Parallel np: Using the npRmpi Package

Jeffrey S. Racine
McMaster University

Abstract

The **npRmpi** package is a parallel implementation of the R (R Development Core Team (2010)) package **np** (Hayfield and Racine (2008)). The underlying C code uses the message passing interface ('MPI') and is MPI2 compliant.

Keywords: nonparametric, semiparametric, kernel smoothing, categorical data.

1. Overview

A common and understandable complaint often voiced about applied nonparametric kernel methods is the amount of computation time required for data-driven bandwidth selection when one has a large data set. There is a certain irony at play here since nonparametric methods are ideally suited to situations involving large data sets yet, computationally speaking, their analysis may lie beyond the reach of many users. Some background may be in order. My co-authors and I favor data-driven methods of bandwidth selection such as cross-validation, among others. These methods possess a number of very desirable properties but have run times that are proportional to the square of the number of observations hence doubling the number of observations will increase run time by a factor of four. For large data sets run time many simply not be feasible in a serial (i.e. single processor) environment.

The solution adopted in the **npRmpi** package is to run the code in a parallel computing environment and exploit the presence of multiple processors when available. The underlying C code for **np** is MPI-aware (MPI denotes the 'message passing interface', a popular parallel programming library that is an international standard), and we merge the **R np** and **Rmpi** packages to form the **npRmpi** package (this requires minor modification of some of the underlying **Rmpi** code which is why we cannot simply load the **Rmpi** package itself).[1]

All of the functions in **np** can exploit the presence of multiple processors. Run time is inversely proportional to the number of processors hence doubling the number of processors will cut run time in half.[2] Given the availability of commodity cluster computers and the presence of multiple cores in desktop and laptop machines, leveraging the **npRmpi** package for large data sets may present a feasible solution to the often lengthy computation times associated with nonparametric kernel methods.

[1]The **npRmpi** package incorporates the **Rmpi** package (Hao Yu <hyu@stats.uwo.ca>) with minor modifications and we are extremely grateful to Hao Yu for his contributions to the R community.

[2]There is minor overhead involved with message passing, and for small samples the overhead can be substantial when the ratio of message passing to computing the kernel estimator increases - this will be negligible for sufficiently large samples.

The code has been tested in the Mac OS X and Linux environments which allow the user to compile R packages on the fly (presuming of course that a C compiler exists on your system). Users running MS Windows will have to consult local tech support personnel and may also wish to consult the resources available for the **Rmpi** package and associated web site for further assistance. I cannot assist with installation issues beyond what is provided in this document and trust the reader will forgive me for this.

2. Differences Between np and npRmpi

There are only a few visible differences between running code in serial versus parallel environments. Typically you run your parallel code in batch mode so the first step would be to get your code running in batch mode using the **np** package (obviously on a subset of your data for large data sets). Once you have properly functioning code, you will next add some 'hooks' necessary for MPI to run (see Section 2.3 below for a detailed example), and finally you will run the job using either `mpirun` or, indirectly, via a batch scheduler on your cluster such as `sqsub` (kindly consult your local support personnel for assistance on using batch queueing systems on your local cluster).

Note that, since the data has to be broadcast to the slave notes, it is a good idea to put it in a dataframe first and it is always a good idea at this stage to cast your variables according to type.

Installation

Installation will depend on your hardware and software configuration and will vary widely across platforms. If you are not familiar with parallel computing you are strongly advised to seek local advice.

That being said, if you have current versions of R and Open MPI properly installed on your system, installation of the **npRmpi** package can be done in the standard manner from within R via

```
> install.packages("npRmpi")
```

or, alternatively, you can download the **npRmpi** tarball from CRAN and, from a command shell, run

```
R CMD INSTALL npRmpi_foo.tar.gz
```

where foo is the version number.

For clusters you may additionally need to provide locations of libraries (kindly see your local sysadmin as there are far too many variations for me to assist). On a local Linux cluster I use the following by way of illustration (for this illustration we use the Intel compiler suite and need to set MPI library paths and MPI root directories):

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/sharcnet/openmpi/1.4.2/intel/lib
export MPI_ROOT=/opt/sharcnet/openmpi/1.4.2/intel
```

where again foo is the version number.

Please seek local help for further assistance on installing and running parallel programs.

2.1. Parallel Batch Execution

To run a parallel **np** job having successfully installed the **npRmpi** program, copy the `Rprofile` file in `npRmpi/inst` to the current directory and name it `.Rprofile` (or copy it to your home directory and again name it `.Rprofile`).[3] Then to run the batch code in the file `npudensml_npRmpi.R` using two processors on an Open MPI system you will enter something like

```
mpirun -np 2 R CMD BATCH npudensml_npRmpi.R
```

You can compare run times and any other differences by examining the files `npudensml_serial.Rout` and `npudensml_npRmpi.Rout` (see Section A below for some illustrative examples). Clearly you could do this with a subset of your data for large problems to judge the extent to which the parallel code reduces run time.

If you have a batch scheduler installed on your cluster you might instead enter something like

```
sqsub -q mpi -n 2 R CMD BATCH npudensml_npRmpi.R
```

Again, kindly consult local tech support personnel for issues concerning the use of batch queueing systems and using compute clusters.

2.2. Essential Program Elements

Here is a simple illustrative example of a serial batch program that you would typically run using the **np** package.

```
## This is the serial version of npudensml_npRmpi.R for comparison
## purposes (bandwidth ought to be identical, timing may
## differ). Study the differences between this file and its MPI
## counterpart for insight about your own problems.

library(np)
options(np.messages=FALSE)

## Generate some data

n <- 2500

set.seed(42)

x <- rnorm(n)

## A simple example with likelihood cross-validation
```

[3] You will need to download the **npRmpi** source code and unpack it in order to get Rprofile from the npRmpi/inst directory.

```
t <- system.time(bw <- npudensbw(~x,
                                 bwmethod="cv.ml"))

summary(bw)

cat("Elapsed time =", t[3], "\n")
```

Below is the same code modified to run in parallel using the **npRmpi** package. The salient differences are as follows:

1. You *must* copy the `Rprofile` file from the npRmpi/inst directory of the tarball/zip file into either your root directory or current working directory and rename it `.Rprofile`.
2. You will notice that there are some `mpi.foo` commands where `foo` is, for example, `bcast.cmd`. These are the **Rmpi** commands for telling the slave nodes what to run.

   The first thing we do is initialize the master and slave nodes using the `np.mpi.initialize()` command.
3. Next we broadcast our data to the slave nodes using the `mpi.bcast.Robj2slave()` command which sends an R object to the slaves.
4. After this, we might compute the data-driven bandwidths. Note we have wrapped the **np** command `npudensbw()` in the `mpi.bcast.cmd()` with the option `caller.execute=TRUE` which indicates it is to execute on the master and slave nodes simultaneously.
5. Finally, we clean up gracefully by broadcasting the `mpi.quit()` command.
6. There are a number of example files (including that above and below) in the `npRmpi/demo` directory that you may wish to examine. Each of these runs and has been deployed in a range of environments (Mac OS X, Linux).

```
## Make sure you have the .Rprofile file from npRmpi/inst/ in your
## current directory or home directory. It is necessary.

## To run this on systems with OPENMPI installed and working, try
## mpirun -np 2 R CMD BATCH npudensml_npRmpi. Check the time in the
## output file foo.Rout (the name of this file with extension .Rout),
## then try with, say, 4 processors and compare run time.

## Initialize master and slaves.

mpi.bcast.cmd(np.mpi.initialize(),
              caller.execute=TRUE)

## Turn off progress i/o as this clutters the output file (if you want
## to see search progress you can comment out this command)
```

```
mpi.bcast.cmd(options(np.messages=FALSE),
              caller.execute=TRUE)

## Generate some data and broadcast it to all slaves (it will be known
## to the master node)

n <- 2500

mpi.bcast.cmd(set.seed(42),
              caller.execute=TRUE)

x <- rnorm(n)
mpi.bcast.Robj2slave(x)

## A simple example with likelihood cross-validation

t <- system.time(mpi.bcast.cmd(bw <- npudensbw(~x,
                                               bwmethod="cv.ml"),
                               caller.execute=TRUE))

summary(bw)

cat("Elapsed time =", t[3], "\n")

## Clean up properly then quit()

mpi.bcast.cmd(mpi.quit(),
              caller.execute=TRUE)
```

For more examples including regression, conditional density estimation, and semiparametric models, see the files in the `npRmpi/demo` directory. Kindly study these files and the comments in each in order to extend the parallel examples to your specific problem.

Note that the output from the serial and parallel runs ought to be identical save for execution time. If they are not there is a problem with the underlying code and I would ask you to kindly report such things to me immediately along with the offending code.

3. Summary

The **npRmpi** package is a parallel implementation of the **np** package that can exploit the presence of multiple processors and the MPI interface for parallel computing to reduce the computational run time associated with kernel methods. Run time is inversely proportional to the number of available processors, so two processors will complete a job in roughly one half the time of one processor, ten in one tenth and so forth.[4] Though installation of a working

[4]There is minor overhead involved with message passing, and for small samples the overhead can be substantial as the ratio of message passing to computing the kernel estimator increases - this will be negligible for sufficiently large samples.

MPI implementation requires some familiarity with computer systems, local expertise exists for many and help is to be found there. That being said, the Mac OS X operating system comes stock with a fully functioning version of Open MPI so there is zero additional effort required for the user in order to get up and running in this environment. Finally, any feedback for improvements for this document, reporting of errors and bugs and so forth is always encouraged and much appreciated.

References

Hayfield T, Racine JS (2008). "Nonparametric Econometrics: The np Package." *Journal of Statistical Software*, **27**(5). URL http://www.jstatsoft.org/v27/i05/.

R Development Core Team (2010). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

A. Illustrative Timed Runs

The times reported in Table 1 were generated using R 2.11-0 and Open MPI 1.2.8 on a 2008 vintage MacBook running Snow Leopard 10.6.3 on a 2.4 GHz Intel Core 2 Duo (a completely stock installation). Code was first run in serial mode using the np package version 0.30-9 then in parallel mode with 2 processors using the npRmpi package version 0.30-9. Elapsed time for the np functions is provided (seconds) as is the ratio of the elapsed time for the parallel run to the serial run.

Note that many of these illustrative examples use smallish sample sizes hence the run time with 2 processors will not be 1/2 that with 1 processor due to overhead. But for larger samples (i.e. the ones you actually need parallel computing for, not these toy illustrations) you ought to see an improvement that is inversely related to the number of processors.

B. Known Issues

1. It would be wise to cast all variables when read into R (always good practice) and not do so using the formula interface.

   Casting responses (i.e. stuff to the left of the `~`) works in the serial version but does not appear to work for `npcdensbw`.

2. The functions `npdeptest`, `npdeptest`, `npsymtest` and `npunitest` currently run orders of magnitude slower under MPI when using `method="integration"` (default). This might be related to a limit on the number of nested function calls in R and/or the **cubature** package as I have encountered this in a serial environment in the past (solutions welcomed!).

   For the time being you are advised to use the serial **np** package if you require this function or select `method="summation"` (see `?npdeptest`, `?npdeptest`, `?npsymtest`

Table 1: Illustrative timed runs (seconds) with 1 processor (serial, np package) and 2 processors (parallel, npRmpi package).

Function	Secs(1)	Secs(2)	Ratio
npcdens (ls)	126.3	112.3	0.89
npcdens (ml)	50.1	25.5	0.51
npcdistccdf	68.0	35.7	0.53
npcmstest	73.9	51.9	0.70
npconmode	90.8	37.2	0.41
npdeneqtest	43.2	23.9	0.55
npdeptest	69.0	37.9	0.55
npindex (Ichimura)	36.7	19.9	0.54
npindex (Klein/Spady)	50.2	27.7	0.55
npqreg	95.9	51.7	0.54
npreg (lc, aic)	77.8	57.3	0.74
npreg (lc, ls)	81.8	49.7	0.61
npreg (ll, aic)	86.0	43.7	0.51
npreg (ll, ls)	89.3	46.0	0.52
npscoef	38.1	31.8	0.84
npsdeptest	79.4	58.6	0.74
npsigtest	143.2	87.3	0.61
npsymtest	43.2	30.5	0.71
npudens (ls)	60.9	30.6	0.50
npudens (ml)	25.3	14.5	0.57
npunitest	41.6	28.1	0.67

and `?npunitest` for caveats, though these functions are quite efficient computationally speaking and can handle quite large data sets by default via the serial implementation).

3. The `C` code underlying regression cross-validation currently differs between **np** and **npRmpi**. Both are correct with the latter being a tad slower (on one processor that is - see the timings above for a comparison). Note that, due to the different numerical implementations, each may deliver slightly different bandwidths. I hope to enlist the assistance of my co-creator on this package in the near future so that the same code underlies both packages throughout.

Affiliation:

Jeffrey S. Racine
Department of Economics
McMaster University
Hamilton, Ontario, Canada, L8S 4L8
E-mail: racinej@mcmaster.ca
URL: http://www.mcmaster.ca/economics/racine/