R + HPC

David Walling

Data Group

Texas Advanced Computing Center

walling@tacc.utexas.edu

What to do if the computation is too big for a single desktop

- A common user question:
  - I have an existing R solution for my research work. But the data is growing too big. Now my R program runs days to finish or simply runs out of memory.
- 3 strategies
  - Move to more powerful hardware
  - Automatic offloading with multicore/GPU/MIC
  - Implement code using parallel packages

Strategy 1: Powerful Hardware

- Stampede - HPC
  - normal queue: 16 cores/32GB mem/48 hour max
  - largemem queue: 32 cores/ 1TB mem/48 hour max
  - normal-mic queue: Access to Intel MIC co-processors
- Lonestar5 - HPC
  - normal queue: 24 cores/64GB mem/48hour max
- Maverick - Vis
  - vis queue: 32 cores/128GB mem/4 hour max
  - gpu queue: has access to GPUs
- Wrangler - Data
  - normal queue: 24 cores/128GB mem/48hour max
  - Reservations: Up to 1 month reserved nodes
  - Hadoop/Spark + R Streaming
  - Dedicated Flash storage for IO intensive tasks

Linux

- We run linux
- More command line driven
- Daunting for Windows only users
- Filezilla allows for editing R scripts remotely
- Can use VNC access to get a Linux desktop
- RStudio helps the transition
- TACC Linux Training and Courses

Modules

- We provide an optimized build of R called Rstats
- Compiled with Intel compilers (vs. gnu) and linked against MKL math library
- Managing the linux environment is done via TACC modules commands

```
login1.wrangler(1)$ ml

Currently Loaded Modules:
  1) TACC-paths   2) Linux   3) cluster-paths   4) intel/15.0.3   5) mvapich2/2.1   6) cluster   7) TACC

login1.wrangler(2)$ ml Rstats
login1.wrangler(3)$ which R
/opt/apps/intel15/mvapich2_2_1/Rstats/3.2.1/bin/R
```

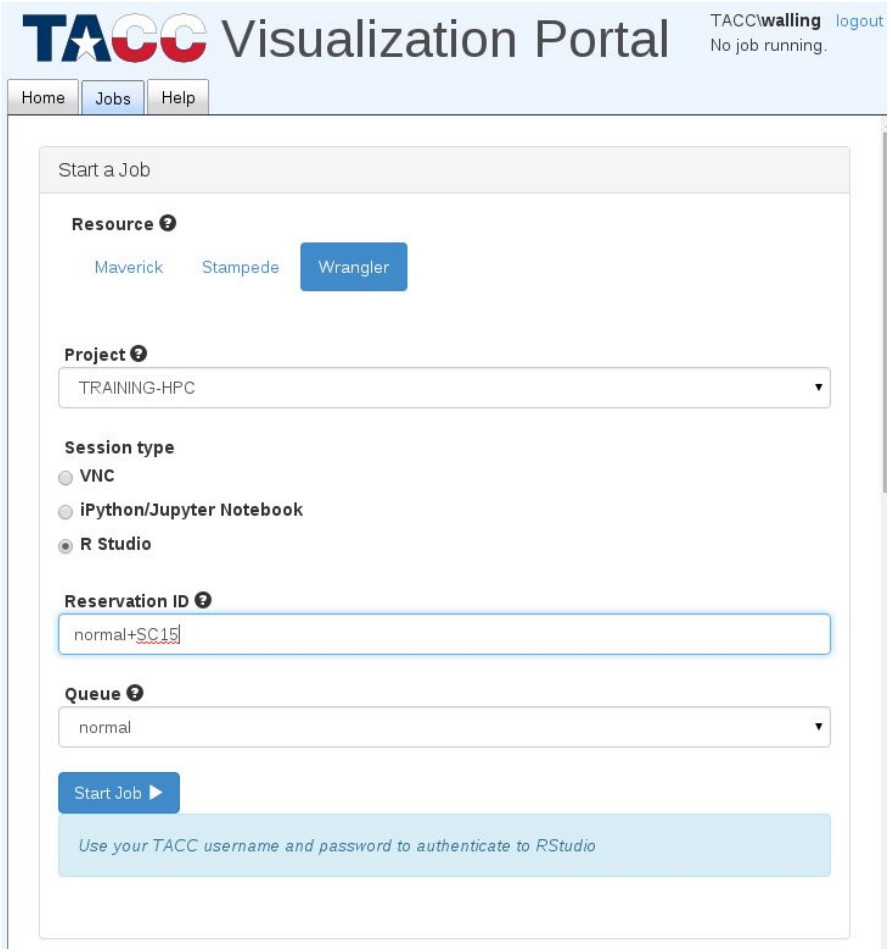
Runnings Jobs

- DO NOT RUN on login nodes!!!
- Use idev for interactive sessions
- sbatch for batch submissions
- ‘Read’ User Guides

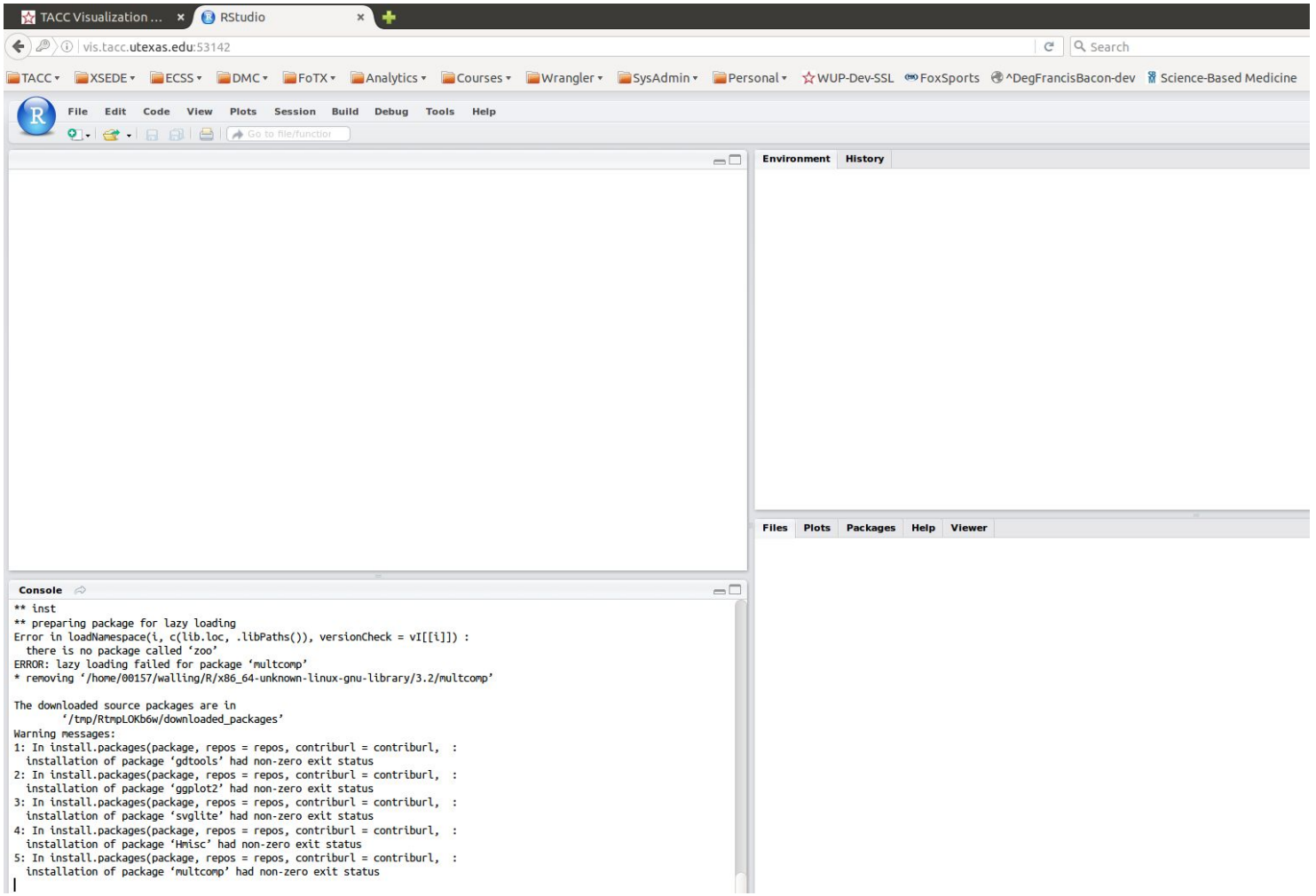
```
login1.wrangler(4)$ idev -p normal -t 04:00:00

Defaults file    : ~/.idevrc
Default  project : TG-STA110019S
Default  time    : 30 min.
Default  queue   : debug
System           : Wrangler
Using queue      : -p normal
time (hh:mm:ss)  : -t 04:00:00

 Just a Note: Your reservation hadoop+SSI2016+1663 is INACTIVE.
 Use idev -r to use it when it becomes ACTIVE.
 To see when it begins, execute:  scontrol show reservations

------------------------------------------------------------
         Welcome to Wrangler at TACC
------------------------------------------------------------

--> Verifying valid submit host (login1)...OK
--> Verifying valid jobname...OK
--> Enforcing max jobs per user...OK
--> Verifying availability of your home dir (/home/00157/walling)...OK
--> Verifying availability of your work dir (/work/00157/walling/wrangler)...OK
--> Verifying valid ssh keys...OK
--> Verifying access to desired queue (normal)...OK
--> Verifying job request is within current queue limits...OK
--> Checking available allocation (TG-STA110019S)...OK
Submitted batch job 17216

 After your idev job begins to run, a command prompt will appear,
 and you can begin your interactive development session.
 We will report the job status every 4 seconds: (PD=pending, R=running).

job status:  PD
job status:  R
--> Job is now running on masternode= c251-124...OK
--> Sleeping for 7 seconds...OK
--> Checking to make sure your job has initialized an env for you....OK
--> Creating interactive terminal session (login) on master node c251-124.
c251-124.wrangler(1)$
```

RStudio @ TACC

- URL: http://vis.tacc.utexas.edu
- Click ‘Jobs’
- Select
  - Resource = Wrangler
  - Project = SSI2016
  - Session Type = R Studio
  - Queue = hadoop
  - ReservationID = hadoop+SSI2016+1663
- Click ‘Start Job’
- Once job is running, click ‘Open in browser’



RStudio @ TACC

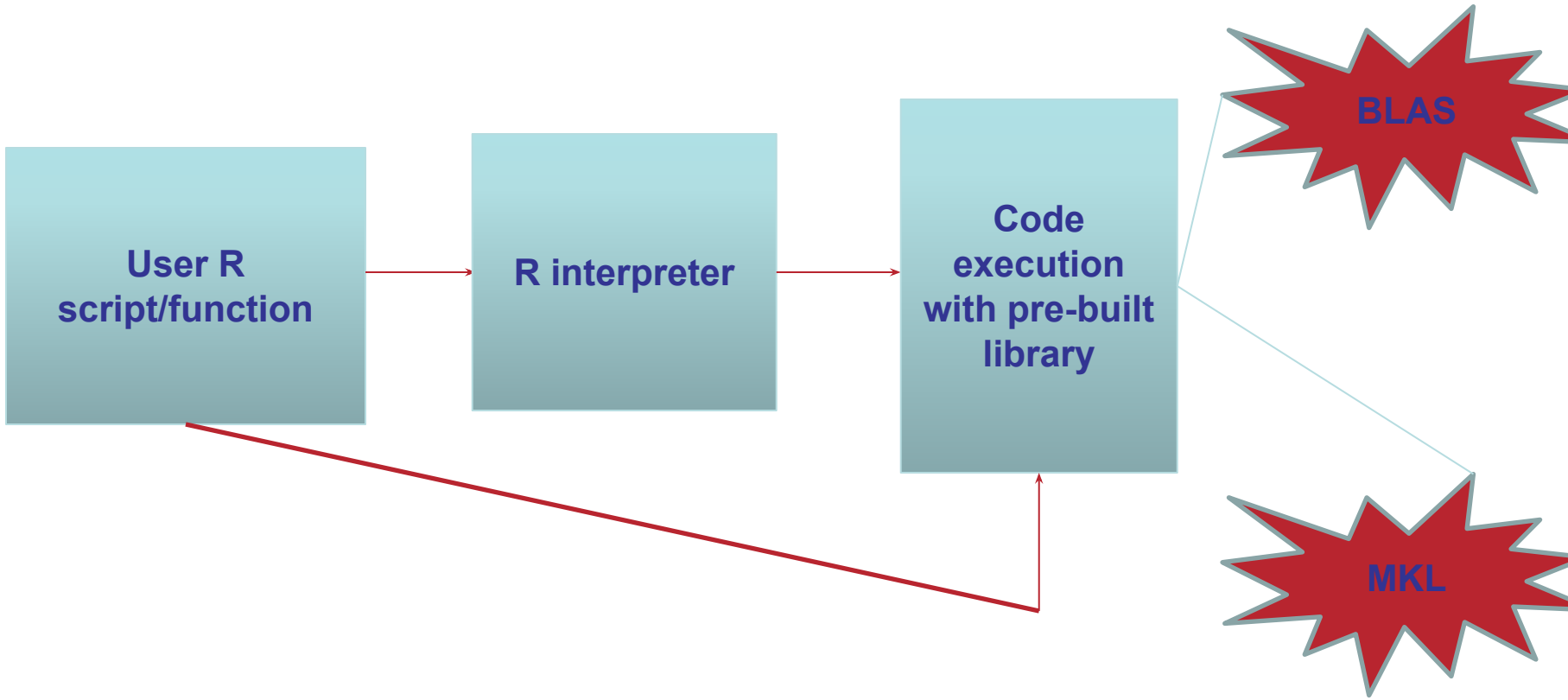


Strategy 2: Automatic Offloading

- R is originally designed for single threaded execution.
  - Slow performance
  - Not scalable with large data
- R can be built and linked to libraries that utilize latest multi-core technology
- This enables automatic parallel execution for some operations, most commonly, linear algebra related computations, i.e.
  - $Ax = b$
  - $B^* = (X^T X)^{-1} X^T Y$

Dynamic Library & R

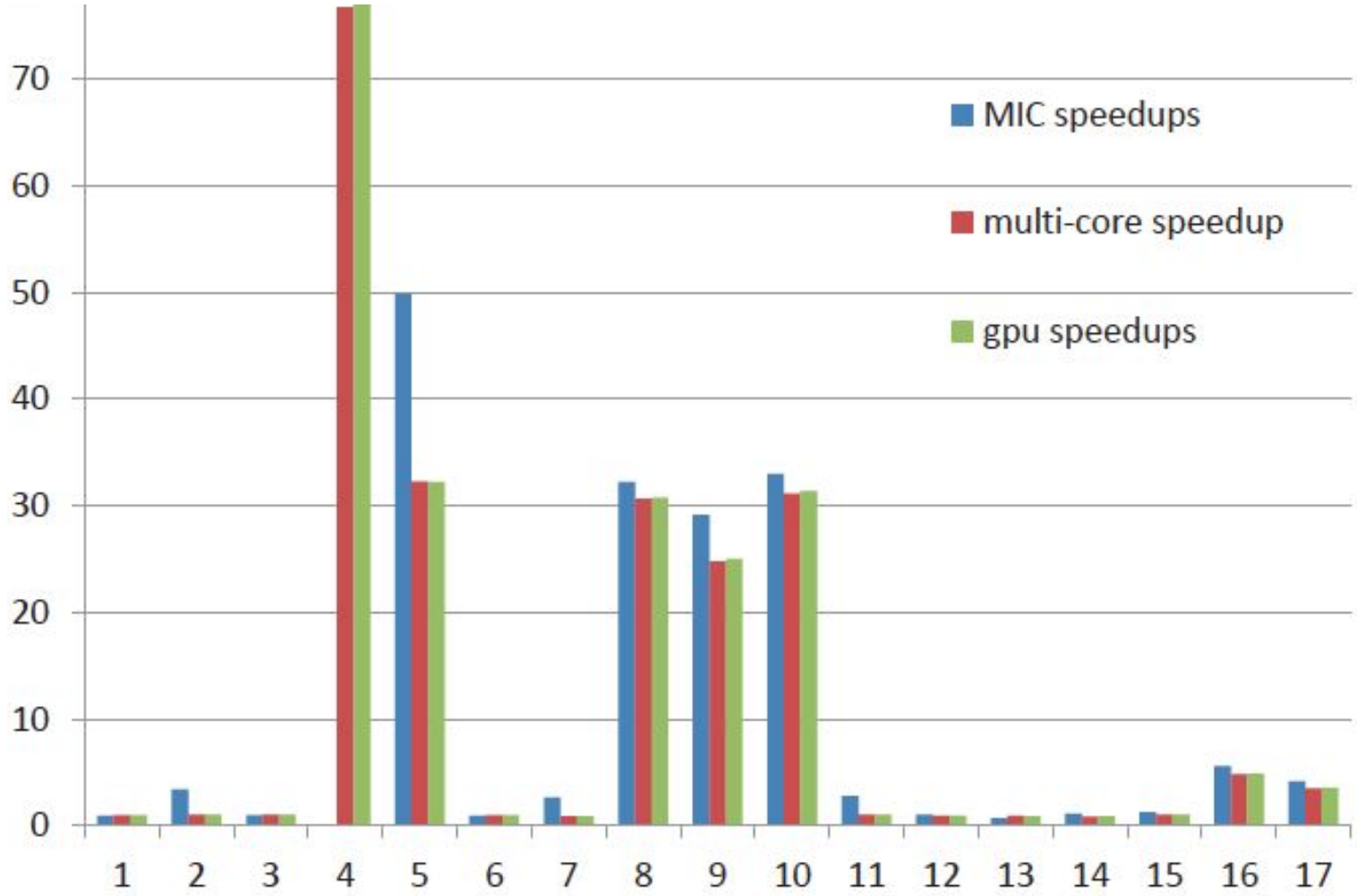
- MKL provides BLAS/LAPACK routines that can “offload” to the Xeon Phi Coprocessor, reducing total time to solution



Automatic Offloading

- Hardware supported:
  - Multiple cores on CPU
  - Intel Xeon Phi coprocessor (on Stampede)
  - GPGPU (on Stampede/Maverick) - not automatic
- Libraries supporting automatic offloading
  - Intel Math Kernel Library (MKL)
    - Available on stampede and maverick for users
  - HiPlarB
    - Open source and freely available
    - http://www.hiplar.org/hiplar-b.html

R-2.5 benchmark performance with automatic hardware acceleration



Automatic Offloading

- Advantage:
  - No code changes needed
  - User can run R solution as before without knowledge of the parallel execution.
- Limitations:
  - Only support limited (but fairly common) computational operations.

Automatic Offloading

- How?
  - On Stampede: Use ‘module load Rstats’: built with Intel compilers using all optimizations + linking to MKL
  - Re-install R packages that also have C code. For the interested, you can see C code being compiled as the package is installed.
  - Set environment variables
    - **export MKL_MIC_ENABLE=1**
    - **export OMP_NUM_THREADS=16**
    - **export MIC_OMP_NUM_THREADS=240**

Automatic Offloading

```
$ cat AutoOffload.R

size <- 1000000
data <- data.frame(y=rnorm(size))
data <- cbind(data, sapply(1:100, function(i) { rnorm(size) }))
ptm <- proc.time()
model <- lm(y~., data=data)
proc.time() - ptm
```

Automatic Offloading - Off

```
$ cat run-AutoOffload-Off.slurm
#!/bin/bash
#SBATCH -J AutoOffload-Off-R
#SBATCH -o AutoOffload-Off.out%j
#SBATCH -p normal-mic
#SBATCH -t 00:30:00
#SBATCH -A TRAINING-HPC
#SBATCH
--reservation=SC15-R-Training
#SBATCH -e AutoOffload-Off.err%j
#SBATCH -N 1
#SBATCH -n 16

#set environment
module purge
module load TACC
module load intel/15.0.2
module load Rstats
```

Automatic Offloading - On

```
$ cat run-AutoOffload-On.slurm
#!/bin/bash
#SBATCH -J AutoOffload-On-R
#SBATCH -o AutoOffload-On.out%j
#SBATCH -p normal-mic
#SBATCH -t 00:30:00
#SBATCH -A TRAINING-HPC
#SBATCH
--reservation=SC15-R-Training
#SBATCH -e AutoOffload-Off.err%j
#SBATCH -N 1
#SBATCH -n 16

#set environment
module purge
module load TACC
module load intel/15.0.2
module load Rstats
……

# add path
export MKL_MIC_ENABLE=1
export MKL_HOST_WORKDIVISION=0.3
export MKL_MIC_WORKDIVISION=0.7
export OFFLOAD_REPORT=2
export     OMP_NUM_THREADS=16
export     MKL_NUM_THREADS=16
export MIC_OMP_NUM_THREADS=240
export MIC_MKL_NUM_THREADS=240
export       KMP_AFFINITY=compact   #
scatter or compact
export    MIC_KMP_AFFINITY=balanced

# Run a script
Rscript AutoOffload.R
```

Automatic Offloading - Comparison

```
stampede$ squeue -u walling
             JOBID   PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           5867653  normal-mic AutoOffl  walling R       0:37      1 c401-403
           5867656  normal-mic AutoOn  walling  R       1:05      1 c401-702


stampede$ cat AutoOffload-Off.out5867656
  user  system elapsed
16.532   1.797  18.492


stampede$ cat AutoOffload-On.out5867653
  user  system elapsed
62.919   2.484  11.458
```

Strategy 3: Parallel Packages

- There are many parallel packages available to enable parallelism with R
- Two most common approaches included with R distribution
  - Multicore
  - Snow/Rmpi
- Many other packages utilize these under the hood

Examples:
/work/00157/walling/wrangler/training/MSU-August-2016

Multicore

- Utilizes multiple processing core within the same node.
- Replace several common functions with parallel implementations
- No need of significant changes of existing code.
- Scalability is limited by the number of core and memory available within single node

Multicore - mcapply

- lapply → mclapply
  - lapply(1:30, rnorm)
  - mclapply(1:30, rnorm)
- mc.cores
  - The maximum number of cores to use
- mc.preschedule
  - TRUE, computation is first divided by the number of cores.
  - FALSE, one job is spawned for each value sequentially

Multicore - mcapply

```
login2.stampede(179)$ cat Multicore.R
size <- 10000

# Single core
ptm <- proc.time()
data <- lapply(1:size, rnorm)
proc.time() - ptm

# Multiple cores
library(parallel)
ptm <- proc.time()
data <- mclapply(1:size, rnorm, mc.cores=16)
proc.time() - ptm
```

Multicore - mcapply

```
c557-602.stampede(1)$ Rscript Multicore.R
   user  system elapsed
  4.154   0.158   4.328
   user  system elapsed
  0.129   0.432   0.996
```

Multicore

- For loops are natural candidates to replace with mclapply.
- Each iteration of the loop should be ‘embarrassingly parallel’, i.e. doesn’t depend on any of the other iterations.
- Examples:
  - Randomly compute a statistic 100k times and test the resulting distribution.
  - Process each file in a large directory structure

Multicore - mcapply

```
wrangler$> cat Multicore-CLT.R

set.seed(1) # Reproducible randomness
population <- (1:1000000)
B <- 10000
n <- 1000

# For loop
print('for loop')
ptm <- proc.time()
result_loop = numeric(N) # initialize vector
for(i in 1:B) {
  sample = sample(x=population, size=n)
  result_loop[i] = mean(sample)
}
proc.time() - ptm

# Print resulting distribution, should be
'normal'
hist(result_loop)
```

```
# *apply family
print('apply')
ptm <- proc.time()
result_apply <- sapply(1:B, function(i) {
  mean(sample(x=population, size=n))
})
proc.time() - ptm
hist(result_apply)

# multicore - 16 cores
library(parallel)
print('multicore')
ptm <- proc.time()
result_16mc <- mclapply(1:B, function(i) {
  mean(sample(x=population, size=n))
}, mc.cores=2)
proc.time() - ptm
hist(unlist(result_16mc))
```

Multicore - mcapply

```
c251-141.wrangler(6)$ Rscript Multicore-CLT.R
[1] "for loop"
   user  system elapsed
  4.627   5.021   9.639
[1] "apply"
   user  system elapsed
  4.798   5.113   9.902
[1] "Multicore"
   user  system elapsed
  0.007   0.011   1.403
```

Snow/Rmpi

- Developed Based on Rmpi package, but also supports socket connections.
- Simplify the process to initialize parallel process over cluster.
- While sockets are possible, must use a dirty hack to properly set your environment
- Recommend using RMPISNOW instead to launch job
- Stampede: Must use Rstats 3.0.3 under Intel14.0.1.106 (Mvapich Bug w/ 3.2.1)

Snow/Rmpi

```
wrangler$ cat SimpleSNOW.R
library(Rmpi)
library(snow)

cluster <- getMPIcluster()

# Print the hostname for each cluster member
sayhello <- function()
{
      info <- Sys.info()[c("nodename", "machine")]
      paste("Hello from", info[1], "with CPU type", info[2])
}

names <- clusterCall(cluster, sayhello)
print(unlist(names))

# Stop cluster will also
# call mpi finalize
# no need for mpi.exit
stopCluster(cluster)
```

Snow/Rmpi

```
wrangler$ cat run-SNOW_Rmpi.slurm
#!/bin/bash
#SBATCH -J SNOW_Rmpi-R
#SBATCH -o SNOW_Rmpi.out%j
#SBATCH -p normal
#SBATCH -t 00:30:00
#SBATCH -A TRAINING-HPC
#SBATCH -e SNOW_Rmpi.err%j
#SBATCH -N 2
#SBATCH -n 10

#set environment
module purge
module load TACC
echo "say hello"
ibrun  RMPISNOW < SimpleSnow.R
echo "done"
```

Snow/Rmpi

```
wrangler$ sbatch run-SNOW_Rmpi.slurm

wrangler(246)$ squeue -u train###
             JOBID   PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           5867739      normal SNOW_Rmp  walling CG       0:13      2 c402-[301-302]

wrangler(389)$ cat SNOW_Rmpi.out5867843
say hello
TACC: Starting up job 5867843
TACC: Setting up parallel environment for MVAPICH2+mpispawn.
TACC: Starting parallel tasks...
……
> print(unlist(names))
[1] "Hello from c401-803.stampede.tacc.utexas.edu with CPU type x86_64"
[2] "Hello from c401-803.stampede.tacc.utexas.edu with CPU type x86_64"
[3] "Hello from c401-803.stampede.tacc.utexas.edu with CPU type x86_64"
….
[15] "Hello from c401-803.stampede.tacc.utexas.edu with CPU type x86_64"
[16] "Hello from c401-902.stampede.tacc.utexas.edu with CPU type x86_64"
[17] "Hello from c401-902.stampede.tacc.utexas.edu with CPU type x86_64"
……
```

Recommend Packages

- data.table
  - Data.frames implement in C, very fast
  - Supports indexing, fast joins
  - Fread provides very fast import from file to data.table structure
- Rcpp
  - Write C in R for slow portions of code
  - Optionally reference existing C code
- BigMemory
  - On disk matrices processed in strides
- pbdR
  - Various tools for large data developed out of PSC
  - http://r-pbd.org
- Others
  - http://cran.r-project.org/web/views/HighPerformanceComputing.html

David Walling

walling@tacc.utexas.edu