Enable Multithread with data.table in Mac/Intel chips

Jorge Roa
jorgeroa@stanford.edu

Fernando Alarid-Escudero
falarid@stanford.edu

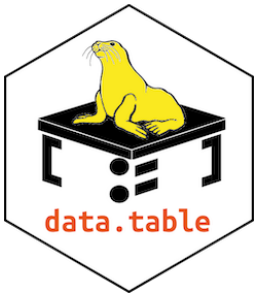


Table of contents

1 Introduction

This document shows you how to enable the use of multiple cores on Macs with Intel/Apple silicon chips (M1, M2, and M3). With the increasing shift towards multi-core processors, is crucial to employ them for improving performance, especially for data-intensive operations. Many applications, such as data.table in R, can benefit significantly from multithreading, allowing them to distribute tasks across multiple processor cores simultaneously.

Historically, enabling multithreading on macOS with Intel chips was straightforward. However, with the introduction of Apple silicon chips (like the M1, M2, and M3), there have been changes in the architecture that require a different approach to activate them.

2 Initial setup

If we load data.table library in R studio, this message will appear in your console:

```
> library(data.table)
data.table 1.14.6 using 1 threads (see ?getDTthreads).  Latest news: r-datatable.com
**********
This installation of data.table has not detected OpenMP support. It should still work but in single-threade
d mode.
This is a Mac. Please read https://mac.r-project.org/openmp/. Please engage with Apple and ask them for sup
port. Check r-datatable.com for updates, and our Mac instructions here: https://github.com/Rdatatable/data.
table/wiki/Installation. After several years of many reports of installation problems on Mac, it's time to
gingerly point out that there have been no similar problems on Windows or Linux.
**********
```

As you can see, OpenMP support is needed to use multiple cores in Macs. Therefore, we must install those packages through the terminal and set the required paths to run OpenMP.

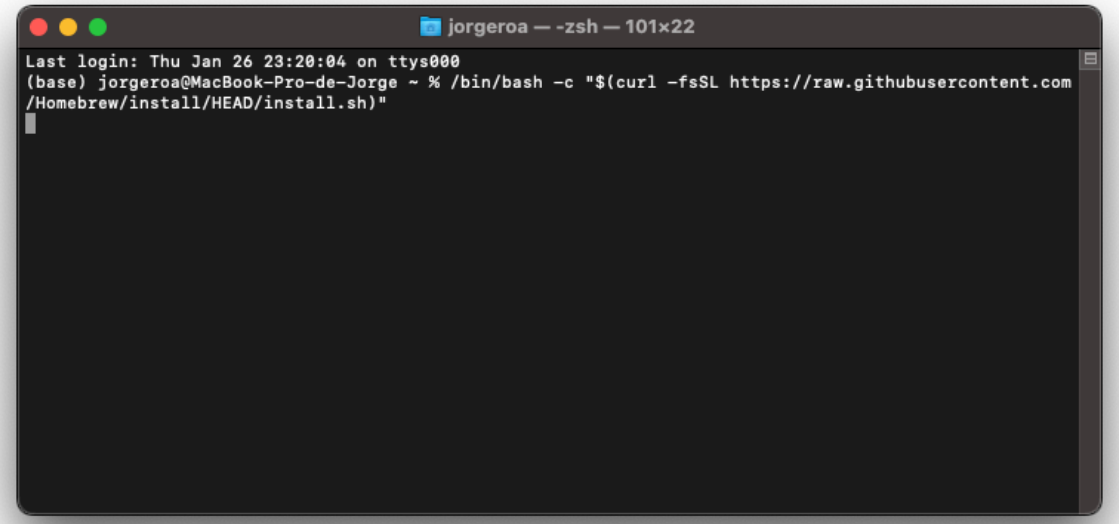
3 Prerequisites

1) Have the latest version of < fa brands r-project > studio.

2) Have the latest version of Mac Ventura

4 First Step

1) Open your terminal. It should see like this:



5 Second Step (Install homebrew)

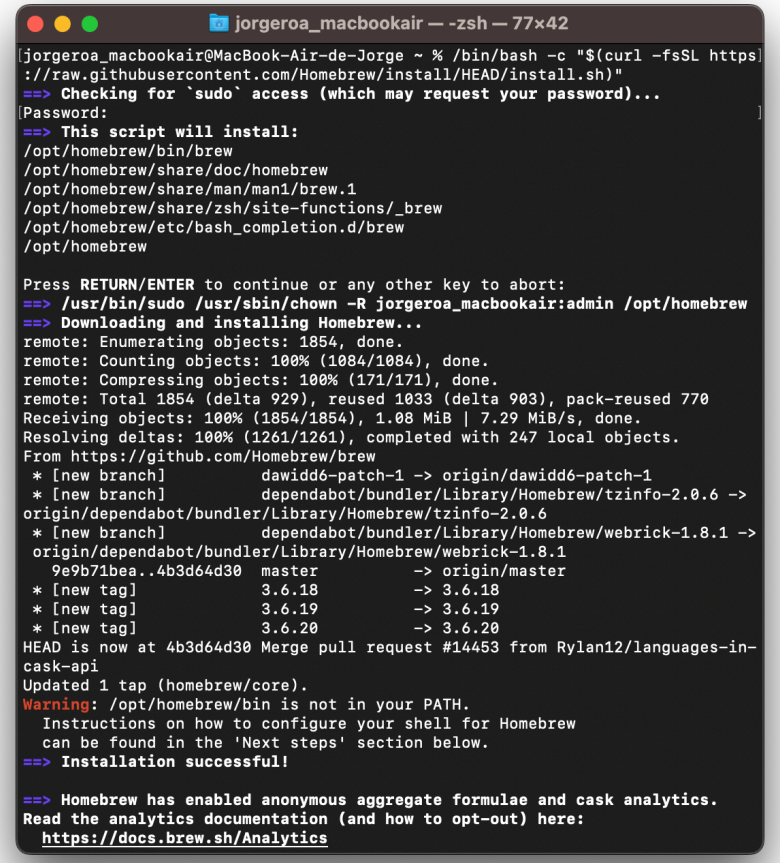
is an open-source software package management system that makes installing applica-

tions on Linux and Apple's macOS operating systems easier. Homebrew will help us install `OpenMP`. Open Multi-Processing allows us to run applications in parallel to efficient processes. In this case, we want to employ the multiple cores that Mac has for `data.table` wrangling.

As the Homebrew page states, Homebrew installs the stuff you need that Apple (or your Linux system) didn't. So, once we open our terminal, we need to paste this command on our terminal.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

To install `homebrew`, you need to enter your password and press `ENTER`. After installing homebrew, you should see in your terminal this screen:



6 Third Step (Check package and change paths)

We need to be sure that homebrew was installed in the correct path. For this, we can type in the terminal:

```
brew help
```

Set our paths properly if the command is not recognized.

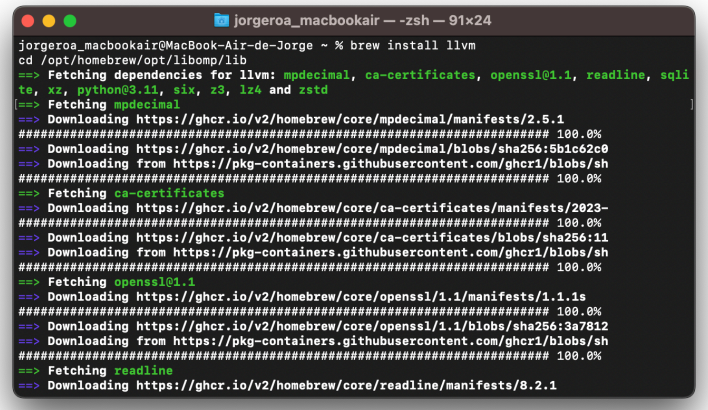
Chip Type	Command to Set Path
Apple Silicon (M1, M2, M3)	`export PATH=/opt/homebrew/bin:$PATH`
Intel	`export PATH=/usr/local/opt/homebrew/bin:$PATH`

7 Fourth Step: Installing required packages (libopenmp, libopenmpt, llvm, cask, ggc)

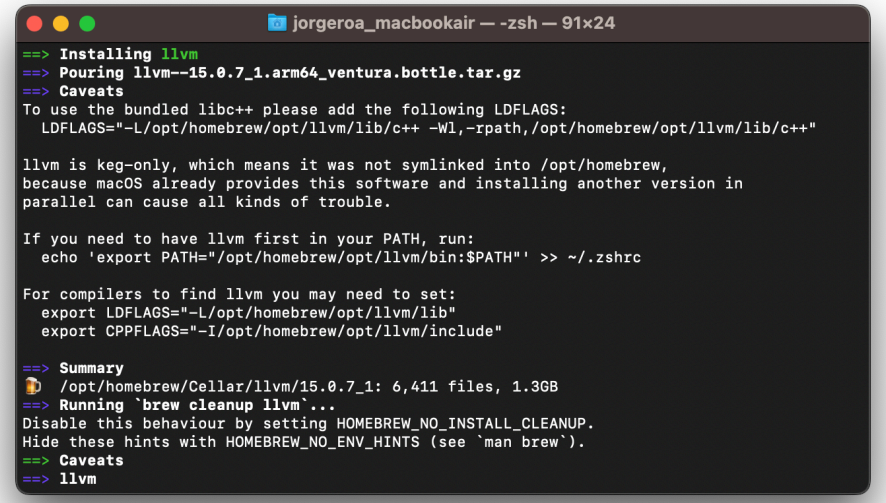
7.1 Install llvm

We need to install the llvm package to access the clang compiler, which helps us to set multithreading.

```
brew install llvm
```



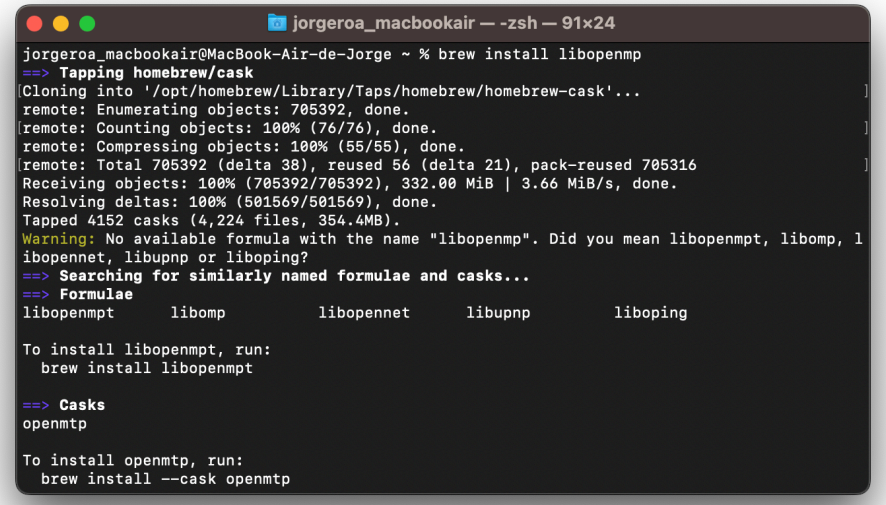
Once the installation is finished, we should see this screen in the terminal. It's worth mention that this library



7.2 Install libopenmp

```
brew install libopenmp
```

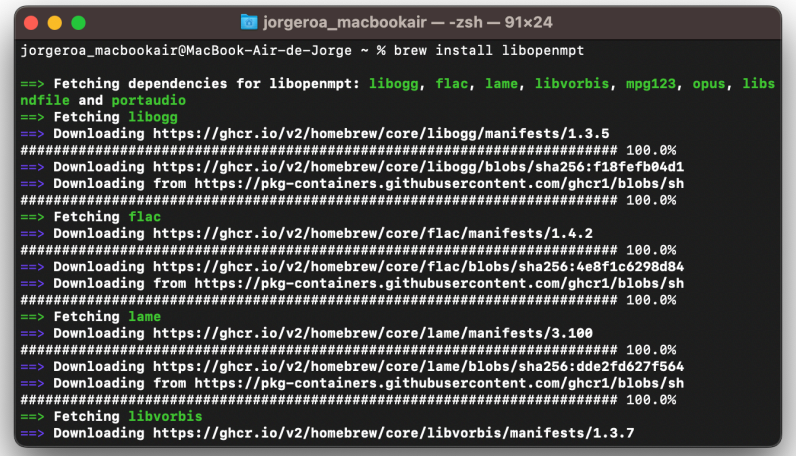
This is the screen that you should see once the installation is done.



7.3 Install libopenmpt

```
brew install libopenmpt
```

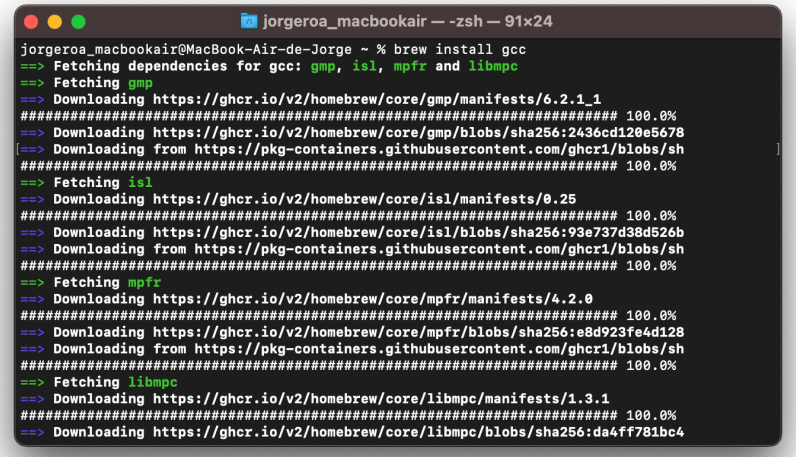
This is the screen that you should see once the installation is done.



7.4 Install gcc

```
brew install gcc
```

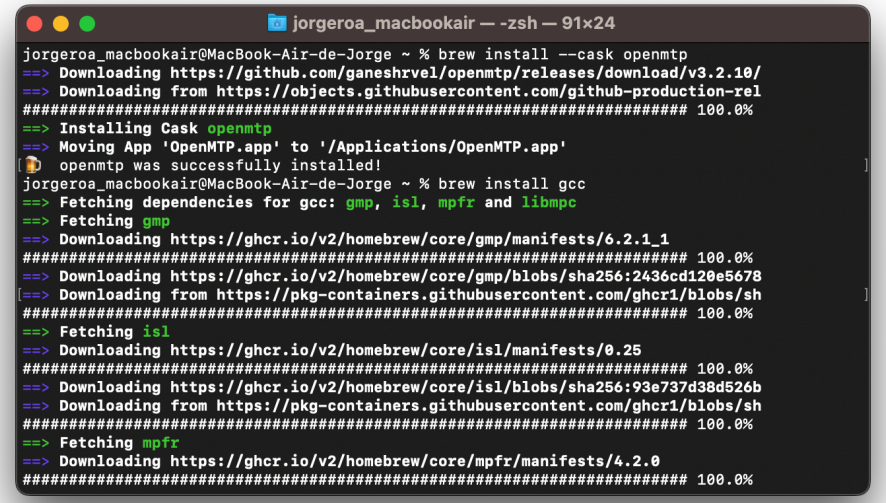
This is the screen that you should see once the installation is done.



7.5 Install –cask openmtp

```
brew install --cask openmtp
```

This is the screen that you should see once the installation is done.



8 Fifth Step: Create .R folder and Makevars file.

We need to create a text file called `Makevars`. This file is necessary because we need to set a file with multiple paths where we will retrieve the various packages we have installed in our R environment. We need to open a new script in our R studio project or open our terminal to execute the next commands:

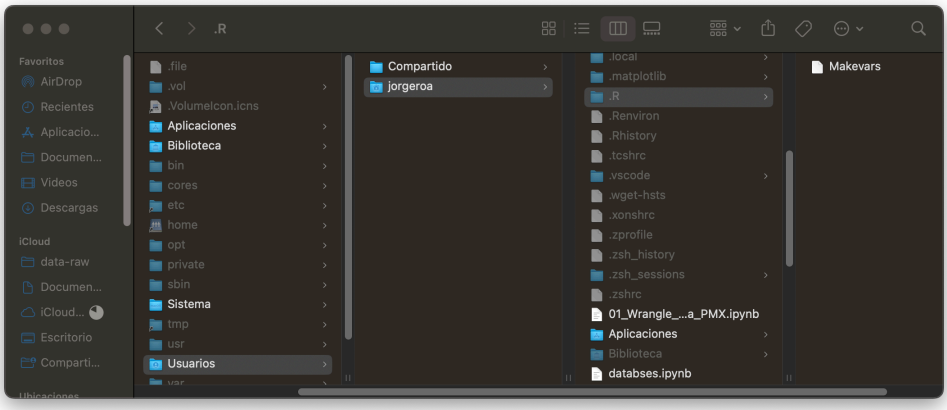
```
dir.create('~/.R')


file.create('~/.R/Makevars')
```

`dir.create` creates a hidden folder in our environment, and `file.create` creates a text file in the folder we just created. This folder is located in our usrs folder. You can access your user folder and your hidden folders with the next shortcuts:

Action	Mac Shortcut	Windows Shortcut
Go to your home folder	`CMD + Shift + H`	`Win + E`
Show your hidden folders	`CMD + Shift + .`	`Ctrl + H` or `Alt + V, H`
Open File Explorer/Finder	`CMD + N (in Finder)`	`Win + E`
Search files	`CMD + Space`	`Win + S` or `Ctrl + F`
Open Task Manager/Activity Monitor	`CMD + Option + Esc`	`Ctrl + Shift + Esc`
Switch between windows	`CMD + Tab`	`Alt + Tab`

You should have this path (according to your chip).



9 MOST IMPORTANT STEP: create makevars file and specify paths

This is the most important step to make the multithreading work. We need to modify correctly the `Makevars` file and specifiy the paths.

10 Sixth Step: Create our paths in our Makevars file.

Once we created our `Makevars` file, we need to edit it with the next command:

```
file.edit('~/.R/Makevars')
```

A new window will open with the name of the file. Now, we must paste the following paths according to our chip (Apple Silicon or Intel). this step is also important because here we

are indicating `data.table` where the libraries installed from `brew` are. If we don't follow this steps, `data.tabe`will be never able to unlock the multithreading.

11 Apple Silicon

11.1 Latest version (October 2024)

```
LDFLAGS += -L/opt/homebrew/opt/libomp/lib
CPPFLAGS += -I/opt/homebrew/opt/libomp/include
```

11.2 Older versions

If by any reasons, the latest version doesn't work once you completed all the steps of this document, what you can do is replace this for paths that used to work in other MacOS versions.

11.2.1 First option

```
LDFLAGS += -L/opt/homebrew/opt/libomp/lib -lomp
CPPFLAGS += -I/opt/homebrew/opt/libomp/include -Xclang -fopenmp
```

11.2.2 Second option

```
HOMEBREW_LOC=/opt/homebrew
LLVM_LOC=$(HOMEBREW_LOC)/opt/llvm
CC=$(LLVM_LOC)/bin/clang -fopenmp
CXX=$(LLVM_LOC)/bin/clang++ -fopenmp
CFLAGS=-g -O3 -Wall -pedantic -std=gnu99 -mtune=native -pipe
CXXFLAGS=-g -O3 -Wall -pedantic -std=c++11 -mtune=native -pipe
LDFLAGS=-L$(HOMEBREW_LOC)/opt/gettext/lib -L$(LLVM_LOC)/lib -Wl,-rpath,$(LLVM_LOC)/lib
CPPFLAGS=-I$(HOMEBREW_LOC)/opt/gettext/include -I$(LLVM_LOC)/include
```

12 Intel

12.1 Latest version (October 2024)

```
LDFLAGS += -L/opt/homebrew/opt/libomp/lib
CPPFLAGS += -I/opt/homebrew/opt/libomp/include
```

12.2 Older versions

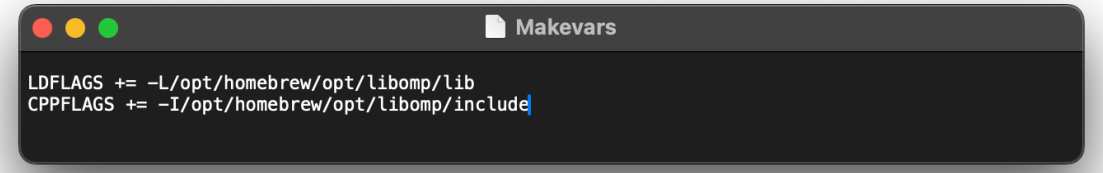
12.2.1 First option

```
LDFLAGS += -L/opt/homebrew/opt/libomp/lib -lomp
CPPFLAGS += -I/opt/homebrew/opt/libomp/include -Xclang -fopenmp
```

12.2.2 Second option

```
HOMEBREW_LOC=/usr/local
LLVM_LOC=$(HOMEBREW_LOC)/opt/llvm
CC=$(LLVM_LOC)/bin/clang -fopenmp
CXX=$(LLVM_LOC)/bin/clang++ -fopenmp
CFLAGS=-g -O3 -Wall -pedantic -std=gnu99 -mtune=native -pipe
CXXFLAGS=-g -O3 -Wall -pedantic -std=c++11 -mtune=native -pipe
LDFLAGS=-L$(HOMEBREW_LOC)/opt/gettext/lib -L$(LLVM_LOC)/lib -Wl,-rpath,$(LLVM_LOC)/lib
CPPFLAGS=-I$(HOMEBREW_LOC)/opt/gettext/include -I$(LLVM_LOC)/include
```

At the end, the `Makevars`file should look like this:



Once you put the paths, save the text file and close it. The difference between Apple Silicon and Intel is just the path; everything else remains similar. For Apple Silicon, the path is `/opt/homebrew` and for Intel is `/usr/local`.

13 Seventh Step: Reinstall data.table and set your cores.

Finally, we need to remove data.table and reinstall it via `source`. THIS STEP IS NECESSARY.

```
remove.packages("data.table")
install.packages('data.table', type='source')
```

If everything was done correctly, you should load the package with library(data.table) and see the following message.

```
** building package indices
** installing vignettes
** testing if installed package can be loaded from temporary location
** checking absolute paths in shared objects and dynamic libraries
** testing if installed package can be loaded from final location
** testing if installed package keeps a record of temporary installation path
* DONE (data.table)

The downloaded source packages are in
        '/private/var/folders/7r/cc5chw4n1yn8ny0c_nq8bny80000gn/T/RtmpY8sEUe/downloaded_packages'
> library(data.table)
data.table 1.14.6 using 4 threads (see ?getDTthreads).  Latest news: r-datatable.com
```

Remember that you can set more threads with `setDTthreads` (according to the specifications of your computer). Happy coding!