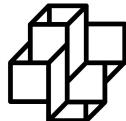


Escola Supercomputador



O R EM AMBIENTE HPC



National Laboratory for Scientific Computing

L|N|C|C

Laboratório Nacional de Computação Científica - LNCC
Professores: Raquel L. Costa (quelopes@lncc.br)
Guilherme Gall (gmgall@lncc.br)

Agosto de 2018 (2º edição)

SUMÁRIO

- Introdução ao R
 - ◆ O que é o R?
 - ◆ Por que usar R? E por que utiliza-lo em ambientes HPC?
- Carregar o módulo R no supercomputador (Santos Dumont, SD)
 - ◆ Instalar um pacote R no SD
 - ◆ Submetendo um script para o SD
- Família apply
- Pacotes de paralelismo no R
 - ◆ Análogos à família apply (parallel, snow)
 - ◆ foreach e seus backends paralelos (foreach, doParallel)
- Benchmark no R
- Estudo de caso 1: Exemplo k-means
- Melhorando a eficiência do código R
- Estudo de caso 2: Model-R

O QUE É O R?

O QUE É O R?

- Desenvolvido a partir da linguagem S pelos estatísticos **Ross Ihaka** e **Robert Gentleman** da Universidade de Auckland (Nova Zelândia) em 1995
- Objetivo inicial era desenvolver um programa estatístico de domínio público, ou seja, com o código aberto e disponível para toda comunidade (Free Software Foundation's GNU General Public License - GPL)
- Atualmente, o R é muito mais do que um ambiente estatístico...
 - ◆ Uma das linguagens mais utilizadas em Big Data
- Enquanto programas estatísticos fornecem um resultado de uma regressão por exemplo, R fornece uma saída mínima e armazena o resultado em um objeto que pode ser integrado a outras funções
 - ◆ Encadeamento de tarefas

POR QUE USAR O R?

Além da vantagem de **software livre**, o R também apresenta:

- Multiplataforma (Linux, macOS, Windows, ...)

- Manipulação de dados eficaz e facilidade de armazenamento

- Uma série de operadores para cálculos com arranjos, especialmente matrizes

- Extensa, coerente e integrada coleção de ferramentas intermediárias para análise de dados

- Instalações gráficas para análises de dados e exibição tanto direta no computador quanto para cópia permanente (impressões)

- Linguagem que inclui condições, *Loops*, funções recursivas definidas pelo usuário e instalações de entradas e saídas

- Possibilidade de criar e compartilhar pacotes

R NO TERMINAL

```
quelopes@quelopes-Inspiron-5458: ~/Downloads/JabotMSTR
main.tex:193 Encoding '/tmp/Consulta_10_-_Bioma_e_vegetacao_por_agregacao.png'

quelopes@quelopes-Inspiron-5458:~/Downloads/JabotMSTR$ R
  
R version 3.4.2 (2017-09-28) -- "Short Summer"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

Acessando R via terminal. Janela de comando ou console.

- Os comandos em R podem ser digitados após o *prompt* de comando **>**
- Para finalizar uma sessão utiliza-se a função **q()** 'quit'

INSTALAÇÃO DO AMBIENTE INTEGRADO DO R, O RSTUDIO

→ O RStudio consiste numa ambiente integrado de desenvolvimento

- ◆ Inclui um console
- ◆ Editor de *syntax-highlighting*
- ◆ Ferramentas para gráficos
- ◆ *Debugging*
- ◆ Gerenciador de *workspace*
- ◆ Outros

O RSTUDIO

Editor de código

The screenshot shows the RStudio interface with several panes:

- Editor de código (Code Editor):** Contains an R script named `Parameters.R` with code related to gene expression analysis.
- Console:** Displays the R command-line interface with the current working directory set to `~/Development-house/ProgramaVeroINCA-2018/`. It shows the execution of the script and some initial setup messages.
- Workspace:** Shows the global environment with various objects defined, such as `dirR_raw`, `GPL`, `method`, `nameExp`, `orgAnnot`, `overallDesign`, `platAccess`, and `platname`.
- Files:** Shows the project structure with files like `Annotation`, `Dockerfile`, `GNetShiny`, `GSE62232`, `import.cql`, `Module-A`, `Parameters.R`, `README.md`, and `shiny-server.sh`.

Console

Gráficos e arquivos

Exemplo do ambiente RStudio.

INSTALAÇÃO DE PACOTES

- Uma outra vantagem do R é o uso de pacotes que o torna altamente extensível
- Pacotes são bibliotecas para funções gerais ou áreas de estudo específicas
- Um conjunto de pacotes é incluído com a instalação do R
- Outros pacotes estão disponíveis em repositórios de pacotes como o CRAN (The Comprehensive R Archive Network, CRAN), Bioconductor ou mesmo no github



REPOSITÓRIO DE PACOTES DO R

The screenshot shows the CRAN homepage with a sidebar on the left containing links for GRAN, Mirrors, What's new?, Task Views, Search, About R, R Homepage, The R Journal, Software, R Sources, R Binaries, Packages, Other, Documentation, Manuals, FAQs, and Contributed. The main content area is titled "The Comprehensive R Archive Network" and includes sections for "Download and Install R", "Source Code for all Platforms", "Questions About R", "What are R and CRAN?", "Submitting to CRAN", and "Note".

Página inicial do CRAN.

The screenshot shows the Bioconductor homepage with a navigation bar at the top for Home, Install, Help, Developers, and About. The main content area features a section for the "International Workshop on Bioinformatics" with details about the location (Port of Spain, Trinidad), dates (2011-01-19 ~ 2011-01-21), instructors (Martin Morgan, Vincent Carey, others), a description of the 3-day course, materials (presentations and lab exercises), and a code block for installing the IW2011 package. A sidebar on the right lists packages, documentation, and other resources.

Página inicial do Bioconductor.

O PACOTE DEVTOOLS

O pacote **devtools** não apenas facilita o processo para desenvolver pacotes R, mas também fornece outra maneira de distribuir pacotes R.

Mac/Linux:

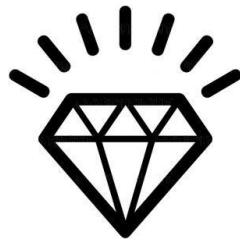
```
devtools::install_github("hadley/devtools")
```

Windows:

```
library(devtools)
build_github_devtools()
#### Restart R before continuing ####
install.packages("devtools.zip", repos = NULL)
```

```
# Remove the package after installation
unlink("devtools.zip")
```

ANTES DE COMEÇARMOS... DICAS PRECIOSAS



- As funções em R são sempre acompanhadas por parênteses `()`
- Verifique seu diretório atual e se necessário altere-o
 - ◆ `getwd()`
 - ◆ `setwd('/home/aluno/Dados')`
- Em caso de dúvida `help('sqrt')` ou `?sqrt` ou `help.search('sqrt')`
- O buscador <http://www.rseek.org/> restringe a busca para os sites que possuem conteúdo relacionado apenas à linguagem R

POR QUE USAR O R EM HPC ?

POR QUE USAR R?

O conhecimento sobre MPI, SOCKET, C, C++ e Fortran podem ser **barreiras** para grande parte dos estatísticos que lidam com cálculos estatísticos intensivos e estão tentando acelerar os cálculos implementando algoritmos paralelos.

POR QUE USAR R?

- Muitos problemas de análise computacional estatística envolvem avançados algoritmos com conjunto de dados (*datasets*) com grande número de parâmetros necessários para ser estimado.
 - ◆ Análise de sequência de DNA em bioinformática
 - ◆ Bootstrap
 - ◆ Redes neurais
 - ◆ Validação cruzada (*cross-validation*)

CRAN: MATERIAL SOBRE R HPC, BIG DATA, ...

CRAN Task View: High-Performance and Parallel Computing with R

Maintainer: Dirk Eddelbuettel

Contact: [Dirk.Eddelbuettel at R-project.org](mailto:Dirk.Eddelbuettel@R-project.org)

Version: 2018-02-07

URL: <https://CRAN.R-project.org/view=HighPerformanceComputing>

This CRAN task view contains a list of packages, grouped by topic, that are useful for high-performance computing (HPC) with R. In this context, we are defining 'high-performance computing' rather loosely as just about anything related to pushing R a little further: using compiled code, parallel computing (in both explicit and implicit modes), working with large objects as well as profiling.

Unless otherwise mentioned, all packages presented with hyperlinks are available from CRAN, the Comprehensive R Archive Network.

Several of the areas discussed in this Task View are undergoing rapid change. Please send suggestions for additions and extensions for this task view to the [task view maintainer](#).

Suggestions and corrections by Achim Zeileis, Markus Schmidberger, Martin Morgan, Max Kuhn, Tomas Radovoyevitch, Jochen Knaus, Tobias Verbeke, Hao Yu, David Rosenberg, Marco Enea, Ivo Welch, Jay Emerson, Wei-Chen Chen, Bill Cleveland, Ross Boylan, Ramon Diaz-Uriarte, Mark Zeligman, Kevin Ushey, Graham Jeffries, Will Landau, and Tim Flutre (as well as others I may have forgotten to add here) are gratefully acknowledged.

Contributions are always welcome, and encouraged. Since the start of this CRAN task view in October 2008, most contributions have arrived as email suggestions. The source file for this particular task view file now also reside in a GitHub repository (see below) so that pull requests are also possible.

The `cvt` package supports these Task Views. Its functions `install.views` and `update.views` allow, respectively, installation or update of packages from a given Task View; the option `coreOnly` can restrict operations to packages labeled as *core* below.

Direct support in R started with release 2.14.0 which includes a new package `parallel` incorporating (slightly revised) copies of packages `multicore` and `snow`.
Some types of clusters are not handled directly by the base package 'parallel'. However, and as explained in the package vignette, the parts of parallel which

Escola Supercomputador



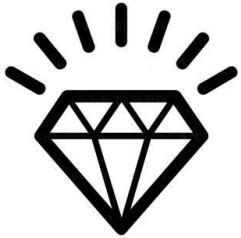
UTILIZANDO O R NO SANTOS DUMONT

CARREGAR O MÓDULO R NO SANTOS DUMONT

```
$ source /scratch/app/modulos/intel-psxe-2016.sh (PSXE 2016) ou  
$ source /scratch/app/modulos/intel-psxe-2017.sh (PSXE 2017)  
$ module load openmpi/1.10_intel  
$ module load R/3.3.1_intel
```

É com esses módulos que os exemplos estão funcionando, mas existem outros módulos e outras versões.

Use `$ module avail` para ver todos os módulos disponíveis.



\$SCRATCH

IMPORTANTE!!! DIRECIONAR PARA O SCRATCH

CD \$SCRATCH

PACOTES INSTALADOS GLOBALMENTE

- Vários pacotes já encontram-se instalados, inclusive os comentados no minicurso.
- Alguns deles:
 - `bigmemory`
 - `doMPI`
 - `doParallel`
 - `doSNOW`
 - `foreach`
 - `microbenchmark`
 - `parallel`
 - `rbenchmark`
 - `snow`
 - `Snowfall`
- Para ver a lista completa, use a função `installed.packages()`

INSTALAÇÃO DE PACOTES NO SANTOS DUMONT

Se desejar usar um pacote não instalado, é possível fazer uma instalação local de um pacote da seguinte maneira:

1. Crie um diretório no scratch para manter os pacotes
 - `mkdir /scratch/projeto/usuario/R`
2. Instale os pacotes no diretório criado
 - `R -e "install.packages('raster', repos='http://cran.rstudio.com/', lib='/scratch/projeto/usuario/R')"`
3. Use a variável de ambiente R_LIBS_USER para indicar o path do diretório com os pacotes
 - `export R_LIBS_USER='/scratch/projeto/usuario/R'`

INSTALAÇÃO DE PACOTES NO SANTOS DUMONT

Da documentação oficial:

The library search path is initialized at startup from the environment variable ‘R_LIBS’ (which should be a colon-separated list of directories at which R library trees are rooted) followed by those in environment variable ‘R_LIBS_USER’. Only directories which exist at the time will be included.

INSTALAÇÃO DE PACOTES NO SANTOS DUMONT

- Também é possível modificar `.libPaths()` manualmente dentro de um script:

```
>.libPaths(c('/scratch/projeto/usuario/minha_library', .libPaths()))
```

- Útil para testar outras versões de pacotes já instalados, sem perder a instalação anterior.

FAMÍLIA APPLY

FUNÇÃO APPLY

- Usada para aplicar uma função às linhas ou colunas de uma matriz
- O retorno é um vetor ou array
- Exemplo

```
> M <- matrix(seq(1,16), 4, 4)
```

```
> M
```

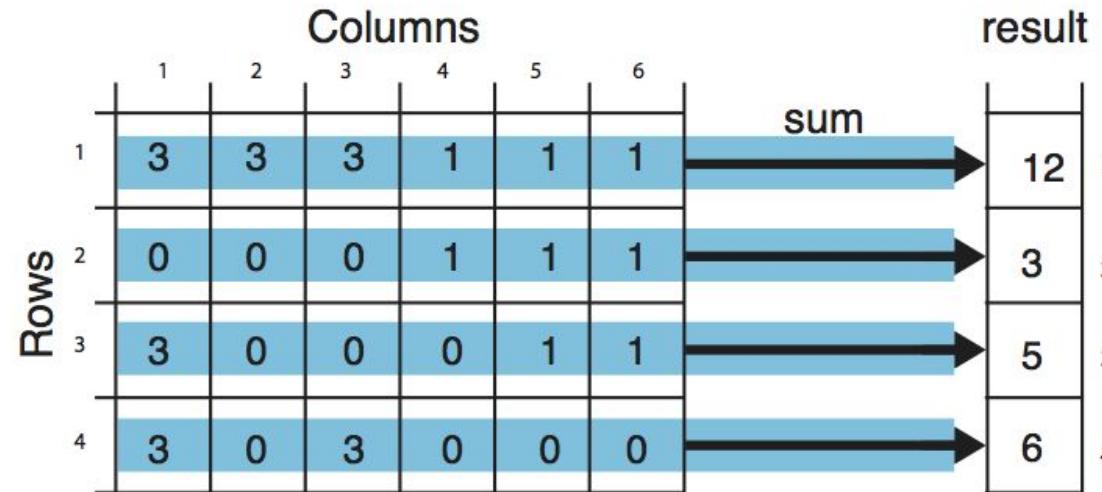
	[,1]	[,2]	[,3]	[,4]
[1,]	1	5	9	13
[2,]	2	6	10	14
[3,]	3	7	11	15
[4,]	4	8	12	16

FUNÇÃO APPLY

```
> # Aplica min() às linhas  
> apply(M, 1, min)  
[1] 1 2 3 4  
  
> # Aplica max() às colunas  
> apply(M, 2, max)  
[1] 4 8 12 16
```

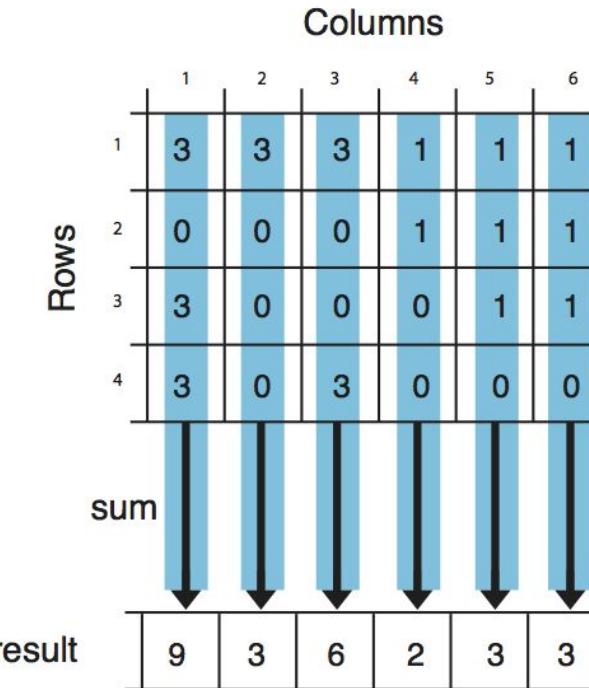
FUNÇÃO APPLY

```
result<-apply(mat,1,function(x) sum(x) )  
resut<-apply(mat,1,sum)
```



FUNÇÃO APPLY

```
result<-apply(mat,2,function(x) sum(x))  
result<-apply(mat,2,sum)
```



FUNÇÃO APPLY

- Execute `apply.R` para ver `apply()` em ação.
- Gere uma matriz 100 por 100 de números aleatórios e encontre o menor valor de cada linha e de cada coluna.
- Dica: funções `runif()` e `min()`.

FUNÇÃO LAPPLY

- Aplica uma função à cada elemento de uma lista
- **Retorna sempre uma lista**
- Exemplo

```
> X <- list(a = 1, b = 1:10, c = c("Olá", "Mundo"))
```

```
> X
```

```
$a
```

```
[1] 1
```

```
$b
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
$c
```

```
[1] "Olá"    "Mundo"
```

FUNÇÃO LAPPLY

```
> # Aplica a função length() a cada elemento  
> lapply(X, FUN = length)
```

```
$a
```

```
[1] 1
```

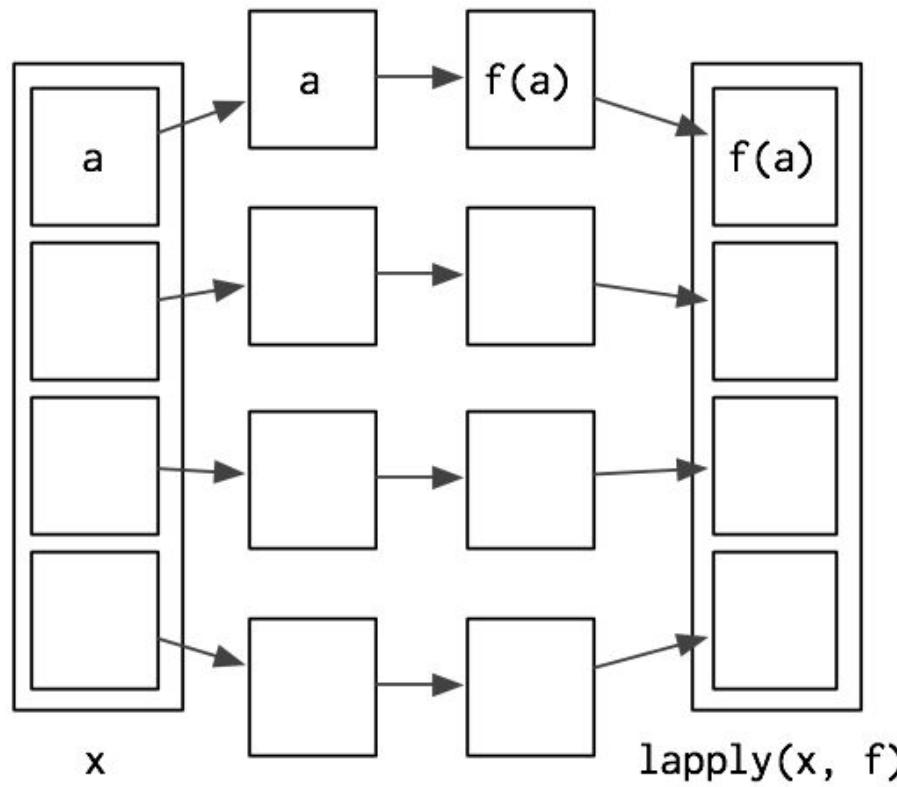
```
$b
```

```
[1] 10
```

```
$c
```

```
[1] 2
```

FUNÇÃO LAPPLY



Do livro [Advanced R](#) de Hadley Wickham

FUNÇÃO LAPPLY

- Veja lapply() em ação em **lapply.R**
- Gere uma lista com 3 elementos...
 - 10 números aleatórios
 - uma matriz 4 x 4
 - Uma string
- e gere uma lista de 3 elementos em que cada elemento terá as dimensões de cada um dos objetos listados acima

FUNÇÃO SPLY

- Aplica uma função à cada elemento de uma lista
- **Retorna um vetor ou matriz ao invés de uma lista**
- Exemplo

```
> X <- list(a = 1, b = 1:10, c = c("Olá", "Mundo"))
```

```
> X
```

```
$a
```

```
[1] 1
```

```
$b
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
$c
```

```
[1] "Olá"    "Mundo"
```

FUNÇÃO SPLY

```
> # Aplica a função length() a cada elemento  
> # retornando um vetor  
> sapply(X, FUN = length)  
a   b   c  
1 10   2
```

- Considere o uso de sapply() no lugar de unlist(lapply())

```
> unlist(lapply(X, FUN=length))  
a   b   c  
1 10   2
```

FUNÇÃO SAPPY

```
> # sapply() pode retornar uma matriz se a função aplicada  
> # retornar vetores de mesmo tamanho  
> estados <- function(x) sample(state.name, 3)  
  
> estados(1)  
[1] "Hawaii"      "South Dakota" "Montana"  
  
> sapply(1:4, FUN=estados)  
          [,1]          [,2]          [,3]          [,4]  
[1,] "Kentucky"    "Oregon"     "Nebraska"   "Oregon"  
[2,] "West Virginia" "Hawaii"    "Louisiana"  "Kansas"  
[3,] "Iowa"        "Minnesota"  "Oregon"     "Tennessee"
```

FUNÇÃO SAPPY

- Veja sapply() em ação em **sapply.R**
- Gere uma lista com 3 elementos...
 - 10 números aleatórios
 - uma matriz 4 x 4
 - Uma string
- e gere **um vetor** de 3 elementos em que cada elemento terá as dimensões de cada um dos objetos listados acima

FUNÇÃO MAPPLY

- `mapply()` é a versão multivariável de `sapply()`.
- Ela aplica a função aos primeiros elementos das estruturas de dados, depois aos segundos, aos terceiros e assim por diante.
- A função deve aceitar múltiplos argumentos.
- **Retorna vetor ou matriz sempre que possível.**
- **Se não, retorna lista.**

FUNÇÃO MAPPLY

```
> # Retorna um vetor de 5 elementos com  
> # sum(1, 5, 1) na 1a posição,  
> # sum(2, 4, 2) na 2a posição,  
> # sum(3, 3, 3) na 3a posição,  
> # sum(4, 2, 4) na 4a posição e  
> # sum(5, 1, 5) na 5a posição.  
  
> mapply(sum, 1:5, 5:1, 1:5)  
[1] 7 8 9 10 11
```

FUNÇÃO MAPPLY

```
> # Faz rep(1, 4), rep(2, 3), rep(3, 2), rep(4, 1)
> # Nesse caso, só é possível retornar uma lista
> mapply(rep, 1:4, 4:1)
[[1]]
[1] 1 1 1 1

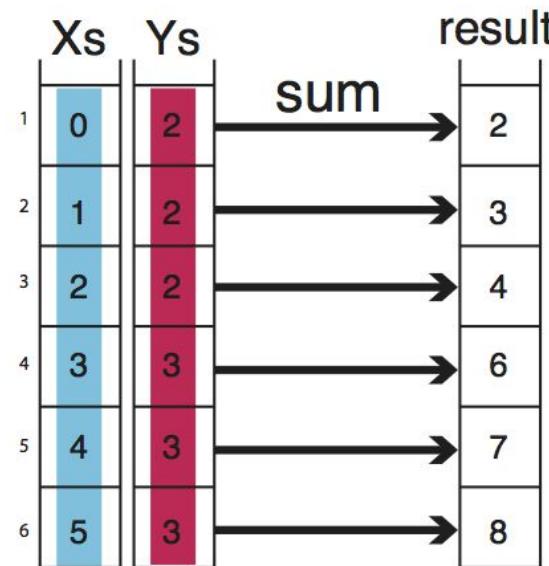
[[2]]
[1] 2 2 2

[[3]]
[1] 3 3

[[4]]
[1] 4
```

FUNÇÃO MAPPLY

```
result<-mapply(function(x,y) sum(x,y),Xs,Ys)
                result<-mapply(sum,Xs,Ys)
```



FUNÇÃO MAPPLY

- Veja `mapply()` em ação em [mapply.R](#)
- Gere 2 vetores
 - X: um com números de 1 a 10
 - Y: um com números de 10 a 1
- e gere **um vetor** de 10 elementos em que cada elemento será o resultado de $X[1]^Y[1]$, $X[2]^Y[2]$... $X[10]^Y[10]$

PACOTES DE PARALELISMO

PACOTE PARALLEL

- Disponível na instalação base do R desde a versão 2.14.0.
- Baseado nos pacotes **snow** e **multicore**.
- Fornece substitutos para a maioria das funções desses pacotes.

PACOTE PARALLEL

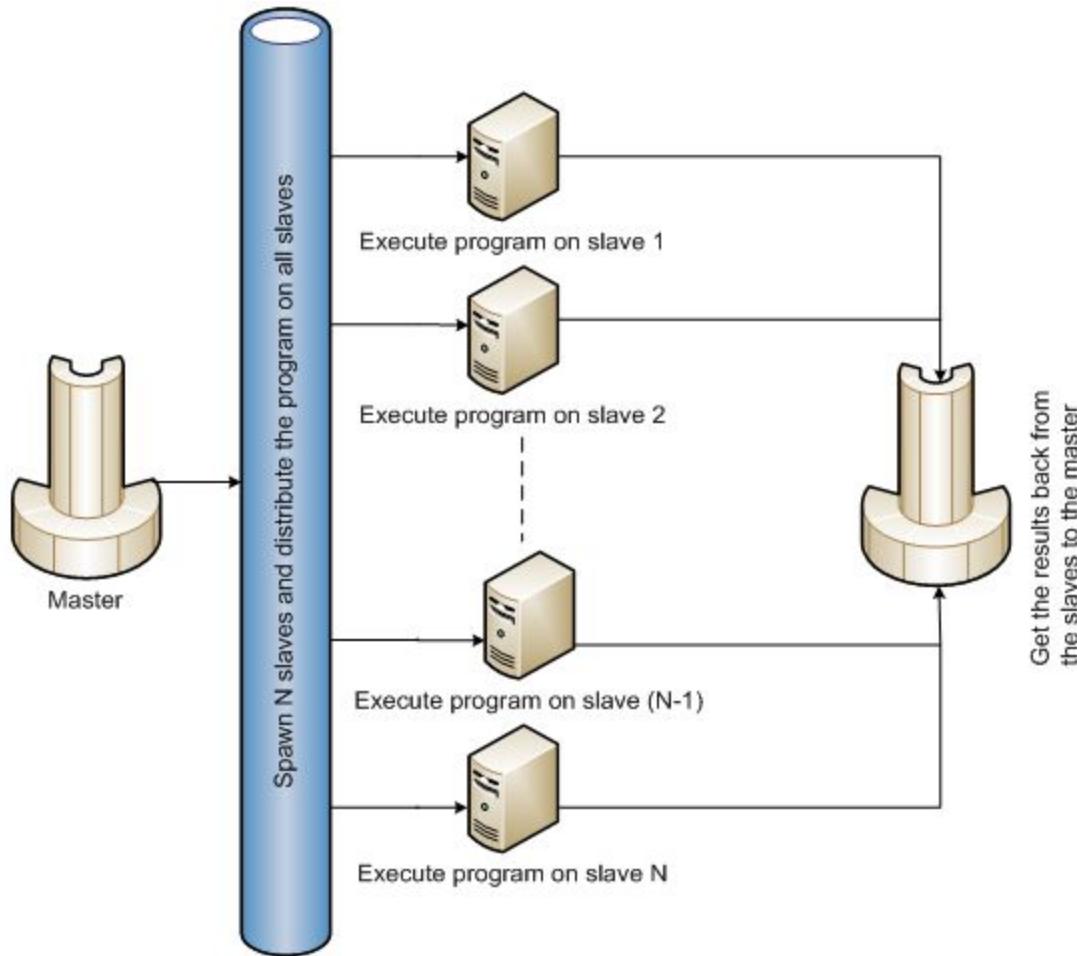
O processamento com o pacote parallel segue o seguinte modelo:

- a. M processos *workers* são iniciados
- b. Os dados necessários para completar a tarefa são enviados para cada *worker*
- c. A tarefa é dividida em M pedaços de tamanho aproximadamente iguais e eles são enviados para os *workers*
- d. Espera-se até que todos os *workers* completem suas tarefas
- e. Os passos b-d são repetidos para as tarefas seguintes
- f. Os processos *workers* são terminados

PACOTE PARALLEL

- Esse modelo é implementado em funções que são as contrapartes paralelas para as funções da família `apply`, possibilitando o processamento de cada elemento do vetor/lista de entrada em um *core* diferente.
- Um modelo ligeiramente diferente é dividir a tarefa em $M_1 > M$ pedaços, enviar os primeiros M pedaços para os *workers* e repetidamente aguardar um *worker* terminar e enviar o pedaço restante.
 - “Balanceamento de carga” (*load balancing*)

PACOTE PARALLEL



PACOTE PARALLEL

- Para iniciar os processos *workers*, use a função `makeCluster()`
 - Ela recebe por parâmetro o número de *workers* e o tipo de comunicação entre eles
- Para encerrar os *workers*, use a função `stopCluster()`
 - Ela recebe por parâmetro o objeto criado via `makeCluster()`

PACOTE PARALLEL

```
# Cria um cluster de 4 workers  
# com comunicação MPI  
cl <- makeCluster(4, type = "MPI")  
  
# Faz trabalho em paralelo  
  
# Para os workers  
stopCluster(cl)
```

FUNÇÃO PARAPPLY

- parApply() é análoga à apply()
- A diferença no uso é o 1º argumento: ele deve ser o objeto cluster criado via makeCluster()
- parRapply() é oferecida como conveniência: ela opera automaticamente nas **linhas**
- parCapply() é oferecida como conveniência: ela opera automaticamente nas **colunas**
- A documentação cita que parRapply() e parCapply() podem ser mais eficientes que parApply() em algumas situações

FUNÇÃO PARAPPLY

- Execute `parApply.R` para ver `parApply()` em ação.
- Gere uma matriz 100 por 100 de números aleatórios e encontre o menor valor de cada linha e de cada coluna.
- Dica: funções `runif()` e `min()`.
- Faça isso num cluster de 10 *workers*.

FUNÇÃO PARLAPPLY

- `parLapply()` é análoga à `lapply()`
- Também deve receber o objeto cluster criado via `makeCluster()` via 1º parâmetro
- **Retorna sempre uma lista**
- Possui uma versão com balanceamento de carga:
`parLapplyLB()`
 - Use-a quando aplicar a função a diferentes elementos da lista de entrada levar tempos muito diferentes
 - Adiciona *overhead* de comunicação

FUNÇÃO PARLAPPLY

- Veja `parLapply()` em ação em `parLapply.R`
- Gere uma lista com 3 elementos...
 - 10 números aleatórios
 - uma matriz 4 x 4
 - Uma string
- e gere uma lista de 3 elementos em que cada elemento terá as dimensões de cada um dos objetos listados acima.
- Faça isso num cluster de 3 *workers*.

FUNÇÃO PAR\\$APPLY

- parSapply() é análoga à sapply()
- Também deve receber o objeto cluster criado via makeCluster() via 1º parâmetro
- **Retorna um vetor ou matriz**
- Possui uma versão com balanceamento de carga:
parSapplyLB()
 - Use-a quando aplicar a função a diferentes elementos da lista de entrada levar tempos muito diferentes
 - Adiciona *overhead* de comunicação

FUNÇÃO PAR\\$APPLY

- Veja `parSapply()` em ação em `parSapply.R`
- Gere uma lista com 3 elementos...
 - 10 números aleatórios
 - Uma matriz 4 x 4
 - Uma string
- Gere **um vetor** de 3 elementos em que cada elemento terá as dimensões de cada um dos objetos listados acima
- Faça isso num cluster de 3 *workers*.

FUNÇÃO CLUSTERMAP

- clusterMap() é análoga à mapply()
- Também deve receber o objeto cluster criado via makeCluster() via 1º parâmetro.
- A função chamada deve receber múltiplos argumentos.
- **Retorna sempre uma lista, ao contrário de mapply()**

FUNÇÃO CLUSTERMAP

- Veja `clusterMap()` em ação em `clusterMap.R`
- Gere 2 vetores
 - X: um com números de 1 a 10
 - Y: um com números de 10 a 1
- Gere **uma lista** de 10 elementos em que cada elemento será o resultado de $X[1]^Y[1]$, $X[2]^Y[2]$... $X[10]^Y[10]$
- Faça isso num cluster de 10 *workers*.

FUNÇÃO CLUSTEREXPORT

- `clusterExport()` faz a(s) variável(is) do mestre passadas por parâmetro disponíveis aos processos *workers*.
- Um vetor de strings com os nomes das variáveis a serem disponibilizadas devem ser passadas por parâmetro, não as variáveis em si.
- Exemplo

```
> expoente <- 3  
  
> clusterExport(cl, "expoente")
```

FUNÇÃO CLUSTEREXPORT

- Veja os efeitos de clusterExport() em `clusterExport.R`
- Comente a linha...

```
expoente <- 3
```

- e teste novamente. O que aconteceu? Por quê?

FUNÇÃO CLUSTERCALL

- `clusterCall()` chama uma função em cada *worker* do cluster, com argumentos idênticos.
- Os argumentos **são avaliados no mestre** e seus valores são transmitidos aos *workers* que executam a função.

FUNÇÃO CLUSTER EVALQ

- `clusterEvalQ()` avalia uma expressão literal em **cada worker do cluster**.
- É uma versão paralela de `evalq()`
- Disponibilizada como conveniência: chama `clusterCall(cl, evalq, expr)` para cada *worker*.
- Muito usada para carregar um pacote nos workers:
`clusterEvalQ(cl, library(PACOTE))`

FUNÇÕES CLUSTER`EVALQ` E CLUSTER`CALL`

- Execute `clusterEvalQ.R` para ver a diferença entre `clusterEvalQ()` e `clusterCall()`.

FUNÇÃO CLUSTERAPPLY

- `clusterApply()` recebe uma sequência de argumentos (vetor ou lista) e uma função.
- Chama a função com o 1º elemento da sequência como parâmetro da função no 1º *worker* do cluster, o 2º elemento da sequência como argumento da função no 2º *worker* do cluster e assim por diante.

FUNÇÃO CLUSTERAPPLY

- O tamanho da lista de argumentos deve ser menor ou igual à quantidade de *workers* no cluster.
 - Existe uma versão com balanceamento de carga, `clusterApplyLB()`, que não possui essa limitação.
- **Retorna sempre uma lista.**
- Muito parecida com `parLapply()`, mas não retorna uma lista nomeada.
- Exercício: substitua `parLapply()` por `clusterApply()` em `parLapply.R`

FUNÇÃO CLUSTERSPLIT

- `clusterSplit()` divide uma sequência em pedaços.
- Retorna um pedaço para cada *worker* em um cluster.
- Cada pedaço traz elementos consecutivos.
- **Retorna sempre uma lista.**
- Toda a computação é feita no mestre.

FUNÇÃO CLUSTERSPLIT

- Exemplo

```
> # cluster com 4 workers  
clusterSplit(cl, 1:8)  
[[1]]  
[1] 1 2
```

```
[[2]]  
[1] 3 4
```

```
[[3]]  
[1] 5 6
```

```
[[4]]  
[1] 7 8
```

FUNÇÃO CLUSTERSPLIT

- Usada para fatiar uma entrada em partes iguais a serem processadas pelos *workers*.
- É importante dividir seu trabalho de forma a reduzir o *overhead* de comunicação.
- Veja isso na prática em `clusterSplit.R`

FUNÇÃO DETECTCORES

- `detectCores()` tenta retornar a quantidade de cores da máquina.
- Pode ser útil para descobrir que quantidade de *workers* criar.
- Não muito útil em clusters.

FUNÇÃO CLUSTERSETRNGSTREAM

- `clusterSetRNGStream()` configura a seed do gerador de números aleatórios.
- Use para garantir reproduzibilidade em execuções em paralelo.
- Execute `clusterSetRNGStream.R` múltiplas vezes para testar.

CORRESPONDÊNCIA APPLY - PARALLEL

apply()	parApply()
apply(,1,)	parRapply()
apply(,2,)	parCapply()
lapply()	parLapply() / parLapplyLB()
sapply()	parSapply() / parSapplyLB()
mapply()	clusterMap()
evalq()	clusterEvalQ()
set.seed()	clusterSetRNGStream()

FOREACH

PACOTE FOREACH

- Oferece suporte a uma construção de loop que permite iterar sobre elementos de uma coleção, sem o uso de uma variável contadora explícita.
- Usado pelo seu **valor de retorno**, não pelos seus efeitos colaterais.
- Permite execuções em paralelo após o registro de um *backend* paralelo.

PACOTE FOREACH

Retorna um valor,
ao contrário do
for tradicional.

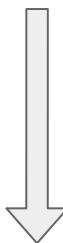
Por padrão,
retorna uma
lista

```
> x <- foreach(i=1:3) %do% sqrt(i)  
> x
```

```
[[1]]  
[1] 1  
  
[[2]]  
[1] 1.414214  
  
[[3]]  
[1] 1.732051
```

PACOTE FOREACH

É possível especificar uma função para combinar a saída



```
> x <- foreach(i=1:3, .combine='c') %do% exp(i)  
> x
```

```
[1] 2.718282 7.389056 20.085537
```

PACOTE FOREACH

Se a expressão retorna um vetor, é possível combiná-los numa matriz.



```
> x <- foreach(i=1:4, .combine='cbind') %do% rnorm(4)
> x
```

	result.1	result.2	result.3	result.4
[1,]	1.351555	0.6232773	1.3019642	0.4170242
[2,]	-1.032726	1.5233215	-1.6029630	-0.8573203
[3,]	1.399842	-0.1439332	0.1347221	0.2907462
[4,]	1.343903	1.7250600	1.4653007	0.2795696

PACOTE FOREACH

- Para fazer execuções em paralelo é necessário registrar um *backend* paralelo.
- Alguns disponíveis:
 - doMC
 - doSNOW
 - **doParallel**

PACOTE FOREACH

```
library(foreach)
library(doParallel)

cl<-makeCluster(...)
registerDoParallel(cl)

foreach(...) %dopar% {
  # instruções
}

stopCluster(cl)
```

A instrução muda
para **%dopar%**

PACOTE FOREACH

- Execute `doParallel.R` para ver `foreach()` em ação.
- Meça o tempo de execução de...

```
out <- foreach(x=a, y=b, z=c) %dopar% {  
  (x + y)**z  
}
```

- com a técnica empregada em `clusterSplit.R`
- Troque o operador `%dopar%` por `%do%` e meça novamente.

BENCHMARK NO R

EXEMPLO 1: FUNÇÃO SYS.TIME (R-BASE)

```
do_lm = function(value){  
  X <- matrix(rnorm(value), 100, 10)  
  y <- X %*% sample(1:10, 10) + rnorm(100)  
  b <- lm(y ~ X + 0)$coef  
}  
  
start_time <- Sys.time()  
do_lm(1000)  
end_time <- Sys.time()  
  
print(end_time - start_time)  
  
## Time difference of 0.00741148 secs
```

Execute o script **t1_bench_sys-time.R**

EXEMPLO 2: FUNÇÃO SYSTEM.TIME (R-BASE)

```
do_lm = function(value){  
  X <- matrix(rnorm(value), 100, 10)  
  y <- X %*% sample(1:10, 10) + rnorm(100)  
  b <- lm(y ~ X + 0)$coef  
}  
  
system.time({do_lm(1000)})  
  
##    user  system elapsed  
##  0.002   0.000   0.002
```

Execute o script **t2_bench_systemTime.R**

EXEMPLO 3: PACOTE RBENCHMARK

Execute o script
t3_bench_rbenchmark.R

```
library(rbenchmark)

benchmark("lm" = {
  X <- matrix(rnorm(1000), 100, 10)
  y <- X %*% sample(1:10, 10) + rnorm(100)
  b <- lm(y ~ X + 0)$coef
},
"pseudoinverse" = {
  X <- matrix(rnorm(1000), 100, 10)
  y <- X %*% sample(1:10, 10) + rnorm(100)
  b <- solve(t(X) %*% X) %*% t(X) %*% y
},
"linear system" = {
  X <- matrix(rnorm(1000), 100, 10)
  y <- X %*% sample(1:10, 10) + rnorm(100)
  b <- solve(t(X) %*% X, t(X) %*% y)
},
replications = 1000,
columns = c("test", "replications", "elapsed", "relative",
           "user.self", "sys.self")
)

##          test replications elapsed relative user.self sys.self
## 3 linear system      1000    0.272    1.000    0.272    0.000
## 1      lm            1000    1.386    5.096    1.373    0.012
## 2 pseudoinverse      1000    0.285    1.048    0.284    0.000
```

EXEMPLO 4: PACOTE MICROBENCHMARK

Execute o script

t4_bench_microbenchmark.R

```
## Unit: milliseconds
##          expr      min       lq     mean   median      uq      max
##        lm 335.73504 344.59359 365.34562 353.33344 387.12642 426.62201
## pseudoinverse 47.61561 50.06029 61.59383 52.35680 59.71544 109.18576
## linear system 28.24556 31.32272 40.82109 32.58404 35.16226 87.50643
## neval cld
##    100  c
##    100  b
##    100  a
```

```
library(microbenchmark)

set.seed(2017)
n <- 10000
p <- 100
X <- matrix(rnorm(n*p), n, p)
y <- X %*% rnorm(p) + rnorm(100)

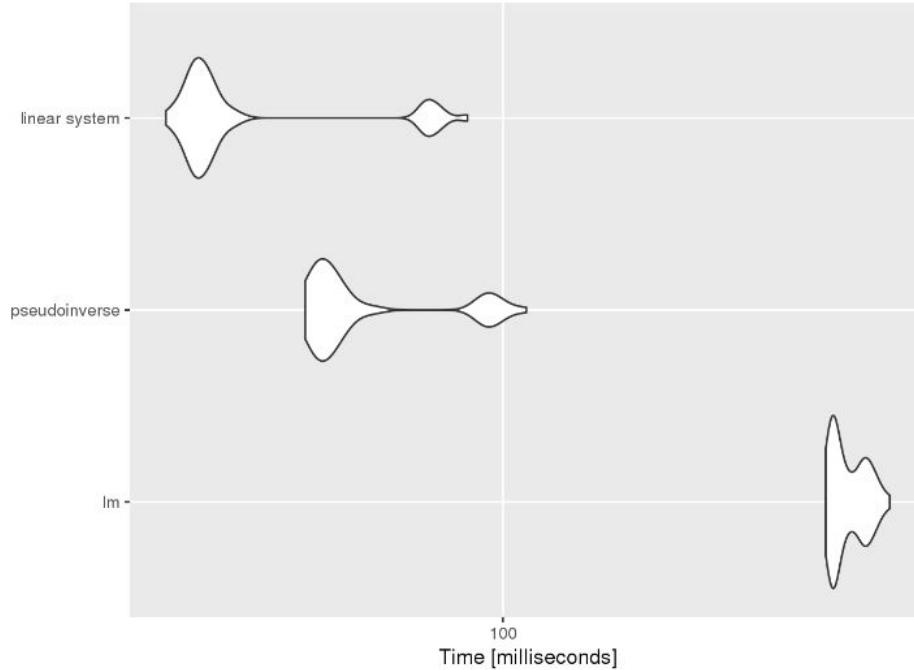
check_for_equal_coefs <- function(values){
  tol <- 1e-12
  max_error <- max(c(abs(values[[1]] - values[[2]]),
                      abs(values[[2]] - values[[3]]),
                      abs(values[[1]] - values[[3]])))
  max_error < tol
}

mbm <- microbenchmark("lm" = { b <- lm(y ~ X + 0)$coef },
                      "pseudoinverse" = {
                        b <- solve(t(X) %*% X) %*% t(X) %*% y
                      },
                      "linear system" = {
                        b <- solve(t(X) %*% X, t(X) %*% y)
                      },
                      check = check_for_equal_coefs)
```

mbm

EXEMPLO 4: PACOTE MICROBENCHMARK

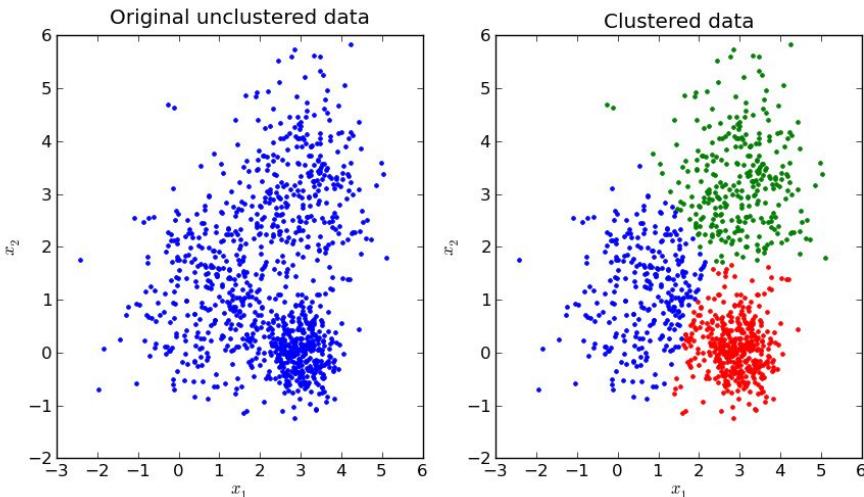
```
library(ggplot2)
autoplot(mbm)
```



EXEMPLO K-MEANS

ALGORITMO K-MEANS

- Proposto por MacQueen em 1967
- Consiste em particionar n observações em k conjuntos/grupos. É um problema NP-difícil, entretanto técnicas heurísticas são suficientes na conversão de um mínimo local



```
Especifique k
Selecione os k objetos que serão os centróides dos agrupamentos
para todos os objetos restantes faça
    Calcule a distância entre o elemento e os centróides
    Adicione o elemento ao agrupamento que possuir a menor distância
    Recalcule o centróide do agrupamento
fim para
para todos os k agrupamentos faça
    Calcule a Soma de Quadrados Residual
fim para
repita
    para todos os n elementos faça
        Mova o elemento para os outros agrupamentos
        Recalcule a Soma de Quadrados Residual
        se soma dos Quadrados Residual diminuiu então
            O objeto passa a fazer parte do grupo que produzir maior ganho
            Recalcule a Soma de Quadrados Residual dos grupos alterados
        fim se
    fim para
até Número de interações = i ou Não ocorra mudança de objetos
```

EXEMPLO 1: K-MEANS EXECUÇÃO SEQUENCIAL

```
#!/usr/bin/env Rscript

data <- read.csv('dataset.csv')
result <- kmeans(dt, centers=4, nstart=100)

print(result)
```

Execute o script **t1_kmeans_seq.R**

EXEMPLO 2: K-MEANS EXECUÇÃO COM LAPPLY

```
#!/usr/bin/env Rscript

data <- read.csv('dataset.csv')

parallel.function <- function(i){
  kmeans(x=dt, centers=4, nstart=i)
}

results <- lapply(c(25, 25, 25, 25), FUN=parallel.function)
results[[1]]$tot.withinss
temp.vector <- sapply(results, function(result){result$tot.withinss})
result <- results[[which.min(temp.vector)]]

print(result)
```

Execute o script **t2_kmeans_lapply.R**

EXEMPLO 3: K-MEANS EXECUÇÃO COM FOREACH

```
#!/usr/bin/env Rscript

data <- read.csv('dataset.csv')

results <- foreach(i=c(25,25,25,25)) %do%{
  kmeans(x=dt,centers=4,nstart=i)
}
temp.vector <- sapply(results,function(result){result$tot.withinss})
result <- results[[which.min(temp.vector)]]

print(result)
```

Execute o script **t3_kmeans_foreach.R**

EXEMPLO 4: K-MEANS COM DOSNOW

```
#!/usr/bin/env Rscript

library(foreach)
library(doSNOW)

data <- read.csv('dataset.csv')

cl <- makeCluster(4,type="MPI")
clusterExport(cl,c('dt'))
registerDoSNOW(cl)
results <- foreach(i=c(25,25,25,25)) %dopar%{
  kmeans(x=dt,centers=4,nstart=i)
}
temp.vector <- sapply(results,function(result){result$tot.withinss})
result <- results[[which.min(temp.vector)]]
stopCluster(cl)

print(result)
```

Execute o script **t4_kmeans_doSNOW.R**

EXEMPLO 5: K-MEANS COM DOMPI

```
#!/usr/bin/env Rscript

library(foreach)
library(doMPI)

data <- read.csv('dataset.csv')

cl <- startMPIcluster(count=4)
registerDoMPI(cl)
results <- foreach(i=c(25,25,25,25)) %dopar%{
  kmeans(x=dt,centers=4,nstart=i)
}
temp.vector <- sapply(results,function(result){result$tot.withinss})
result <- results[[which.min(temp.vector)]]
closeCluster(cl)

print(result)
```

Execute o script **t5_kmeans_doMPI.R**

EXEMPLO 6: K-MEANS COM DO PARALLEL

```
#!/usr/bin/env Rscript

library(foreach)
library(doParallel)

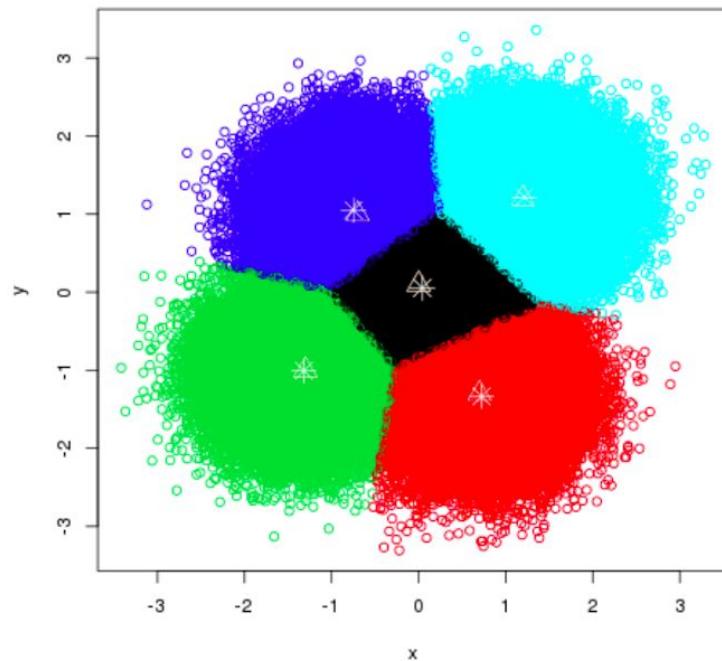
data <- read.csv('dataset.csv')

cl <- makeCluster(4,type="MPI")
clusterExport(cl,c('dt'))
registerDoParallel(cl)
results <- foreach(i=c(25,25,25,25)) %dopar%{
  kmeans(x=dt,centers=4,nstart=i)
}
temp.vector <- sapply(results,function(result){result$tot.withinss})
result <- results[[which.min(temp.vector)]]
stopCluster(cl)

print(result)
```

Execute o script **t6_kmeans_doParallel.R**

RESULTADO DO K-MEANS



COMPARANDO O TEMPO NAS DIFERENTES ESTRATÉGIAS DE EXECUÇÃO DO K-MEANS

Execute o script `kmeans_bench.R`

MELHORANDO O CÓDIGO R

MELHORANDO O CÓDIGO R

- Pequenas alterações no código R podem ter grandes efeitos no tempo de execução das funções



EXEMPLO 1: UM LOOP INEFICIENTE

```
system.time({
  for (i in 1:nrow(df)) {
    if ((df[i, 'col1'] + df[i, 'col2'] + df[i, 'col3'] + df[i, 'col4']) > 4) { # verifica se o num e > 4
      df[i, 5] <- "greater_than_4"
    } else {
      df[i, 5] <- "lesser_than_4"
    }
  }
})

##    user  system elapsed
##  603.174 18.983 623.079
```

Execute o script
speedupWithoutPar.R

EXEMPLO 2: UTILIZANDO UMA ESTRUTURA PRÉ ALOCADA

```
output <- character (nrow(df)) # inicializa um vetor de saida
system.time({
  for (i in 1:nrow(df)) {
    if ((df[i, 'col1'] + df[i, 'col2'] + df[i, 'col3'] + df[i, 'col4']) > 4) {
      output[i] <- "greater_than_4"
    } else {
      output[i] <- "lesser_than_4"
    }
  }
  df$output})

```

```
##    user  system elapsed
##  23.062   0.002  23.073
```

EXEMPLO 3: VERIFICANDO DETERMINADA CONDIÇÃO FORA DO LOOP

```
output <- character (nrow(df))
condition <- (df$col1 + df$col2 + df$col3 + df$col4) > 4 # verifica a condicao fora do loop
system.time({
  for (i in 1:nrow(df)) {
    if (condition[i]) {
      output[i] <- "greater_than_4"
    } else {
      output[i] <- "lesser_than_4"
    }
  }
  df$output <- output
})

##      user  system elapsed
##  0.091   0.000   0.091
```

EXEMPLO 4: UTILIZANDO O WHICH

```
system.time({  
    want = which(rowSums(df) > 4)  
    output = rep("less than 4", times = nrow(df))  
    output[want] = "greater than 4"  
})
```

EXEMPLO 5: USANDO O IFELSE

```
system.time({  
  output <- ifelse ((df$col1 + df$col2 + df$col3 + df$col4) > 4, "greater_than_4", "lesser_than_4")  
  df$output <- output  
})  
  
##    user  system elapsed  
##  0.092   0.010   0.102
```

Outras dicas:

- Remova objetos não usados e que carregam a memória. A função `rm()` remove objetos carregados no R;
- Sempre que possível, utilize estruturas que consumam menos memória. Por exemplo, `data.table`

MELHORANDO O CÓDIGO R, BARREIRA COMPUTACIONAL

→ *Speeding up*

- ◆ Lentidão nos loops ✓
- ◆ Rcpp, inline

```
1 library(Rcpp)
2
3 cppFunction('int g(int n) { if (n < 2)
4   return(n); return(g(n-1) + g(n-2)); }')
5
6 ## Using it on first 11 arguments
7 sapply(0:10, g)
8
9 f <- function(n){
10   if (n < 2) return(n)
11   return(f(n-1) + f(n-2))
12 }
13
14 library(rbenchmark)
15 benchmark(f(20), g(20))[,1:4]
16
18:1 (Top Level) ⇡
```

Console ~/Parallel_tests/ ↗

```
+ return(f(n-1) + f(n-2))
+ }
> library(rbenchmark)
> benchmark(f(20), g(20))[,1:4]
  test replications elapsed relative
1 f(20)           100    1.693  282.167
```

OUTROS FRAMEWORKS (NÃO COBERTOS NESSE MÓDULO)

- Big data
 - ◆ **SparkR** (R + Apache Spark), **Rhipe**
 - ◆ HadoopR
 - ◆ **bigmemory**, **ff**
 - ◆ Data management
 - DBMS: MySQL, PostgreSQL, RNeo4j
- GPU
 - ◆ **gpuR**, **gputools**
- Computação em grid: **multiR**
- Bibliotecas científicas como **RcppEigen**, **RcppArmadillo**

REFERÊNCIAS, APOSTILAS, CURSOS, ...

Apostilas

- **Introdução ao R nas Ciências da Vida:** <https://quelopes.github.io>
- **CRAN Task View: High-Performance and Parallel Computing with R:** <https://cran.r-project.org/web/views/HighPerformanceComputing.html>
- **Curso introdutório de R:** <http://statmath.wu.ac.at/~schwendinger/HPC/>
- **R em HPC:** <http://www.glennclockwood.com/data-intensive/r/on-hpc.html>
- **Benchmarking in R:** http://www.alexejgossmann.com/benchmarking_r/

Códigos

- <https://github.com/eddelbuettel/ctv-hpc>
- <https://github.com/glennclockwood/paraR>

Cursos online

Via coursera:

- **R Programming (Johns Hopkins University):** <https://www.coursera.org/learn/r-programming>
- **Statistics with R Specialization (Duke University):** <https://www.coursera.org/specializations/statistics>

Referências bibliográficas

- State of the Art in Parallel Computing with R. *Journal of Statistical Software*. August 2009, Vol. 31(1).
- A Survey of R Software for Parallel Computing. *American Journal of Applied Mathematics and Statistics*. 2014, Vol. 2(4), pg. 224-230.
- Parallel R. (Book). Q. Ethan McCallum e Stephen Weston. 2012. O'reilly.

ESTUDO DE CASO: MODEL-R

FIM!
OBRIGADO

Código: https://github.com/quelopes/R_for_HPC

Apresentação: https://quelopes.github.io/files/courses/RforHPC_Lncc2018/

Apostila: (em breve!)

Guilherme Gall (gmgall@lncc.br)
Raquel L. Costa (quelopes@gmail.com)