HPC with R

Jiyan Jonsdotter
Betreuer: Julian Kunkel

Structure

- Introduction to R
- Packages
- Code samples
- Conclusion

Introduction to R

- General Purpose Programming Language
- Designed to fit for statistics
  - Statistic functions (high level operations)
  - Plots
  - Used for Bioconductor
- Features
  - High abstraction
  - Interactive programming
  - High Performance (C-Modules)
  - Extendable through packages
- Community develops new packages

Packages

- Abstract support for parallelism
  - Functions available to work with any parallelism (cluster, multicore, etc. …)
  - snow
  - snowfall
  - foreach
- MPI
- No OpenMP at the moment (romp)
- Support for OpenCL
- Support for Hadoop (MapReduce)
- Support for GPUs (f.e. CUDA)

HPC in R

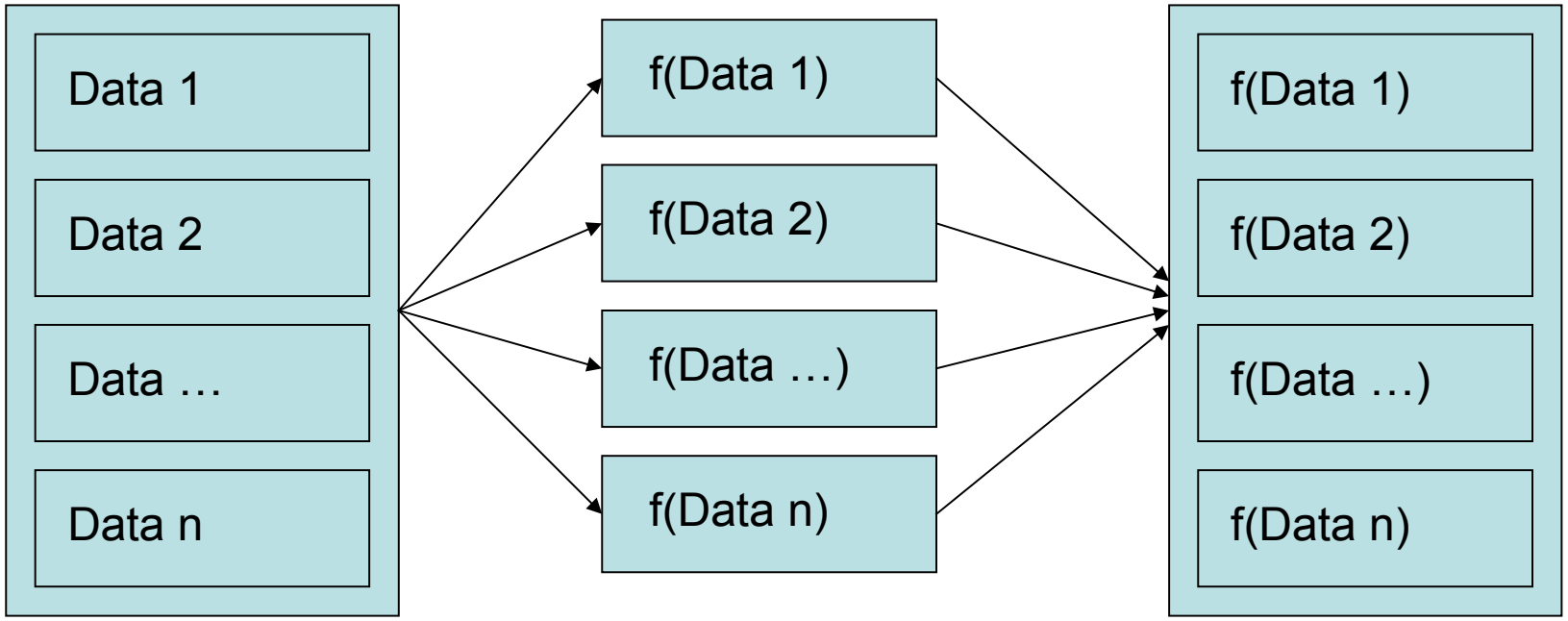
Parallel since version 2.1.4 (seems since 31.10.11) – Version today: 3.2.3

Shared	**Mixed**	**Distributed**
(romp)	snow	rmpi
	snowfall	pbdmpi
doMC	doSNOW	doMPI
	rlecuyer	
	doRNG	

Snow

- Easy to start with
- Abstraction from parallelism
  - makeCluster
  - stopCluster
- On top of a technology stack:
  - MPI
  - Sockets
  - NWS (NetWorkSpaces)
  - PVM (outdated)
- Access through apply functions

Snow – How it works



Snow

- makeCluster()
- stopCluster()
- clusterCall(cl,fun,…)
- clusterApply(cl,x,fun,…)
- clusterMap(cl,fun,…)

Rmpi

- Bases on MPI
- Functions inspired from MPI-Interface
- Execution via

  mpiexec –np 2 Rscript main.r

  or through interactive access
- Mightier than snow

Rmpi

- mpi.send()
- mpi.recv()
- mpi.comm.size()
- mpi.comm.rank()
- mpi.barrier()
- mpi.comm.spawn()

pbdMPI

- Classes for MPI
- Execution via mpiexec
- A lot of functions like in Rmpi
- Not interactive
- Not master/slave
- No spawning required

Snowfall

- Similar to snow
- But does not need parallelism
- sfInit()-call decides parallelism
- sfClusterApply : clusterApply

Snowfall

- sfInit(parallel,cpus)
- sfStop()
- sfClusterCall(fun,…)
- sfClusterApply(x,fun,…)
- sfClusterMap(fun,…)

Foreach

- Package for iterating over collections
- Designed to be sequentiell or parallel
- Parallel backends

Foreach – infix operator

- Backends defined over infix %between% operator
- What is better, a+b or +(a,b)?
- Similar to operator overloading
  - C++
  - Java Strings (+ operation)

Foreach

- doMC, doSNOW, doMPI
- registerDoMC() works with fork()-Call
- Backend gets used by function call

Rlecuyer

- For random numbers
- Based on a C++ class
- Internal state with 6 integer values
- Internal state defines next state and produces random value
- Call via runif()
- Use n cores and n streams

Rlecuyer

- .lec.CreateStram(names)
- .lec.AdvanceState()
- .lec.DeleteStream(names)
- .lec.GetState()
- .lec.IncreasedPrecis()
- .lec.SetSeed(name,seed)

doRNG

- Random number Generation for foreach
- Reproducible
- Important if you want to follow the steps in a simulation
- General problem of period of numbers

Inline

- Possible to execute C code from R
- Rcpp and RInside for R in C code
- So R code is mixable with MPI

Benchmark

- Packet benchmark
- benchmark(f1,f2) executes the given functions 100 times
- Gives relative running times

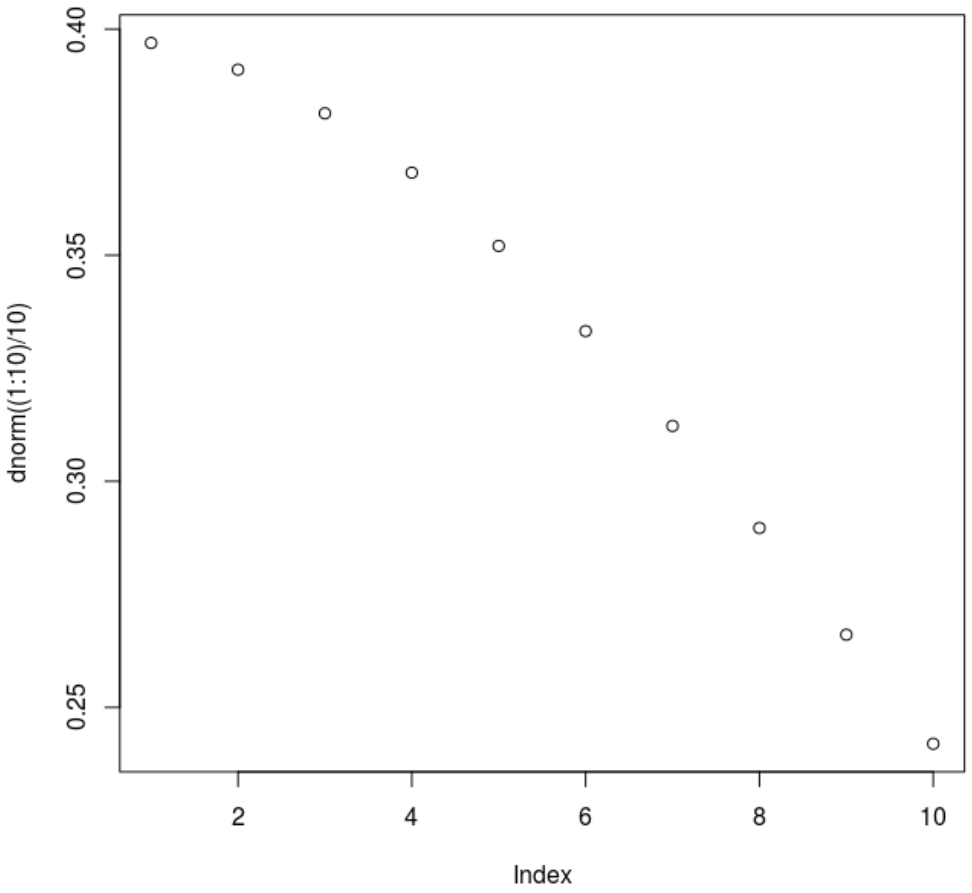
Code Samples

- General sample
- Snow
- Snowfall
- Rmpi
- pbdMPI
- Foreach
- Infix Operator
- Rlecuyer
- doRNG

Code Example

```
x <- 1:10

y <- x/10

randomVal <- runif(1)

Val1 <- dnorm(-1) – dnorm(1)

Val2 <- pnorm(1) + pnorm(1)

plot(dnorm((1:10)/10))
```

plot(dnorm((1:10)/10))



Snow

```
cl <- makeSOCKCluster(c(„localhost“,
“localhost“))
f <- function(x){x**2}
clusterApply(cl,1:2,f)
stopCluster(cl)
```

Snowfall

```
sfInit(parallel=TRUE,cpus=2)
f <- function(x){x**2}
sfClusterApply(1:2,f)
sfStop()
```

Rmpi

```
mpi.comm.spawn(1) # 1 slave node
if(mpi.comm.rank(0) == 0)
    # master node
else
    #do send or recv
mpi.finalize()
```

pbdMPI

```
suppressMessages(library("pbdMPI"))
init()

.comm.size <- comm.size()
.comm.rank <- comm.rank()

if(.comm.rank==0)
{
y <- send(matrix((1:100), nrow = 1))
print(y)
}
if(.comm.rank == 1){
y <- recv()
}

finalize()
```

Foreach

```
foreach(i=1:10) %do%{i**2}
# wont be doing anything without backend
foreach(i=1:10) %dopar%{i**2}
library(„doMC“)
registerDoMC(2) # now it works
foreach(i=1:10) %dopar%{i**2}
```

Infix Operator

```
`%add%`<-

function(x,y) x+y

1 %add% 2

# gives back 3
```

Rlecuyer

.lec.CreateStream(„random“)

runif() # random value

. lec.DeleteStream(„random“)

doRNG

```
set.seed(200)
foreach(i=1:3) %dopar% {runif()}
set.seed(200)
foreach(i=1:3) %dopar% {runif()}
#won‘t be identical
```

doRNG

```
set.seed(200)
foreach(i=1:3) %dorng% {runif()}
set.seed(200)
foreach(i=1:3) %dorng% {runif()}
#will be identical / reproducible
```

Conclusion

- Support for MPI but not (yet) OpenMP
- Support for parallel backends in code (foreach and snow)
- Random number generation
- PVM not supported any longer (still available in the archive)

References

- https://www.sharcnet.ca/help/index.php/Using_R_and_MPI
- https://en.wikipedia.org/wiki/Programming_with_Big_Data_in_R
- https://de.wikipedia.org/wiki/R_%28Programmiersprache%29
- https://cran.r-project.org/web/views/HighPerformanceComputing.html

References

- http://www.iro.umontreal.ca/~lecuyer/myftp/papers/streams00.pdf