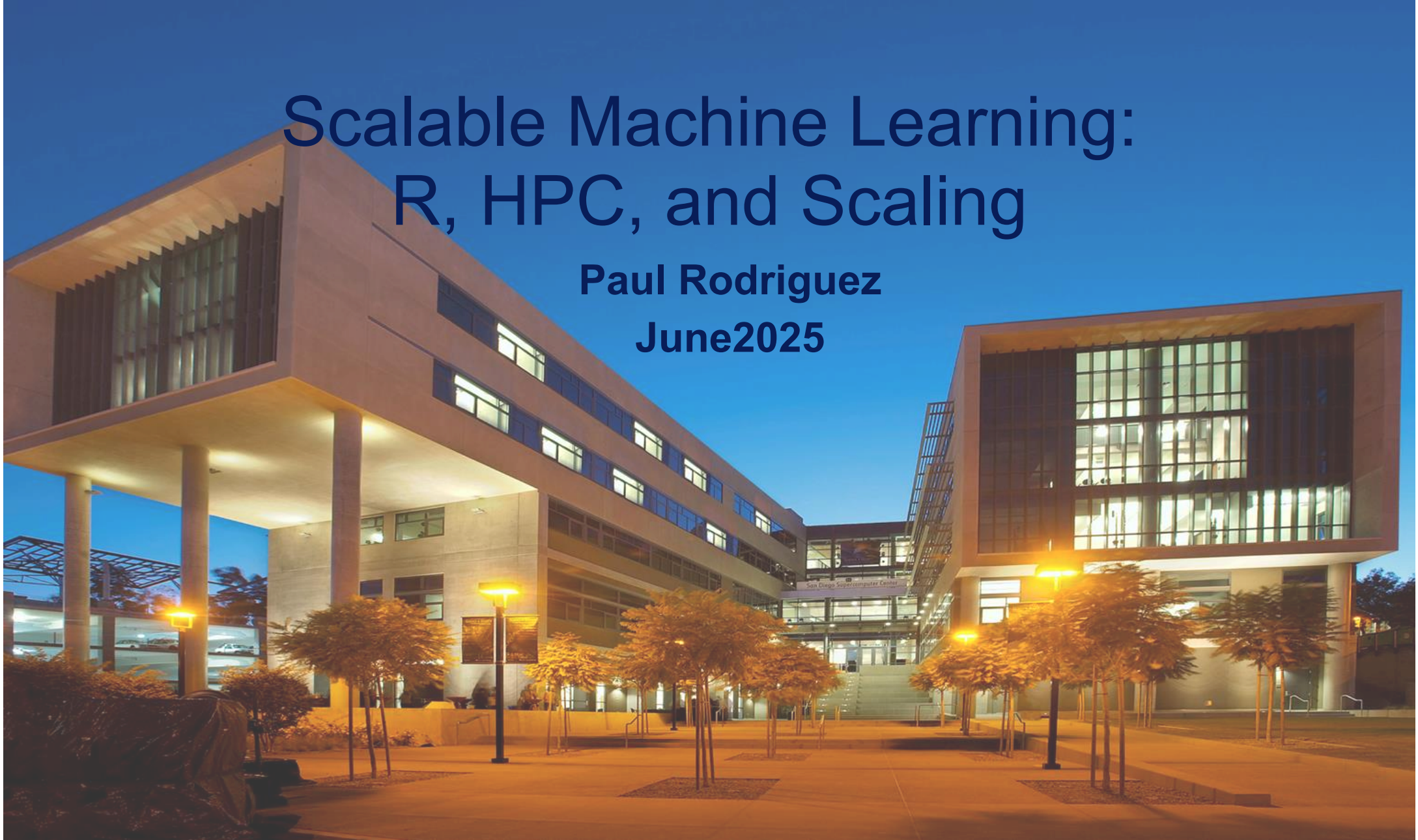


Scalable Machine Learning: R, HPC, and Scaling

Paul Rodriguez
June 2025



Outline

- **R on Expanse**
- **R and Scaling**
- **Parallel R**
- **Embarrassingly Parallel R**

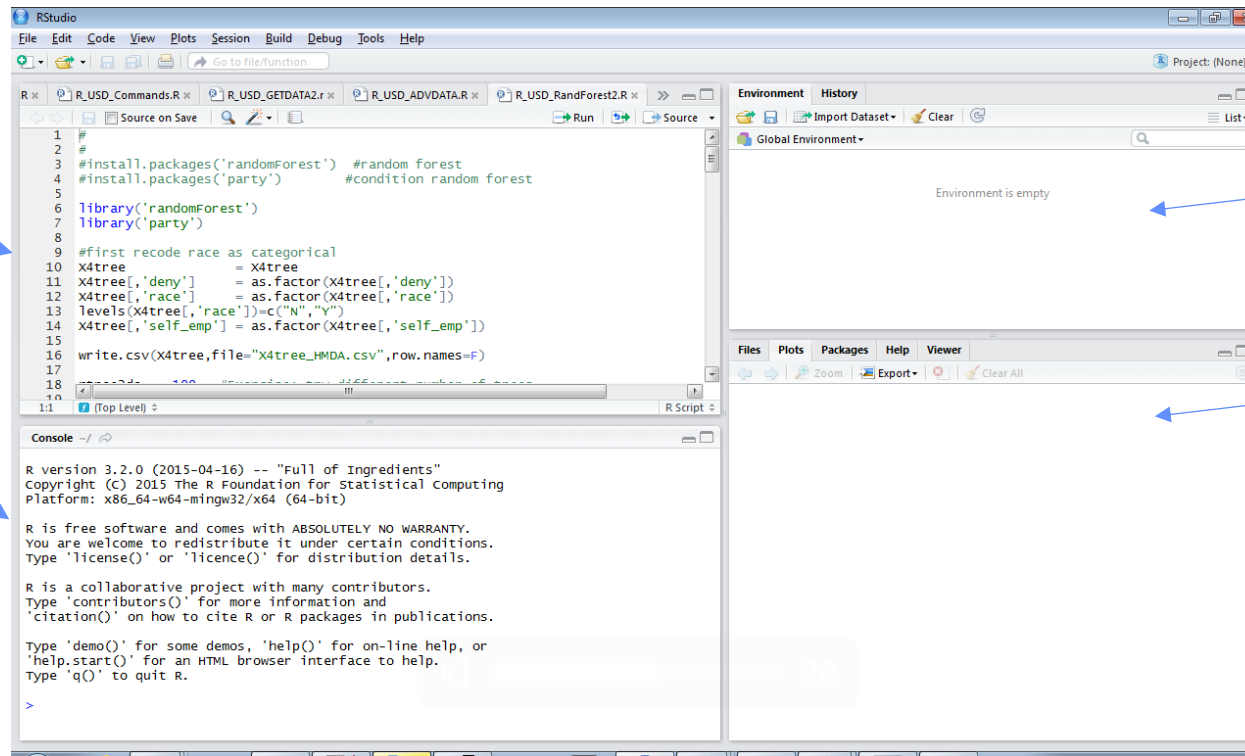
3 ways to run R on Expanse

1. Run a SLURM batch script of R commands
 2. Acquire a node with the “srun” command and use terminal window
 3. Use Expanse portal to start R-studio
(see user guide for details)
- (4. Also you can run R in a Jupyter notebook)

Rstudio: an integrated development environment

*Edit window to
Build scripts*

R console



*Environment
Information on
variables and
command
history*

*Plots, help
docs,
package
lists*

R interactively on Expanse command line

1. Get an interactive compute node (see demo slides):

2. Try

`$ module spider r` *(this tells you what modules you need)*

3. Enter

`$ module load cpu/0.15.4`

`$ module load gcc/9.2.0`

`$ module load r/4.0.2-openblas`

`$ R`

R version 4.0.2 (2020-06-22) -- "Taking Off Again"

Copyright (C) 2020 The R Foundation for Statistical Computing

Platform: x86_64-pc-linux-gnu (64-bit)

.....

Type 'q()' to quit R.

>

```
[p4rodrig@login02 ~]$ module spider r
-----
r: r/4.0.2-openblas
-----

Other possible modules matches:
  AMDuProf, amber, aria2, arm-forge, berkeley-db, bism

You will need to load all module(s) on any one of the li
"r/4.0.2-openblas" module is available to load.

  cpu/0.15.4  gcc/9.2.0

Help:
```

R strengths for HPC (IMHO)

- **Data Wrangling -**
- **Particular statistical procedure implementations -**
 - Imputation methods (for missing data)
 - Sampling methods
 - Instrument Variable (2 stage) Regression
 - Matching subjects for pairwise analysis
 - Generalized Linear Model (e.g. logistic regression)
 - MCMC routines (but Stan is likely better package)
 - Some ML models (e.g. randomForest, LASSO)

R strengths for HPC (IMHO)

- **Data Wrangling –**

- **Particular statistical procedure implementations -**

Imputation methods (for missing data)

Sampling methods

Instrument Variable (2 stage) Regression

Matching subjects for pairwise analysis

Generalized Linear Model (e.g. logistic regression)

MCMC routines (but Stan is likely better package)

Some ML models (e.g. randomForest, LASSO)

*Inferential Stats,
Experimental Studies*

*Biostats/BioInformatics
Data Science,
Machine Learning*

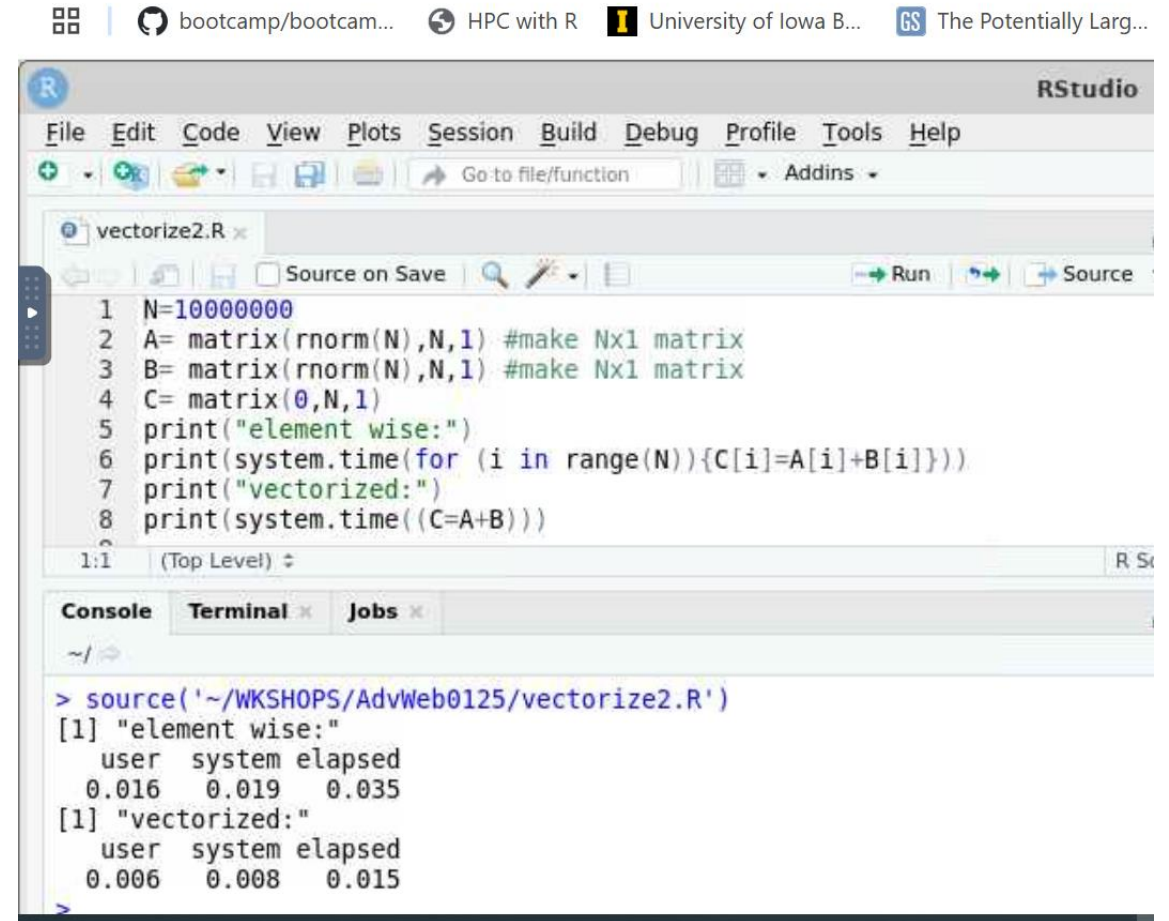
R Scaling In a nutshell

- R uses BLAS/LAPACK math libraries for operations on vectors
[Same for Matlab and Python]
- R packages provide multicore, out-of-core, multinode, or distributed data (SparkR) options
[Same for Matlab and Python]
- Some ML model implementations may be built to use parallel backends (review the available options)

Optimizing R code

- In general:
uses vector operations
instead of for loops on
individual vector elements

(eg add 2 vectors directly,
instead of each element 1...N)



The screenshot shows the RStudio interface. The top toolbar includes icons for file operations, running code, and debugging. The menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, and Help. The source editor shows a file named 'vectorize2.R' with the following R code:

```
1 N=10000000
2 A= matrix(rnorm(N),N,1) #make Nx1 matrix
3 B= matrix(rnorm(N),N,1) #make Nx1 matrix
4 C= matrix(0,N,1)
5 print("element wise:")
6 print(system.time(for (i in range(N)){C[i]=A[i]+B[i]}))
7 print("vectorized:")
8 print(system.time((C=A+B)))
```

The console output shows the execution of the code:

```
> source('~/.WKSHP0PS/AdvWeb0125/vectorize2.R')
[1] "element wise:"
   user system elapsed
0.016  0.019  0.035
[1] "vectorized:"
   user system elapsed
0.006  0.008  0.015
```

R multicore processing


- ‘doParallel’ package – provides the back end to the ‘for each’ parallel processing command
- uses threads across CPU cores to pass data & commands
- It also works for multinode (runs on top of RMPI)

See <https://cran.r-project.org/web/packages/doParallel/vignettes/gettingstartedParallel.pdf>

R multicore coding

```
install.packages(doParallel)  
library(doParallel)  
registerDoParallel(cores=24)
```


1. allocate workers



R multicore coding

```
install.packages(doParallel)
library(doParallel)
registerDoParallel(cores=24)
```

1. allocate workers



```
my_data_frame = .....
```

2. Make 'foreach' loop

R multicore coding

```
install.packages(doParallel)  
library(doParallel)  
registerDoParallel(cores=24)
```

1. allocate workers



```
my_data_frame = ..... 2. Make 'foreach' loop
```

```
my_results = foreach(i=1:24,.combine=rbind)
```

**3. specify how to
combine results**



R multicore coding

```
install.packages(doParallel)
library(doParallel)
registerDoParallel(cores=24)

my_data_frame = .....
my_results = foreach(i=1:24,.combine=rbind) %dopar%
{ ... }
```

1. allocate workers

2. Make 'foreach' loop

3. specify how to combine results

4. %dopar% runs it across cores, (%do% runs it serially)

R multicore coding

```
install.packages(doParallel)
library(doParallel)
registerDoParallel(cores=24)

my_data_frame = .....
my_results = foreach(i=1:24,.combine=rbind) %dopar%
{
  ...
  your code here
  return( a variable or object )
}
```

1. allocate workers

2. Make 'foreach' loop

3. specify how to combine results

4. %dopar% runs it across cores, (%do% runs it serially)

R multicore coding

```
install.packages(doParallel)
library(doParallel)
registerDoParallel(cores=24)
```

1. allocate workers

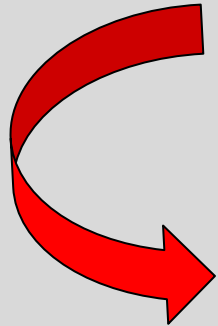
```
my_data_frame = .....
```

2. Make 'foreach' loop

```
my_results = foreach(i=1:24,.combine=rbind) %dopar%
{ ...
  your code here
  return( a variable or object )
})
```

3. specify how to combine results

4. %dopar% runs it across cores, (%do% runs it serially)



BEWARE: foreach will copy data to every core if it seems necessary

R multinode uses same steps, but:

BEWARE: copying data across nodes is *MUCH* higher communication costs

```
library(doParallel)
```

1. allocate cluster

```
cl <- makeCluster(48)  
registerDoParallel(cl)
```



```
my_data_frame = .....
```

2. Use foreach and %dopar%

```
results = foreach(i=1:48,.combine=rbind) %dopar%  
{ ... your code here
```



```
    return( a variable or object )  
  })  
stopCluster(cl)
```

Exercise for home: Testing R parallel, command line

Step:

Log into expanse terminal

```
$ srun --account=__  
      --partition=shared  
      --nodes=1 --ntasks-per-node=1  
      --cpus-per-task=8 --mem=16G  
      --time=04:00:00 --pty --wait=0  
      /bin/bash
```

```
$ cd github repo: _r_on_HPC folder
```

```
$ 'module spider r' to see what to load
```

```
$ R (run one time)
```

```
> install.packages("doParallel") (say 'yes' for local install)
```

```
7. $ Rscript --vanilla TestDoParallel_v1.R
```

```
4rodrig@login01 RHPC]$ module load cpu/0.15.4 gcc/9.2.0  
  
The following have been reloaded with a version change:  
1) cpu/0.17.3b => cpu/0.15.4  
  
4rodrig@login01 RHPC]$ module load r/4.0.2-openblas  
4rodrig@login01 RHPC]$
```

```
train113@exp-1-24 4.2.RandHPC]$ Rscript --vanilla ./TestDoParallel_v1.R  
loading required package: doParallel  
loading required package: foreach  
loading required package: iterators  
loading required package: parallel  
1] "starting dopar test"  
1] "Using N rows= 10000 P cols= 200"  
1] "X size is: 15.3 Mb"
```

Exercise: Testing R parallel, 'top' command

Also try this:

Login to Expanse with a second terminal window

```
$ queue -u $USER
```

```
$ ssh exp-#-## (ssh into that compute node)
```

```
$ top -u $USER
```

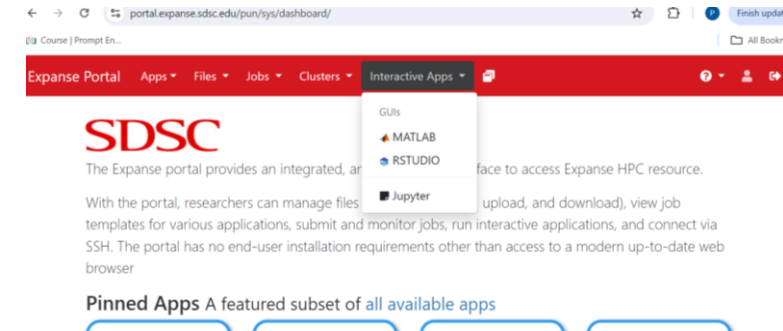
- how's memory usage?

enter H to see threads,

enter f -> down arrow -> space -> esc to toggle cpuid

Exercise: Testing R parallel in portal

- Log into expanse portal and start R studio
- goto URL: <https://portal.expanse.sdsc.edu/training>
- We can run a test script while monitoring usage
- Also, look for tradeoffs in memory vs execution as matrix size varies (see next slides)



The screenshot shows the SDSC Expanse Portal interface. The 'Interactive Apps' menu is open, and 'RSTUDIO' is selected. Below, the 'batch_connect/sys/rstudio/session_contexts/new' page is shown, where the 'Partition' dropdown is set to 'compute'.

1 Open portal ->
Interactive Apps ->
Rstudio

Enter
Node: "compute"
Cores: "64"
Memory: 124 Gb
Res: ciml25cpu
(other fields defaults ok)

2 Also login to Expanse terminal window

\$ queue -u \$USER
\$ ssh exp-##-##
\$ top -u \$USER

The terminal window shows the following commands and output:

```

Last login: Fri Jun  4 15:01:29 2021 from 71.128.8.73
[p4rodrig@login02 ~]$ queue -u p4rodrig
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
3246260 compute sys/dash p4rodrig R 0:27 1 exp-2-15
[p4rodrig@login02 ~]$ ssh exp-2-15
Last login: Sat Jun  5 13:09:04 2021
[p4rodrig@exp-2-15 ~]$ top -u $USER

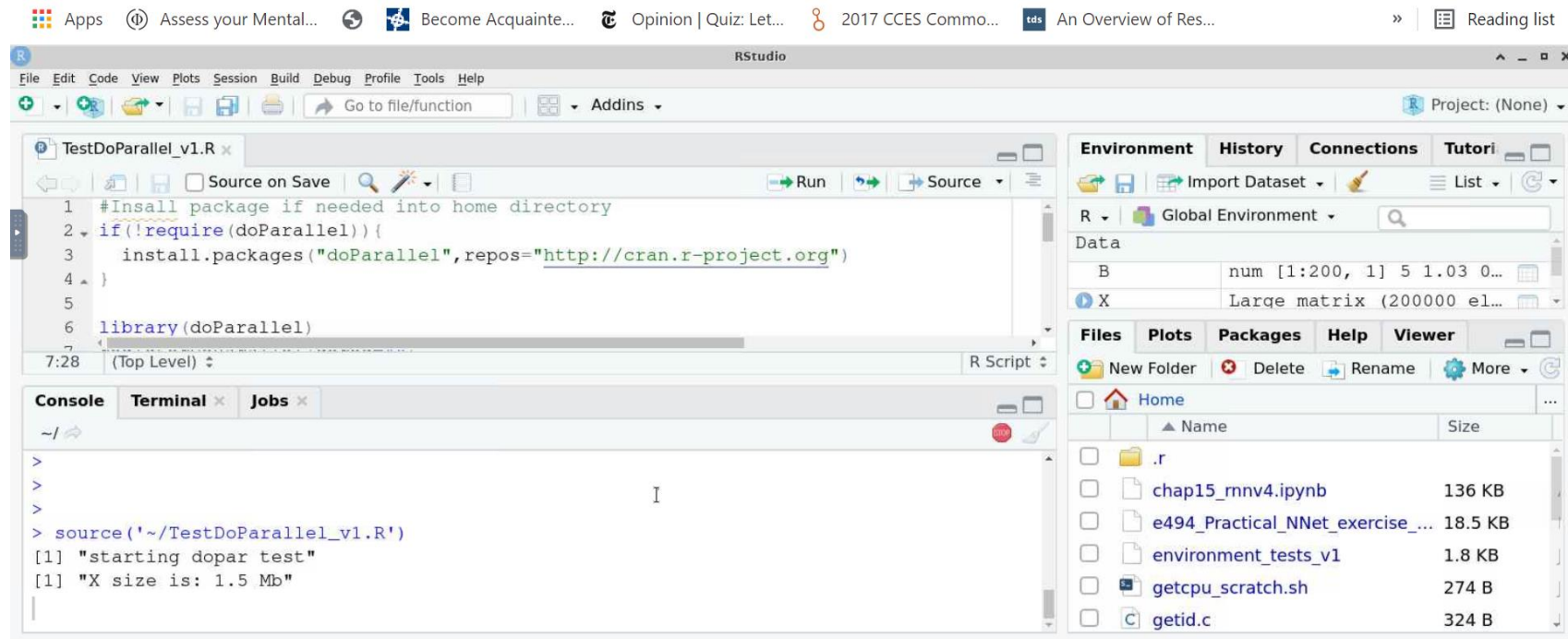
```

'H' will toggle threads
'f', downarrow to P, space, esc.

3 Open the 'Test_doParallel' Rscript

Select 'source' to run the whole script, it will install 'doParallel' package (if the R installation doesn't have it already)

look for # <<< ----- comments to change data parameters



The screenshot shows the RStudio interface with the 'TestDoParallel_v1.R' script open in the editor. The script contains the following code:

```
1 #Install package if needed into home directory
2 if(!require(doParallel)){
3   install.packages("doParallel",repos="http://cran.r-project.org")
4 }
5
6 library(doParallel)
```

The console output shows the results of running the script:

```
>
>
>
> source('~/.TestDoParallel_v1.R')
[1] "starting dopar test"
[1] "X size is: 1.5 Mb"
```

The Environment pane on the right shows the Global Environment with variables B (num [1:200, 1] 5 1.03 0...) and X (Large matrix (200000 el...)). The Files pane on the right shows the current directory with files like chap15_mnv4.ipynb, e494_Practical_NNet_exercise..., environment_tests_v1, getcpu_scratch.sh, and getid.c.

Review the top output

```
p4rodrig@exp-9-27:~  
top - 15:35:51 up 87 days, 21:07, 1 user, load average: 20.97, 5.76, 4.37  
Tasks: 1788 total, 41 running, 1747 sleeping, 0 stopped, 0 zombie  
%cpu(s): 32.6 us, 0.1 sy, 0.0 ni, 67.0 id, 0.0 wa, 0.2 hi, 0.1 si, 0.0 st  
MiB Mem : 257517.8 total, 210968.9 free, 42132.2 used, 4416.7 buff/cache  
MiB Swap: 0.0 total, 0.0 free, 0.0 used, 210846.5 avail Mem  
  
  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND  
54587 p4rodrig  20   0 17.5g 896016 4196 R 100.0  0.3  0:37.27 rsession  
54562 p4rodrig  20   0 17.5g 896016 4196 R 99.7  0.3  0:37.48 rsession  
54568 p4rodrig  20   0 17.5g 896016 4196 R 99.7  0.3  0:37.46 rsession  
54571 p4rodrig  20   0 17.5g 896016 4196 R 99.7  0.3  0:37.35 rsession  
54572 p4rodrig  20   0 17.5g 896016 4196 R 99.7  0.3  0:37.33 rsession  
54574 p4rodrig  20   0 17.5g 896016 4196 R 99.7  0.3  0:37.38 rsession  
54579 p4rodrig  20   0 17.5g 896016 4196 R 99.7  0.3  0:37.33 rsession  
54591 p4rodrig  20   0 17.5g 896016 4196 R 99.7  0.3  0:37.24 rsession
```

Notice the elapsed time and memory size

Change the NxP matrix size and rerun

(start with N=10K, P=2K)

```
7:28 (Top Level)   
Console Terminal x Jobs x  
~/   
>  
> source('~/.TestDoParallel_v1.R')  
[1] "starting dopar test"  
[1] "X size is: 1.5 Mb"  
      user system elapsed  
      1.176   1.969   30.620  
> |
```

```
10 # Make up some random data and lis  
11 N=100000;      #N rows start with  
12 P=2000;       #P columns 200 for  
13  
14 #make random data with 1 column ar  
15 X =matrix(rnorm(N*P),N,P)  
16 X[,1] =X[,1]+1  
17  
16:28 (Top Level)   
Console Terminal x Jobs x  
~/   
Loading required package: foreach  
Loading required package: iterators  
Loading required package: parallel  
[1] "starting dopar test"  
[1] "X size is: 1.5 Gb"
```

Try this at home:

Let $N=100K$, $P=2000$

Notice the memory used is close to 124Gb we asked for

```
p4rodrig@exp-9-27:~  
top - 15:38:40 up 87 days, 21:10, 1 user, load average: 10.77, 6.29, 4.76  
Tasks: 1749 total, 19 running, 1730 sleeping, 0 stopped, 0 zombie  
%Cpu(s): 14.0 us, 0.0 sy, 0.0 ni, 85.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st  
MiB Mem : 257517.8 total, 130239.0 free, 123199.7 used, 4079.0 buff/cache  
MiB Swap: 0.0 total, 0.0 free, 0.0 used, 129947.3 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND	P
55219	p4rodrig	20	0	24.2g	7.6g	2696	R	100.0	3.0	0:24.52	rsession	68
55227	p4rodrig	20	0	24.2g	7.6g	3064	R	100.0	3.0	0:24.55	rsession	88
55235	p4rodrig	20	0	24.2g	7.6g	2696	R	100.0	3.0	0:24.56	rsession	80
55236	p4rodrig	20	0	24.2g	7.6g	2696	R	100.0	3.0	0:24.70	rsession	100
55237	p4rodrig	20	0	24.2g	7.6g	2696	R	100.0	3.0	0:24.50	rsession	47
55242	p4rodrig	20	0	24.2g	7.6g	2696	R	100.0	3.0	0:24.36	rsession	32
55253	p4rodrig	20	0	24.2g	7.6g	2696	R	100.0	3.0	0:24.69	rsession	126
55259	p4rodrig	20	0	24.2g	7.6g	2696	R	100.0	3.0	0:24.00	rsession	16
55261	p4rodrig	20	0	24.2g	7.6g	2696	R	100.0	3.0	0:24.25	rsession	24
55265	p4rodrig	20	0	24.2g	7.6g	2696	R	100.0	3.0	0:23.96	rsession	6
55239	p4rodrig	20	0	24.2g	7.6g	2696	R	99.7	3.0	0:24.61	rsession	20
55241	p4rodrig	20	0	24.2g	7.6g	2696	R	99.7	3.0	0:24.43	rsession	8
55243	p4rodrig	20	0	24.2g	7.6g	2836	R	99.7	3.0	0:24.53	rsession	104

If you ask for 248Gb will it run?

What if you use only 24 cores?

Parallelizing for loops

(pseudo code)

R with
doParallel

registerDoParallel()

foreach with dopar,

combine results

Parallelizing for loops

(pseudo code)

R with
doParallel

registerDoParallel()

foreach with dopar,

combine results

Matlab with
parallel toolbox

parpool()

parfor

or

*'spmd' with
distributed arrays*

gather array

Parallelizing for loops

(pseudo code)

R with
doParallel

registerDoParallel()

foreach with dopar,

combine results

Matlab with
parallel toolbox

parpool()

parfor

or

*'spmd' with
distributed arrays*

gather array

Python with
dask.distributed

Client(numwkr)

for i in range(numwkr):

A=delayed(my_func)(i)

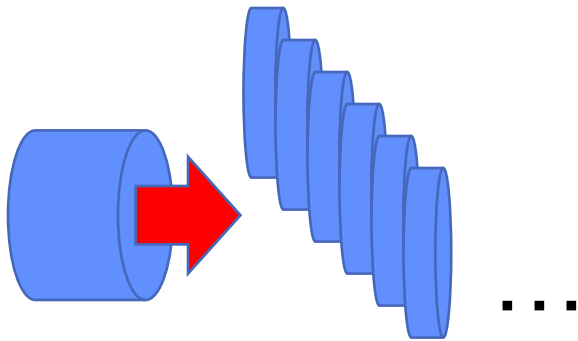
Acombine.append(A)

Acombined.compute()

An option for (embarrassingly) Parallel R

If you can process data parts independently then you can parallelize in an embarrassingly simple way

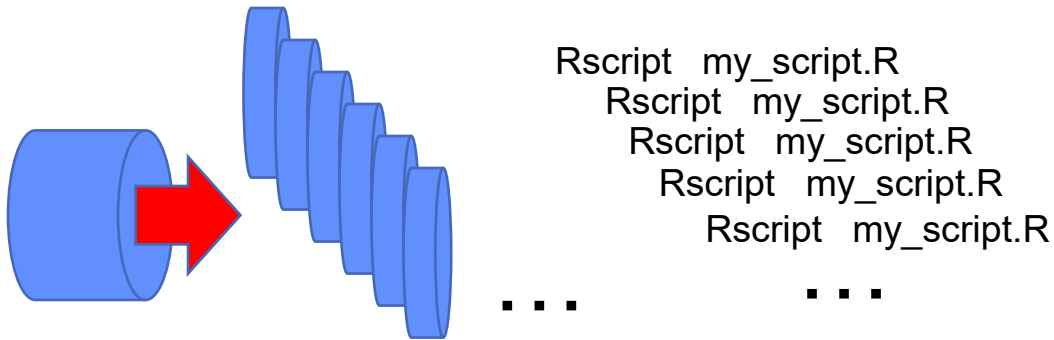
1. First, s: Split up data into N parts



An option for (embarrassingly) Parallel R

If you can process data parts independently then you can parallelize in an embarrassingly simple way

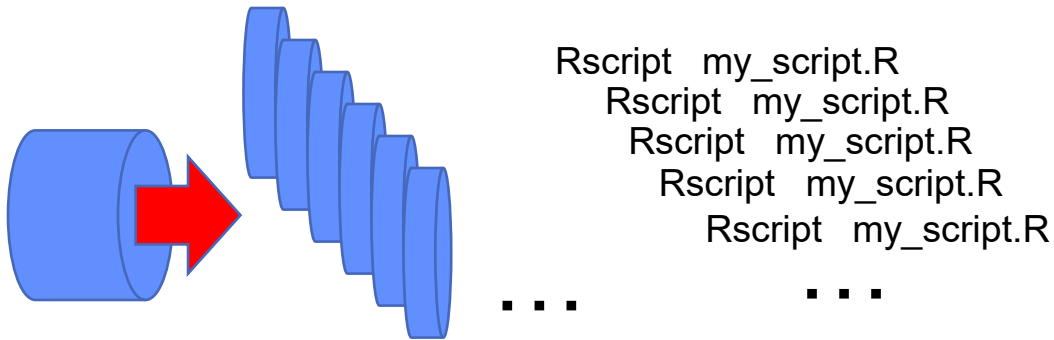
1. First, s: Split up data into N parts
2. Launch N instances of your R script, one per CPU core. Each instance processes one (or more) datasets



An option for (embarrassingly) Parallel R

If you can process data parts independently then you can parallelize in an embarrassingly simple way

1. First, s: Split up data into N parts
2. Launch N instances of your R script, one per CPU core. Each instance processes one (or more) datasets

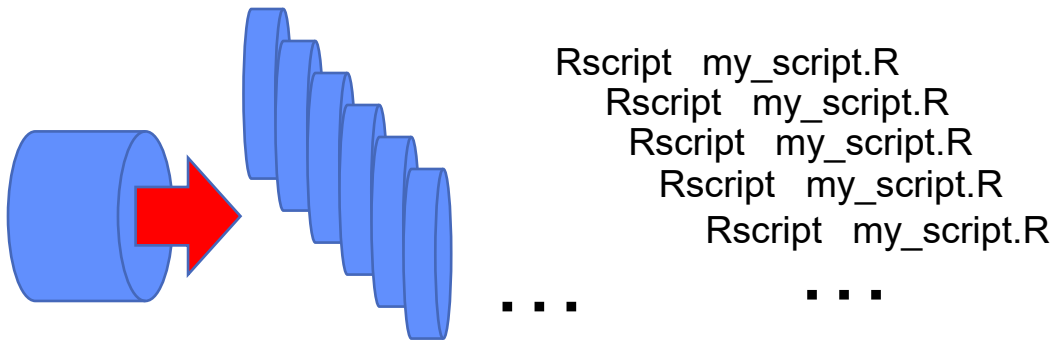


ISSUE: how to launch R scripts and pass arguments about which dataset to process?

An option for (embarrassingly) Parallel R

If you can process data parts independently then you can parallelize in an embarrassingly simple way

1. First, s: Split up data into N parts
2. Launch N instances of your R script, one per CPU core. Each instance processes one (or more) datasets



ISSUE: how to launch R scripts and pass arguments about which dataset to process?

2 ways:

mpirun command (more flexible)
parallel command (Unix tool, maybe simpler)

Example Using GNU tool “parallel”

Get list of files

*|' pipe list to
parallel command*

```
$ ls *dataset*.txt | parallel ...
```

```
p4rodrig@exp-7-27:~/WKSHOPS/AdvWeb0125/PARTEST
[p4rodrig@exp-7-27 PARTEST]$ ls *dataset*.txt
fakedataset.1.txt  fakedataset.2.txt  fakedataset.3.txt  fakedataset.4.txt
[p4rodrig@exp-7-27 PARTEST]$
[p4rodrig@exp-7-27 PARTEST]$
```


Example Using GNU tool “parallel”

Get list of files

*‘|’ pipe list to
parallel command*

*‘-j’ 4 means run 4 jobs
(1 job on each CPU core)*

```
$ ls *dataset*.txt | parallel -j 4 Rscript your-r-script.R
```

The parallel command passes file names as arguments to Rscript

```
p4rodrig@exp-7-27:~/WKSHOPS/AdvWeb0125/PARTEST
[p4rodrig@exp-7-27 PARTEST]$ ls *dataset*.txt
fakedataset.1.txt  fakedataset.2.txt  fakedataset.3.txt  fakedataset.4.txt
[p4rodrig@exp-7-27 PARTEST]$
[p4rodrig@exp-7-27 PARTEST]$
```

```
#!/usr/bin/env Rscript
args = commandArgs(trailingOnly=TRUE)

print(paste('starting script'))
for (i in seq(length(args))) {
  print(paste('arg: ', i, ' value:', args[i]))
  Sys.sleep(5)
}
```

Example Using GNU tool “parallel”

Get list of files

*‘|’ pipe list to
parallel command*

*‘-j’ 4 means run 4 jobs
(1 job on each CPU core)*

```
$ ls *dataset*.txt | parallel -j 4 Rscript your-r-script.R
```

The parallel command passes file names as arguments to Rscript

```
p4rodrig@exp-7-27:~/WKSHOPS/AdvWeb0125/PARTEST
[p4rodrig@exp-7-27 PARTEST]$ ls *dataset*.txt
fakedataset.1.txt fakedataset.2.txt fakedataset.3.txt fakedataset.4.txt
[p4rodrig@exp-7-27 PARTEST]$
[p4rodrig@exp-7-27 PARTEST]$
[p4rodrig@exp-7-27 PARTEST]$ ls *dataset*.txt | parallel -j 4 Rscript TestArgs.R
[1] "starting script"
[1] "arg: 1 value: fakedataset.1.txt"
[1] "starting script"
[1] "arg: 1 value: fakedataset.2.txt"
[1] "starting script"
[1] "arg: 1 value: fakedataset.3.txt"
[1] "starting script"
[1] "arg: 1 value: fakedataset.4.txt"
[p4rodrig@exp-7-27 PARTEST]$
```

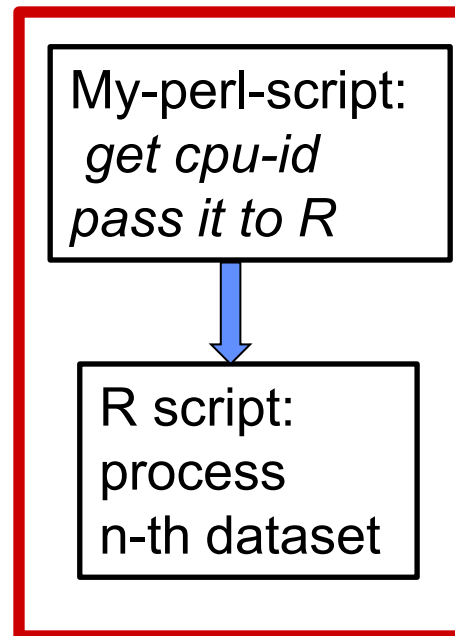
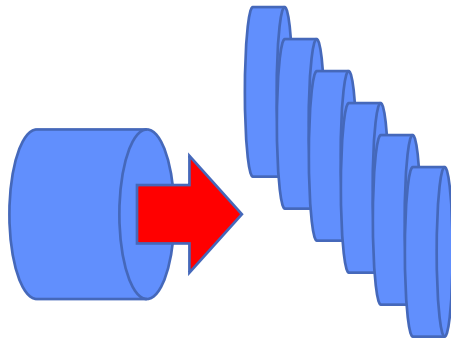
```
#!/usr/bin/env Rscript
args = commandArgs(trailingOnly=TRUE)
```

```
print(paste('starting script'))
for (i in seq(length(args))) {
  print(paste('arg: ', i, ' value:', args[i]))
  Sys.sleep(5)
}
```

mpirun option for (embarrassingly) Parallel R

1. Split up
data into N
parts

2. In slurm batch script or command line:
`mpirun ... my-perl-script`



*This gets
distributed
across nodes
and cores by
slurm & mpi
parameters*

Slurm parameters: one R instance per core across all nodes

Normal
batch
job info

```
...  
#SBATCH --partition=compute  
#SBATCH --nodes=2  
#SBATCH --ntasks-per-node=128  
#SBATCH --cpus-per-task=1
```

2 x 128 = 256 mpi ranks

```
module load slurm  
module load cpu  
module load gcc  
module load intel-mpi
```

256 perl script/R instances
1 core each

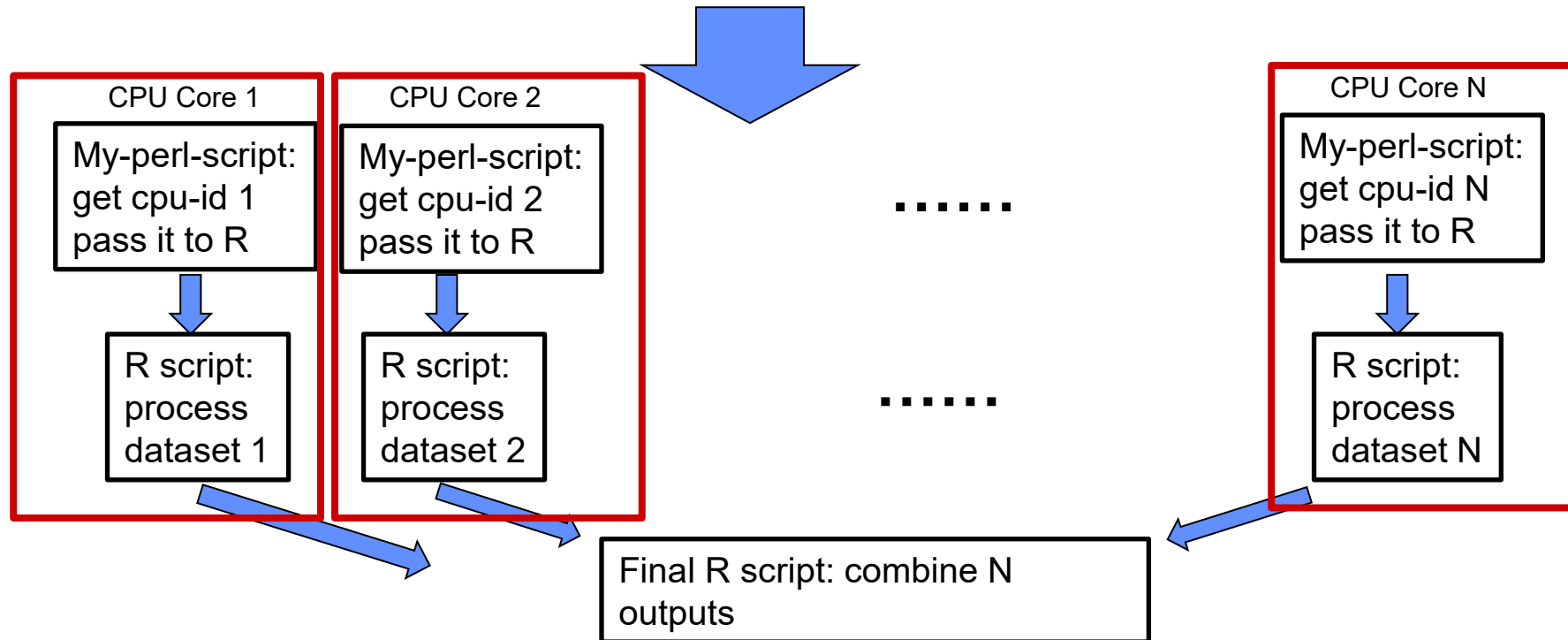
```
mpirun -genv I_MPI_PIN_DOMAIN=omp:compact ./get_mpirank_runcmd.pl
```

(based on /cm/shared/examples/sdsc/mpi-openmp-hybrid/hybrid-slurm.sb)

one R instance per core across all nodes

In slurm batch script:

```
mpirun ... my-perl-script
```



Slurm parameters: one R instance per node with 128 cores per R instance

Normal
batch
job info

```
...  
#SBATCH --partition=compute  
#SBATCH --nodes=2  
#SBATCH --ntasks-per-node=1  
#SBATCH --cpus-per-task=128  
  
module load slurm  
module load cpu  
module load gcc  
module load intel-mpi  
  
module load r  
  
mpirun -genv I_MPI_PIN_DOMAIN=omp:compact ./get_mpirank_runcmd.pl
```

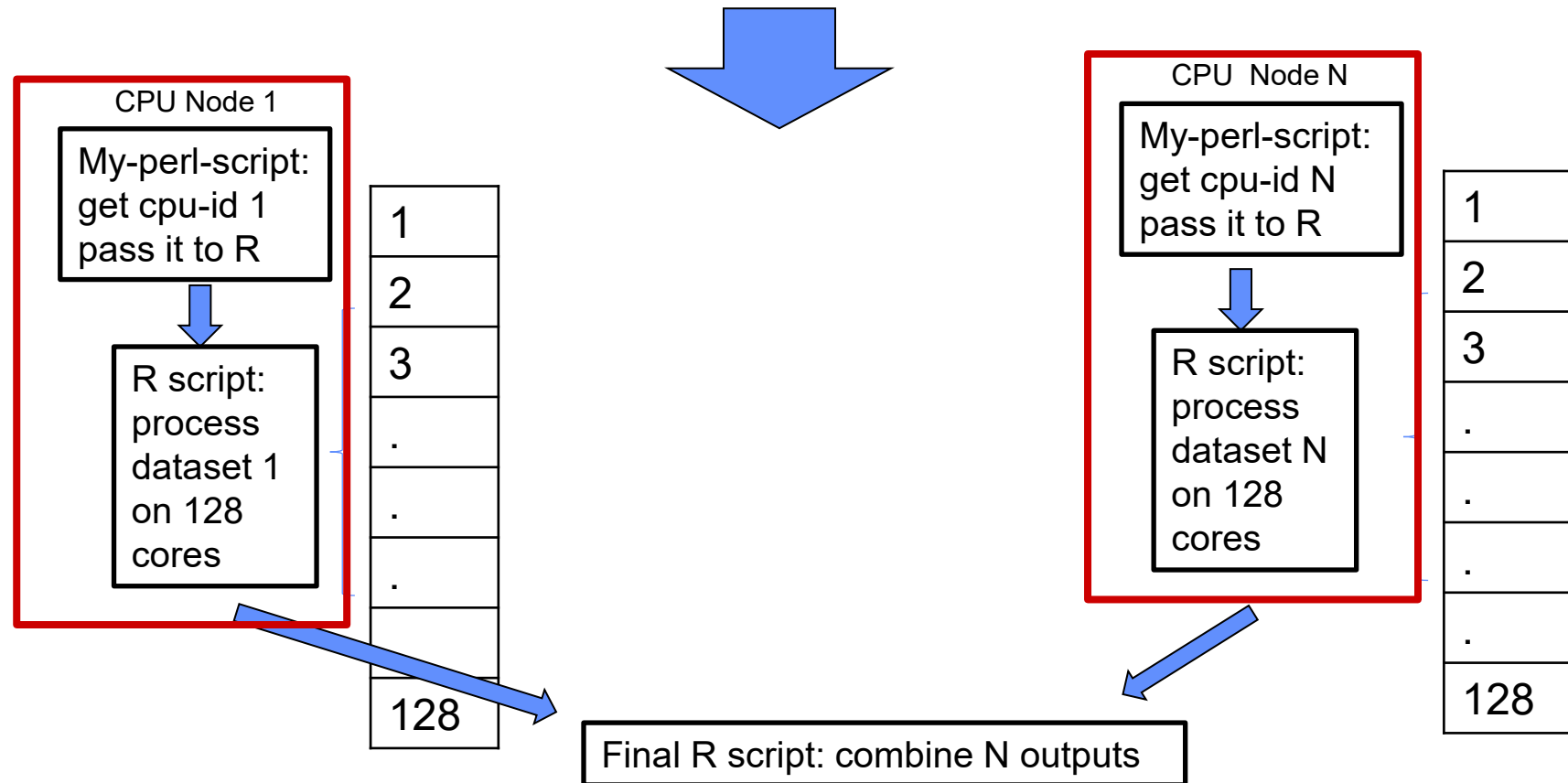
2 x 1 = 2 mpi ranks

2 perl script/R instances
128 cores each
(doParallel can use them)

Example: One R instance per node, doParallel across all cores in each node

In slurm batch script:

```
mpirun ... my-perl-script
```



- **Some example parallelizations that launch R instances onto CPU cores**

Example 1: scaling MCMC

*Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models,
F. Bumbaca, UCIrvine*

- Probabilities of individual's web activity and clicks
- Used *mpirun* launch parallelization for MCMC model on each individual
- Then combined results via a hierarchical model
(*rhierMnlRwMixturefunction* in the R package *bayesm*)

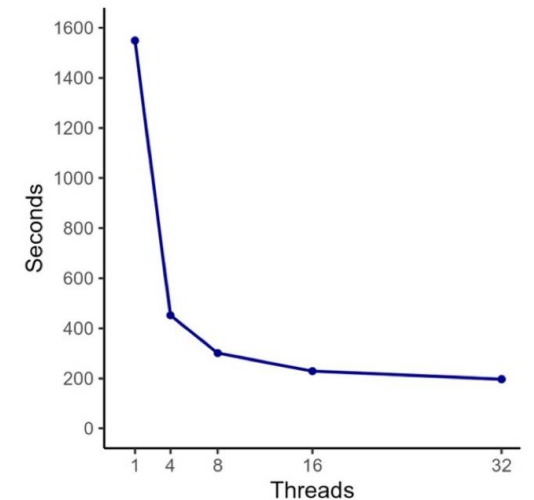
Example 2: gene expression prediction models

MAGEPRO <https://github.com/kaiakamatsu/MAGEPRO>

Kai Akamatsu, *Amariuta Lab, UCSD Medicine*

<https://www.medrxiv.org/content/10.1101/2024.09.25.24314410v1.full.pdf>

- Goal: Enable the identification of novel gene-trait associations through transcriptome-wide association studies.
- Created a pipeline for each gene
 - a) eQTL statistics (plink)*
 - b) gene expression heritability*
 - c) Lasso based prediction model*
- Ran 22 batch jobs, 1 per chromosome
- Within each batch job, used GNU tool 'parallel' to run steps with 1 core per gene



Launching Independent R sessions

- **Placing *independent* R sessions onto cores is more flexible for:**
 - data management
 - large number of separate models
 - large variation in time per model
 - hybrid multimode/multicore scripts

But perhaps requires more programming or pre/postprocessing than 'doParallel'

A note on installing R Packages (into your own directories)

- In R (might help to be on interactive node):

install.packages('package-name')

(see <https://cran.r-project.org/> for package lists and reviews)

- Sometimes you have to be explicit:

*install.packages('ggmap',
 repos='http://cran.us.r-project.org',dependencies=TRUE)*

If compiling is required and you get an error, call support

Packages are put into your /home/user/R directory

Installing R Packages: Conda example

- Conda is a general package manager
- If install.packages not working in R, Conda and a new base R might work better.
- Example: from unix prompt on Expanse node:

```
module load anaconda3
```

```
conda create -n myrenv          #create your conda environment
```

```
conda activate myrenv
```

```
conda update -n base -c defaults conda
```

```
conda install r-base=4.3.1      #install R and your packages in your environment
```

```
conda install r-biomod2
```

End