Running R Jobs on HPCC Resources

Introduction

R is a language and software environment for statistical computing and graphics. R provides a wide variety of statistical (linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, clustering, etc...) and graphical techniques and is highly extensible. R is a software package especially suited for data analysis and graphical representation. Functions and results of the analysis are all stored as objects, allowing easy function modification and model building.

Table of Contents

1. Setting up the environment

Modules are used to set up a user's environment. Typically, users initialize their environment when they log in by setting environment information for every application they will reference during the session. The HPCC uses the Environment Modules package as a tool to simplify shell initialization and allow users to easily modify their environment during the session with module files.

For more information about how to load and maintain your software environment using modules, please refer to the user guide "Software Environment Setup".

In order to set the environment variables for R, you will first need to check which versions of R are installed using the "`module spider R`" command. This command will return a description of the software and which versions are currently installed. Alternatively, you can use the HPCC Red Raider Cluster Software Packages search tool online. To load the most recent version of R,

you can run "`module load gcc/10.1.0 r/4.0.2`" on the Nocona partition and "`module load intel R`" or "`module load gnu R`" command on the Quanah partition. You can then run the "`module list`" command to verify that the R module has been successfully loaded. This can be done using one of the following sets of commands:

```
#--- Nocona ---
#To load the version of R compiled with GNU compiler run the following:
module load gcc/10.1.0 r/4.0.2
module list


#--- Quanah ---
#To load the Intel compiled version of R run the following:
module load intel R
module list

#To load the GNU compiled version of R run the following:
module load gnu R
module list
```

2. How to install R in Conda

You may find that you often want to use a different version of R that is not yet built on our cluster module. This can be accomplished by creating a separate Conda environment where your desired version of R can be installed. If you don't have Conda installed in your system, please read through the installing Conda and setting up Conda environments guide before following through with the steps given below.

The following are the steps to install R in conda.

Step 1. Create a conda environment with a name, `R_nocona` for example.

```
conda create -n R_nocona
```

Step 2. Activate the created conda environment.

```
conda activate R_nocona
```

Step 3. Install R and the essential R packages through R channel.

```
conda install -c r r r-essentials
```

(Note: The above command will install the latest version of R that is available in the r channel. HPCC recommends every user to use the latest version of R as it gives better performance, and more functionalities will be included in the latest versions.)

If you want to install a specific version of R, use the following syntax.

```
conda install -c r r=version_number r-essentials
```

The following command gives the list of R versions available.

```
conda search -c r r
```

E.g:

```
conda install -c r r=3.6.0 r-essentials
```

Once done, you may type "R" to start running R.

Note: *The Conda installation of R will compile the R packages on the local system. For example, Nocona is equipped with AMD processors while Quanah uses Intel processors, meaning the CPU instruction sets are different among those partitions. Therefore, you should get into a habit of creating a different Conda environment for each partition.*

3. How to write a parallel multi-core R script

Running R jobs in parallel can improve the performance of your code by allowing multiple tasks to be executed simultaneously. The HPCC strongly recommends that you have a good grasp of what your code is doing and what resource requirements it will need to function properly before running it on the cluster. By default, R will always attempt to run serially, using only a single core. If you wish to run across multiple cores, there are several ways to achieve this in R using different packages.

3.1. Parallel packages

R packages provide built-in functions for parallel computing, including `mclapply()`, `mcmapply()`, and `parLapply()`. These packages allow you to run multiple R processes in parallel and can significantly speed up your computations.

The below is a list of useful parallel packages:

- **parallel:** The parallel package is included in R's base installation and provides a high-level interface for parallel computing on local machines or clusters.
- **snow:** The snow package provides a simple interface for parallel computing using a variety of parallel backends, including local sockets, MPI, and PVM.
- **foreach:** The foreach package provides a parallel foreach loop construct that can be used with a variety of parallel backends, including parallel, snow, and doParallel.

- **doParallel:** The doParallel package provides a backend for foreach that enables parallel computing using the parallel package.
- **doMPI:** The doMPI package provides a backend for foreach that enables parallel computing using the MPI standard.
- **doSNOW:** The doSNOW package provides a backend for foreach that enables parallel computing using the snow package.
- **future:** The future package provides a simple and unified API for parallel computing using a variety of parallel backends, including parallel, snow, and doParallel.
- **BiocParallel:** The BiocParallel package provides a parallel computing framework for use with bioinformatics tools and workflows.

3.2. How to use R parallel packages

When writing your R code, you may want to pay close attention to the packages you make use of. There often exists numerous packages that attempt to solve the same problem, and in many of these cases you will find that some are serial while others are parallel. Whenever possible, you should try to use parallel packages versus serial ones. You may also want to look into the documentation for the packages you are using to see if they have the ability to run in a parallel fashion.

Several packages are available in R for parallel computing, including `foreach`, `doParallel`, and `parallelMap`. These packages provide additional functionality and control over parallel execution, such as specifying the number of workers and managing the distribution of data.

Before using a parallel package, make sure you install it using the following command.

```
install.packages("package_name")
```

To view a description of a particular package, use the following command.

```
packageDescription("package_name")
```

For instance:

To install "`doParallel`" package:

```
install.packages("doParallel")
```

By default R installs these packages in a system directory that requires admin privileges, If you want to install the package in a separate directory that you have write access use the following command

```
install.packages("package_name", lib="path")
```

E.g: Installing the "doParallel" package in R_packages directory in home.

```
install.packages("doParallel", lib="~/R_packages")
```

You should expect a list of mirrors to select for your installation. You can go with the closest mirror:

```
--- Please select a CRAN mirror for use in this session ---
Secure CRAN mirrors

 1: 0-Cloud [https]
 2: Australia (Canberra) [https]
 3: Australia (Melbourne 1) [https]
 4: Australia (Melbourne 2) [https]
 5: Australia (Perth) [https]
 6: Austria [https]
 7: Belgium (Brussels) [https]
 8: Brazil (PR) [https]
        ……
       ……
       ……
 70: USA (IA) [https]
 71: USA (MI) [https]
 72: USA (MO) [https]
 73: USA (OH) [https]
 74: USA (OR) [https]
 75: USA (TN) [https]
```

If you don't want to select the CRAN mirror every time you install a package, follow the steps.

Step 1. Go to your home directory.

```
cd ~
```

Step 2. open .Rprofile file.

```
vi .Rprofile
```

Step 3. paste the script "options(repos = c(CRAN = "https://cran.rstudio.com/"))" and save the file.

```
options(repos = c(CRAN = "https://cran.rstudio.com/"))
```

This sets the default repository to RStudio's CRAN mirror.

Now, open r terminal and install the required packages.

To view description of "doParallel" package:

```
packageDescription("doParallel")
```

You should expect the description of the package as shown below:

```
> packageDescription("doParallel")
Package: doParallel
Type: Package
Title: Foreach Parallel Adaptor for the 'parallel' Package
Version: 1.0.17
Authors@R: c(person("Folashade", "Daniel", role="cre",
        email="fdaniel@microsoft.com"), person("Microsoft",
        "Corporation", role=c("aut", "cph")), person("Steve", "Weston",
        role="aut"), person("Dan", "Tenenbaum", role="ctb"))
Description: Provides a parallel backend for the %dopar% function using
        the parallel package.
Depends: R (>= 2.14.0), foreach (>= 1.2.0), iterators (>= 1.0.0),
        parallel, utils
Suggests: caret, mlbench, rpart, RUnit
Enhances: compiler
License: GPL-2
URL: https://github.com/RevolutionAnalytics/doparallel
BugReports: https://github.com/RevolutionAnalytics/doparallel/issues
NeedsCompilation: no
Packaged: 2022-01-16 17:54:13 UTC; folashade
Author: Folashade Daniel [cre], Microsoft Corporation [aut, cph], Steve
        Weston [aut], Dan Tenenbaum [ctb]
Maintainer: Folashade Daniel <fdaniel@microsoft.com>
Repository: CRAN
Date/Publication: 2022-02-07 12:50:02 UTC
Built: R 4.0.2; ; 2023-04-20 18:02:47 UTC; unix
-- File: /home/lganji/R/x86_64-pc-linux-gnu-
library/4.0/doParallel/Meta/package.rds
```

The following are a few examples of usage of Parallel packages:

Example 1: "foreach" package

The foreach package in R provides a simple and flexible way to perform parallel computing using the "foreach" loop construct. It allows you to execute R code on multiple cores or even distributed across multiple machines. Here are the steps to use the foreach package in R:

Step 1. Load the `foreach` package and its parallel backend `doParallel` using the `library()` function.

Step 2. Define the number of cores to use for parallel processing (35 in this case).

Step 3. Create a cluster object using the `makeCluster()` function, specifying the number of cores to use.

Step 4. Register the cluster with the `doParallel` package using the `registerDoParallel()` function.

Step 5. Use the `foreach()` function to perform parallel processing on a list or vector.

E.g:

```
library(doParallel)
library(foreach)
n.cores <- 35
my.cluster <- parallel::makeCluster(n.cores)
doParallel::registerDoParallel(cl = my.cluster)
data <- list(a = 1:10, b = 11:20, c = 21:30)
results <- foreach(x = data) %dopar% mean(x)
results
```

In this example, `foreach()` applies the `mean()` function to each element of the data list in parallel using 35 cores. The `%dopar%` operator indicates that the loop should be executed in parallel.

Example 2: "`parallel`" package

The parallel package in R provides a convenient way to perform parallel computing. It allows you to execute multiple R processes in parallel on a single machine, or on a cluster of machines. Here are the steps to use the parallel package in R.

E.g:

```
library(parallel)
n_cores <- 35
cl <- makeCluster(n_cores)
x <- 1:10
result <- parLapply(cl, x, function(i) i^2)
```

The `parLapply()` function splits the input vector x into chunks and distributes them across the cores in the cluster for processing. The results are then combined into a list result.

Note that there are other functions in the `parallel` package, such as `parSapply()`, `parLapplyLB()`, and `mclapply()`, that can also be used for parallel processing in R.

Example 3: "`future`" package

This package provides a simple and unified way to create and manage parallel processes in R. It is also able to process results via a future framework without blocking other current R processes. **future** package provides other functions and options for parallel and asynchronous processing, such as `future_lapply()`, `future_map()`, and `future_imap()`, which are similar to the `lapply()`, `map()`, and `imap()` functions in the `purrr` package. You can use the `future_lapply()` function to apply a function to a list in parallel.

E.g:

```
library(future)
plan(multiprocess, workers = 2)
data <- list(a = 1:10, b = 11:20, c = 21:30)
results <- future_lapply(data, function(x) mean(x))
```

In this example, `future_lapply()` applies the `mean()` function to each element of the data list in parallel using 2 cores. The `plan()` function specifies the parallel backend to be used.

Note: Not all R code is easily parallelizable, and parallel execution may not always result in faster performance. Benchmarking your code and experimenting with different parallelization strategies is important to find the most effective approach for your specific use case.

To know more about R packages, please visit R packages

4. Submitting R jobs to the Nocona Partition

For this tutorial we will be relying on an example script that will perform some R benchmarking tests:

4.1. Submitting a Serial R job to the Nocona partition:

Step 1. Copy the serial R example directory into your home directory.

```
cp -r /lustre/work/examples/nocona/R_serial ~/R_serial_nocona
```

Step 2. Enter the newly created directory.

```
cd ~/R_serial_nocona
```

Step 3. Review the serial R job submission script and the R script files.

```
cat R_nocona.sh
cat serTest.R
```

Step 4. Submit serial R example script into Nocona partition.

```
sbatch R_nocona.sh
```

Your R script is now queued to run on the Nocona partition. You can use the "`squeue --me`" command to check the status of the job. Please read the Job Submission Guide for more information about running jobs and checking their status.

4.2. Submitting a Parallel R job to the Nocona partition:

The tutorial steps for submitting parallel R jobs to the Nocona partition is similar to the serial R job submission, except you need to copy a parallel R example directory into your home directory.

```
cp -r /lustre/work/examples/nocona/R-parallel ~/R-parallel_nocona
```

Under this directory, you may see different parallel R example subdirectories.

```
~/R-parallel_nocona/R_foreach
~/R-parallel_nocona/R_mclapply
~/R-parallel_nocona/R_parLapply
```

You can follow the step 2-4 from the serial R job submission section in order to test each parallel R job on Nocona partition. Please read the Job Submission Guide for more information about running jobs and checking their status.

5. Submitting R jobs to the Quanah Partition

For this tutorial we will be relying on an example scripts that will perform some R benchmarking tests:

5.1. Submitting a Serial R job to the Quanah partition

Step 1. Copy the serial R example directory into your home directory.

```
cp -r /lustre/work/examples/quanah/R_serial ~/R_serial_quanah
```

Step 2. Enter the newly created directory.

```
cd ~/R_serial_quanah
```

Step 3. Review the serial R job submission script and the R script files.

```
cat R_quanah.sh
cat serTest.R
```

Step 4. Submit serial R example script into Quanah partition.

```
sbatch R_quanah.sh
```

Your R script is now queued to run on the Quanah partition. You can use the "`squeue --me`" command to check the status of the job. Please read the Job Submission Guide for more information about running jobs and checking their status.

5.2. Submitting a Parallel R job to the Quanah partition

Step 1. Copy the Parallel R example directory into your home directory.

```
cp -r /lustre/work/examples/quanah/R-parallel ~/R-parallel_quanah
```

Step 2. Enter the newly created directory.

```
cd ~/R-parallel_quanah
```

Under this directory you can find different parallel R example subdirectories.

```
~/R-parallel_quanah/R_foreach
~/R-parallel_quanah/R_mclapply
~/R-parallel_quanah/R_parLapply
```

Step 3. Go to any of those directories and review the scripts.

```
cat R_quanah.sh
cat parTest.R
```

Step 4. Submit parallel example script to Quanah partition.

```
sbatch R_quanah.sh
```

Your R script is now queued to run on the Quanah partition. You can use the "`squeue --me`" command to check the status of the job. Please read the Job Submission Guide for more information about running jobs and checking their status.

5. Checking If a Job is Running in Parallel

To know the status and details of your submitted jobs, use the following command:

```
squeue --me
```

The above command provides information about your jobs on the Slurm scheduling queue, including your job ID, partition name, number of CPUs assigned, name of the assigned node, etc.

```
-bash-4.2$ sbatch R_quanah.sh
Submitted batch job 8970097
-bash-4.2$
-bash-4.2$
-bash-4.2$ squeue --me
     JOBID  PARTITION  PRIORI  ST      USER      NAME        TIME  NODES   CPUS      QOS  NODELIST(REASON)

   8970097     quanah    4859   R    lganji    r-test        0:05      1     36   normal  cpu-7-43
```

```
-bash-4.2$ ls
parTest.R  R_quanah.sh  r-test.e8970097  r-test.o8970097
-bash-4.2$
```

Once your job is completed, you may check your results from the error and output files.

From the above example, your job is assigned to worker node `cpu-7-43`. You may check how your job is running on that node by first logging into the node with the command "`ssh cpu-7-43`" (The name of the CPU node list might differ in your system)

E.g:

```
ssh cpu-7-43
```

Then, type the command "`top`" to see the running processes of that node.

```
-bash-4.2$ ssh cpu-7-43
Last login: Tue Mar 14 13:52:07 2023 from quanah.localdomain
-bash-4.2$ top

top - 13:54:19 up 33 days, 18 min,  1 user,  load average: 0.32, 2.93, 9.13
Tasks: 550 total,   2 running, 548 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2.1 us,  0.7 sy,  0.0 ni, 97.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 19797656+total, 19274220+free,  4490936 used,   743412 buff/cache
KiB Swap: 24366796+total, 24366796+free,        0 used. 19227814+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
110412 lganji    20   0  307012  45424   6852 S   9.6  0.0   0:00.29 R
110477 lganji    20   0  307020  45428   6848 S   9.6  0.0   0:00.29 R
110386 lganji    20   0  307020  47440   6848 S   9.3  0.0   0:00.28 R
110490 lganji    20   0  307016  45424   6848 S   9.3  0.0   0:00.28 R
110399 lganji    20   0  307012  45420   6848 S   8.9  0.0   0:00.27 R
110425 lganji    20   0  307012  45424   6852 S   8.9  0.0   0:00.27 R
110438 lganji    20   0  307012  45420   6848 S   8.9  0.0   0:00.27 R
110451 lganji    20   0  307020  45428   6848 S   8.9  0.0   0:00.27 R
110464 lganji    20   0  307016  45424   6848 S   8.9  0.0   0:00.27 R
110373 lganji    20   0  307016  45424   6848 S   5.0  0.0   0:00.26 R
110503 lganji    20   0  204196  20328   4856 R   2.3  0.0   0:00.07 R
110050 lganji    20   0  314160  52564   6804 S   1.0  0.0   0:00.55 R
110368 lganji    20   0   58756   2620   1488 R   0.7  0.0   0:00.04 top
```

You can then confirm that the job is running in parallel with multiple processes running R.