AquaFortR: Streamlining Atmospheric Science, Oceanography, Climate, and Water Research with Fortran-accelerated R

Ahmed Homoudi[1]

[1]TUD Dresden University of Technology, Institute of Hydrology and Meteorology

08 April 2024

Table of contents

Welcome to the enlightening journey through the pages of **"AquaFortR: Streamlining Atmospheric Science, Oceanography, Climate, and Water Research with Fortran-accelerated R"**. In this book, you will gain invaluable insights into seamlessly speeding up R scripts by harnessing the power of Fortran. You will acquire essential perspectives into speeding up your package using simple C and Fortran codes. Furthermore, you will accumulate tweaks to accelerate your scripts or packages, and supplementary reading will prove to be both advantageous and highly beneficial for further optimization, and efficiency.

This work has been funded by the German Research Foundation (DFG) through the project NFDI4Earth (DFG project no. 460036893, https://www.nfdi4earth.de/) within the German National Research Data Infrastructure (NFDI, https://www.nfdi.de/).

Preface

Chapter 1

Introduction

This chapter briefly introduces the R programming language and how to install R on different operating systems. Furthermore, we will have a look at the installation of RStudio. Finally, we will learn briefly about Fortran.

The book is designed to cater to individuals with a foundational understanding of programming, irrespective of their familiarity with a specific programming language.

1.1 R

R is a powerful and versatile open-source language and environment for statistical computing and graphics (R Core Team, 2023). R was developed to facilitate data manipulation, exploration, and visualisation, providing various statistical and graphical tools.

The R environment is designed for effective data handling, array and matrix operations, and is a well-developed and simple programming language (R Core Team, 2023). Its syntax is concise and expressive, making it an accessible language for newbies and seasoned programmers. R can perform tasks either by executing scripts or interactively. The latter is advantageous for beginners and during the first stages of script development. The interactive environment is accessible through command lines or various integrated development environments (IDEs), such RStudio.

Many factors play an important role in the popularity of R. For example, the ability to produce graphics with a publication's quality. Simultaneously, the users maintain full control over customising the plots according to their preferences and intricate details. Another significant factor is its ability to be extended to fit the user's demand. The Comprehensive R Archive Network (CRAN) contains an extensive array of libraries and packages to extend R's functionality for various tasks. With over 20447 available packages, the CRAN package repository is a testament to R's versatility and adaptability.

Tidyverse is the most popular bundle of R packages for data science. It was developed by Hadley Wickham and his team. The common shared design among tidyverse packages increases the consistency across functions and makes each new function or package a little easier to comprehend and utilise (Wickham et al., 2023).

Many packages have been developed for spatial data analysis to address various challenging tasks. R-spatial provides a rich set of packages for handling spatial or spatio-temporal data. For example, sf provides simple features access in R, which is a standardized method for encoding spatial vector data. The stars package aims to handle spatiotemporal arrays. Additionally, the terra package works with spatial data and has the ability to process large datasets on the disk when loading into memory (RAM) is not feasible.

Furthermore, research produces a large data sets; therefore, it is essential to store them according to the FAIR principles (**F**indability, **A**ccessibility, **I**nteroperability, and **R**euse of digital assets; Wilkinson et al. (2016)). NetCDF and HDF5 are among the most prominent scientific data formats owing to their numerous capabilities. The R package ncdf4 delivers a high-level R interface to data files written using Unidata's netCDF library. Additionally, rhdf5 provides an interface between HDF5 and R.

Regarding the integration of other programming languages, R has diverse interfaces, which are either a fundamental implementation of R or attainable via another R package (Chambers, 2016). The fundamental interfaces are `.Call()`, `.C()`, and `.Fortran()` to C and Fortran. The development of the Rcpp package has revolutionised seamless access to C++. Python is also accessible using the reticulate package. Finally, the Java interface was granted using the rJava package.

In conclusion, R is a valuable tool in the Earth System, as it can effectively tackle multifarious scientific tasks and address numerous outstanding research inquiries.

1.2 Fortran

Fortran, short for **For**mula **Tran**slation, is one of the oldest high-level programming languages. It was first developed in the 1950s, and is nonetheless widely used in scientific and engineering applications. Key features of Fortran include its ability to efficiently handle arrays and matrices, making it well-suited for numerical computations. Additionally, Fortran has a simple and straightforward syntax that makes it easy to learn and use, because it is possible to write mathematical formulas almost as they written in mathematical texts (Metcalf et al., 2018).

Fortran has been revised multiple times, with the most recent iteration being Fortran 2023. Another important feature of Fortran is its support for parallel programming, enabling developers to take advantage of multicore processors and high-performance computing architectures.

There are frequently numerous good reasons to integrate different programming languages to achieve tasks. Interoperability with C programming language is a feature that was introduced with Fortran 95 (Metcalf et al., 2018). Given that C is widely used for system-level programming, many of the other languages include support for C. Therefore, the C Application Programming Interface (API) can also be used to connect two non-C languages (Chirila & Lohmann, 2014). For instance, in atmospheric modelling, Fortran is used for its high performance and capcity to handle large data sets, while C is utilised for its efficiency and control over memory usage.

Noteworthy, developing software using Fortran necessitates utilisation of its primitive procedures and developing from scratch. This is because Fortran is not similar to scripting languages (i.e. R) that requires a special environment (Masuda, 2020). However, Fortran is privileged with its persistent backward compatibility, resulting in the usability of countless (legacy) codes written decades ago.

Since R was designed to streamline data analysis and statistics (Wickham, 2015) and Fortran is renowned for its high performance, it makes seance to integrate the two languages. It should be noted that both programming languages present arrays in column-major order, which makes it easier to bridge without causing confusion. Additionally, R is developed using Fortran, C, and R programming languages, and it features `.Call` and `.External()` functions that allows users to utilise compiled code from other R packages.

Despite the development of newer programming languages, Fortran remains a popular choice for many scientists and engineers due to its reliability, efficiency, and ability to handle large amounts of data.

1.3 Installation

R

Installation of R differs according to the operating system:

- The webpage here provides information on installing R according to the Linux distribution.
- For Windows, The executable can be downloaded from here. Additionally, Previous releases of R for Windows exit.
- For macOS, various releases and versions are accessible here.

RStudio

As mentioned earlier, RStudio is an IDE for R. Although it is the most prominent, other IDEs exist, such as Jupyter Notebook and Visual Studio Code. In this book, RStudio will be the main IDE. To install Rstudio, visit the posit page to download the suitable installers.

Fortran

Fortran doesn't require an explicit installation, unlike interpreted languages such as R. The source code would be translated to the machine language using the Fortran compiler, and then it can be executed. Therefore, it is important to make sure that a Fortran compiler, i.e., the GNU Fortran (gfortran) compiler, exists in the working machine.

- In the majority of Linux distributions, the GCC compilers, including gfortran, come pre-installed.
- For Windows, installing RTools should ensure the existence of gfortran
- For macOS, binaires for gfortran are avaiable here. Furthermore, more information is available on R for macOS here

Chapter 2

Accelerate R Scripts with Fortran

In this chapter, we will compare the efficiency of running three computationally demanding examples in pure R script versus using another R script with the core computations performed in Fortran.

2.1 2D Coss-Correlation

In signal processing, cross-correlation is a measure of the similarity between two signals as a function of the displacement of one relative to the other (Wang, 2019). It can deliver information about the time lag between the two signals. Cross-correlation is often applied in computer vision for visual tracking. For example, it is used in template matching, feature detection, and motion tracking. 2D cross-correlation also plays an important role in convolutional networks and machine learning.

In atmospheric science, oceanography, climate, and water research, 2D cross-correlation can be applied in various ways. For example, it can be used to estimate ocean surface currents (Warren et al., 2016), cloud tracking using satellite imagery (Seelig et al., 2021), and Particle Image Velocimetry (PIV) in fluid dynamics applications (Willert & Gharib, 1991).

The 2D cross-correlation of an array F, with dimension (M, N), and array G, with dimension (P, Q), can be given as the array CC that has a dimension of $(M+P-1, N+Q-1)$.

$$CC_{(s,t)} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} F_{(m,n)} G_{(m-s,n-t)} \tag{2.1}$$

where s varies between $-(P-1)$ and $(M-1)$ while t varies between $-(Q-1)$ and $(N-1)$. F and $G \in R$.

Now, let us define the `xcorr2D_r` function as shown in Listing 2.1. The function receives two matrices or arrays `a` & `b` and return the full cross-correlation plane `cc`.

Moving forward, we can define the `xcorr2d_f` subroutine in Fortran as shown in Listing 2.2. Subroutines are generally the approach for integrating Fortran in R. Function in Fortran return a single value with no option of altering the input arguments, while subroutines have the ability to perform complex tasks while altering input arguments. This proofs to be helpful e.g., in solving equations system.

Another imperative point is to define the dimension of the arrays when passing them to Fortran (i.e. explicit-shape arrays). To illustrate, `m, n, p, q, k, l` are the dimension of input arrays `a`and `b`, and the out array `cc`.

Since Fortran is a compiled language, we need to save the subroutine in `xcorr2D.f90` file and compile it using: `R CMD SHLIB xcorr2D.f90`

Note

Please use the terminal tab in Rstudio or open a new terminal using `Alt+Shift+R`

As mentioned earlier, we need to pass the dimension of the arrays to Fortran. Therefore, it would logical to write a wrapping function for Fortran subroutine that provides other input arguments.

In the wrapper function (Listing 2.3), we initially require loading the shared object, which is the compiled Fortran subroutine, as `dyn.load("path/to/xcorr2D.so")`. Furthermore, it is important to prepare other input variables for Fortran such as the dimensions of the input and output arrays. Imperatively, data types should be approached carefully. Before calling `.Fortran()`, all storage mode of the variables in R was converted to the appropriate type using either `as.double()` or `as.integer()`. If the wrong type is passed, it can result in a hard-to-catch error or unexpected results[1].

Example

Now, we can use an example to compare the performance of the two functions. In order to do so, `microbenchmark` package needs to be installed, and `ggplot2` is required for plotting.

The obtained benchmarking data allows (`mbm`) for a quantitative comparison of the computational efficiency between the two methods. By printing “mbm” in the console (`print(mbm)`) it is evident that Fortran outperforms the R implementation of 2D cross-correlation by a factor of ~10. The significance of leveraging Fortran becomes evident in Figure 2.1.

[1]Writing R Extensions, 5.2 Interface functions .C and .Fortran

```r
library(microbenchmark)
library(ggplot2)

set.seed(72)
# Assume a
a <- structure(runif(64), dim = c(8L, 8L))
# Assume b
b <- structure(runif(64), dim = c(8L, 8L))
mbm <- microbenchmark(
  xcorr2D_r = xcorr2D_r0(a, b),
  xcorr2D_f = xcorr2D_f0(a, b)
)

autoplot(mbm) +
  stat_summary(
    fun = "median",
    geom = "crossbar",
    width = 0.6,
    colour = "red"
  )
```

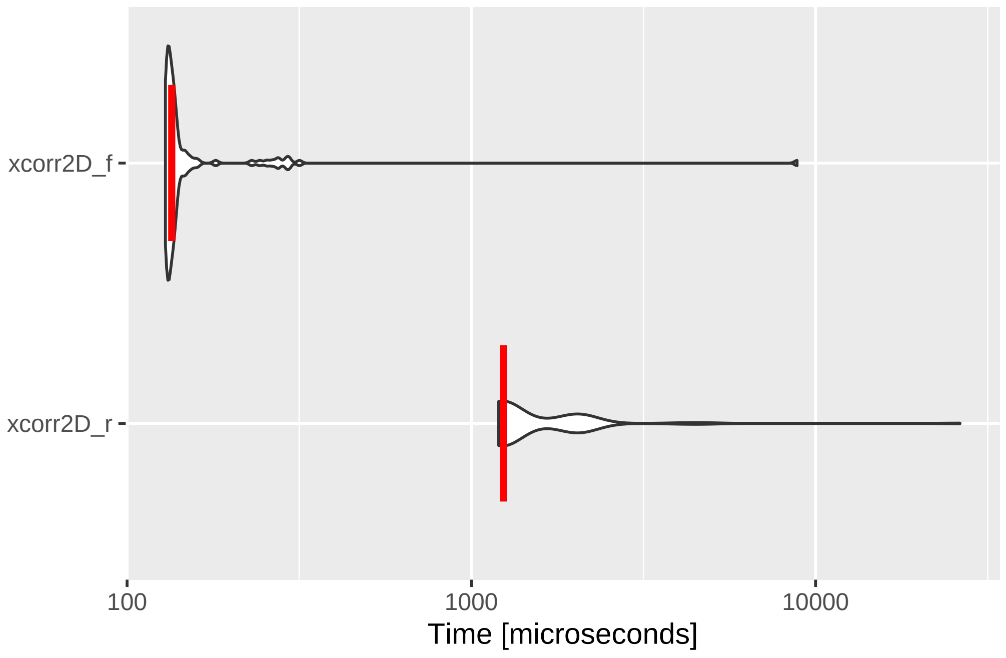

Figure 2.1: Performance comparison of 2D Cross-correlation in R and Fortran. Median is shown as red vertical line.

2.2 2D Convolution

Convolution and cross-correlation are both operations applied to images. Cross-correlation involves sliding a kernel (filter) across an image, while convolution involves sliding a flipped kernel across an image. (Draelos, 2019). Most spatial data in earth science are is discretised resulting in large data sets. Sometimes, these data sets include noise which can obscure meaningful patterns and relationships. One of the prominent methods to remove this nose while preserving important features and structures is the Gaussian smoothing filter. Gaussian smoothing is often achieved by convolution where F is the original data, and G is the kernel representing the 2D Gaussian coefficients.

The 2D convolution of an array F, with dimension (M, N), and array G, with dimension (P, Q), can be given as the array CC that has a dimension of $(M + P - 1, N + Q - 1)$. hv means that G is flipped.

$$CC_{(s,t)} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} F_{(m,n)} G^{hv}_{(m-s,n-t)}$$

where s varies between $-(P-1)$ and $(M-1)$ while t varies between $-(Q-1)$ and $(N-1)$. F and $G \in R$.

Indeed, it is possible to flip the second array and utilise the functions from Section 2.1. Nevertheless, our focus is on the comprehensive workflow. Listing 2.4 presents the implementation of convolution in R, whereas Listing 2.5 demonstrates the Fortran version.

The gfortran compiler is also capable of creating shared libraries. It allows for easy addition of other flags, such as enabling the generation of the run-time check (`-fcheck=all`). The code below shows two options for compiling `conv2D.f90` by R or the gfortran compiler.

```
# R
R CMD SHLIB conv2D.f90

# gfortran
gfortran -fpic -shared conv2D.f90 -o conv2D.so
```

In R wrapper function, `.C64()` from dotCall64 package will be used instead of `.Fortran()`. According to Gerber et al. (2018), `.C64()` transcends other foreign function interfaces in many aspects: - it supports long vectors. - The `SIGNATURE` argument ensures that the interfaced R objects are of the specified types - The `INTENT` argument helps avoid unnecessary copies of R objects between languages. It is very important when working with large data sets.

In Listing 2.6, the basic input arguments, such as the dimensions of input and output arrays, are prepared. Afterwards, the `SIGNATURE` is defined as six integers and three doubles corresponding to the required types in the subroutine. `INTENT` will ensure that only the `conv` argument is copied between R and Fortran. This is particularly important in large productions, where coping the subroutine parameters can extend beyond the available memory (RAM).

Note

A Gaussian smoothing filter can be applied to an array `a` using `b` as the Gaussian kernel or the 2D Gaussian coefficients. However, the convolution and cross-correlation can be optimised using the Fast Fourier Transform (FFT). See (**chap-summary?**).

```r
library(microbenchmark)
library(ggplot2)


set.seed(72)
# Assume a
a <- structure(runif(64), dim = c(8L, 8L))
# Assume b
b <- structure(runif(64), dim = c(8L, 8L))
mbm <- microbenchmark(
  conv2D_r = conv2D_r0(a, b),
  conv2D_f = conv2D_f0(a, b)
)

autoplot(mbm) +
  stat_summary(
    fun = "mean",
    geom = "crossbar",
    width = 0.6,
    colour = "red"
  )
```

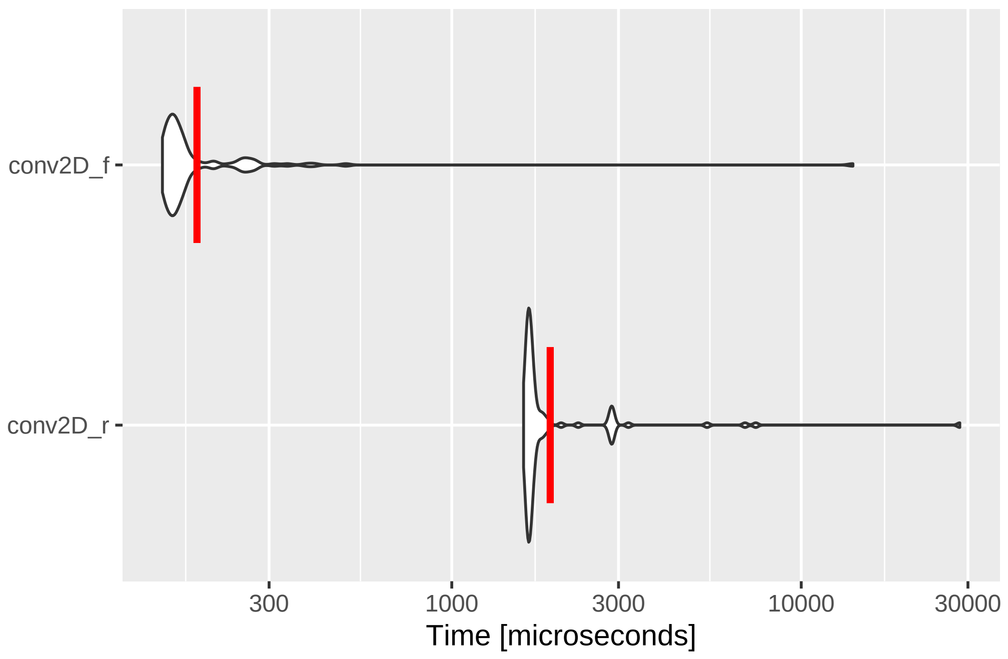

Figure 2.2: Performance comparison of 2D Convolution in R and Fortran. Median is shown as red vertical line.

Similar to cross-correlation calculation, the Fortran implementation of convolution outperforms the R one by a factor of ~10. Performing convolution in large data set using R and Fortran is beneficial since it reduce the required computational resources.

Question

After learning about `.Fortran()` and `.C64()`, you can use one of the two examples above and compare the performance of the two interfaces using `microbenchmark()`. Which function is faster?

2.3 Convective Available Potential Energy (CAPE)

According to the Glossary of Meteorology, CAPE is "the potential energy of an air parcel due to positive buoyancy, which is a useful tool for forecasting, parameterising, and estimating the potential updraft strength of convective clouds." CAPE is calculated as follows (Stull, 2016):

$$CAPE = R_d \sum_{p_{LFC}}^{p_{EL}} (T_p - T_v).ln(\frac{p_{bottom}}{p_{top}}) \tag{2.2}$$

where R_d is the gas constant for dry air, T_p is the parcel temperature, T_e is the environment temperature, p is pressure, LFC is the Level of Free Convection, and EL is the Equilibrium Level.

Under a warming climate, CAPE is expected to increase (Chen et al., 2020), which can result in an elevated risk of thunderstorms. It is crucial for humanity to quantify the future risk for proper preparation and mitigation. Typically, thunderstorms are investigated with convective-permitting modelling (CPM) where the horizontal resolution is less than 4km. CPM simulations produce vast amount of data sets, and CAPE estimation requires atmospheric vertical profiles at a specific gridpoint and time.

Given the rapid advancements in computing power, it is anticipated that CPM is expected to be performed at finer horizontal and vertical resolution, thereby increasing the complexity of estimating CAPE. It is essential that the enhancement of computing power is accompanied by responsible management and allocation of these resources.

Since CAPE calculation scripts are highly complex and lengthy, the necessary codes are only available in supplementary materials. Additionally, to test the two implementations of CAPE, the AquaFortR package should be installed to utilise the example data. See Listing 2.7.

! Important

Foremost, the Fortran subroutine need to be complied as shown in previous sections. The path to the shared library cape_f.so in cape_f.R file should be adapted to the correct path.

Exploring Figure 2.3, it is evident that the implementation of Fortran is faster than R by a factor of ~28, proofing that integrating Fortran in R is vital for performance and beneficial for the environment.

```r
library(microbenchmark)
library(ggplot2)

mbm <- microbenchmark(
  cape_r = cape_r0(t_parcel, dwpt_parcel, mr_parcel,
    Pressure, Temperature, MixingRatio,
    vtc = TRUE
  ),
  cape_f = cape_f0(t_parcel, dwpt_parcel, mr_parcel,
    Pressure, Temperature, MixingRatio,
    vtc = TRUE
  )
)

autoplot(mbm) +
  stat_summary(
    fun = "mean",
    geom = "crossbar",
    width = 0.6,
    colour = "red"
  )
```

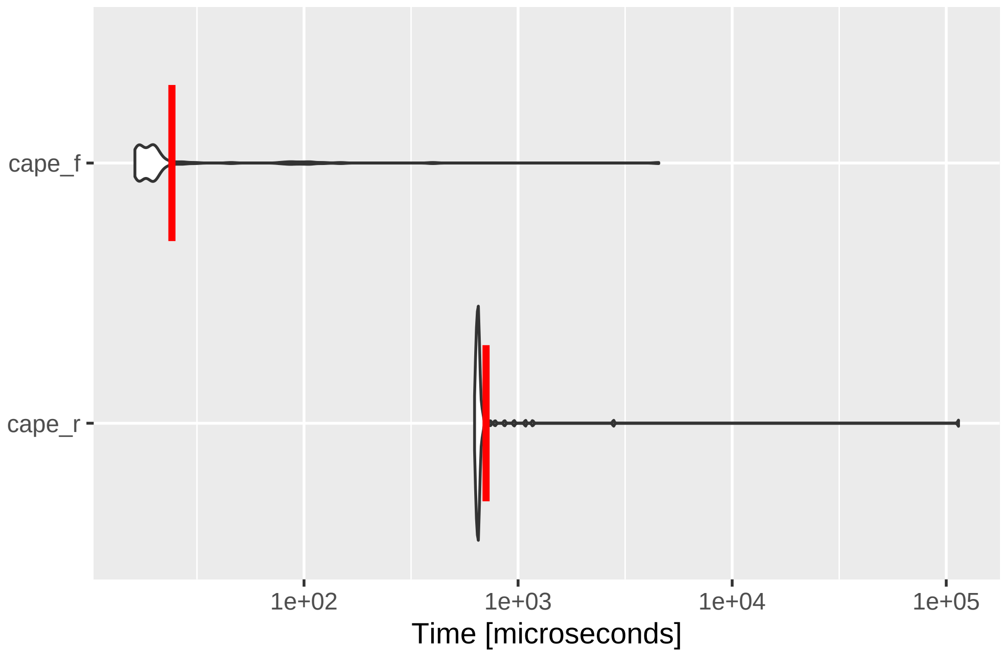

Figure 2.3: Performance comparison of CAPE in R and Fortran

Listing 2.1 Cross-correlation in R

```
xcorr2D_r0 <- function(a, b) {
  # the full CC matrix
  cc_row <- nrow(a) + nrow(b) - 1
  cc_col <- ncol(a) + ncol(b) - 1
  cc <- matrix(1:c(cc_row * cc_col),
    byrow = FALSE, ncol = cc_col
  )

  # obtain possible shifts
  min_row_shift <- -(nrow(b) - 1)
  max_row_shift <- (nrow(a) - 1)
  min_col_shift <- -(ncol(b) - 1)
  max_col_shift <- (ncol(a) - 1)

  # Padded matrix
  rows_padded <- abs(min_row_shift) +
    nrow(a) + abs(max_row_shift)
  cols_padded <- abs(min_col_shift) +
    ncol(a) + abs(max_col_shift)
  # a
  padded_a <- matrix(0,
    nrow = rows_padded,
    ncol = cols_padded
  )
  padded_a[
    (abs(min_row_shift) + 1):(abs(min_row_shift) + nrow(a)),
    (abs(min_col_shift) + 1):(abs(min_col_shift) + ncol(a))
  ] <- a

  for (icol in 1:cc_col) {
    for (irow in 1:cc_row) {
      icc <- irow + ((icol - 1) * cc_row)
      cols <- (icol):(icol + ncol(b) - 1)
      rows <- (irow):(irow + nrow(b) - 1)
      # b
      padded_b <- array(0,
        dim = c(rows_padded, cols_padded)
      )
      padded_b[rows, cols] <- b

      cc[irow, icol] <- sum(padded_a * padded_b)
    }
  }

  return(cc)
}
```

Listing 2.2 Cross-correlation in Fortran

```
subroutine xcorr2d_f(m, n, p, q, k, l, a, b, cc)
   implicit none
   integer                                :: m, n, p, q, k, l
   double precision, dimension(m, n)      :: a
   double precision, dimension(p, q)      :: b
   double precision, dimension(k, l)      :: cc
   !     dummy vars
   integer                               :: min_row_shift, min_col_shift
   integer                               :: max_row_shift, max_col_shift
   integer                               :: rows_padded, cols_padded
   integer                               :: icol, irow, icc, icol2, irow2
   real, allocatable, dimension(:, :)    :: padded_a, padded_b

   !     obtain possible shfits
   min_row_shift = -1*(p - 1)
   max_row_shift = m - 1
   min_col_shift = -1*(q - 1)
   max_col_shift = n - 1

   !   Padded arrray
   rows_padded = abs(min_row_shift) + m + abs(max_row_shift)
   cols_padded = abs(min_col_shift) + n + abs(max_col_shift)
   !    A
   allocate (padded_a(rows_padded, cols_padded))
   padded_a = 0.0
   padded_a((abs(min_row_shift) + 1):(abs(min_row_shift) + m), &
            (abs(min_col_shift) + 1):(abs(min_col_shift) + n)) = a

   !    B
   allocate (padded_b(rows_padded, cols_padded))
   padded_b = 0.0
   do icol = 1, l
      do irow = 1, k
         icc = irow + ((icol - 1)*k)
         icol2 = icol + q - 1
         irow2 = irow + p - 1
         padded_b(irow:irow2, icol:icol2) = b
         cc(irow, icol) = sum(padded_a*padded_b)
         padded_b = 0.0
      end do
   end do
end subroutine xcorr2d_f
```

Listing 2.3 Cross-correlation wrapping function

```
xcorr2D_f0 <- function(a, b) {
  # Please adjust the path to your setup. In my machine,
  dyn.load("AquaFortR_Codes/xcorr2D.so")

  # the full CC matrix
  cc_row <- nrow(a) + nrow(b) - 1
  cc_col <- ncol(a) + ncol(b) - 1
  cc <- matrix(1:c(cc_row * cc_col), byrow = FALSE, ncol = cc_col)

  cc<- .Fortran("xcorr2d_f",
    m = as.integer(dim(a)[1]),
    n = as.integer(dim(a)[2]),
    p = as.integer(dim(b)[1]),
    q = as.integer(dim(b)[2]),
    k = as.integer(cc_row),
    l = as.integer(cc_row),
    a = as.double(a),
    b = as.double(b),
    cc = as.double(cc)
  )$cc

  return(cc)
}
```

Listing 2.4 Convolution in R

```
conv2D_r0 <- function(a, b) {
  # the full convolution matrix
  conv_row <- nrow(a) + nrow(b) - 1
  conv_col <- ncol(a) + ncol(b) - 1
  conv <- matrix(1:c(conv_row * conv_col), byrow = FALSE, ncol = conv_col)

  # obtain possible shifts
  min_row_shift <- -(nrow(b) - 1)
  max_row_shift <- (nrow(a) - 1)
  min_col_shift <- -(ncol(b) - 1)
  max_col_shift <- (ncol(a) - 1)

  # Padded matrix
  rows_padded <- abs(min_row_shift) + nrow(a) + abs(max_row_shift)
  cols_padded <- abs(min_col_shift) + ncol(a) + abs(max_col_shift)
  # a
  padded_a <- matrix(0, nrow = rows_padded, ncol = cols_padded)
  padded_a[
    (abs(min_row_shift) + 1):(abs(min_row_shift) + nrow(a)),
    (abs(min_col_shift) + 1):(abs(min_col_shift) + ncol(a))
  ] <- a

  for (icol in 1:conv_col) {
    for (irow in 1:conv_row) {
      iconv <- irow + ((icol - 1) * conv_row)
      cols <- (icol):(icol + ncol(b) - 1)
      rows <- (irow):(irow + nrow(b) - 1)
      # b
      padded_b <- array(0, dim = c(rows_padded, cols_padded))
      # flip the kernel i.e. b
      padded_b[rows, cols] <- b[nrow(b):1, ncol(b):1]

      conv[irow, icol] <- sum(padded_a * padded_b)
    }
  }

  return(conv)
}
```

Listing 2.5 Convolution in Fortran

```
subroutine conv2d_f(m, n, p, q, k, l, a, b, conv)
   implicit none
   integer                                :: m, n, p, q, k, l, i, j
   double precision, dimension(m, n)      :: a
   double precision, dimension(p, q)      :: b
   double precision, dimension(k, l)      :: conv
   !     dummy vars
   integer                               :: min_row_shift, min_col_shift
   integer                               :: max_row_shift, max_col_shift
   integer                               :: rows_padded, cols_padded
   integer                               :: icol, irow, iconv, icol2, irow2
   real, allocatable, dimension(:, :)    :: padded_a, padded_b

   !     obtain possible shfits
   min_row_shift = -1*(p - 1)
   max_row_shift = m - 1
   min_col_shift = -1*(q - 1)
   max_col_shift = n - 1



   !   Padded arrray
   rows_padded = abs(min_row_shift) + m + abs(max_row_shift)
   cols_padded = abs(min_col_shift) + n + abs(max_col_shift)
   !    A
   allocate (padded_a(rows_padded, cols_padded))
   padded_a = 0.0
   padded_a((abs(min_row_shift) + 1):(abs(min_row_shift) + m), &
            (abs(min_col_shift) + 1):(abs(min_col_shift) + n)) = a

   !    B
   allocate (padded_b(rows_padded, cols_padded))
   padded_b = 0.0
   do icol = 1, l
      do irow = 1, k
         iconv = irow + ((icol - 1)*k)
         icol2 = icol + q - 1
         irow2 = irow + p - 1
         padded_b(irow:irow2, icol:icol2) = b(p:1:-1,q:1:-1)
         conv(irow, icol) = sum(padded_a*padded_b)
         padded_b = 0.0
      end do
   end do
end subroutine conv2d_f
```

Listing 2.6 Convolution wrapping function

```
conv2D_f0 <- function(a, b) {
  require(dotCall64)
  dyn.load("AquaFortR_Codes/conv2D.so")

  m <- nrow(a)
  n <- ncol(b)

  p <- nrow(b)
  q <- ncol(b)
  # the full convolution matrix
  conv_row <- m + p - 1
  conv_col <- n + q - 1
  conv <- matrix(0,
    ncol = conv_col,
    nrow = conv_row
  )

  conv <- .C64("conv2d_f",
    SIGNATURE = c(
      rep("integer", 6),
      rep("double", 3)),
    INTENT = c(rep("r",8), "rw"),
    m, n, p, q,
    k = conv_row,
    l = conv_col,
    a = a, b = b,
    conv = conv
  )$conv

  return(conv)
}
```

Listing 2.7 CAPE implementation in R and Fortran

```
if (!require(AquaFortR)) {
  remotes::install_github("AHomoudi/AquaFortR", subdir = "RPackage")
}

library(AquaFortR)
data("radiosonde")

Temperature <- radiosonde$temp + 273.15 # K
Dewpoint <- radiosonde$dpt + 273.15 # K
Pressure <- radiosonde$pressure # hPa
# Mixing ratio
MixingRatio <- mixing_ratio_from_dewpoint(Dewpoint, Pressure)
t_parcel <- Temperature[1]
dwpt_parcel <- Dewpoint[1]
mr_parcel <- MixingRatio[1]

source("AquaFortR_Codes/cape_r.R")
source("AquaFortR_Codes/cape_f.R")
```

Chapter 3

Accelerate R Packages with Fortran

This chapter will focus on wrapping the developed routines from the previous chapter in an R package.

3.1 Brief Introduction to R Packages

3.1.1 devtools package

3.2 Developing AquaFortR Package

3.2.1 Fortran subroutine

iso_c_binidng

3.2.2 R function

3.2.3 C funtions

- C to Fortran
- R to C

3.2.4 Package's files

Chapter 4

Conclusions and Optimization Insights

- positive features of R
- positive features of Fortran
- Why .Fortran() should not be used. dotCall64 is recommended for scripts.
- Mention the memory allocation issues related to .Fortran
- OpenMp implementation in Fortran, Romp
- Explain how beneficial R, Fortran with OpenMP is. For examples calculating cape for each vertical profile in a high resolution (1-4km) climate simulation (years) output (nlat, lon, nlevels, ntime) can be accelerated by passing the whole input arrays from R to Fortran, then loop over (nlat, lon, ntime) with OpenMP.
- The cross-correlation can be speed up by using FFT. Also, in Fortran, the FFTW C library can be used.
- Possibility of calling BLAS, LAPACK and LINPACK linear algebra functions
- Fortran using C (R_ext/BLAS.h, R_ext/Lapack.h and R_ext/Linpack.h)
- Final words why packaging is better.

Chapter 5

Further Reading

- Fortran and R – Speed Things Up by Steve Pittard
- The Need for Speed Part 1: Building an R Package with Fortran (or C) by Avraham Adler
- The Need for Speed Part 2: C++ vs. Fortran vs. C by Avraham Adler
- Writing R Extensions by R Core Team
- Modern Fortran Tutorial by Yutaka Masuda
- Extend R with Fortran by Yutaka Masuda
- Advanced R by Hadley Wickham
- Fortran 90 Tutorial by Stanford University
- Fortran Libraries by Fortran Wiki
- Fortran Best Practices by Fortran Community
- Fortran 90 Reference Card by Michael Goerz.
- Hands-On Programming with R by Garrett Grolemund
- r-spatial by Edzer Pebesma, Marius Appel, and Daniel Nüst
- Spatial Data Science
- R for Data Science (2e) by Hadley Wickham, Mine Çetinkaya-Rundel, and Garrett Grolemund
- Introduction to Environmental Data Science by Jerry Davis

References

Chambers, F., John M. CRC Press Boca Raton. (2016). *Extending r*. http://slubdd.de/katalog?TN_libero_mab2

Chen, J., Dai, A., Zhang, Y., & Rasmussen, K. L. (2020). Changes in convective available potential energy and convective inhibition under global warming. *Journal of Climate*, *33*(6), 2025–2050. https://doi.org/10.1175/JCLI-D-19-0461.1

Chirila, D. B., & Lohmann, G. (2014). *Introduction to modern fortran for the earth system sciences*. Springer Berlin Heidelberg. https://books.google.de/books?id=bZCeBQAAQBAJ

Draelos, R. (2019). Convolution vs. Cross-correlation. In *Glass Box*. https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/

Gerber, F., Mösinger, K., & Furrer, R. (2018). dotCall64: An r package providing an efficient interface to compiled c, c++, and fortran code supporting long vectors. *SoftwareX*, *7*, 217–221. https://doi.org/https://doi.org/10.1016/j.softx.2018.06.002

Masuda, Y. (2020). *Modern fortran tutorial*. Introduction. https://masuday.github.io/fortran_tutorial/introduction.html

Metcalf, M., Reid, J., & Cohen, M. (2018). *Modern fortran explained: Incorporating fortran 2018*. OUP Oxford. https://books.google.de/books?id=sB1rDwAAQBAJ

R Core Team. (2023). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Seelig, T., Deneke, H., Quaas, J., & Tesche, M. (2021). Life Cycle of Shallow Marine Cumulus Clouds From Geostationary Satellite Observations. *Journal of Geophysical Research: Atmospheres*, *126*(22). https://doi.org/10.1029/2021JD035577

Stull, R. (2016). *Practical meteorology: An algebra-based survey of atmospheric science*. AVP International, University of British Columbia. https://books.google.de/books?id=xP2sDAEACAAJ

Wang, C. (2019). *Kernel learning for visual perception*. https://doi.org/10.32657/10220/47835

Warren, M. A., Quartly, G. D., Shutler, J. D., Miller, P. I., & Yoshikawa, Y. (2016). Estimation of ocean surface currents from maximum cross correlation applied to GOCI geostationary satellite remote sensing data over the tsushima (korea) straits. *Journal of Geophysical Research: Oceans*, *121*(9), 6993–7009. https://doi.org/https://doi.org/

10.1002/2016JC011814

Wickham, H. (2015). *Advanced r*. CRC Press. https://books.google.de/books?id=FfsYCwAAQBAJ

Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for data science import, tidy, transform, visualize, and model data* (2nd edition). O'Reilly. http://slubdd.de/katalog?TN_libero_mab2

Wilkinson, M. D., Dumontier, M., Aalbersberg, Ij. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C. T., Finkers, R., … Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, *3*(1), 160018. https://doi.org/10.1038/sdata.2016.18

Willert, C. E., & Gharib, M. (1991). Digital particle image velocimetry. *Experiments in Fluids*, *10*(4), 181–193. https://doi.org/10.1007/BF00190388