FortranCon 2020

Dr. Erin Hodgess

Western Governors Univ

July 2, 2020

Using R with High Performance Fortran on a Windows Laptop

Overview

- Introduction
- Ordinary Kriging for Spatio-Temporal Data: rmpiFort
- Theoretical Background
- Empirical Background
- The Irish Wind Data Study
- A Discussion of the Computer Setup: Open Accelerators (OpenACC), the Message Passing Interface (MPI)
- Conclusion

Introduction

- R has been a language of choice in the last 20 years for statistical computing.
- There are literally many thousands of libraries with various applications.
- However, R does occasionally suffer from problems with data size and speed.
- So we turn to Fortran to extend the capabilities.
- R uses Fortran 95

Introduction

- Several years ago, I was fortunate enough to work on supercomputers for geostatistics research on ordinary kriging.
- I worked with spatial data and was able to achieve excellent speedup.
- I wondered if laptops would work for those who didn't have access to supercomputers.

Introduction

- Why Fortran? Why not C or C++?
- Initial work on the supercomputer was with C and the rmpi package.
- The speedup was negligible.
- I experimented with Fortran, and the process began to show great promise in terms of time.

Ordinary Kriging for Spatio-Temporal Data: rmpiFort

- I constructed a R package with high performance tools on Windows such as MPI and OpenACC.
- `R` has an existing function, `krigeST`
- I ran comparisons on a MacBook Air and a Windows laptop
- In the past few days, I was able to harness WSL with Ubuntu to replicate this work. Until now, the WSL could not access the graphics cards.
- I obtained surprisingly good results

Theoretical Background

- I have a spatio-temporal vector:

$$\boldsymbol{z} = \mathbf{z}(s_1, t_1), \mathbf{z}(s_2, t_2), \ldots, \mathbf{z}(s_n, t_n)$$

  where $\mathbf{z}(s_i, t_i)$ are the locations and times of the response variable.
- Goal: To produce a grid of interpolated values in space and time
- This is similar to a weighted regression model

Theoretical Background

- First Order Stationarity

$$E[\boldsymbol{z}] = \boldsymbol{\mu}$$

- Second Order Stationarity: Covariance for the S-T model

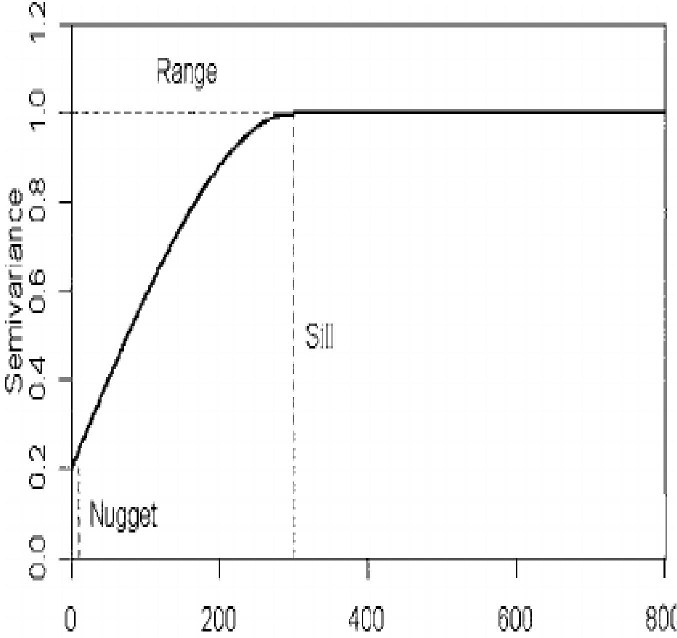
$$\boldsymbol{C}_{st} = Cov(\mathbf{z}(s,t), \mathbf{z}(s+h, t+u))$$

$$\boldsymbol{C}_{st} = Cov(\mathbf{z}(0,0), \mathbf{z}(h,u))$$

Theoretical Background

- In spatial and spatio-temporal processes, there is a semivariogram which aids in selecting an appropriate model.
- This semivariogram is used to build the Covariance matrix.

Theoretical Background



Empirical Background

- I obtain the estimates for the sill, nugget, range and the anistropy parameter from standard `R` functions.
- I can now calculate the mean, the covariance matrix, and the predicted values with our high performance tools.

Empirical Background

- I produce the overall sample mean:

$$\hat{\mu} \;=\; \left(\mathbf{1}' \cdot \boldsymbol{C}_{\text{sm}}^{-1} \cdot \mathbf{1}\right)^{-1} \cdot \left(\mathbf{1}' \cdot \boldsymbol{C}_{\text{sm}}^{-1} \cdot \mathbf{z}\right)$$

where $\boldsymbol{C}_{\text{sm}}$ is

$$\boldsymbol{C}_{\text{sm}} \;=\; \boldsymbol{C}_s(h) + \boldsymbol{C}_t(u) + \boldsymbol{C}_{\text{joint}}\left(\sqrt{h^2 + (\kappa \cdot u)^2}\right).$$

- For a one point forecast, I obtain:

$$\hat{\boldsymbol{z}} \;=\; \hat{\mu} + c_0 \boldsymbol{C}_{\text{sm}}^{-1}\left(\boldsymbol{z} - \hat{\mu}\mathbf{1}\right).$$

This is easily generalized for multiple points in space and time.

The Irish Wind Data Study

- I looked at the Irish Wind data, as discussed in Pebesma(2004), and which originally appeared in Haslett and Raftery (1989).
- The data are daily wind speeds at 12 stations around Ireland.
- The data run from January 1961 - December 1978.
- I began with 3 months, 1 year, and in one year increments until December 1969.

The Irish Wind Data Study

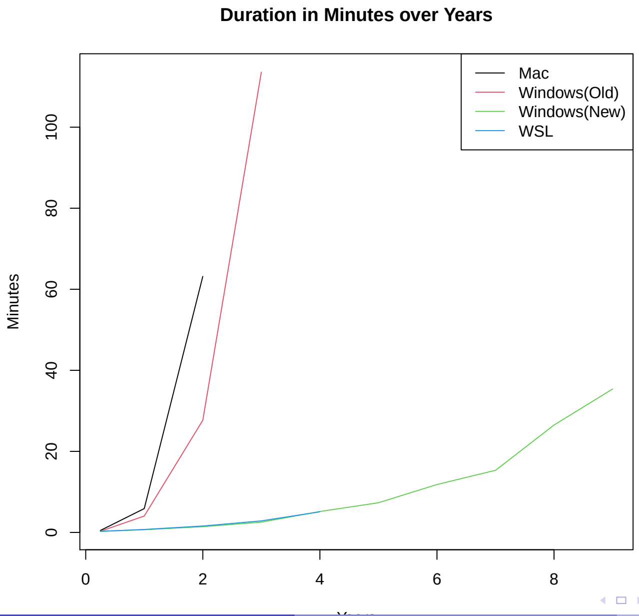
- I ran the existing process on a 2 core MacBook Air, and a 6 core Windows laptop.
- I ran the new process on the 6 core Windows laptop in which R had been compiled with the OpenBLAS package.
- I re-ran our study within the last two weeks with R Version 4.0.0.
- Last minute update: I ran our study on Tuesday with WSL/Ubuntu with R Version 4.0.2.
- The output is a spatio-temporal grid with interpolated values.

The Irish Wind Data Study

Duration	Mac	Windows(Old)	Windows(New)	WSL
3 months	0.48	0.29	0.28	0.26
1 year	5.88	4.04	0.67	0.73
2 years	63.17	27.68	1.42	1.58
3 years		113.59	2.53	2.85
4 years			5.16	5.09
5 years			7.32	
6 years			11.82	
7 years			15.33	
8 years			26.50	
9 years			35.39	

Time in Minutes

The Irish Wind Data Study



A Discussion of the Computer Setup

- What tools are needed?
- I download the Portland Group Inc. (PGI) Community Edition compiler.
- I must also download the latest version of CUDA drivers, from the NVIDIA website.
- I download Microsoft MPI (Message Passing Interface) as well.
- Finally, I obtain the free version of Visual Studio (2019).

A Discussion of the Computer Setup

- Everything can be done with free software.
- I begin with Avraham Adler's website on building OpenBLAS.
- Then I compile R with this OpenBLAS.

A Discussion of the Computer Setup

- Typically, when compiling an R package with foreign programs in the src directory, the Makevars file is constructed in that src directory for Windows
- However, when using the PGI compiler, the Makevars does not work.
- For compilation of the R package, I use the Makeconf file, which is located in the `R-4.0.0/etc/x64` directory.
- The compiler and the flags are updated in the Makeconf file to reflect the PGI compiler.
- If producing a single program with OpenACC or MPI, we can use the PGI compiler from the command line.

A Discussion of the Computer Setup

- How does OpenACC work?
- This is done via Fortran with OpenACC directives.
- The OpenACC moves the data from the host (CPU) to the GPU device for quick processing.

A Discussion of the Computer Setup

- How complicated is OpenACC?

```
  !$acc loop seq
do i=1,n1
 do j=1,n1
  mat1(i,j) = (((x(i)-x(j))**2 + (y(i)-y(j))**2))**0.5
 enddo
enddo
 !$acc end loop
```

- I use our OpenACC subroutines to produce the matrices.
- I actually return the matrices as an extended vectors.

A Discussion of the Computer Setup: OpenACC Example

```
    subroutine ac1(n,y,xt)
  !DEC$ ATTRIBUTES DLLEXPORT :: ac1
  use cudafor
  implicit none
  integer :: n, y(n),i
  real :: xt,xa,xb
  xa=0.0;xb=0.0
  call cpu_time(xa)
  !$acc loop seq
       do i=1,n
          y(i) = i + 2
  enddo
  !$acc end loop
call cpu_time(xb)
xt = xb-xa
end subroutine ac1
```

A Discussion of the Computer Setup: OpenACC Example

```
> dyn.load("cudarama.dll")
> n <-  1024^2
> y <- numeric(length=n)
> xt1 <- .Fortran("ac1",n=as.integer(n),y=as.integer
xt=as.single(0.0))
> xt1$xt
[1] 0.002
attr(,"Csingle")
[1] TRUE
```

A Discussion of the Computer Setup

- I use MPI to initialize our next step in the procedure.
- When we use MPI with `R`, we use `R` functions to call the subroutines

A Discussion of the Computer Setup

```
init1 <- function(ierror=0)
return(.Fortran("finit",as.integer(ierror)))
```

A Discussion of the Computer Setup

```
    subroutine finit(ierror)
!DEC$ ATTRIBUTES DLLEXPORT :: finit
implicit none
include 'mpif.h'

integer :: ierror

call MPI_INIT(ierror)

end subroutine finit
```

A Discussion of the Computer Setup

- The main function in our process is inverting the final covariance matrix.
- The `pbdDMAT` package uses MPI, and will split our matrices into submatrices based on the process rank.
- This package creates a special structure for large matrices.
- In the final segment of our study, I am inverting a 39420×39420 matrix on a laptop.
- This happens, along with all of the other calculations, in 35 minutes.

A Discussion of the Computer Setup: MPI Example

```
      mpitest2a <- function(ierror=0,comm=1,size=1,rank=0)
  library(rmpiFort)
  dyn.load("sum1.dll")
  y <- 0
nSteps <- 2048
        x <- 0
        integral <- 0
        xlow <- 0
        xupp <- 1
        dx <- (xupp - xlow)/nSteps
        nb <- 5
zz <- init1(ierror=ierror)
sz1 <- univ1(comm=comm,size=size)[[2]]
rk1 <- unlist(rank1(comm=comm,rank=rank)[[2]])
```

```
      iMin <- (rk1*nSteps)/sz1
      iMax <- ((rk1+1)*nSteps)/sz1


for(i in iMin:(iMax-1))
      x <- xlow + dx*(i+0.5)
      integral <- integral + (x^nb)*dx


      total_int <- unlist(.Fortran("sum1",x=as.single(integra
      rank=as.integer(rk1),
       y=as.single(y))[[3]])

      if(rk1==0)
      print("final")
      print(total_int)

       za <- fin1(ierror=ierror)
      subroutine sum1(x,rank,y)
!DEC$ ATTRIBUTES DLLEXPORT :: sum1
implicit none
include 'mpif.h'



integer rank,size,ierror,tag,status(MPI_STATUS_SIZE),i,np
real :: x
real, intent(inout) :: y



     call MPI_REDUCE(x,y,1,MPI_REAL,MPI_SUM,0,MPI_COMM_WORL


end subroutine sum1
```

```
      > mpiexec -np 4 Rscript nextmpi.in >mpi2a.out
      >cat mpi2a.out
[1] "final"
[1] 0.1666666
attr(,"Csingle")
[1] TRUE
```

Conclusion

- The speedup was unexpectedly good.
- The data size was also better than I thought.
- I would like to try this on a laptop with more cores and more memory.

Thank you!

- Questions, please?