `omp_set_num_threads(threads)`

Examples on supercomputer

Export performance

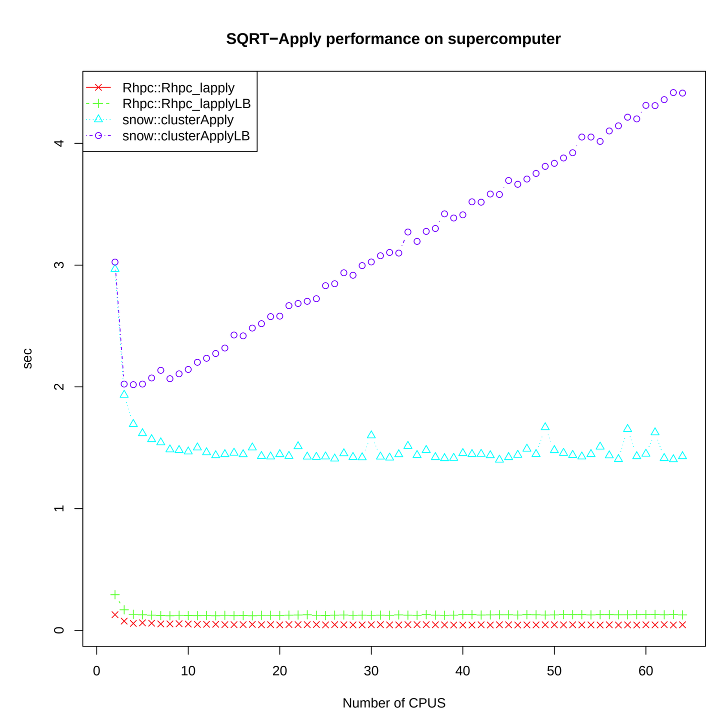


Listing 1: Rhpc_Export

```
> library(Rhpc)
Loading required package: rlecuyer
> Rhpc_initialize()
> cl<-Rhpc_getHandle()
Detected communication size 64
> set.seed(123)
> N<-4e3
> Rhpc_numberOfWorker(cl)
[1] 63
> M<-matrix(runif(N^2),N,N)
> system.time(Rhpc_Export(cl,"M"))
   user system elapsed
  1.012 0.116 1.139
> f<-function()sum(M)
> all.equal(rep(sum(M), Rhpc_numberOfWorker(cl)),
   unlist(Rhpc_worker_call(cl,f)))
[1] TRUE
> Rhpc_finalize()
>
>
> proc.time()
   user system elapsed
 15.559 0.281 16.452
```

As `clusterCall` of **snow** starts workers sequentially, it becomes slow when the number of workers increases. As **Rhpc** uses collective communication by `MPI_Bcast`, data transportation to workers is still fast even when the number of workers increases.

SQRT performance



Listing 2: Rhpc_lapply

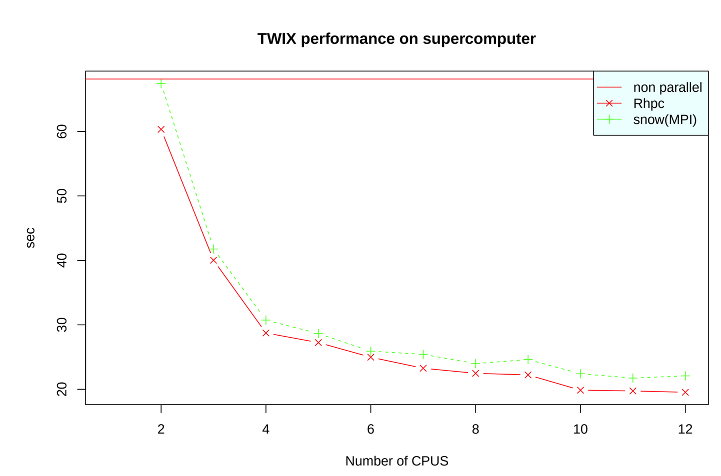
```
> library(Rhpc)
Loading required package: rlecuyer
> Rhpc_initialize()
> cl<-Rhpc_getHandle()
Detected communication size 64
> system.time(
+ ans<-Rhpc_lapply(cl, 1:10000, sqrt))
   user system elapsed
  0.045 0.001 0.046
> all.equal(sqrt(1:10000),unlist(ans))
[1] TRUE
> Rhpc_finalize()
```

Listing 3: Rhpc_lapplyLB

```
> library(Rhpc)
Loading required package: rlecuyer
> Rhpc_initialize()
> cl<-Rhpc_getHandle()
Detected communication size 64
> system.time(
+ ans<-Rhpc_lapplyLB(cl, 1:10000, sqrt))
   user system elapsed
  0.125 0.001 0.127
> all.equal(sqrt(1:10000),unlist(ans))
[1] TRUE
> Rhpc_finalize()
```

snow and **Rmpi** are largely written in **R** language, and are rather slow. As the main part of **Rhpc** is written in C language, it is efficient.

TWIX performance



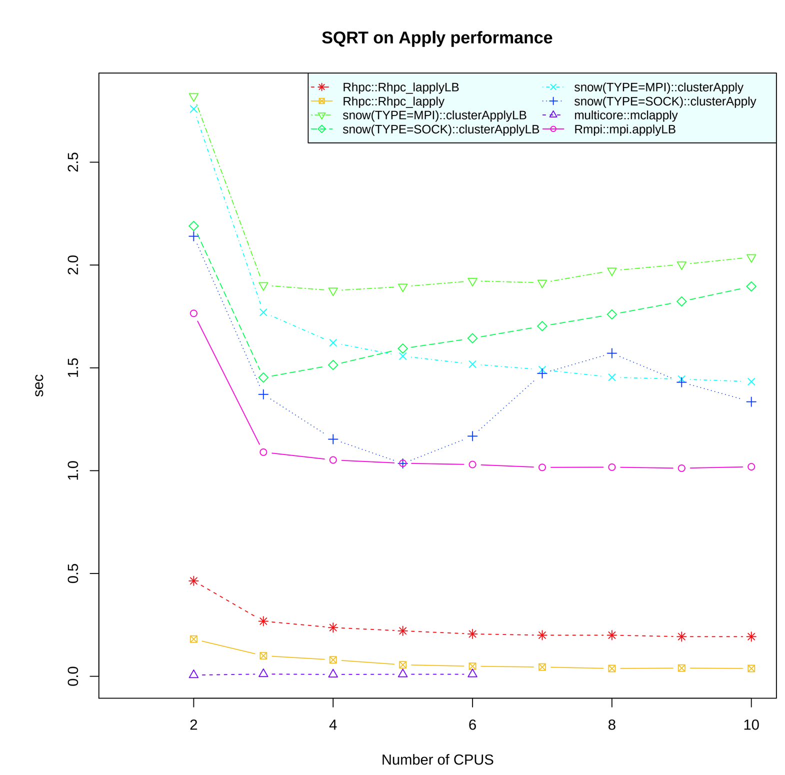
Listing 4: TWIX using parallel computing

```
> library(TWIX)
> r<-read.csv("http://archive.ics.uci.edu/ml/machine-
  learning-databases/wine-quality/winequality-red.csv
  ",sep=";")
> w<-read.csv("http://archive.ics.uci.edu/ml/machine-
  learning-databases/wine-quality/winequality-white.
  csv",sep=";")
> x<-rbind(r,w)
> x$quality=factor(x$quality)
> set.seed(123)
> i <- sample(nrow(x),nrow(x)/4)
> ic <- setdiff(1:nrow(x),i)
> training <- x[ic,]
> test <- x[i,]
>
> library(Rhpc)
Loading required package: rlecuyer
> Rhpc_initialize()
> cl<-Rhpc_getHandle()
Detected communication size 12
> system.time(TWIX(quality~. ,data=training[,1:12],
+ topN=c(12,12), method="local",
+ cluster=cl))
n = 1685
Deviance gain and TIC of the best TWIX-tree: 645.3553
  0.7482044
Deviance gain and TIC of the greedy tree(Nr.1116):
  849.0834 0.7297353
   user system elapsed
 19.444 0.082 19.534
> Rhpc_finalize()
```

TWIX is a binary-split decision tree algorithm for classification and data mining developed by Sergej Potapov, Martin Theus and Simon Urbanek. It has **snow** and **multicore** codes, and just small changes are required for **Rhpc**.

Example on personal cluster

SQRT performance



Listing 5: Rhpc_lapply

```
> library(Rhpc)
Loading required package: rlecuyer
> Rhpc_initialize()
> cl<-Rhpc_getHandle(as.integer(Sys.getenv("NPROCS"))-1)
> system.time(ans<-Rhpc_lapply(cl, 1:10000, sqrt))
   user system elapsed
  0.036 0.004 0.038
> all.equal(sqrt(1:10000),unlist(ans))
[1] TRUE
> Rhpc_finalize()
>
> proc.time()
   user system elapsed
  0.620 2.020 2.648
```

At present, **Rhpc** is a little slower than **multicore**.