Some Parallel R

Timothy H. Kaiser, PH.D
tkaiser2@nrel.gov

Slides

https://github.com/timkphd/slides

Claims and Disclaimers

* I am not an expert in R
* I am considered an expert in Parallel Programming
* I have developed and published parallel packages in several languages including Python, C++, and Fortran
* I am not an expert in Earthquake simulation (have done it)
* Have publications where resulting from when someone said they wanted to do Parallel R. I convinced him otherwise and he actually finished his Ph.D.
* The people that developed parallel R are R programmers
* Not so sure this is a good idea

Paths for Today

* Rmpi
* Parallel R library
* Combine the two

Hope to cover today

- Briefly Rmpi
  - Simple examples
  - Example from documentation (Don't think you should be doing this)
  - Three full examples
    - Bag of Task
    - California Earthquake Hazard Map
    - Finite Difference (Stommel) Code (won't get this far)

Hope to cover today

* Briefly "Normal" Parallel R
  * library(parallel)
  * library(dplyr)
  * library(doParallel)

Hope to cover today

* Some tricks outside of R
  * Forcing tasks to cores
  * Monitoring tasks
  * Splitting files
  * Writing files to be read as binary

References

https://cran.r-project.org/web/packages/foreach/index.html

https://cran.r-project.org/web/packages/foreach/foreach.pdf

https://cran.r-project.org/web/packages/doParallel/index.html

https://cran.r-project.org/web/packages/dplyr/

https://cran.r-project.org/web/packages/Rmpi/index.html

https://mpi4py.readthedocs.io/en/stable/

https://github.com/rstudio/cheatsheets

MPI - search Message Passing Interface

https://insights.dice.com/2019/07/29/5-programming-languages-probably-doomed/

Examples run on:

* CU/CSU Summit
* NREL Eagle
* Mac laptop and desktop
* and for today...

My Raspberry Pi Cluster

* Currently two 4 core nodes
  * clr & blk
* Ethernet interconnect
* lmod
* gcc 10.x
* OpenMPI & MPICH
* R 4.0.5
* Python 3.9.4
* spack

Source

```
hexagon:~ tkaiser$ mkdir tut
hexagon:~ tkaiser$ cd tut
hexagon:tut tkaiser$ git clone https://github.com/timkphd/examples.git
Cloning into 'examples'...
remote: Enumerating objects: 83, done.
remote: Counting objects: 100% (83/83), done.
remote: Compressing objects: 100% (63/63), done.
remote: Total 1236 (delta 39), reused 45 (delta 19), pack-reused 1153
Receiving objects: 100% (1236/1236), 2.47 MiB | 3.64 MiB/s, done.
Resolving deltas: 100% (390/390), done.
hexagon:tut tkaiser$ mv examples/r .
hexagon:tut tkaiser$ rm -rf examples
hexagon:tut tkaiser$ cd r

hexagon:r tkaiser$ ./getdat.py
downloading file
True
split -l 158563 start start
hexagon:r tkaiser$ tar -xzf laser.tgz
hexagon:r tkaiser$

hexagon:r tkaiser$ Rscript doinstall.R
. . .
. . .
hexagon:r tkaiser$ R CMD SHLIB mapit.c
```

Guide for installing Rmpi

```
curl --insecure https://cran.r-project.org/src/contrib/Rmpi_0.6-9.1.tar.gz -o Rmpi.tar.gz

MY_MPI_PATH=`which mpicc| sed s,/bin/mpicc,,`

echo MY_MPI_PATH= $MY_MPI_PATH

export TYPE=MPICH2

R CMD INSTALL --configure-args="\
--with-Rmpi-include='$MY_MPI_PATH/include' \
--with-Rmpi-libpath='$MY_MPI_PATH/lib' \
--with-mpi='$MY_MPI_PATH/bin/mpicc' \
--with-Rmpi-type='$TYPE'"  \
Rmpi.tar.gz
```

* Latest version number can be found at:
  * https://cran.r-project.org/web/packages/Rmpi/index.html
  * You can also specify OPENMPI for the type, which most likely work for Intel

R codes

bcast.R
: MPI bcast,gather,reduce,scatter

bigcor.R
: Testing some correlation ideas, work in progress.

bind.R
: foreach with combine=rbind and combine=cbind

bininvert.R
: Simple matrix inversion, not parallel

bot.R
: MPI Bag of tasks

doin.R
: Install script

doinstall.R
: Install script

flower.R
: Iris data set analysis in MPI with bcast.Robj, send.Robj, reduce

hellop.R
: foreach hello world

invert1.R
: Another serial matrix inversion program

invert.R
: Matrix inversion program, forcing task to a given core, called by spawn

ircmdr.R
: Install Script

iris.R
: Compares random data to iris data set with foreach

pcor.R
: Testing some correlation ideas, work in progress.

new.R
: Real research code work in progress

pcor.R
: Testing some correlation ideas, work in progress.

pieit.R
: Make a pie chart

piris.R
: Iris data set analysis with foreach, see flower.R

qsort.R
: Simple recursive sort with foreach

quake01.R
: Earthquake hazard map with foreach

quake02.R
: Earthquake hazard map with MPI

quake03.R
: Earthquake hazard map with MPI, faster

r_ex01.R
: Send/Recv in MPI, see c_ex01.c

semantics.R
: A bunch of foreach examples, some show how it can be broken.

spawn2.R
: MPI spawn program calls invert.R

spawn.R
: Simple MPI spawn program

st_00.R
: Finite difference code

st_01.R
: Finite difference code with MPI

st2.R
: Real research code, work in progress

st3.R
: Real research code, work in progress

st4.R
: Real research code, work in progress

sys.R
: Utility functions, sys - system and srun

tymer.R
: Nice timing routine, matches Tim's python version

C codes

c_ex01.c
: Send/Recv in MPI, see r_ex01.R

mapit.c
: Forces tasks to a specified core. See spawn2.R and invert.R

stc_01.c
: Finite difference code with MPI

Python

getdat.py
: Download and preprocess earthquake data.

Fortran

quake.f90
: Earthquake hazard map

readq.f90
: Shows how to read/write data in binary in Fortran

Data

bounds
: Bounds for earthquake hazard map calaulation

st.in
: Input for finite difference calculation

redoit
: Build Rmpi

laser.tgz
: Input for Bag of Task program

infile
: Input for Bag of Task program

Other

: rcheat.html

Some ways to do things in R
: readme.html

This file

Parallel Programming
Basic MPI

Talk Overview

* Background on MPI
* Documentation
* Hello world in MPI
* Some differences between Rmpi and normal MPI
* Basic communications
* Simple send and receive program
* Spawn

Background on MPI

- MPI - Message Passing Interface
  - Library standard defined by a committee of vendors, implementers, & parallel programmers
  - Used to create parallel programs based on message passing
- Portable: one standard, many implementations
- Available on almost all parallel machines in C and Fortran
- R is not one for the officially supported languages
- Well over 100 advanced routines but 6 basic
- Rmpi is a very small subset of full MPI but it adds a few extensions (56 pages vs 856)

Documentation

* MPI home page (contains the library standard): www.mcs.anl.gov/mpi
* https://cran.r-project.org/web/packages/Rmpi/index.html
* Books
  * "MPI: The Complete Reference" by Snir, Otto, Huss-Lederman, Walker, and Dongarra, MIT Press (also in Postscript and html)
  * "Using MPI" by Gropp, Lusk and Skjellum, MIT Press
* Tutorials
* many online, just do a search

MPI Implementations

- Most parallel supercomputer vendors provide optimized implementations
- Compiler vendors
  - Intel
  - Portland Group
- OpenMPI
  - www.open-mpi.org (default on Mio and RA)

MPI Implementations

- MPICH:
  - http://www-unix.mcs.anl.gov/mpi/mpich1/download.html
  - http://www.mcs.anl.gov/research/projects/mpich2/index.php
- MVAPICH
  - Infiniband optimized version from Ohio State
  - http://mvapich.cse.ohio-state.edu/index.shtml

Launching a MPI job

* Most (all?) RMACC sites use the batch scheduler slurm
* Create a batch file, for example "myscript"
* In the file myscript you will have a line of the form:
  * srun -n 8 ./myprog.R
* You might also use
  * mpiexec -n 8 ./myprog.R

Key Concepts of MPI

- Used to create parallel programs based on message passing
  - Normally the same program is running on several different processors
  - Processors communicate using message passing
- Typical methodology:

```
start job on n processors
do i=1 to j
    each processor does some calculation
    pass messages between processor
end do
end job
```

Messages

* Simplest message: an array of data of one type.
* Predefined types correspond to commonly used types in a given language
    * MPI_REAL (Fortran), MPI_FLOAT (C)
    * MPI_DOUBLE_PRECISION (Fortran), MPI_DOUBLE (C)
    * MPI_INTEGER (Fortran), MPI_INT
    * User can define derived (compound) types and send them.
* R
    * 1-Integer 2- double
    * 3- character Rmpi adds serialized "pickled" data for DFs

Communicators

- Communicator
  - A collection of processors working on some part of a parallel job
  - Used as a parameter for most MPI calls
  - MPI_COMM_WORLD includes all of the processors in your job
  - Processors within a communicator are assigned numbers (ranks) 0 to n-1
  - Can create subsets of MPI_COMM_WORLD

Include files

* The MPI include file
  * C: mpi.h
  * Fortran: mpif.h (a f90 module is a good place for this)
* Defines many constants used within MPI programs
* In C defines the interfaces for the functions
* Compilers know where to find the include files
* Rmpi - library(Rmpi) sort of takes the place of include files

Minimal MPI program in C

* Every MPI program needs these...

```
/* the mpi include file */
   #include <mpi.h>
   int nPEs,ierr,iam;
/* Initialize MPI */
   ierr=MPI_Init(&argc, &argv);
/* How many processors (nPEs) are there?*/
   ierr=MPI_Comm_size(MPI_COMM_WORLD, &nPEs);
/* What processor am I (what is my rank)? */
   ierr=MPI_Comm_rank(MPI_COMM_WORLD, &iam);
...
   ierr=MPI_Finalize();
```

In C MPI routines are functions and return an error value

Minimal MPI program in R

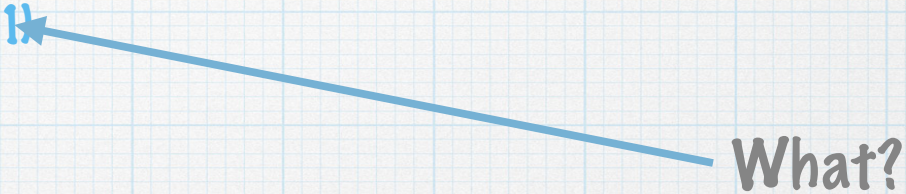
* Every MPI program needs these...

```
/* Initialize MPI */
if (!is.loaded("mpi_initialize")) {
    library("Rmpi")
    }
/* How many processors (nPEs) are there?*/
   myid <- mpi.comm.rank(comm=mpi_comm_world)
/* What processor am I (what is my rank)? */
   numprocs <- mpi.comm.size(comm=mpi_comm_world)
...
   bonk<-mpi.finalize()
```

In Rmpi loading the library does the initialization

Synchronous Send

* Rmpi
  * mpi.send(x, type, dest, tag, comm = 1)
* C
  * MPI_Send(&buffer, count ,datatype, destination, tag,communicator);
* Fortran
  * Call MPI_Send(buffer, count, datatype, destination,tag,communicator, ierr)
* Call blocks until message on the way

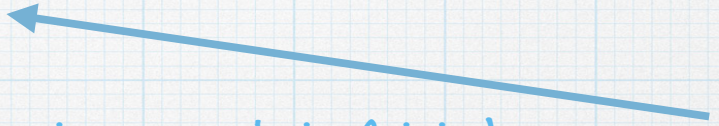


Call MPI_Send(buffer, count, datatype, destination, tag, communicator, ierr)

- **Buffer**: The data array to be sent
- **Count** : Length of data array (in elements, 1 for scalars)
- **Datatype** : Type of data, for example : MPI_DOUBLE_PRECISION, MPI_INT, etc
  - Rmpi<-integer=1 double=2 character=3
- **Destination** : Destination processor number (within given communicator)
- **Tag** : Message type (arbitrary integer)
- **Communicator** : Your set of processors

Synchronous Receive

* Rmpi
  * mpi.recv(x, type, source, tag, comm = 1, status = 0)
* C
  * MPI_Recv(&buffer,count, datatype, source, tag, communicator, &status);
* Fortran
  * Call MPI_ RECV(buffer, count, datatype, source,tag,communicator, status, ierr)



* Call blocks the program until message is in buffer
* Status - contains information about incoming message

Call MPI_Recv(buffer, count, datatype, source, tag, communicator, status, ierr)

- **Buffer**: The data array to be received
- **Count** : Maximum length of data array (in elements, 1 for scalars)
- **Datatype** : Type of data, for example : MPI_DOUBLE_PRECISION, MPI_INT, etc
  - Rmpi<-integer=1 double=2 character=3
- **Source** : Source processor number (within given communicator)
- **Tag** : Message type (arbitrary integer)
- **Communicator** : Your set of processors
- **Status**: Information about message
- **Ierr** : Error return (Fortran only)

Basic Send and Receive in R
r_ex01.R

```
if (!is.loaded("mpi_initialize")) {     library("Rmpi")      }
mpi_comm_world<-0
myid <- mpi.comm.rank(comm=mpi_comm_world)
numprocs <- mpi.comm.size(comm=mpi_comm_world)
myname <- mpi.get.processor.name()
tag<-1234;
source<-0;
destination<-1;
count<-1;
paste("I am",myid,"of",numprocs,"on",myname)
if(myid == source){
    buffer<-5678.9
    mpi.send(buffer, 2, destination, tag,  comm=mpi_comm_world)
    }
if(myid == destination){
    x<-1234.5
    x<-mpi.recv(x, 2, source, tag,  comm=mpi_comm_world, status=0)
    print(c("got",x))
    }
bonk<-mpi.finalize()
```

```
 mpiexec -n 2 Rscript r_ex01.R
[1] "I am 1 of 2 on semikln"
[1] "I am 0 of 2 on semikln"
NULL
[1] "got"    "5678.9"
semikln:eq timk$
```

A few other calls

- mpi.bcast
  - broadcasts a message from the specified rank to all members
- mpi.reduce
  - combines each member's result, using the operation op
- mpi.gather
  - gather each member's message to the member specified by the argument root.
- mpi.gatherv
  - same as gather but each member can send different amounts of data
- mpi.scatter
  - split a vector and send parts to each process

Bcast

Description

mpi.bcast is a collective call among all members in a comm. It broadcasts a message from the specified rank to all members.

Usage

```
mpi.bcast(x, type, rank = 0, comm = 1, buffunit=100)
```

Reduce

Description

mpi.reduce is aglobal reduction operation. mpi.reduce combines each member's result, using the operation op, and returns the combined value(s) to the member specified by the argument dest.

Usage

```
mpi.reduce(x, type=2,
        op=c("sum","prod","max","min","maxloc","minloc"),
        dest = 0, comm = 1)
```

Gather & Gatherv

Description

mpi.gather and mpi.gatherv (vector variant) gather each member's message to the member spec- ified by the argument root. The root member receives the messages and stores them in rank order. mpi.allgather and mpi.allgatherv are the same as mpi.gather and mpi.gatherv except that all members receive the result instead of just the root.

Usage

```
mpi.gather(x, type, rdata, root = 0, comm = 1)
mpi.gatherv(x, type, rdata, rcounts, root = 0, comm = 1)
```

```
if (!is.loaded("mpi_initialize")) {library("Rmpi")}

mpi_comm_world<-0
myid <- mpi.comm.rank(comm=mpi_comm_world)
numprocs <- mpi.comm.size(comm=mpi_comm_world)
myname <- mpi.get.processor.name()

paste("I am",myid,"of",numprocs,"on",myname)

source=0
count<-4 #not needed

buffer1<-c(0,0,0,0)
if (myid == source){
    buffer1<-c(10,200,3000,40000)
}

buffer1<-mpi.bcast(buffer1,type=1,comm = mpi_comm_world)

print(c(myid,buffer1))

buffer2<-as.integer(myid+1)

stuff<-vector("integer",numprocs)

stuff<-mpi.gather(buffer2, 1, stuff,root = 0, comm = mpi_comm_world)

mysum<-mpi.reduce(buffer2, type=1, op="sum",dest = 0, comm = mpi_comm_world)

myprod<-mpi.reduce(buffer2, type=1, op="prod",dest = 0, comm = mpi_comm_world)

if(myid == 0){
      cat("stuff=",stuff,"\n")
      cat("the sum",mysum,"\n")
      cat("the product",myprod,"\n")

    stuff<-append(stuff,stuff*10)
    print(stuff)
}

twovals<-vector("integer",2)
twovals<-mpi.scatter(stuff, type=1, twovals,root = 0, comm = mpi_comm_world)

cat("on ",myid," i got these two values ",twovals,"\n")

bonk<-mpi.finalize()
```

The file "bcast.R" contains these calls

```
mac:r> mpiexec -n 4 Rscript bcast.R
[1] "I am 0 of 4 on tkaiser2-31606s"
[1] "I am 3 of 4 on tkaiser2-31606s"
[1] "I am 2 of 4 on tkaiser2-31606s"
[1] "I am 1 of 4 on tkaiser2-31606s"
[1]     0    10   200  3000 40000
[1]     2    10   200  3000 40000
[1]     1    10   200  3000 40000
[1]     3    10   200  3000 40000
stuff= 1 2 3 4
the sum 10
the product 24
[1]  1  2  3  4 10 20 30 40
on  0  i got these two values  1 2
on  1  i got these two values  3 4
on  2  i got these two values  10 20
on  3  i got these two values  30 40
mac:r>
```

flower.R

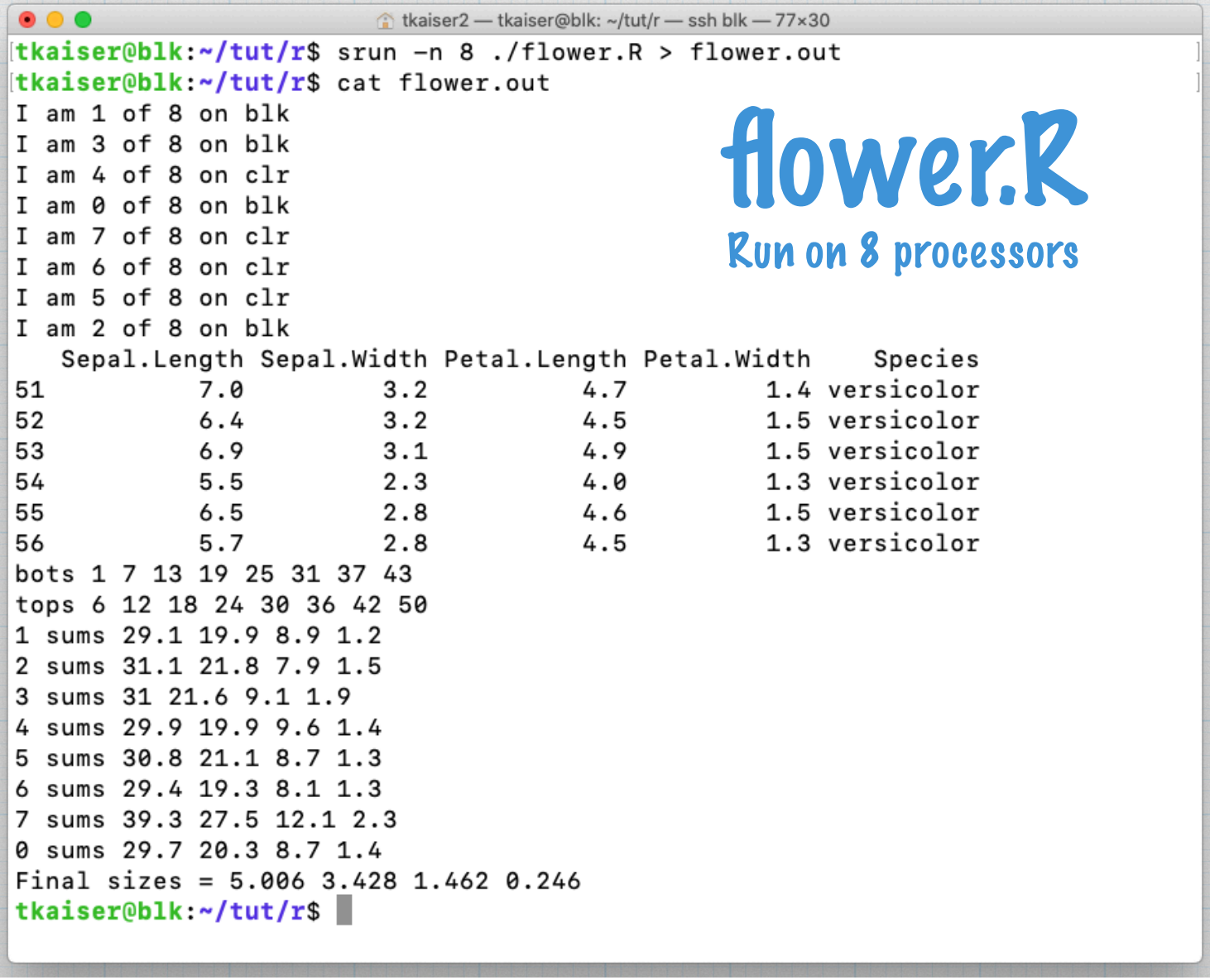
- Proc 0 gets the "iris" data set
  - Broadcasts Versicolor to all procs
  - Sends a portion of Setosa to each
- Each does "math" on their portion
- All send data back to proc 0 for final sums

```
#!/usr/bin/env Rscript
if (!is.loaded("mpi_initialize")) {
    library("Rmpi")
    }
mpi_comm_world<-0
myid <- mpi.comm.rank(comm=mpi_comm_world)
numprocs <- mpi.comm.size(comm=mpi_comm_world)
myname <- mpi.get.processor.name()
cat("I am",myid,"of",numprocs,"on",myname,"\n")
if (myid == 0 ) {
  library(datasets)
  set2<-iris[iris$Species == "versicolor", ]
} else {
# For the bcast we need to set input
# to something on every processor
# even though technically it is not used.
  set2<-integer(1)
}
Sys.sleep(2)
set2<-mpi.bcast.Robj(set2,0,comm=mpi_comm_world)
if (myid == 1) {
    head(set2)
}
Sys.sleep(2)
```

Sending Objects

```
Sys.sleep(2)
if (myid == 0 ) {
  set1<-iris[iris$Species == "setosa", ]
  each=as.integer(nrow(set1)/numprocs)
# We are going to send a section of "setosa"
# to each processor. This gives the ranges.
  bots<-integer(numprocs)
  tops<-integer(numprocs)
  for (n in 1:numprocs) {
    bots[n]<-(n-1)*each+1
    tops[n]<-bots[n]+each-1
  }
  tops[numprocs]=nrow(set1)
  cat("bots",bots,"\n")
  cat("tops",tops,"\n")
  myset<-set1[bots[1]:tops[1],]
  for (n in 1:(numprocs-1)) {
    tosend<-set1[bots[n+1]:tops[n+1],]
    mpi.send.Robj(tosend,n,tag=1234,comm=mpi_comm_world)
  }
} else {
   status<-as.integer(0)
   myset<-mpi.recv.Robj(0,tag=1234,comm=mpi_comm_world,status)
}
#head(myset)
#Sepal.Length Sepal.Width Petal.Length Petal.Width
sizes<-c(sum(myset[["Sepal.Length"]]),sum(myset[["Sepal.Width"]]),
         sum(myset[["Petal.Length"]]),sum(myset[["Petal.Width"]]))
cat(myid,"sums",sizes,"\n")
thetot<-mpi.reduce(sizes, type=2, op="sum",dest = 0, comm = mpi_comm_world)
if(myid == 0) {
  thetot=thetot/nrow(set1)
  cat("Final sizes =",thetot,"\n")
}
bonk<-mpi.finalize()
```

Sending Array


tkaiser2 — tkaiser@blk: ~/tut/r — ssh blk — 77×30
tkaiser@blk:~/tut/r$ srun -n 8 ./flower.R > flower.out
tkaiser@blk:~/tut/r$ cat flower.out
I am 1 of 8 on blk
I am 3 of 8 on blk
I am 4 of 8 on clr
I am 0 of 8 on blk
I am 7 of 8 on clr
I am 6 of 8 on clr
I am 5 of 8 on clr
I am 2 of 8 on blk
Sepal.Length Sepal.Width Petal.Length Petal.Width Species
51 7.0 3.2 4.7 1.4 versicolor
52 6.4 3.2 4.5 1.5 versicolor
53 6.9 3.1 4.9 1.5 versicolor
54 5.5 2.3 4.0 1.3 versicolor
55 6.5 2.8 4.6 1.5 versicolor
56 5.7 2.8 4.5 1.3 versicolor
bots 1 7 13 19 25 31 37 43
tops 6 12 18 24 30 36 42 50
1 sums 29.1 19.9 8.9 1.2
2 sums 31.1 21.8 7.9 1.5
3 sums 31 21.6 9.1 1.9
4 sums 29.9 19.9 9.6 1.4
5 sums 30.8 21.1 8.7 1.3
6 sums 29.4 19.3 8.1 1.3
7 sums 39.3 27.5 12.1 2.3
0 sums 29.7 20.3 8.7 1.4
Final sizes = 5.006 3.428 1.462 0.246
tkaiser@blk:~/tut/r$
flower.R
Run on 8 processors

```
tkaiser@clr:~/tut/r$ cat both
192.168.0.17
192.168.0.17
192.168.0.17
192.168.0.17
192.168.0.40
192.168.0.40
192.168.0.40
192.168.0.40

tkaiser@clr:~/tut/r$ ml python/3.9.4_o
tkaiser@clr:~/tut/r$ mpirun -x R_LIBS_USER -x LD_LIBRARY_PATH -hostfile ~/bin/both -n 8 ./r_ex01.R
[1][1] "I R am 2 of 8 on clr"
[1] "I R am 3 of 8 on clr"
[1][1] "I R am 5 of 8 on blk"
 "I R am 4 of 8 on blk"
 "I R am 0 of 8 on clr"
NULL
[1] "I R am 1 of 8 on clr"
[1] "got"  "5678"
[1] "I R am 7 of 8 on blk"
[1] "I R am 6 of 8 on blk"

tkaiser@clr:~/tut/r$ ml python/3.9.4
tkaiser@clr:~/tut/r$ mpirun -envlist R_LIBS_USER,LD_LIBRARY_PATH -f both -n 8 ./r_ex01.R
[1] "I R am 0 of 8 on clr"
[1][1] "I R am 4 of 8 on clr"
[1] "I R am 6 of 8 on clr"
 "I R am 2 of 8 on clr"
[1] "I R am 1 of 8 on blk"
[1] "I R am 5 of 8 on blk"
[1][1] "I R am 3 of 8 on blk"
 "I R am 7 of 8 on blk"
NULL
[1] "got"  "5678"
tkaiser@clr:~/tut/r$
```

By the way...

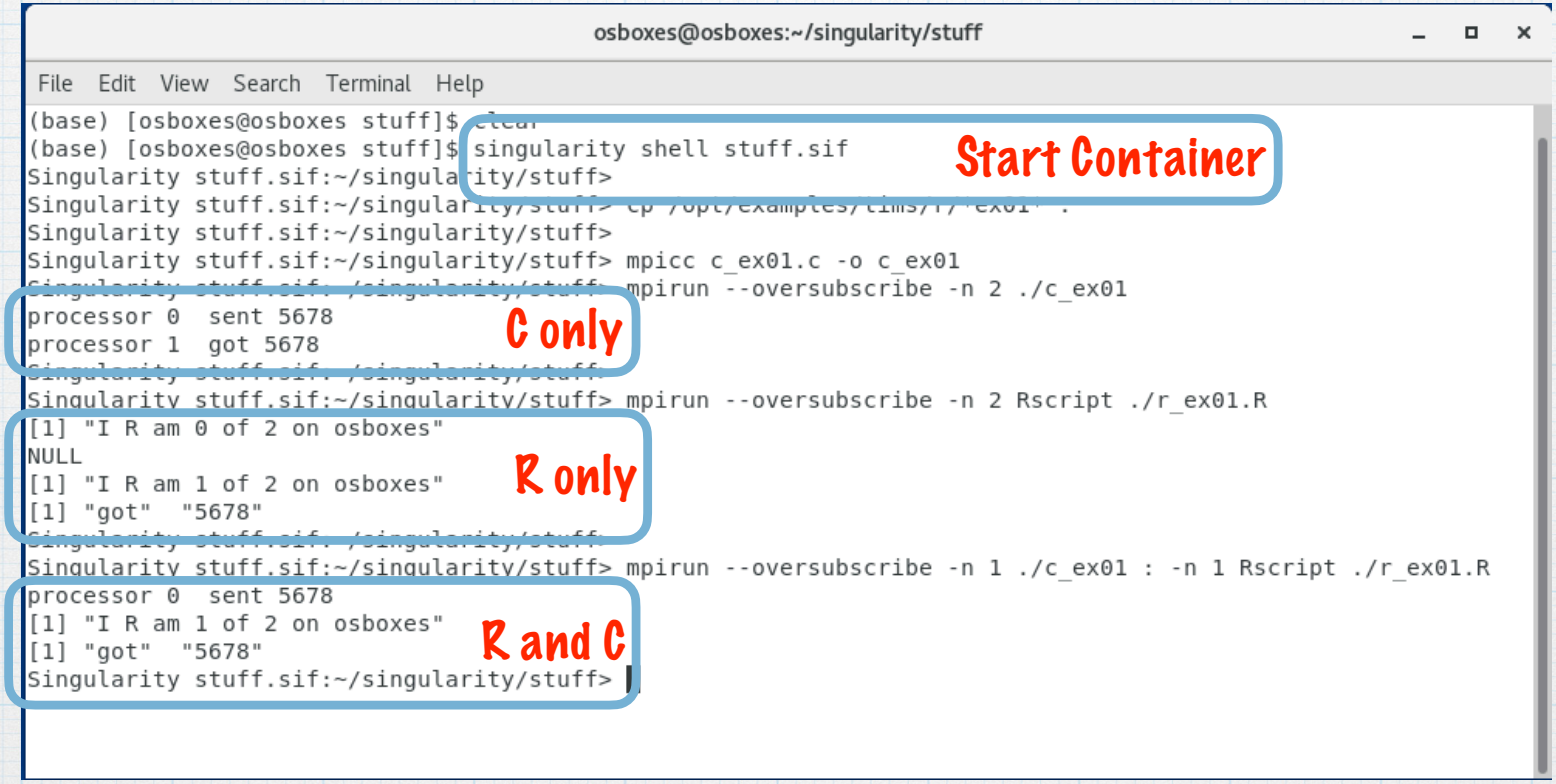
* You can "carefully" mix Rmpi with normal MPI programs
* I have a container that has all the "stuff" from todays talks

Running C and R together

```
tkaiser@clr:~/tut/r$ ml python/3.9.4            # also loads MPICH2 version of MPI

tkaiser@clr:~/tut/r$ mpicc c_ex01.c -o c_ex01  # build the C version

tkaiser@clr:~/tut/r$ mpirun -np 1 ./c_ex01 : -np 1 ./r_ex01.R. # run them together
 C Hello from clr 0 2
 C processor 0  sent 5678
[1] "I R am 1 of 2 on clr"
[1] "got"  "5678"tkaiser@clr:~/tut/r$
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <math.h>

/*************************************************
This is a simple send/receive program in MPI
*************************************************/
int main(int argc,char *argv[])
{
    int myid, numprocs;
    int tag,source,destination,count,resultlen;
    int buffer;
    MPI_Status status;
    char myname[MPI_MAX_PROCESSOR_NAME] ;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(myname,&resultlen);
    printf("C Hello from %s %d %d\n",myname,myid,numprocs);
    tag=1234;
    source=0;
    destination=1;
    count=1;
    if(myid == source){
      buffer=5678;
      MPI_Send(&buffer,count,MPI_INT,destination,tag,MPI_COMM_WORLD);
      printf(" C processor %d  sent %d\n",myid,buffer);
    }
    if(myid == destination){

MPI_Recv(&buffer,count,MPI_INT,source,tag,MPI_COMM_WORLD,&status);
        printf("C processor %d  got %d\n",myid,buffer);
    }
    MPI_Finalize();
}
```

```r
#!/usr/bin/env Rscript
if (!is.loaded("mpi_initialize")) {
    library("Rmpi")
    }
mpi_comm_world<-0
myid <- mpi.comm.rank(comm=mpi_comm_world)
numprocs <- mpi.comm.size(comm=mpi_comm_world)
myname <- mpi.get.processor.name()
tag<-1234;
source<-0;
destination<-1;
count<-1;
paste("I R am",myid,"of",numprocs,"on",myname)
if(myid == source){
        buffer<-as.integer(5678)
        mpi.send(buffer, 1, destination, tag,  comm=mpi_comm_world)
        }
if(myid == destination){
        x<-as.integer(1234)
        x<-mpi.recv(x, 1, source, tag,  comm=mpi_comm_world, status=0)
        print(c("got",x))
        }
bonk<-mpi.finalize()
```




```
singularity pull --name stuff.sif shub://timkphd/stuff:05221357
```

https://petra.acns.colostate.edu/docs/containers/

Rmpi has extensions for launching "workers"

```
# Load the R MPI package if it is not already loaded.
if (!is.loaded("mpi_initialize")) { library("Rmpi") }

print(mpi.universe.size())
# Spawn 4 slaves (change this number appropriately)
mpi.spawn.Rslaves(nslaves=4)
#
# In case R exits unexpectedly, have it automatically clean up
# resources taken up by Rmpi (slaves, memory, etc...)
.Last <- function(){
       if (is.loaded("mpi_initialize")){
           if (mpi.comm.size(1) > 0){
               print("Please use mpi.close.Rslaves() to close slaves.")
               mpi.close.Rslaves()
           }
           print("Please use mpi.quit() to quit R")
           .Call("mpi_finalize")
       }
}
# Tell all slaves to return a message identifying themselves
mpi.remote.exec(paste("I am",mpi.comm.rank(),"of",mpi.comm.size()))

# Tell all slaves to close down, and exit the program
mpi.close.Rslaves()
mpi.quit()
```

This is from the Rmpi web page...
Don't blame me.

What I have seen...

```
semikln:eq timk$ mpiexec -n 1 Rscript spawn.R
[1] 6
   4 slaves are spawned successfully. 0 failed.
master (rank 0, comm 1) of size 5 is running on: semikln
slave1 (rank 1, comm 1) of size 5 is running on: semikln
slave2 (rank 2, comm 1) of size 5 is running on: semikln
slave3 (rank 3, comm 1) of size 5 is running on: semikln
slave4 (rank 4, comm 1) of size 5 is running on: semikln
$slave1
[1] "I am 1 of 5"

$slave2
[1] "I am 2 of 5"

$slave3
[1] "I am 3 of 5"

$slave4
[1] "I am 4 of 5"

[1] 1
semikln:eq timk$
```

* Some systems will not allow spawn
* Launching tasks without the knowledge of the schedule can lead to contention
* Other ways to do this - Bag of tasks
* However, we will come back to this example

Most of the routines & values in Rmpi

Robj routines are extensions for sending data frames
Master/Slave routines are not part of the standard

mpi.abort	mpi.cart.shift	mpi.gatherv	mpi.is.master	mpi.remote.exec
mpi.allgather	mpi.cartdim.get	mpi.get.count	mpi.isend	mpi.request.maxsize
mpi.allgather.Robj	mpi.close.Rslaves	mpi.get.processor.name	mpi.isend.Robj	mpi.scatter
mpi.allgatherv	mpi.comm.c2f	mpi.get.sourcetag	mpi.parApply	mpi.scatter.Robj
mpi.allreduce	mpi.comm.disconnect	mpi.hostinfo	mpi.parCapply	mpi.scatter.Robj2slave
mpi.any.source	mpi.comm.dup	mpi.iapply	mpi.parLapply	mpi.scatterv
mpi.any.tag	mpi.comm.free	mpi.iapplyLB	mpi.parMM	mpi.send
mpi.apply	mpi.comm.get.parent	mpi.info.create	mpi.parRapply	mpi.send.Robj
mpi.applyLB	mpi.comm.is.null	mpi.info.free	mpi.parReplicate	mpi.sendrecv
mpi.barrier	mpi.comm.maxsize	mpi.info.get	mpi.parSapply	mpi.setup.rngstream
mpi.bcast	mpi.comm.rank	mpi.info.set	mpi.parSim	mpi.spawn.Rslaves
mpi.bcast.cmd	mpi.comm.remote.size	mpi.intercomm.merge	mpi.probe	mpi.status.maxsize
mpi.bcast.data2slave	mpi.comm.set.errhandler	mpi.iparApply	mpi.proc.null	mpi.test
mpi.bcast.Rfun2slave	mpi.comm.size	mpi.iparCapply	mpi.quit	mpi.testall
mpi.bcast.Robj	mpi.comm.spawn	mpi.iparLapply	mpi.realloc	mpi.testany
mpi.bcast.Robj2slave	mpi.comm.test.inter	mpi.iparMM	mpi.realloc.comm	mpi.testsome
mpi.cancel	mpi.dims.create	mpi.iparRapply	mpi.realloc.request	mpi.universe.size
mpi.cart.coords	mpi.exit	mpi.iparReplicate	mpi.realloc.status	mpi.wait
mpi.cart.create	mpi.finalize	mpi.iparSapply	mpi.recv	mpi.waitall
mpi.cart.get	mpi.gather	mpi.iprobe	mpi.recv.Robj	mpi.waitany
mpi.cart.rank	mpi.gather.Robj	mpi.irecv	mpi.reduce	mpi.waitsome
mpi.abort	mpi.cart.shift	mpi.gatherv	mpi.is.master	mpi.remote.exec

Shifting Directions

- We'll next look at the "parallel" collection of libraries
- We'll come back to MPI to:
  - Compare performance
  - Show how we can combine both
  - Look at more examples
    - Earthquake hazard map
    - Bag of task
    - Finite difference code

"normal" Parallel

* library(parallel)
* library(dplyr)
* library(doParallel)
* Not discussed but probably should "snow"

parallel-package {parallel} R Documentation

Support for Parallel Computation

Description

Support for parallel computation, including random-number generation.

Details

This package was first included with R 2.14.0 in 2011.

There is support for multiple RNG streams with the `"L'Ecuyer-CMRG"` RNG: see `nextRNGStream`.

It contains functionality derived from and pretty much equivalent to that contained in packages **multicore** (formerly on CRAN, with some low-level functions renamed and not exported) and **snow** (for socket clusters only, but MPI and NWS clusters generated by **snow** are also supported). There have been many enhancements and bug fixes since 2011.

This package also provides `makeForkCluster` to create socket clusters by forking (not Windows).

For a complete list of exported functions, use `library(help = "parallel")`.

Author(s)

Brian Ripley, Luke Tierney and Simon Urbanek

Maintainer: R Core Team R-core@r-project.org

See Also

Parallel computation involves launching worker processes: functions `psnice` and `pskill` in package **tools** provide means to manage such processes.

Information of package 'Parallel'

```
Description:

Package:            parallel
Version:            3.6.0
Priority:           base
Title:              Support for Parallel computation in
                    R
Author:             R Core Team
Maintainer:         R Core Team <R-core@r-project.org>
Description:        Support for parallel computation, including by forking (taken from  package
                    multicore), by sockets (taken from package snow) and random-number generation.
License:            Part of R 3.6.0
Imports:            tools, compiler
Suggests:           methods
Enhances:           snow, nws, Rmpi
NeedsCompilation:   yes
Built:              R 3.6.0; x86_64-apple-darwin15.6.0;
                    2019-04-26 16:31:55 UTC; unix

Index:

children                Low-level Functions for Management of Forked
                        Processes
clusterApply            Apply Operations using Clusters
detectCores             Detect the Number of CPU Cores
makeCluster             Create a Parallel Socket Cluster
mcaffinity              Get or Set CPU Affinity Mask of the Current
                        Process
mcfork                  Fork a Copy of the Current R Process
mclapply                Parallel Versions of 'lapply' and 'mapply'
                        using Forking
mcparallel              Evaluate an R Expression Asynchronously in a
                        Separate Process
nextRNGStream           Implementation of Pierre L'Ecuyer's RngStreams
parallel-package        Support for Parallel Computation
pvec                    Parallelize a Vector Map Function using Forking
splitIndices            Divide Tasks for Distribution in a Cluster

Further information is available in the following vignettes in directory
‘/Library/Frameworks/R.framework/Resources/library/parallel/doc’:

parallel: Package 'parallel' (source, pdf)
```

Information of package 'doParallel'

```
Description:

Package:                    doParallel
Type:                       Package
Title:                      Foreach Parallel Adaptor
                            for the 'parallel' Package
Version:                    1.0.14
Authors@R:                  c(person("Rich", "Calaway",role="cre",email="richcala@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
Author:                     Rich Calaway [cre],Microsoft Corporation [aut,cph], Steve Weston [aut], Dan Tenenbaum [ctb]
Maintainer:                 Rich Calaway <richcala@microsoft.com>
Repository:                 CRAN
Repository/R-Forge/Project:
                            doparallel
Repository/R-Forge/Revision:
                            21
Repository/R-Forge/DateTimeStamp:
                            2018-09-21 22:41:46
Date/Publication:           2018-09-24 19:20:09 UTC
NeedsCompilation:           no
Packaged:                   2018-09-21 22:51:03 UTC;
                            rforge
Built:                      R 3.6.0; ; 2019-04-26
                            19:55:12 UTC; unix

Index:

doParallel-package      The doParallel Package
registerDoParallel      registerDoParallel

Further information is available in the following
vignettes in directory
‘/Users/timk/Library/R/3.6/library/doParallel/doc’:

gettingstartedParallel: Getting Started with doParallel
      and foreach (source, pdf)
```

doParallel-package {doParallel} R Documentation

The doParallel Package

Description

The doParallel package provides a parallel backend for the foreach/%dopar% function using the parallel package of R 2.14.0 and later.

Details

Further information is available in the following help topics:

registerDoParallel register doParallel to be used by foreach/%dopar%

To see a tutorial introduction to the doParallel package, use vignette("gettingstartedParallel"). To see a tutorial introduction to the foreach package, use vignette("foreach").

To see a demo of doParallel computing the sinc function, use demo(sincParallel).

Some examples (in addition to those in the help pages) are included in the "examples" directory of the doParallel package. To list the files in the examples directory, use list.files(system.file("examples", package="doParallel")). To run the bootstrap example, use source(system.file("examples", "bootParallel.R", package="doParallel")). This is a simple benchmark, executing both sequentally and in parallel. There are many more examples that come with the foreach package, which will work with the doParallel package if it is registered as the parallel backend.

For a complete list of functions with individual help pages, use library(help="doParallel").

Information of package 'foreach'

```
Description:

Package:                    foreach
Type:                       Package
Title:                      Provides Foreach Looping
                            Construct for R
Version:                    1.4.4
Authors@R:                  c(person("Rich", "Calaway",
                            role="cre",
                            email="richcala@microsoft.com"),
                            person("Microsoft",
                            role=c("aut", "cph")),
                            person("Steve", "Weston",
                            role="aut"))
Description:                Support for the foreach looping construct.  Foreach is an idiom that allows for
                            iterating over elements in a collection, without the use of an explicit loop
                            counter.  This package in particular is intended to be used for its return
                            value, rather than for its side effects.  In that sense, it is similar to the
                            standard lapply function, but doesn't require the evaluation of a function.
                            Using foreach without side effects also facilitates executing the loop in parallel.
Depends:                    R (>= 2.5.0)
Imports:                    codetools, utils, iterators
Suggests:                   randomForest
Enhances:                   compiler, doMC, RUnit, doParallel
License:                    Apache License (== 2.0)
Author:                     Rich Calaway [cre], Microsoft [aut, cph], Steve Weston [aut]
Maintainer:                 Rich Calaway <richcala@microsoft.com>
Repository:                 CRAN
Repository/R-Forge/Project:
                            foreach
Repository/R-Forge/Revision:
                            31
Repository/R-Forge/DateTimeStamp:
                            2017-12-08 23:08:19
Date/Publication:           2017-12-12 22:37:36 UTC
NeedsCompilation:           no
Packaged:                   2017-12-08 23:33:48 UTC;
                            rforge
Built:                      R 3.6.0; ; 2019-04-26 19:54:56 UTC; unix
```

```
Index:

foreach                 foreach
foreach-ext             Foreach Extension Functions
foreach-package         The Foreach Package
getDoParWorkers         Functions Providing Information on the doPar Backend
getDoSeqWorkers         Functions Providing Information on the doSeq Backend
registerDoSEQ           registerDoSEQ
setDoPar                setDoPar
setDoSeq                setDoSeq

Further information is available in the following
vignettes in directory
‘/Users/timk/Library/R/3.6/library/foreach/doc’:

foreach: foreach Manual (source, pdf)
nested: Nesting Foreach Loops (source, pdf)
```

dplyr-package {dplyr} R Documentation

dplyr: a grammar of data manipulation

Description

dplyr provides a flexible grammar of data manipulation. It's the next iteration of plyr, focused on tools for working with data frames (hence the *d* in the name).

Details

It has three main goals:

- Identify the most important data manipulation verbs and make them easy to use from R.
- Provide blazing fast performance for in-memory data by writing key pieces in C++ (using Rcpp)
- Use the same interface to work with data no matter where it's stored, whether in a data frame, a data table or database.

To learn more about dplyr, start with the vignettes: `browseVignettes(package = "dplyr")`

Package options

`dplyr.show_progress`

Should lengthy operations such as `do()` show a progress bar? Default: `TRUE`

Package configurations

These can be set on a package-by-package basis, or for the global environment. See `pkgconfig::set_config()` for usage.

`dplyr::na_matches`

Should `NA` values be matched in data frame joins by default? Default: `"na"` (for compatibility with dplyr v0.5.0 and earlier, subject to change), alternative value: `"never"` (the default for database backends, see `join.tbl_df()`).

dplyr

* Lots of functionality
* Most is syntactic sugar for other commands
* Parallel for loops use the "pipe" operator
  * `foreach(ijk=1:nt) %dopar% {`

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column**

&

Each **observation**, or **case**, is in its own **row**

pipes

x %>% f(y) becomes **f(x, y)**

https://github.com/rstudio/cheatsheets

Data Transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

pipes
x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

group_by(.data, ..., add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …)
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
distinct(iris, Species)

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …) Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	| xor()
>	>=	!is.na()	!	&

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL) Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1) Extract column values as a vector. Choose by name or index.

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank w ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISC

dplyr::**case_when()** - multi-case if_else()

```
iris %>% mutate(Species = case_when(
                Species == "versicolor" ~ "versi",
                Species == "virginica" ~ "virgi",
                TRUE ~ Species))
```

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

Combine Tables

COMBINE VARIABLES

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(…) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…)
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…)
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…)
Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…)
Join data. Retain all values, all rows.

Use **by = c("col1", "col2", …)** to specify one or more common columns to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns that

COMBINE CASES

Use **bind_rows()** to paste tables below each other as they are.

bind_rows(…, .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

intersect(x, y, …)
Rows that appear in both x and y.

setdiff(x, y, …)
Rows that appear in x but not y.

union(x, y, …)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

Use a "**Filtering Join**" to filter one table against the rows of another.

In my original presentation...

* I had a bunch of slides here
* Contains for loops
  * Starts with simple non-parallel loop
  * Shows how to start a parallel session
* Various "broken" loops
* A discussion of what has gone wrong
* Will skip these for now, do a summary, and go straight to "hello world"
* I moved the slides to the end

Some Points From the skipped slides

myout<-foreach

"foreach" is essentially a function it needs someplace for output to go. It takes on the last value in the loop. I have not been able to get ".final" to work.

You should not have "input" on the left had side.

Don't do this "asum<-asum+ijk"

Inputs are promoted to vectors (copied).

Lots of work per iteration is good. (Long function call.)

Calculations occur on different processors or at lease with different pids - Additions are not commutative

Hello World

hellop.R

```
#!/usr/bin/env Rscript
# start the cluster
library(foreach)
library(doParallel)
numprocs=parallel::detectCores()
cl <- makeCluster(numprocs)
registerDoParallel(cl)

# do something
nt<-900
aset <- foreach (ijk=1:nt) %dopar% {
    aset<-c(ijk,Sys.getpid())
}
# stop the cluster
stopCluster(cl)

# use the results
df<-data.frame(iter=integer(),pid=integer())
nresults=length(aset)
for (ijk in 1:nresults){
        r<-unlist(aset[ijk])
        df[nrow(df) + 1,]<-c(r)
    }

str(df)
df2<-data.frame(pid=integer(),its=integer(),load=double())
pids=unique(df['pid'])
for (p in pids[,]) {
    n<-nrow(df[df['pid'] == p,])
    load<-100.0*(n/nt)
    df2[nrow(df2)+1,]<-list(p,n,load)
}
df2<-df2[order(df2$pid),]
print(df2)
```

1. We load the libraries
2. Find out how many cores we have
3. Make and Register our processors

1. Our "work" here is collecting "ijk" and the PID running the iteration
2. We run 900 times
3. Here the results are retuned as a list

1. We convert the list to a data frame
2. We print a summary of # iterations each PID ran

```
tkaiser@clr:~/tut/r$ ./hellop.R
Loading required package: iterators
Loading required package: parallel
'data.frame': 900 obs. of  2 variables:
 $ iter: int  1 2 3 4 5 6 7 8 9 10 ...
 $ pid : int  8691 8692 8690 8689 8691 8692 8691 8692 8691 8692 ...
   pid its       load
4 8689   6  0.6666667
3 8690   9  1.0000000
1 8691 444 49.3333333
2 8692 441 49.0000000
tkaiser@clr:~/tut/r$
```

piris.R

```
#!/usr/bin/env Rscript
# start the cluster

library(foreach)
library(doParallel)
library(tictoc)
numprocs=parallel::detectCores()
cl <- makeCluster(numprocs)
registerDoParallel(cl)

#get our dataset
library(datasets)
set1<-iris[iris$Species == "setosa", ]

#figure out how we are going to split it
each=as.integer(nrow(set1)/numprocs)
bots<-integer(numprocs)
  tops<-integer(numprocs)
  for (n in 1:numprocs) {
    bots[n]<-(n-1)*each+1
    tops[n]<-bots[n]+each-1
  }
tops[numprocs]=nrow(set1)
cat("bots",bots,"\n")
cat("tops",tops,"\n")

#run analysis in parallel
aset <- foreach (ijk=1:length(bots)) %dopar% {
    myset<-set1[bots[ijk]:tops[ijk],]
    aset<-c(sum(myset[["Sepal.Length"]]),
           sum(myset[["Sepal.Width"]]),
           sum(myset[["Petal.Length"]]),
           sum(myset[["Petal.Width"]]),
           tops[ijk]-bots[ijk]+1,Sys.getpid())
    }

str(aset)

stopCluster(cl)

#reduce the data
df<-data.frame(sl=double(),sw=double(),pl=double(),pw=double(),vals=double(),pid=integer())
for (ijk in 1:4){
     df[nrow(df) + 1,] <-aset[[ijk]]
  }
df
colSums(df[,1:4])/sum(df[['vals']])
```

We load the libraries
Find out how many cores we have
Make and Register our processors

1. This is similar in concept to the MPI version
2. We "manually" break up our data set by creating the arrays bots & tops which are indices into our data
3. Our foreach loop is over our "subsets" i.e. "myset"

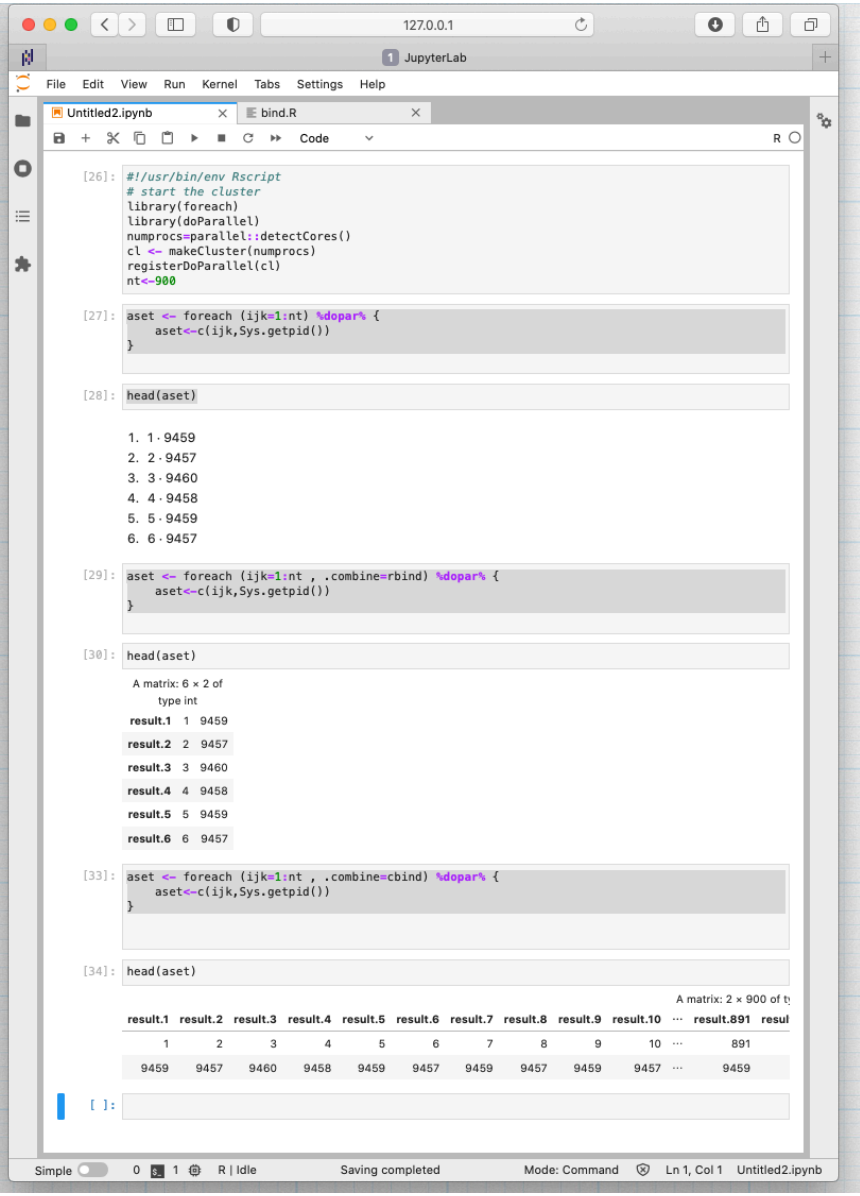
piris.R

```
#!/usr/bin/env Rscript
# start the cluster

library(foreach)
library(doParallel)
library(tictoc)
numprocs=parallel::detectCores()
cl <- makeCluster(numprocs)
registerDoParallel(cl)

#get our dataset
library(datasets)
set1<-iris[iris$Species == "setosa", ]

#figure out how we are going to split it
each=as.integer(nrow(set1)/numprocs)
bots<-integer(numprocs)
  tops<-integer(numprocs)
  for (n in 1:numprocs) {
    bots[n]<-(n-1)*each+1
    tops[n]<-bots[n]+each-1
  }
tops[numprocs]=nrow(set1)
cat("bots",bots,"\n")
cat("tops",tops,"\n")

#run analysis in parallel
aset <- foreach (ijk=1:length(bots)) %dopar% {
    myset<-set1[bots[ijk]:tops[ijk],]
    aset<-c(sum(myset[["Sepal.Length"]]),
           sum(myset[["Sepal.Width"]]),
           sum(myset[["Petal.Length"]]),
           sum(myset[["Petal.Width"]]),
           tops[ijk]-bots[ijk]+1,Sys.getpid())
    }

str(aset)

stopCluster(cl)

#reduce the data
df<-data.frame(sl=double(),sw=double(),pl=double(),pw=double(),vals=double(),pid=integer())
for (ijk in 1:4){
    df[nrow(df) + 1,] <-aset[[ijk]]
  }
df
colSums(df[,1:4])/sum(df[['vals']])
```

```
tkaiser@clr:~/tut/r$ ./piris.R
Loading required package: iterators
Loading required package: parallel
bots 1 13 25 37
tops 12 24 36 50
List of 4
 $ : num [1:6] 58.8 40.2 17.6 2.6 12 ...
 $ : num [1:6] 62.1 43.4 17 3.4 12 ...
 $ : num [1:6] 60.7 41 18.3 2.7 12 ...
 $ : num [1:6] 68.7 46.8 20.2 3.6 14 ...
    sl   sw   pl  pw vals  pid
1 58.8 40.2 17.6 2.6   12 8905
2 62.1 43.4 17.0 3.4   12 8904
3 60.7 41.0 18.3 2.7   12 8906
4 68.7 46.8 20.2 3.6   14 8907
   sl    sw    pl    pw
5.006 3.428 1.462 0.246
tkaiser@clr:~/tut/r$
```

Foreach options

* Foreach has many options
* .combine = what to do with your results on return
* Can specify a function
* cbind and rind will return results as a matrix



Hello world Again

bind.R

```
#!/usr/bin/env Rscript
# start the cluster
library(foreach)
library(doParallel)
numprocs=parallel::detectCores()
cl <- makeCluster(numprocs)
registerDoParallel(cl)
nt<-900

# default is a list
aset <- foreach (ijk=1:nt) %dopar% {
    aset<-c(ijk,Sys.getpid())
}

head(aset)

aset <- foreach (ijk=1:nt , .combine=rbind) %dopar% {
    aset<-c(ijk,Sys.getpid())
}

head(aset)

aset <- foreach (ijk=1:nt , .combine=cbind) %dopar% {
    aset<-c(ijk,Sys.getpid())
}

head(aset)
```

Digression

Forcing tasks to cores in
a more or less portable way

Link

You can combine MPI and parallel foreach

* Multilevel parallelism
* Abstract Example
  * MPI can be used to grab nodes
  * Foreach can launch tasks on nodes
  * Maybe useful if you have GPU nodes
* Here we use "spawn" to get MPI tasks and then foreach to do multiple matrix inversions

invert.R

These routines can be used to get the core or force a task to a core.

```
dyn.load("mapit.so")
 threadfunc <-function() {
     trd=c(0)
     out <- .C("findcore",ic=as.integer(trd),package="thread")
 return(out$ic)
 }
 forceit <-function(core) {
     trd=c(core)
     out <- .C("forcecore",ic=as.integer(trd),package="core")
 return(out$ic)
 }
```

Load libs.

```
library(foreach)
library(doParallel)
library(tictoc)
```

threads.txt contains "nt" the number of tasks to launch and the number of inverts to do

```
con=file("threads.txt","r")
linn=readLines(con,1)
nt<-strtoi(linn)
```

Start our tasks

```
cl <- makeCluster(nt)
registerDoParallel(cl)
size <- 2000
```

In our parallel loop set up the matrices and do the inverts

invert.R

```
tic()
ptime <- system.time({
   r <- foreach(ijk=1:nt, .combine=cbind) %dopar% {
      dyn.load("mapit.so")
      dia <- 0.1
      docore <- 1+((ijk-1) %% 23)
      forceit(docore)
      # For each row and for each column, assign values based on position
      mymat <- matrix(10.0,nrow=size, ncol=size)
      for(i in 1:dim(mymat)[1]) {
           mymat[i,i] <- dia
      }
      b <- solve(mymat)
      one=c(ijk,threadfunc())
   }
})
```

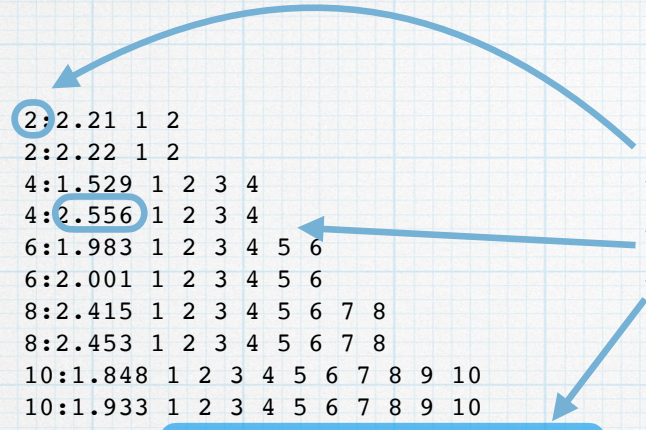
Report results

```
tim<-toc()
dt=tim[[2]][[1]]-tim[[1]][[1]]
stats=c("results",nt,dt)
print(stats)
#print(r)
x<-paste(c(c(nt,dt),r[2,]))
x<-paste(nt,dt,sep=":")
t<-tempfile(pattern="output")
v<-strsplit(t,"/")
con <- file(v[[1]][4],"w")
vect <- 1:nt
for(ijk in 1:nt) {
   vect[ijk]=r[2,ijk]
}
svect <- sort(vect)
#print(r)
#print(svect)
writeLines(paste(c(x)),con=con,sep = "\n")
writeLines(paste(c(svect)),con=con,sep = " ")
writeLines(" ",con=con,sep = "\n")
```

spawn2.R

```
# Load the R MPI package if it is not already loaded.
if (!is.loaded("mpi_initialize")) {
    library("Rmpi")
    }

print(mpi.universe.size())
# Spawn 2 slaves (change this number appropriately)
mpi.spawn.Rslaves(nslaves=2)
#
# In case R exits unexpectedly, have it automatically clean up
# resources taken up by Rmpi (slaves, memory, etc...)
.Last <- function(){
       if (is.loaded("mpi_initialize")){
           if (mpi.comm.size(1) > 0){
               print("Please use mpi.close.Rslaves() to close slaves.")
               mpi.close.Rslaves()
           }
           print("Please use mpi.quit() to quit R")
           .Call("mpi_finalize")
       }
}
# Tell all slaves to return a message identifying themselves
library("parallel")
mpi.remote.exec(paste("I am",mpi.comm.rank(),"of",mpi.comm.size()))
library(tictoc)
tic()
mpi.remote.exec(paste(source("invert.R",verbose=TRUE)))
toc()
# Tell all slaves to close down, and exit the program
mpi.close.Rslaves()
mpi.quit()
```

We wrap invert.R in our MPI spawn example and get two levels of parallelism



```
2:2.21 1 2
2:2.22 1 2
4:1.529 1 2 3 4
4:2.556 1 2 3 4
6:1.983 1 2 3 4 5 6
6:2.001 1 2 3 4 5 6
8:2.415 1 2 3 4 5 6 7 8
8:2.453 1 2 3 4 5 6 7 8
10:1.848 1 2 3 4 5 6 7 8 9 10
10:1.933 1 2 3 4 5 6 7 8 9 10
12:2.537 1 2 3 4 5 6 7 8 9 10 11 12
12:2.565 1 2 3 4 5 6 7 8 9 10 11 12
14:1.971 1 2 3 4 5 6 7 8 9 10 11 12 13 14
14:2.001 1 2 3 4 5 6 7 8 9 10 11 12 13 14
16:1.95 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
16:1.996 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
18:2.012 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
18:2.12 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
20:2.462 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
20:2.549 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
22:1.959 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
22:2.039 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
24:3.793 1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24:3.865 1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
[timk@colostate.edu@shas0137 2440169]$
```

Reported:

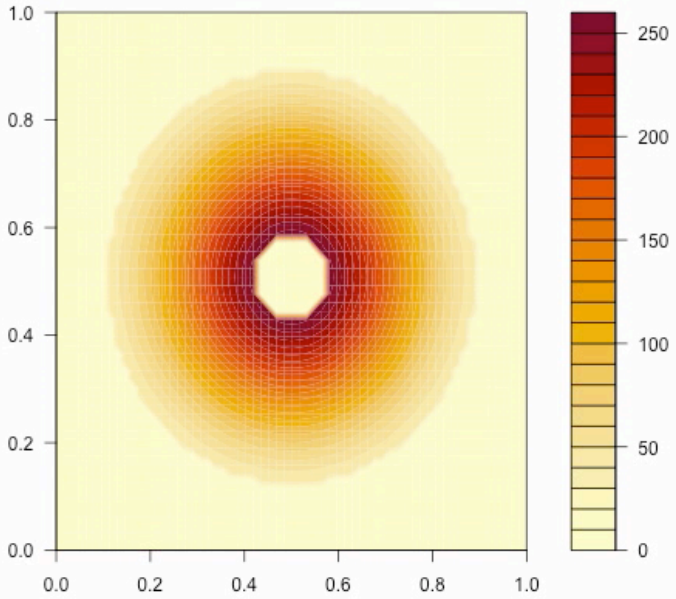
* # tasks/inverts
* runtime
* cores on which tasks are run. 2 lines per set because we are running on two nodes.

We skip core 0 so at 24 tasks we have reused core 1 = slowdown

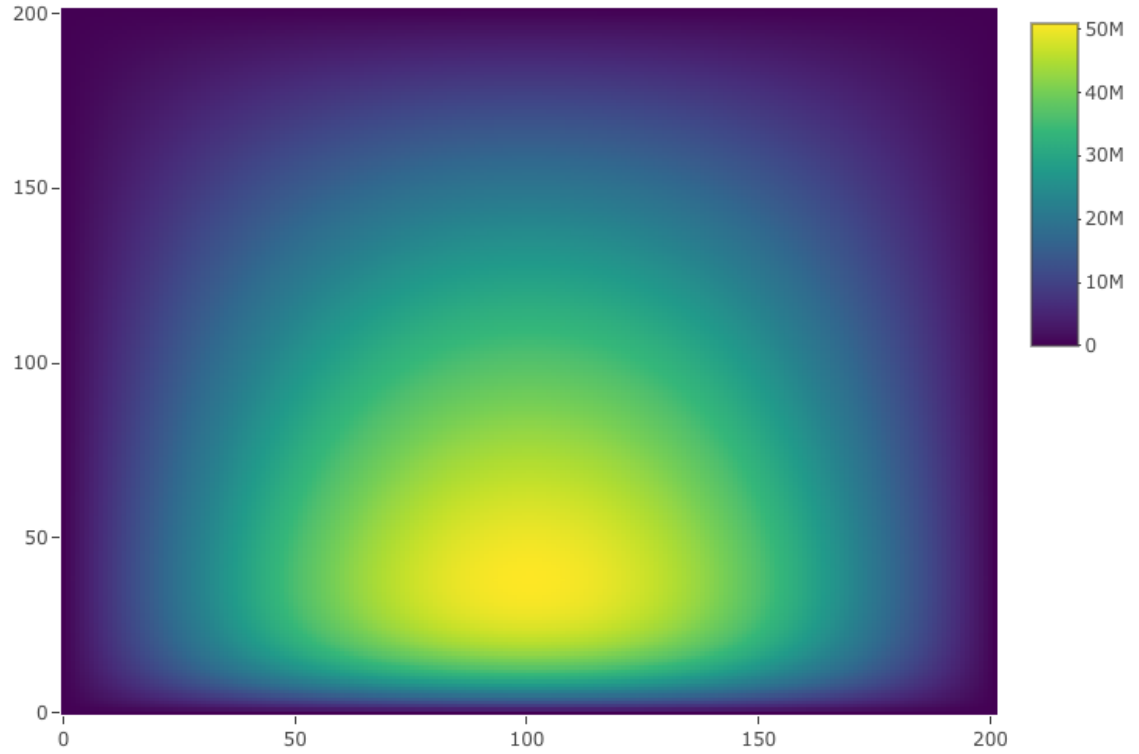
Direction From Here

* Earthquake hazard map for California
  * MPI
  * Parallel R
* Bag of task parallelism
* Finite difference code

Bag of task output

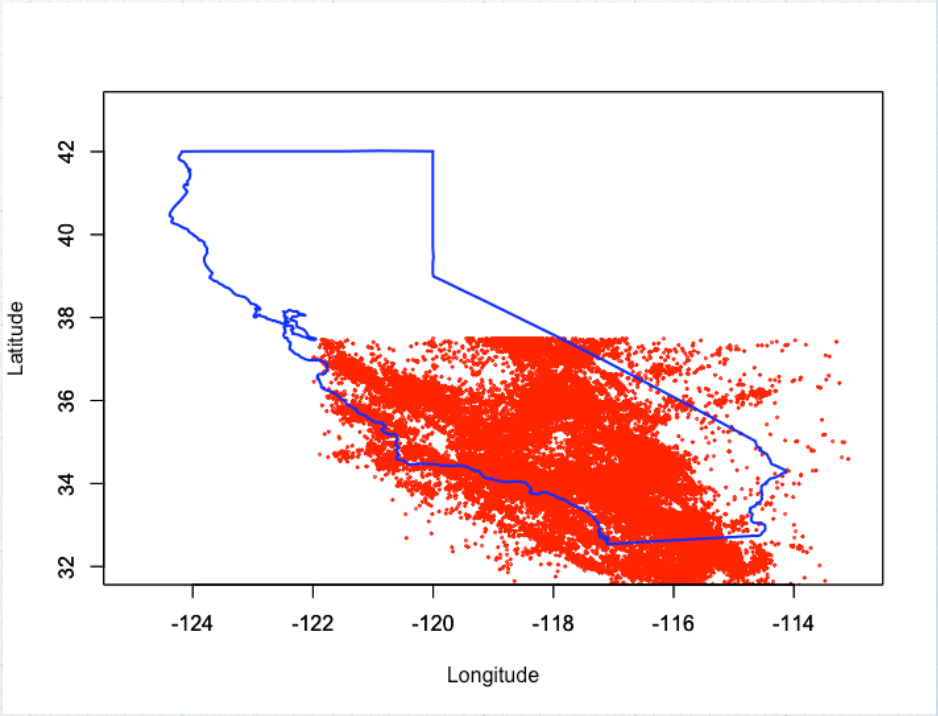


Stommel Output



Earthquake Hazard Map

* Earthquakes in Southern California from 1981 to 2018
* Over 600k in database - 94Mb
* Actually learned something - earthquakes are slightly more likely around lunch time



Data Source

http://scedc.caltech.edu/research-tools/alt-2011-dd-hauksson-yang-shearer.html

http://scedc.caltech.edu/ftp/catalogs/hauksson/Socal_DD/hs_1981_all_comb_K4_A.cat_so_SCSN_v01

http://scedc.caltech.edu/ftp/catalogs/hauksson/Socal_DD/hs_1981_2011_catalog_v01.format

Not Used:

https://ncedc.org/ncedc/catalog-search.html
This is northern CA. There is some overlap and we could combine the two and clean out duplicates.

earthquake intensity as a function of distance

https://web.ics.purdue.edu/~braile/edumod/eqhazard/eqhazard1.htm

634,252 Rows of data

* Earthquakes from 1981 to 2018

```
!       1981 = year
!          1 = month
!          1 = day
!          4 = hour
!         13 = minute
!     55.710 = second
!
!    3301565 = SCSN cuspid (up to 9 digits)
!
!   33.25524 = latitude
! -115.96763 = longitude
!      5.664 = depth (km)
!       2.26 = SCSN calculated preferred magnitude (0.0 if unassigned)
!
!         45 = number of P and S picks used for 1D SSST or 3D location (different from old format)
!         17 =  to nearest statino in km (different from old format)
!       0.21 = rms residual (s) for 1D location; value of 99.0 indicates information not available
!
!          1 = local day/night flag (=0 for day, =1 for night in Calif.)
!
!          4 = location method flag (=1 for SCSN catalog or 1d hypoinverse relocation,
!                =2 for 1D SSST,  =3 for 3D, =4 for waveform cross-correlation)
!              Superseeded by flag below
!        260 = similar event cluster identification number (0 if the event is not relocated with waveform cross-correlation data)
!
!        460 = number of events in similar event cluster (0 if the event is not in similar event clusters)
!
!       76 = number of differential times used to locate this event
!
!      0.300 = est. std. error (km) in absolute horz. position
!      0.800 = est. std. error (km) in absolute depth
!      0.003 = est. std. error (km) in horz. position relative to other events in cluster
!      0.003 = est. std. error (km) in depth relative to other events in cluster
!
!        le = SCSN flag for event type (le=local, qb=quarry, re=regional)
!        ct = for location method (ct=cross-correlation; 3d=3d velocity model; xx= not relocated, SCSN location used)
!       Poly5= the polygon where the earthquake is located.  We used 5 polygons to
!              generate this catlog.
```

http://scedc.caltech.edu/ftp/catalogs/hauksson/Socal_DD/hs_1981_2011_catalog_v01.format

Goal

* Generate a hazard map for a region of SoCal based on historical data
* Hazard:
  * "Sum" of earthquakes
  * Maximum local earthquake
* We add earthquakes by adding maximum V/A
  * Function of distance
  * Function of magnitude

```
for (i in 1:length(lat.seq)){
 for(j in 1:length(lon.seq)) {
  for (row in 1:dorows) {…}
```

tymer package:bonk R Documentation

A nice timing routine: tymer

Description:

A nice timing routine: tymer

tymer.r

Usage:

```
tymer(lab = "", reset = FALSE)
```

Arguments:

lab: A label for the output

reset: Reset the timer so dt is measured from current time.

Value:

Returns time since epoch, date string, dt since last call ,dt since first call and an optional label.'

Examples:

```
 tymer("first call")
[1] "1560955569.153295  2019-06-19 11:02:10       0.000       0.000  first call"
> tymer("second call")
[1] "1560955569.153295  2019-06-19 11:02:48      37.415      37.415  second call"
> tymer("third call")
[1] "1560955569.153295  2019-06-19 11:03:04      16.503      53.918  third call"
> tymer()
[1] "1560955569.153295  2019-06-19 11:03:18      13.544      67.461  "

tymer("do reset",reset=TRUE)
[1] "1560955569.153295  2019-06-19 11:04:46       0.000       0.000  do reset"
> tymer("after reset")
[1] "1560955569.153295  2019-06-19 11:05:01      15.705      15.705  after reset"
```

Our Functions

```
whack <- function(lat1,lon1,lat2,lon2,mag,dep) {
   lat1<-lat1*0.01745329
   lat2<-lat2*0.01745329
   lon1<-lon1*0.01745329
   lon2<-lon2*0.01745329
   ang1<-(sin(lat1) * sin(lat2)) + cos(la      * co      at      * cos(lon2-lon1)
   if(ang1 > 1.0)ang1<-1.0
   d <- 6377.83 * acos(ang1)
   d <- sqrt(d*d+dep*dep)
   if(mag > 0.0){
#  i <-(quake(mag))/(d^2.5)
   i <-atin(d)*(quake(mag))
   } else {
   i=0
   }
   return(i)
}
```

```
atin<-function(dis){
   if(dis > 100.0){
      y<-5.333253/(dis**2)
   }else{
      if(dis < 3.0)dis<-3.0
      x<-log10(dis)
      y<-1.56301016+x*(0.54671034+x*(-0.54724666))
      y<-0.017535744506694952*(10.8**y)
   }
}
```

Attenuation

```
quake <-function(x){
  if(x < 0.1){x=0.1}
  if(x >=0.1 && x <=2.0){
    a<-(-2.146128)
    b<-1.146128
  }
  else {
    if(x <= 3.0) {
      a<-(-0.9058116)
      b<-0.5259698
    } else {
      a<-(-0.279157)
      b<-0.3160013
    }
  t<-a      x
  1(      )
}
```

Magnitude

85% Bogus

Start Up MPI and foreach versions

```
mpi_comm_world<-0
myid <- mpi.comm.rank(comm=mpi_comm_world)
numprocs <- mpi.comm.size(comm=mpi_comm_world)
myname <- mpi.get.processor.name()
source<-0;
paste("I am",myid,"of",numprocs,"on",myname)
```

```
library(parallel)
library(ggplot2)
library(ggmap)
library(maps)
library(mapdata)
library(dplyr)
library(doParallel)
states <- map_data("state")
ca <- states[which(states$region == "california"),]
# or
ca<-states[states$region == "california",]
#plot(ca$lat,ca$lon)
```

Read data

For MPI we break our file into one for each task.

```
l1<-myid %/% 26 +1
l2 <- myid %% 26 +1
sfile<-paste("start",letters[l1],letters[l2],sep="")
print(paste(myid,sfile))
dat<-read.delim(sfile,header=F,sep="")
   thehead<-
c("year","month","day","hour","minute","second","cuspid","latitude","longitude","depth","SCSN","Pan
dS","statino","residual","tod","method","ec","nen","dt","stdpos","stddepth","stdhorrel","stddeprel"
,"le","ct","poly")

colnames(dat)<-thehead
```

```
sfile<-"start"
dat<-read.delim(sfile,header=F,sep="")
   thehead<-
c("year","month","day","hour","minute","second","cuspid","latitude","longitude","depth","SCS
N","PandS","statino","residual","tod","method","ec","nen","dt","stdpos","stddepth","stdhorre
l","stddeprel","le","ct","poly")

colnames(dat)<-thehead
```

Read data

For MPI we break our file into one for each task.

```
l1<-myid %/% 26 +1
l2 <- myid %% 26 +1
sfile<-paste("start",letters[l1],letters[l2],sep="")
print(paste(myid,sfile))
dat<-read.delim(sfile,header=F,sep="")

thehead=c("year","month","day","hour","minute","second","cuspid","latitude","longitude","dep
th","SCSN","PandS","statino","residual","tod","method","ec","nen","dt","stdpos","stddepth","
stdhorrel","stddeprel","le","ct","poly")

colnames(dat)<-thehead
```

getdat.py will do this automatically

```
hexagon:eq tkaiser$ wc start.save
  634252 16432661 98943312 start.save
hexagon:eq tkaiser$ split -l 158563 start start
hexagon:eq tkaiser$ ls -lt | head -5
total 1149272
-rw-r--r--  1 tkaiser  staff   24735828 Jul 17 10:21 startad
-rw-r--r--  1 tkaiser  staff   24735828 Jul 17 10:21 startac
-rw-r--r--  1 tkaiser  staff   24735828 Jul 17 10:21 startab
-rw-r--r--  1 tkaiser  staff   24735828 Jul 17 10:21 startaa
hexagon:eq tkaiser$
```

The loop - MPI version

```
lon.seq<-seq(lonb[1],lonb[2],length=dlon)
lat.seq<-seq(latb[1],latb[2],length=dlat)
df<-data.frame(lat=double(),lon=double(),tot=double(),max=double())
if(myid == 0){tymer(reset=T)}
dorows<-nrow(dat)
dat=subset(dat,,c(latitude,longitude,SCSN,depth))
for(j in 1:length(lon.seq)) {
   mylon<-lon.seq[j]
   for (i in 1:length(lat.seq)){
      mylat=lat.seq[i]
      mymax=0.0
      mytot=0.0
      for (row in 1:dorows) {
         slat<-dat[row,"latitude"]
         slon<-dat[row,"longitude"]
         mag<-dat[row,"SCSN"]
         dep<-dat[row,"depth"]
         ouch<-whack(mylat,mylon,slat,slon,mag,dep)
         if(ouch > mymax){mymax=ouch}
         mytot<-mytot+ouch
      }
      df[nrow(df) + 1,]<-c(mylat,mylon,mytot,mymax)
   }
   if(myid==0){print(tymer(paste("done",mylon)))}
}
# turn our df into two vectors mymax and mytot
   mytot<-df$tot
   mymax<-df$max


thetot<-mpi.reduce(mytot, type=2, op="sum",dest = source, comm = mpi_comm_world)
themax<-mpi.reduce(mymax, type=2, op="max",dest = source, comm = mpi_comm_world)
. . .
bonk<-mpi.finalize()
```

The original "foreach" loop

```
lon.seq<-seq(lonb[1],lonb[2],length=dlon)
lat.seq<-seq(latb[1],latb[2],length=dlat)
cores<-4
cluster <- makeCluster(cores)
registerDoParallel(cluster)
df<-data.frame(lat=double(),lon=double(),tot=double(),max=double())
tymer(reset=T)
jkl=1:length(lat.seq)

dorows<-nrow(dat)
if(TRUE){
   print(tymer(reset=T))
}

for (i in jkl){
   mylat=lat.seq[i]
   aset <- foreach (ijk=1:length(lon.seq)) %dopar% {
      mymax=0.0
      mytot=0.0
      mylon<-lon.seq[ijk]
      for (row in 1:dorows) {
         slat<-dat[row,"latitude"]
         slon<-dat[row,"longitude"]
         mag<-dat[row,"SCSN"]
         dep<-dat[row,"depth"]
         ouch<-whack(mylat,mylon,slat,slon,mag,dep)
         if(ouch > mymax){mymax=ouch}
         mytot=mytot+ouch
      }
      aset<-c(mylat,mylon,mytot,mymax)
   }
   if(TRUE){print(tymer(paste("done",mylat)))}
   nresults=length(aset)
   for (ijk in 1:nresults){
      df[nrow(df) + 1,] <-aset[[ijk]]
   }
}
stopCluster(cluster)
```

```
for (i in jkl){
    mylat=lat.seq[i]
    aset <- foreach (ijk=1:length(lon.seq)) %dopar% {
        mymax=0.0
        mytot=0.0
        mylon<-lon.seq[ijk]
        for (row in 1:dorows) {
            slat<-dat[row,"latitude"]
            slon<-dat[row,"longitude"]
            mag<-dat[row,"SCSN"]
            dep<-dat[row,"depth"]
            ouch<-whack(mylat,mylon,slat,slon,mag,dep)
            if(ouch > mymax){mymax=ouch}
            mytot=mytot+ouch
        }
        aset<-c(mylat,mylon,mytot,mymax,Sys.getpid())
    }
    fprint(aset)
    if(TRUE){print(tymer(paste("done",mylat)))}
    nresults=length(aset)
    for (ijk in 1:nresults){
        r<-unlist(aset[ijk])
        therow=nrow(df) + 1
        r<-unlist(aset[ijk])
        df[therow,1:4]<-r[1:4]
        df[therow,5]<-as.integer(r[5])
    }
}
```

Modified loop to track process ID

```
pids=unique(df['pid'])
for (p in pids[,]) {
    n=nrow(df[df['pid'] == p,])
    print(c(p,n))
}
```

```
tkaiser@clr:~$ while true ; do
> date
> ps -U $USER -L -o pid,lwp,psr,comm,pcpu,%mem | grep -v COMMAND | sort -k3n | grep R
> sleep 2
> done
...
   3239    3239   3 R                100 11.3
Tue May  4 07:37:54 AM MDT 2021
   3239    3239   3 R                100  8.3
Tue May  4 07:37:56 AM MDT 2021
   3366    3366   0 R               36.0  1.4
   3367    3367   0 R               55.5  2.1
   3368    3368   1 R               36.5  1.4
   3369    3369   2 R               37.0  1.4
   3239    3239   3 R               97.9  8.5
Tue May  4 07:37:58 AM MDT 2021
   3366    3366   0 R               18.0  1.4
   3367    3367   0 R               70.5  4.0
   3368    3368   2 R               33.5  2.6
   3369    3369   2 R               18.5  1.4
   3239    3239   3 R               95.7  8.5
Tue May  4 07:38:00 AM MDT 2021
   3366    3366   0 R               12.0  1.4
   3367    3367   0 R               47.0  4.0
   3369    3369   0 R               24.1  2.8
   3239    3239   2 R               93.7  8.5
   3368    3368   2 R               47.5  4.0
Tue May  4 07:38:02 AM MDT 2021
   3239    3239   0 R               91.8  8.5
   3367    3367   0 R               35.2  4.0
   3369    3369   0 R               35.6  4.0
   3366    3366   1 R               19.2  2.9
   3368    3368   2 R               35.6  4.0
Tue May  4 07:38:04 AM MDT 2021
   3366    3366   0 R               36.2  4.3
   3239    3239   1 R               89.3  8.5
   3367    3367   1 R               36.1  4.3
   3368    3368   2 R               36.0  4.3
   3369    3369   3 R               36.3  4.3
...
```

Core utilization while running the loop on a toy problem

Lat/Lon run

Process	Total
3366	12
3367	12
3368	12
3369	12

Saving the data

```
# put our values back in df
mymax<-matrix(df$max,nrow=dlat,ncol=dlon,byrow=TRUE)
mymax<-as.data.frame(mymax,row.names=lat.seq)
colnames(mymax)<-lon.seq
print(mymax)
mytot<-matrix(df$tot,nrow=dlat,ncol=dlon,byrow=TRUE)
mytot<-as.data.frame(mytot,row.names=lat.seq)
colnames(mytot)<-lon.seq
print(mytot)
#sanity check
writeLines(paste("at",lat.seq[1],lon.seq[1],"sum=",mytot[1,1],"max=",mymax[1,1]))
writeLines(paste("at",lat.seq[1],lon.seq[dlon],"sum=",mytot[1,dlon],"max=",mymax[1,dlon]))
writeLines(paste("at",lat.seq[dlat],lon.seq[1],"sum=",mytot[dlat,1],"max=",mymax[dlat,1]))
writeLines(paste("at",lat.seq[dlat],lon.seq[dlon],"sum=",mytot[dlat,dlon],"max=",mymax[dlat,dlon]))
save(mytot,file="mytotp.Rda")
save(mymax,file="mymaxp.Rda")
write.csv(mytot,file="mytotp.csv")
write.csv(mymax,file="mymaxp.csv")
```

Saving the data MPI version

```
if(myid == source){
   # put our values back in df
   mymax<-matrix(themax,nrow=dlat,ncol=dlon)
   mymax<-as.data.frame(mymax,row.names=lat.seq)
   colnames(mymax)<-lon.seq
   writeLines(" ")
   mytot<-matrix(thetot,nrow=dlat,ncol=dlon)
   mytot<-as.data.frame(mytot,row.names=lat.seq)
   colnames(mytot)<-lon.seq
   #sanity check
   writeLines(paste("at",lat.seq[1],lon.seq[1],"sum=",mytot[1,1],"max=",mymax[1,1]))
   writeLines(paste("at",lat.seq[1],lon.seq[dlon],"sum=",mytot[1,dlon],"max=",mymax[1,dlon]))
   writeLines(paste("at",lat.seq[dlat],lon.seq[1],"sum=",mytot[dlat,1],"max=",mymax[dlat,1]))
   writeLines(paste("at",lat.seq[dlat],lon.seq[dlon],"sum=",mytot[dlat,dlon],"max=",mymax[dlat,dlon]))
   save(mytot,file="mytotm.Rda")
   save(mymax,file="mymaxm.Rda")
   write.csv(mytot,file="mytotm.csv")
   write.csv(mymax,file="mymaxm.csv")
}
```

```
dat=subset(dat,,c(latitude,longitude,SCSN,depth))
mytot<-matrix(0,nrow=dlat,ncol=dlon)
mymax<-matrix(0,nrow=dlat,ncol=dlon)
for (row in 1:dorows) {
   slat<-dat[row,"latitude"]
   slon<-dat[row,"longitude"]
   mag<-dat[row,"SCSN"]
   dep<-dat[row,"depth"]
      for (i in 1:length(lat.seq)){
      mylat=lat.seq[i]
      for(j in 1:length(lon.seq)) {
         mylon<-lon.seq[j]
         #if(myid == 0)print(mylon)
         ouch<-whack(mylat,mylon,slat,slon,mag,dep)
         #print(c(ouch,mylat,mylon,slat,slon,mag,dep))
         if(ouch > mymax[i,j]){mymax[i,j]=ouch}
         mytot[i,j]<-mytot[i,j]+ouch
      }
   }
   if(myid == 0) {
      frow<-frow+1
      #if( (frow %% drow) == 0)print(tymer(frow))
   }
}
if(myid == 0)print(tymer(frow))
# pack our dataframe so it is like the original code
for(j in 1:length(lon.seq)) {
for (i in 1:length(lat.seq)){
   mylat=lat.seq[i]
      mylon<-lon.seq[j]
      df[nrow(df) + 1,]<-c(mylat,mylon,mytot[i,j],mymax[i,j])
   }
}
```

MPI - put earthquakes in the outer loop

Small test run on 6 x 8 grid

```
semikln:eq timk$ cat bounds
35.0 32.0 6           lat
-121.0 -115.0 8       lon

[1] -121.0000 -120.1429 -119.2857 -118.4286 -117.5714 -116.7143 -115.8571
[8] -115.0000
[1] 35.0 34.4 33.8 33.2 32.6 32.0
```

Full size 101 x 176 or
555 times bigger

```
semikln:eq timk$ wc start
  634252 16432661 98943312 start
semikln:eq timk$
```

Parallel Foreach

```
> source("works.R")
[1] -121.0000 -120.1429 -119.2857 -118.4286 -117.5714 -116.7143 -115.8571
-115.0000
[1] 35.0 34.4 33.8 33.2 32.6 32.0
Initialising 4 core cluster.
[1] "1564495753.3435  2019-07-30 08:09:13       0.000       0.000  "
[1] "1564495808.12748  2019-07-30 08:10:08      54.784      54.784  done 35"
[1] "1564495865.98115  2019-07-30 08:11:05      57.854     112.638  done 34.4"
[1] "1564495923.93894  2019-07-30 08:12:03      57.958     170.595  done 33.8"
[1] "1564495981.60674  2019-07-30 08:13:01      57.668     228.263  done 33.2"
[1] "1564496039.21519  2019-07-30 08:13:59      57.608     285.872  done 32.6"
[1] "1564496097.07329  2019-07-30 08:14:57      57.858     343.730  done 32"
. . .
. . .
at 35 -121 sum= 847.049686230019 max= 4.51561361845489
at 35 -115 sum= 96.7396093152043 max= 1.4646600693548
at 32 -121 sum= 16.6253217611744 max= 0.155219186870698
at 32 -115 sum= 11557.6639761953 max= 21.1651123970782
>
```

MPI version 1

```
semikln:eq timk$ mpiexec -n 4 Rscript stom.R
[1] "I am 2 of 4 on semikln"
[1] "I am 1 of 4 on semikln"
[1] "I am 0 of 4 on semikln"
[1] "I am 3 of 4 on semikln"
[1] "2 startac"
[1] "0 startaa"
[1] "3 startad"
[1] "1 startab"
[1] -121.0000 -120.1429 -119.2857 -118.4286 -117.5714 -116.7143 -115.8571
[8] -115.0000
[1] 35.0 34.4 33.8 33.2 32.6 32.0
[1] "1564496457.31774  2019-07-30 08:20:57       0.000       0.000  "
[1] "1564496492.34628  2019-07-30 08:21:32      35.029      35.029  done -121"
[1] "1564496526.00923  2019-07-30 08:22:06      33.663      68.691  done -120.142857142857"
[1] "1564496560.41559  2019-07-30 08:22:40      34.406     103.098  done -119.285714285714"
[1] "1564496594.37134  2019-07-30 08:23:14      33.956     137.054  done -118.428571428571"
[1] "1564496628.52846  2019-07-30 08:23:48      34.157     171.211  done -117.571428571429"
[1] "1564496662.30467  2019-07-30 08:24:22      33.776     204.987  done -116.714285714286"
[1] "1564496696.59566  2019-07-30 08:24:56      34.291     239.278  done -115.857142857143"
[1] "1564496730.67589  2019-07-30 08:25:30      34.080     273.358  done -115"

at 35 -121 sum= 847.049686230034 max= 4.51561361845489
at 35 -115 sum= 96.7396093152028 max= 1.4646600693548
at 32 -121 sum= 16.6253217611748 max= 0.155219186870698
at 32 -115 sum= 11557.6639761954 max= 21.1651123970782
```

```
mytot<-matrix(0,nrow=dlat,ncol=dlon)
mymax<-matrix(0,nrow=dlat,ncol=dlon)
for (row in 1:dorows) {
   slat<-dat[row,"latitude"]
   slon<-dat[row,"longitude"]
   mag<-dat[row,"SCSN"]
   dep<-dat[row,"depth"]
      for (i in 1:length(lat.seq)){
      mylat=lat.seq[i]
      for(j in 1:length(lon.seq)) {
         mylon<-lon.seq[j]
         #if(myid == 0)print(mylon)
         ouch<-whack(mylat,mylon,slat,slon,mag,dep)
         #print(c(ouch,mylat,mylon,slat,slon,mag,dep))
         if(ouch > mymax[i,j]){mymax[i,j]=ouch}
         mytot[i,j]<-mytot[i,j]+ouch
      }
   }
   if(myid == 0) {
      frow<-frow+1
      #if( (frow %% drow) == 0)print(tymer(frow))
   }
}
if(myid == 0)print(tymer(frow))
# pack our dataframe so it is like the original code
for(j in 1:length(lon.seq)) {
for (i in 1:length(lat.seq)){
   mylat=lat.seq[i]
      mylon<-lon.seq[j]
      df[nrow(df) + 1,]<-c(mylat,mylon,mytot[i,j],mymax[i,j])
   }
}
```

Reorder the loops so earthquakes are on the outer loop

MPI
earthquakes in outer loop

```
semikln:eq timk$ mpiexec -n 4 Rscript reorder.R
[1] "I am 1 of 4 on semikln"
[1] "I am 0 of 4 on semikln"
[1] "I am 2 of 4 on semikln"
[1] "I am 3 of 4 on semikln"
[1] "2 startac"
[1] "1 startab"
[1] "3 startad"
[1] "0 startaa"
[1] -121.0000 -120.1429 -119.2857 -118.4286 -117.5714 -116.7143 -115.8571
[8] -115.0000
[1] 35.0 34.4 33.8 33.2 32.6 32.0
[1] "1564496773.46741  2019-07-30 08:26:13       0.000       0.000  "
[1] "1564496802.23465  2019-07-30 08:26:42      28.767      28.767  158563"

at 35 -121 sum= 847.049686230034 max= 4.51561361845489
at 35 -115 sum= 96.7396093152028 max= 1.4646600693548
at 32 -121 sum= 16.6253217611748 max= 0.155219186870698
at 32 -115 sum= 11557.6639761954 max= 21.1651123970782
```

Fortran - 1 core

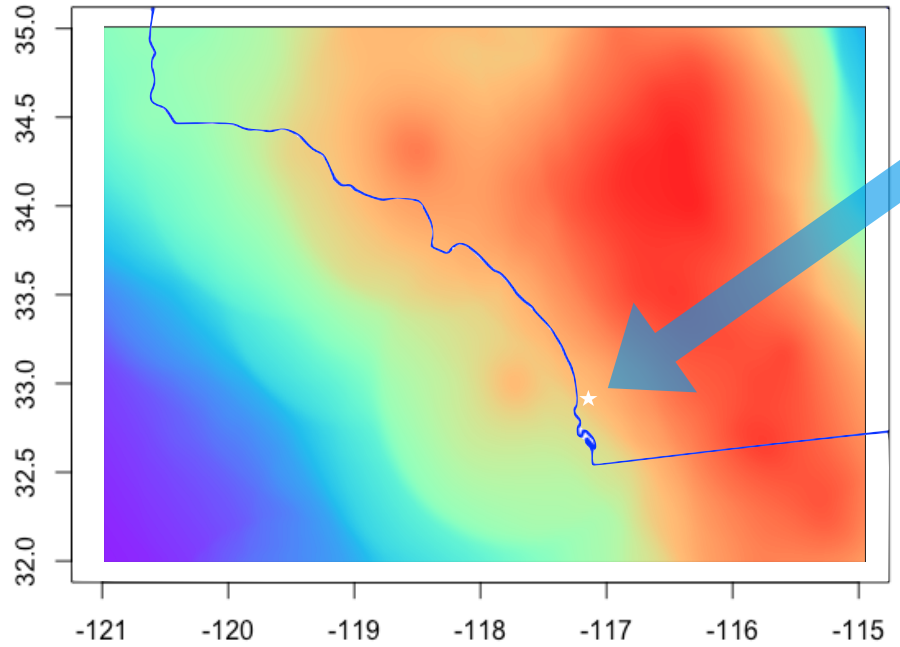
```
tymer fort ; ./a.out start ; tymer fort
1564496341.323570 Tue Jul 30 08:19:01 2019      0.000      0.000
   33.486100000000000       -116.45592000000001        5.1970000000000001       0.59999999999999998
 -121.0000 -120.1429 -119.2857 -118.4286 -117.5714
 -116.7143 -115.8571 -115.0000
   35.0000   34.4000   33.8000   33.2000   32.6000
   32.0000
   35.000000000000000       -121.00000000000000        847.04968623003913        4.5156136184548945
   35.000000000000000       -120.14285714285714        1214.2601087270914        3.8627982934066658
   35.000000000000000       -119.28571428571429        4557.6539045289892        8.6870071415220469
. . .
. . .
   32.000000000000000       -116.71428571428571        3133.5327480740407        6.3111246263069321
   32.000000000000000       -115.85714285714286        15590.168934321318        11.044663681545376
   32.000000000000000       -115.00000000000000        11557.663976195379        21.165112397078222
 at   35.000000000000000   -121.00000000000000  sum=   847.04968623003913  max=   4.5156136184548945
 at   35.000000000000000   -115.00000000000000  sum=   96.739609315251883  max=   1.4646600693548000
 at   32.000000000000000   -121.00000000000000  sum=   16.625321761186740  max=  0.15521918687069841
 at   32.000000000000000   -115.00000000000000  sum=   11557.663976195379  max=   21.165112397078222
1564496345.137718 Tue Jul 30 08:19:05 2019      3.814      3.814
semikln:eq timk$
```

Version	Source	Cores	Time (seconds)
Parallel Foreach	works.R	4	343
MPI	stom.R	4	273
MPI earthquakes in outer loop	reorder.R	4	29
Fortran	fungr.f90	1	3.3

Full Grid 101 x 176

Version	Source	Cores	Time (seconds)
MPI earthquakes in outer loop	quake03.R	24	1093
Fortran	fungr.f90	1	297

Want to buy my house?



@32.93275,-117.0904296

A bit uncouth

Untitled5 5/6/21, 9:39 AM

In [16]:
```
source("mysys.R")
```

In [17]:
```
srun("-n 8 python3 -c '#!/usr/bin/env python3

# Initialize MPI import numpy
from numpy import *
from mpi4py import MPI

# Print id, name, np
comm=MPI.COMM_WORLD
myid=comm.Get_rank()
numprocs=comm.Get_size()
name = MPI.Get_processor_name()
print(myid,name,numprocs)
MPI.Finalize()'
")
```

Run a parallel Python program from within a R Jupyter Notebook

```
0 blk 8
6 clr 8
1 blk 8
7 clr 8
5 clr 8
4 clr 8
2 blk 8
3 blk 8
```

In [ ]:

Works in R also

```
[56]: srun("-n 4 R -q --no-save -e '
      library(\"Rmpi\")
      myid <- mpi.comm.rank(comm=0)
      sprintf(\"myid= %d on %s\",myid,mpi.get.processor.name())
      mpi.finalize()'
      ")
```

```
>
> library("Rmpi")
>
> library("Rmpi")
>
> library("Rmpi")
>
> library("Rmpi")
> myid <- mpi.comm.rank(comm=0)
> myid <- mpi.comm.rank(comm=0)
> myid <- mpi.comm.rank(comm=0)
> myid <- mpi.comm.rank(comm=0)
> sprintf("myid= %d on %s",myid,mpi.get.processor.name())
> sprintf("myid= %d on %s",myid,mpi.get.processor.name())
> [1] "myid= 0 on clr"
> mpi.finalize()
[1] "myid= 1 on clr"
> mpi.finalize()
sprintf("myid= %d on %s",myid,mpi.get.processor.name())
> sprintf("myid= %d on %s",myid,mpi.get.processor.name())
[1] "myid= 3 on clr"
> mpi.finalize()
[1] "myid= 2 on clr"
> mpi.finalize()
```

Exercise for the Student

* Reorder the loops so the outer loop is over earthquakes
* About 12x faster
* Might want to split file and loop over files names

Text to Binary

```
program fung
     integer, parameter:: b8 = selected_real_kind(14) ! basic real types
     integer year,month,day,hour,minute
     real(b8) second
     integer cuspid
     real(b8) latitude,longitude,depth,SCSN
     integer PandS,statino
     real(b8) residual
     integer tod,method, ec,nen,dt
     real(b8) stdpos,stddepth,stdhorrel,stddeprel
     character(len=10)therest
     character(len=2)le,ct
     character(len=5)poly
     character(len=32)fname,fbase
     CALL get_command_argument(1,fname)
     open(11,file=fname,status="old")
     OPEN(UNIT=12, FILE="reals"//trim(fname), STATUS="NEW", ACCESS="STREAM")
     OPEN(UNIT=13, FILE="ints"//trim(fname), STATUS="NEW", ACCESS="STREAM")
     OPEN(UNIT=14, FILE="chars"//trim(fname), STATUS="NEW", ACCESS="STREAM")
     OPEN(UNIT=15, FILE="ascii"//trim(fname), STATUS="NEW")
```

```
      do
      read(11,1,end=2)year,month,day,hour,minute ,&
           second, &
           cuspid, &
           latitude,longitude,depth,SCSN, &
           PandS,statino, &
           residual, &
           tod,method, ec,nen,dt, &
           stdpos,stddepth,stdhorrel,stddeprel, &
           le,ct,poly
!          if(depth .lt. 0.0 .or. SCSN .lt. 0.0)then
      write(*,1)year,month,day,hour,minute ,&
           second, &
           cuspid, &
           latitude,longitude,depth,SCSN, &
           PandS,statino, &
           residual, &
           tod,method, ec,nen,dt, &
           stdpos,stddepth,stdhorrel,stddeprel, &
           le,ct,poly
!          endif
      ! 10 8 byte reals
      write(12)second,latitude,longitude,depth,SCSN, &
           residual,stdpos,stddepth,stdhorrel,stddeprel
      ! 13 4 byte integers
      write(13)year,month,day,hour,minute,cuspid, &
           PandS,statino,tod,method,ec,nen,dt
      write(14)le,ct,poly
      write(15,"(a2,',',a2,',',a5)")le,ct,poly

 1       format(i4,4(1x,i2.2),1x,&
         f6.3,1x,i9,1x,f8.5,1x,f10.5,1x,f7.3,1x,f5.2, &
         1x,i3,1x,i3,1x,f6.2,1x,i3,1x,i3,1x,i5,1x,i5, &
         1x,i7,4(1x,f7.3),2x,a2,1x,a2,1x,a5)
         enddo
 2 continue
end program
```

```
hexdump -v -e '"%f %f %f %f %f %f %f %f %f %f\n"' reals | head
hexdump -v -e '"%c%c %c%c %c%c%c%c%c\n"' chars | head
```

Download data and split it

```
#!/usr/bin/env python
# Program to download earthquake data and to
# split it in to "tasks" more or less equal sized
# files.  "tasks" is taken from the command line
# or defaults to 4.  Our data set should have
# 634252 lines but we check it anyway.
import requests
import os
import sys
# Here's our data
url = ' http://scedc.caltech.edu/ftp/catalogs/hauksson/Socal_DD/hs_1981_all_comb_K4_A.cat_so_SCSN_v01'
exists = os.path.isfile('start')
if exists:
    print("file exixts")
else:
    print("downloading file")
    r = requests.get(url, allow_redirects=True)
    open('start', 'wb').write(r.content)

# get # of tasks
tasks=4
if(len(sys.argv) > 1):
    tasks=int(sys.argv[1])

# count the lines in the file
command="wc -l start"
f=os.popen(command,"r")
out=f.readlines()
out=out[0].split()
lines=int(out[0])
# should be 634252
print(lines == 634252)

# Get a good number of line for each file.
# The last file might be a bit short.
each=lines//tasks
tot=each*tasks
while(tot < lines):
    lines=lines+1
    each=lines/tasks
    tot=each*tasks

# Split the file.
command="split -l X start start"
each=str(each)
command=command.replace("X",each)
print(command)
f=os.popen(command,"r")
out=f.readlines()
```

Forcing tasks to Cores

```
. . .
/* TO COMPILE: R CMD SHLIB mapit.c */
int sched_getcpu();

void findcore (int *ic)
{
   ic[0] = sched_getcpu();
}
void forcecore (int *core) {
     cpu_set_t set;
     pid_t getpid(void);
     CPU_ZERO(&set);        // clear cpu mask
         int bonk;
         bonk=*core;
     CPU_SET(bonk, &set);      // set cpu 0
     sched_setaffinity(getpid(), sizeof(cpu_set_t), &set);
}

void p_to_c (int * pid ,int *core) {
     cpu_set_t set;
     pid_t apid;
     apid=(pid_t)pid;
     CPU_ZERO(&set);        // clear cpu mask
         int bonk;
         bonk=*core;
     CPU_SET(bonk, &set);      // set cpu 0
     sched_setaffinity(getpid(), sizeof(cpu_set_t), &set);
}
```

Forcing tasks to Cores

```
dyn.load("mapit.so")
threadfunc <-function() {
    trd=c(0)
    out <- .C("findcore",ic=as.integer(trd),package="thread")
    return(out$ic)
}
forceit <-function(core) {
    trd=c(core)
    out <- .C("forcecore",ic=as.integer(trd),package="core")
    return(out$ic)
}
```

```
r <- foreach(ijk=1:nt, .combine=cbind) %dopar% {
    dyn.load("mapit.so")
     dia <- 0.1
     docore <- 1+((ijk-1) %% 23)
     forceit(docore)
. . .
}
```

Go back

Monitoring tasks

```
#!/home/timk@colostate.edu/projects/intel/intelpython3/bin/python3
import os
import sys
who=sys.argv[1]
command="squeue -t Running -o%N -h -u " +who
p=os.popen(command,"r")
lines=p.read()
lines=lines.split()
mylist=[]
for l in lines:
   if l.find("[") > 0:
      head,foot=l.split("[")
      foot=foot.replace("]","")
      #print(head,foot)
      foot=foot.split(",")
      for f in foot:
         if f.find("-") < 0:
            node=head+f
            mylist.append(node)
         else:
            ran=f
            ran=ran.split("-")
            ran=[int(ran[0]),int(ran[1])+1]
            ran=range(ran[0],ran[1])
            for r in ran:
               r=str("%4.4d" % (r))
               node=head+r
               mylist.append(node)
   else:
      mylist.append(l.strip())
```

Get list of nodes on which you are running

Run "ps" on all your nodes

```
bc="ps -u $USER -mo pid,tid,%cpu,%mem,psr,comm"
for m in mylist:
   print(m)
   command="ssh "+m+" "+bc
   p=os.popen(command,"r")
   lines=p.read()
   print(lines)
```

process ID	% mem
thread ID	Core
% CPU	command

Skipped Slides

Shows various ways "parallel" can be broken

Import everything we need

```
# (0) #########
rm(list=ls())
library(parallel)
library(dplyr)
#library(multidplyr)
library(doParallel)


print("Use 4 cores for our parallel machine")
cores=4
```

```
> # (0) #########
> rm(list=ls())
> library(parallel)
> library(dplyr)
> #library(multidplyr)
> library(doParallel)
> print("Use 4 cores for our parallel machine")
[1] "Use 4 cores for our parallel machine"
> cores=4
>
```

```r
# (1) #########

print("Do a very simple loop, not in parallel")

nt<-10
asum=0
for(ijk in 1:nt ){
  asum<-asum+ijk
}
print(asum)
#readline(prompt = "NEXT>")
```

```
> source("01.R")
[1] "Do a very simple loop, not in parallel"
[1] 55
>
```

```r
# (2) #########

print("Do a very simple parallel loop")
##
ct <- makeCluster(cores)
registerDoParallel(ct)
##

nt<-10
asum=0
foreach(ijk=1:nt ) %dopar% {
  asum<-asum+ijk
}
print(asum)

print("What happened?")

stopCluster(ct)
#readline(prompt = "NEXT>")
```

This example is invalid.
Any idea why?

```
> source("02.R")
[1] "Do a very simple parallel loop"
Initialising 4 core cluster.
. . .
[1] 0
[1] "What happened?"
>
```

You need to assign "foreach" to a variable

```r
bsum<-foreach(...
```

The variable gets assigned the last values of the loop.
Think of "foreach" as a function that returns a value.

Next two examples are actually broken also .

Read:

* Be careful
* Don't do this

```
# (3) #########  (broken)

print("Same as above but explicitly 'return' a value")
##
ct <- makeCluster(cores)
registerDoParallel(ct)
##

nt<-10
asum=0
bsum<-foreach(ijk=1:nt ) %dopar% {
  asum<-asum+ijk
}
print(c("asum",asum))
print(c("bsum",bsum))


stopCluster(ct)
#readline(prompt = "NEXT>")
```

```
> source("03.R")
[1] "Same as above but explicitly 'return' a value"
[1] "asum" "0"
[[1]]
[1] "bsum"

[[2]]
[1] 1

[[3]]
[1] 2

[[4]]
[1] 3

[[5]]
[1] 4

[[6]]
[1] 6

[[7]]
[1] 8

[[8]]
[1] 10

[[9]]
[1] 14

[[10]]
[1] 17

[[11]]
[1] 20
```

We fixed the problem of foreach not returning a value.

HOWEVER

If you run this again you might get different results

```
# (4) #########  (broken)

print("Same as above but explicitly 'return' a value as rows")
##
ct <- makeCluster(cores)
registerDoParallel(ct)
##

nt<-10
asum=0
bsum<-foreach(ijk=1:nt , .combine=rbind ) %dopar% {
  asum<-asum+ijk
}
print(asum)
print(bsum)

stopCluster(ct)
#readline(prompt = "NEXT>")
```

This example is (still) invalid.
Any idea why?

```
[1] "Same as above but explicitly 'return' a value as rows"
Initialising 4 co
[1] 0
           [,1]
result.1      1
result.2      2
result.3      3
result.4      4
result.5      6
result.6      8
result.7     10
result.8     14
result.9     17
result.10    20
>
```

```
Another Run
[1] 0
           [,1]
result.1      1
result.2      2
result.3      3
result.4      4
result.5      6
result.6      8
result.7     10
result.8     14
result.9     13
result.10    18
>
```

Where's The Food?

At Least 2 Problems

* "Stuff" input to a foreach (asum in this case) can be promoted to a vector
* Calculations will occur on different processors - addition is no longer commutative

https://www.rdocumentation.org/packages/foreach/versions/1.4.4/topics/foreach

Run 10 times get different results

```
# (5) #########

print("Similar to last run")
##
ct <- makeCluster(cores)
registerDoParallel(ct)
##
nt<-10
asum<-0
bsum<-foreach(ijk=1:nt , .combine=cbind ) %dopar% {
  asum<-asum+ijk
}
print(sum(bsum))

stopCluster(ct)
#readline(prompt = "NEXT>")
```

```
for (i in 1:10){source("05.R")}
[1] "Similar to last run"
Initialising 4 core cluster.
[1] 79
[1] "Similar to last run"
Initialising 4 core cluster.
[1] 85
[1] "Similar to last run"
Initialising 4 core cluster.
[1] 85
[1] "Similar to last run"
Initialising 4 core cluster.
[1] 79
[1] "Similar to last run"
Initialising 4 core cluster.
[1] 85
[1] "Similar to last run"
Initialising 4 core cluster.
[1] 79
[1] "Similar to last run"
Initialising 4 core cluster.
[1] 85
[1] "Similar to last run"
Initialising 4 core cluster.
[1] 85
[1] "Similar to last run"
Initialising 4 core cluster.
[1] 85
>
```

```
# (6) #########

print("Add process ID to the output")
##
ct <- makeCluster(cores)
registerDoParallel(ct)
##
if(!exists("doprint")){doprint=FALSE}
nt<-10
asum<-0
bsum<-foreach(ijk=1:nt , .combine=rbind ) %dopar% {
   asum<-asum+ijk
   pid<-Sys.getpid()
   c(pid,asum)
}
if(doprint){
   #print(bsum)
   print(bsum[order(bsum[,1]),])
}
print(sum(bsum[,2]))

stopCluster(ct)
#readline(prompt = "NEXT>")
```

This example is still broken

Here we attach the Process ID to the output.

With doprint= FALSE we get the same output as before

```
> doprint=TRUE
```

Same code but this time we print the output arrays which show the process ID

```
          [,1] [,2]
result.1  5814    1
result.5  5814    6
result.8  5814   14
result.2  5828    2
result.6  5828    8
result.9  5828   17
result.3  5842    3
result.7  5842   10
result.10 5842   20
result.4  5856    4
[1] 85
```

```
          [,1] [,2]
result.1  5874    1
result.5  5874    6
result.9  5874   15
result.2  5888    2
result.6  5888    8
result.10 5888   18
result.3  5902    3
result.7  5902   10
result.4  5916    4
result.8  5916   12
[1] 79
```

Here we don't restart between calls

```
          [,1] [,2]
result.1  6294    1
result.5  6294    6
result.8  6294   14
result.2  6308    2
result.6  6308    8
result.9  6308   17
result.3  6322    3
result.7  6322   10
result.10 6322   20
result.4  6336    4
[1] 85
```

```
          [,1] [,2]
result.1  6294    1
result.5  6294    6
result.7  6294   13
result.10 6294   23
result.2  6308    2
result.6  6308    8
result.9  6308   17
result.3  6322    3
result.8  6322   11
result.4  6336    4
[1] 88
```

Example 1

Index	Return	Value	Path	Pid
1	result.1	1	“0+1”	**89981**
2	result.2	2	“0+2”	**89995**
3	result.3	3	“0+3"	**90009**
4	result.4	4	“0+4"	**90023**
5	result.5	6	“1+5”	**89981**
6	result.6	8	“2+6”	**89995**
7	result.7	10	“3+7”	**90009**
8	result.8	14	“6+8”	**89981**
9	result.9	13	“4+9”	**90023**
10	result.10	18	“8+10”	**89995**
Total		79		

Example 2

Index	Return	Value	Path	Pid
1	result.1	1	“0+1”	**93237**
2	result.2	2	“0+2”	**93260**
3	result.3	3	“0+3"	**93276**
4	result.4	4	“0+4"	**93296**
5	result.5	6	“1+5”	**93237**
6	result.6	8	“2+6”	**93260**
7	result.7	13	“6+7”	**93237**
8	result.8	11	“3+8”	**93276**
9	result.9	17	“8+9”	**93260**
10	result.10	23	“13+10”	**93237**
Total		88		

```
# (8) #########

print("Same as above but explicitly 'return' a value as columns")

##
ct <- makeCluster(cores)
registerDoParallel(ct)
##

nt<-10
mymat<-matrix(nrow=nt,ncol=nt)
mymat[1:nt,]<-0
bsum=0
bsum<-foreach(ijk = 1:nt, .combine=cbind ) %dopar% {
   set.seed(ijk)
   mymat[,ijk]<-rnorm(mymat[,ijk])
   junk<-sum(mymat[,ijk])
}
print(bsum)
print(mymat)

stopCluster(ct)
#readline(prompt = "NEXT>")
```

Here we "fill" a matrix and sum it

```
    result.1 result.2   result.3 result.4   result.5 result.6 result.7  result.8  result.9 result.10
[1,] 1.322028 2.111516 -0.6713568 5.665293 -0.7885155 1.053582 1.039757 -4.493192 -2.190185 -4.906568
> print(mymat)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    0    0    0    0    0    0    0    0    0     0
 [2,]    0    0    0    0    0    0    0    0    0     0
 [3,]    0    0    0    0    0    0    0    0    0     0
 [4,]    0    0    0    0    0    0    0    0    0     0
 [5,]    0    0    0    0    0    0    0    0    0     0
 [6,]    0    0    0    0    0    0    0    0    0     0
 [7,]    0    0    0    0    0    0    0    0    0     0
 [8,]    0    0    0    0    0    0    0    0    0     0
 [9,]    0    0    0    0    0    0    0    0    0     0
[10,]    0    0    0    0    0    0    0    0    0     0
```

```
# (9) #########

writeLines("Now for some examples that actually work")
writeLines("but maybe not the way you expected.")
writeLines("The export may generate a warning.")
##
ct <- makeCluster(cores)
registerDoParallel(ct)
##
if(!exists("doverbose")){doverbose=FALSE}
nt<-10
asum<-0
writeLines("\nasum is zero")
bsum<-foreach(ijk=1:nt , .combine=cbind, .export=(c('asum'))) %dopar% {
   asum<-2**ijk
   asum
}
print(asum)
print(bsum)

nt<-10
asum<-vector(mode="double",length=nt)
writeLines("\nasum is vector of zero")

bsum<-foreach(ijk=1:nt , .combine=cbind,.export=(c('asum')) ) %dopar% {
   asum<-2**ijk
   asum
}
print(asum)
print(bsum)

asum=vector(mode="double",length=nt)+2000
writeLines("\nasum is vector of 2000")
bsum<-foreach(ijk=1:nt , .combine=cbind,.export=(c('asum')) ) %dopar% {
   asum<-2**ijk
   asum
}
print(asum)
print(bsum)

stopCluster(ct)
#readline(prompt = "NEXT>")
```

```
. . .
. . .
asum=vector(mode="double",length=nt)+2000
writeLines("\nasum is vector of 2000")
bsum<-foreach(ijk=1:nt , .combine=cbind,.export=(c('asum')) ) %dopar% {
   asum<-2**ijk
   asum
}
. . .
. . .
```

```
> source("09.r")
Now for some examples that actually work
but maybe not the way you expected.
The export may generate a warning.
Initialising 4 core cluster.

asum is zero
[1] 0
     result.1 result.2 result.3 result.4 result.5 result.6 result.7 result.8 result.9 result.10
[1,]        2        4        8       16       32       64      128      256      512      1024

asum is vector of zero
 [1] 0 0 0 0 0 0 0 0 0 0
     result.1 result.2 result.3 result.4 result.5 result.6 result.7 result.8 result.9 result.10
[1,]        2        4        8       16       32       64      128      256      512      1024

asum is vector of 2000
 [1] 2000 2000 2000 2000 2000 2000 2000 2000 2000 2000
     result.1 result.2 result.3 result.4 result.5 result.6 result.7 result.8 result.9 result.10
[1,]        2        4        8       16       32       64      128      256      512      1024
```

Point is: since asum is not on the left hand side it is essentially ignored

```
# (10) #########

writeLines("We move asum to the right hand size")
##
ct <- makeCluster(cores)
registerDoParallel(ct)

asum<-5
writeLines("\nasum is  scalar on the right hand side")

bsum<-foreach(ijk=1:nt , .combine=rbind ,.export=(c('asum')),.verbose=doverbose) %dopar% {
   xsum<-asum*ijk

}
print(asum)
print(bsum)
```

```
          [,1]
result.1     5
result.2    10
result.3    15
result.4    20
result.5    25
result.6    30
result.7    35
result.8    40
result.9    45
result.10   50
```

Finally we have a "working" code.

```
asum=vector(mode="double",length=5)
writeLines("\nasum is vector of on length 5 on the right hand side")

for (ijk in 1:length(asum))asum[ijk]<-asum[ijk]+ijk

bsum<-foreach(ijk=1:nt , .combine=rbind ,.export=(c('asum')),.verbose=doverbose) %dopar% {
   xsum<-asum*ijk
}
print(asum)
print(bsum)
```

```
> print(asum)
[1] 1 2 3 4 5
> print(bsum)
          [,1] [,2] [,3] [,4] [,5]
result.1     1    2    3    4    5
result.2     2    4    6    8   10
result.3     3    6    9   12   15
result.4     4    8   12   16   20
result.5     5   10   15   20   25
result.6     6   12   18   24   30
result.7     7   14   21   28   35
result.8     8   16   24   32   40
result.9     9   18   27   36   45
result.10   10   20   30   40   50
```

```
asum=vector(mode="double",length=15)
writeLines("\nasum is vector of on length 15 on the right hand side")
writeLines("We add the pid for each task/iteration to the output")
for (ijk in 1:length(asum))asum[ijk]<-asum[ijk]+ijk
bsum<-foreach(ijk=1:nt , .combine=rbind,.verbose=doverbose) %dopar% {
   xsum<-asum+ijk*1000
   pid<-Sys.getpid()
   c(pid,xsum)
}
print(asum)
print(bsum)
print(bsum[order(bsum[,1]),])
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
> print(bsum)
          [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16]
result.1  7201  1001  1002  1003  1004  1005  1006  1007  1008  1009  1010  1011  1012  1013  1014  1015
result.2  7215  2001  2002  2003  2004  2005  2006  2007  2008  2009  2010  2011  2012  2013  2014  2015
result.3  7229  3001  3002  3003  3004  3005  3006  3007  3008  3009  3010  3011  3012  3013  3014  3015
result.4  7243  4001  4002  4003  4004  4005  4006  4007  4008  4009  4010  4011  4012  4013  4014  4015
result.5  7201  5001  5002  5003  5004  5005  5006  5007  5008  5009  5010  5011  5012  5013  5014  5015
result.6  7215  6001  6002  6003  6004  6005  6006  6007  6008  6009  6010  6011  6012  6013  6014  6015
result.7  7229  7001  7002  7003  7004  7005  7006  7007  7008  7009  7010  7011  7012  7013  7014  7015
result.8  7201  8001  8002  8003  8004  8005  8006  8007  8008  8009  8010  8011  8012  8013  8014  8015
result.9  7243  9001  9002  9003  9004  9005  9006  9007  9008  9009  9010  9011  9012  9013  9014  9015
result.10 7215 10001 10002 10003 10004 10005 10006 10007 10008 10009 10010 10011 10012 10013 10014 10015
> print(bsum[order(bsum[,1]),])
          [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16]
result.1  7201  1001  1002  1003  1004  1005  1006  1007  1008  1009  1010  1011  1012  1013  1014  1015
result.5  7201  5001  5002  5003  5004  5005  5006  5007  5008  5009  5010  5011  5012  5013  5014  5015
result.8  7201  8001  8002  8003  8004  8005  8006  8007  8008  8009  8010  8011  8012  8013  8014  8015
result.2  7215  2001  2002  2003  2004  2005  2006  2007  2008  2009  2010  2011  2012  2013  2014  2015
result.6  7215  6001  6002  6003  6004  6005  6006  6007  6008  6009  6010  6011  6012  6013  6014  6015
result.10 7215 10001 10002 10003 10004 10005 10006 10007 10008 10009 10010 10011 10012 10013 10014 10015
result.3  7229  3001  3002  3003  3004  3005  3006  3007  3008  3009  3010  3011  3012  3013  3014  3015
result.7  7229  7001  7002  7003  7004  7005  7006  7007  7008  7009  7010  7011  7012  7013  7014  7015
result.4  7243  4001  4002  4003  4004  4005  4006  4007  4008  4009  4010  4011  4012  4013  4014  4015
result.9  7243  9001  9002  9003  9004  9005  9006  9007  9008  9009  9010  9011  9012  9013  9014  9015
```

```
# (12) #########
print("Work on a matrix")
##
ct <- makeCluster(cores)
registerDoParallel(ct)
##
if(!exists("mymean")){mymean=0}
nt<-8
mymat<-matrix(nrow=nt,ncol=nt)
mymat[1:nt,]<-mymean
csum<-foreach(ijk = 1:nt, .combine=cbind ) %dopar% {
   set.seed(ijk)
# note: the length for rnorm is ignored but needed or you get an error
   junk<-rnorm(nt,mean=mymat[,ijk])
}
print(csum)
bonk=vector(mode="double",length=nt)
for (i in 1:nt){bonk[i]=sum(csum[,i])}
print(bonk)
print(sum(mymat))
print("BIG QUESTION:  WHY IS THIS DIFFERENT FROM OPENMP?")
```

```
     result.1    result.2    result.3   result.4    result.5    result.6   result.7    result.8
[1,] -0.6264538 -0.89691455 -0.96193342  0.2167549 -0.84085548  0.26960598  2.2872472 -0.08458607
[2,]  0.1836433  0.18484918 -0.29252572 -0.5424926  1.38435934 -0.62998541 -1.1967717  0.84040013
[3,] -0.8356286  1.58784533  0.25878822  0.8911446 -1.25549186  0.86865983 -0.6942925 -0.46348277
[4,]  1.5952808 -1.13037567 -1.15213189  0.5959806  0.07014277  1.72719552 -0.4122930 -0.55083500
[5,]  0.3295078 -0.08025176  0.19578283  1.6356180  1.71144087  0.02418764 -0.9706733  0.73604043
[6,] -0.8204684  0.13242028  0.03012394  0.6892754 -0.60290798  0.36802518 -0.9472799 -0.10788140
[7,]  0.4874291  0.70795473  0.08541773 -1.2812466 -0.47216639 -1.30920430  0.7481393 -0.17028915
[8,]  0.7383247 -0.23969802  1.11661021 -0.2131445 -0.63537131  0.73862193 -0.1169552 -1.08833171
[1]  1.0516348  0.2658295 -0.7198681  1.9918898 -0.6408500  2.0571064 -1.3028792 -0.8889656
[1] 0
[1] "BIG QUESTION:  WHY IS THIS DIFFERENT FROM OPENMP?"
```

Some Points From the skipped slides

myout<-foreach

"foreach" is essentially a function it needs someplace for output to go. It takes on the last value in the loop. I have not been able to get ".final" to work.

You should not have "input" on the left had side.

Don't do this "asum<-asum+ijk"

Inputs are promoted to vectors (copied).

Lots of work per iteration is good. (Long function call.)

Calculations occur on different processors or at lease with different pids - Additions are not commutative