PARALLEL PROCESSING IN JULIA USING GRID ENGINE

EMANUEL DIEGO S PENHA

RESEARCH SCHOLAR

http://goo.gl/tkSwLg

INTRODUCTION

Julia is a new-high level computer language built for technical computing with performance in mind. Julia processes can create tasks that utilize large computational resource, these can be ran using clusters.

Clusters are a massive network of computers that can be viewed as a single system. To best harness the power of such machines, code is typically ran in parallel.

While clusters can be viewed as a single system, they required a way to manage processes and processors. One such tool is Grid Engine.

Grid Engine is a program for executing batch jobs on linux clusters. The main idea is to provide a way to schedule big and small jobs and maintain the cluster in a optimal state.The cluster will use available memory and processors as parameters to determine in what order jobs run.

The main objective of this discussion is to describe how one can use another other than Julia and the queuing system of Grid Engine to complete tasks, particularly int the field of bioinformatics.

BINARY AT

```
/home/penhaeds/bin/julia-cb9bcae93a/bin/julia
```

ONLINE AT

https://www.juliabox.org/

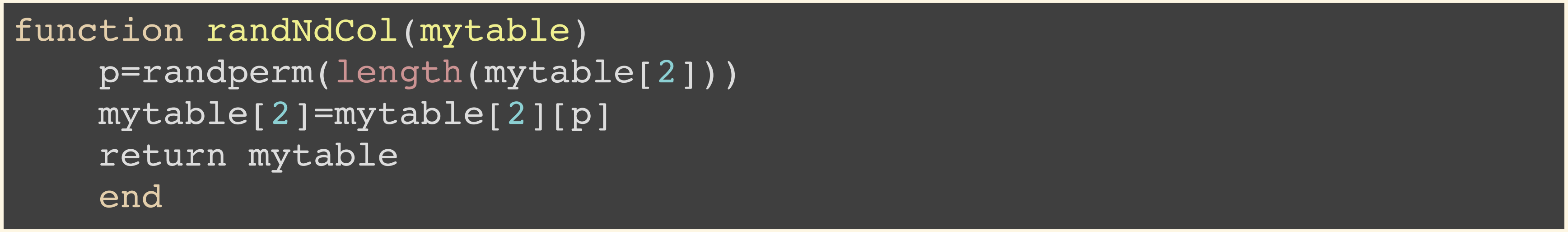
BASIC COMMUNICATION

```
$ ./julia -p 2

julia> r = remotecall(2, rand, 2, 2)
RemoteRef(2,1,5)

julia> fetch(r)
2x2 Float64 Array:
 0.60401   0.501111
 0.174572  0.157411

julia> s = @spawnat 2 1 .+ fetch(r)
RemoteRef(2,1,7)

julia> fetch(s)
2x2 Float64 Array:
 1.60401  1.50111
 1.17457  1.15741
```

Click here for a more complete example

DATAFRAMES

```
julia> using(DataFrames)
julia> df = DataFrame(Origin = ["a","b","c","d","a","d"],
    Target = ["j", "f", "g", "h", "i", "j"])
julia> df[:Target]
julia> p = randperm(length(df[:Target]))
julia> df[:Target]=df[:Target][p]
julia> df
```



```
function randNdCol(mytable)
    p=randperm(length(mytable[2]))
    mytable[2]=mytable[2][p]
    return mytable
    end
```

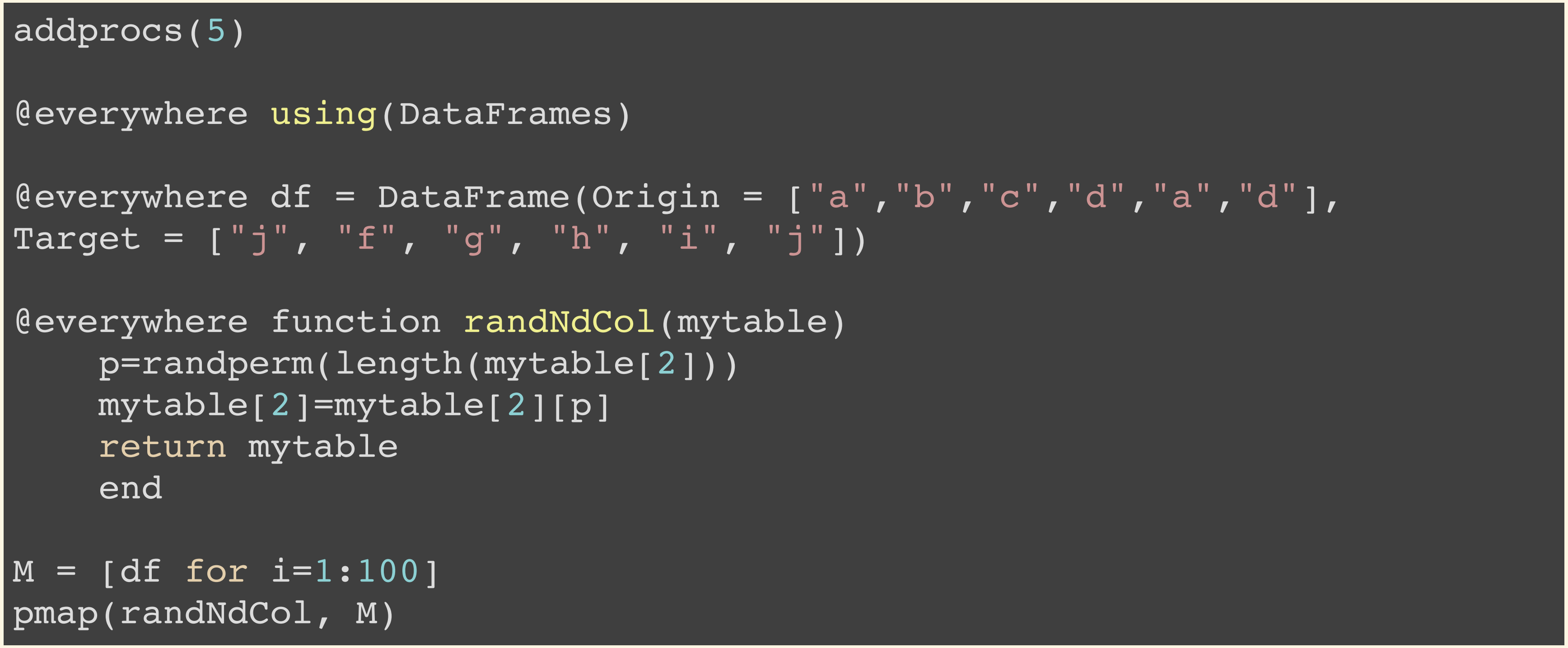
```
randNdCol(df)
```

LOAD DATA/FUNCTIONS TO ALL PROCESSES

```
julia> require("myfile")
@everywhere id = myid()
```

DATA, FUNCTIONS AND PROCESSES

```
addprocs(5)
@everywhere using(DataFrames)
@everywhere df = DataFrame(Origin = ["a","b","c","d","a","d"],
Target = ["j", "f", "g", "h", "i", "j"])
@everywhere function randNdCol(mytable)
    p=randperm(length(mytable[2]))
    mytable[2]=mytable[2][p]
    return mytable
    end
newDf = @spawn randNdCol(df)
fetch(newDf)
```



```
addprocs(5)

@everywhere using(DataFrames)

@everywhere df = DataFrame(Origin = ["a","b","c","d","a","d"],
Target = ["j", "f", "g", "h", "i", "j"])

@everywhere function randNdCol(mytable)
    p=randperm(length(mytable[2]))
    mytable[2]=mytable[2][p]
    return mytable
    end

M = [df for i=1:100]
pmap(randNdCol, M)
```

take a look at the output

USING CLUSTERMANAGERS

```
julia> Pkg.update()
julia> Pkg.add("ClusterManagers")
julia> using ClusterManagers
julia> ClusterManagers.addprocs_sge(5)

job id is 512301, waiting for job to start .................
5-element Array{Any,1}:
 2
 3
 4
 5
 6
```

THE DIFERENCE NOW IS

```
ClusterManagers.addprocs_sge(5)
```

instead of

```
addprocs(5)
```

NEW CODE

```
ClusterManagers.addprocs_sge(5)
@everywhere using(DataFrames)
@everywhere df = DataFrame(Origin = ["a","b","c","d","a","d"],
Target = ["j", "f", "g", "h", "i", "j"])
@everywhere function randNdCol(mytable)
    p=randperm(length(mytable[2]))
    mytable[2]=mytable[2][p]
    return mytable
    end

newDf=[]

M = [df for i=1:10]
pmap(randNdCol, M)
```

TAKE A LOOK AT THE WORKERS ON SGE

```
qstat
```

TAKE A LOOK AT THE WORKERS ON JULIA

```
workers()
```

REMOVE WORKERS FROM JULIA

```
rmprocs(workers());
```

REMOVE WORKERS FROM SGE

```
qdel -u username
```

NODES

```
julia> @parallel for i=1:5
       run(`hostname`)
       end

julia>  From worker 2:  alto
        From worker 5:  alto
        From worker 4:  alto
        From worker 6:  alto
        From worker 3:  alto
```

REDUCE EXAMPLE

```
nheads = @parallel (+) for i=1:200000000
randbool()
end
```

THANK YOU!

penhaeds@uab.edu