Notas de Notebooks - Multicore

Primera versión paralela para el cálculo de componentes disjuntos

Se paralelizó el cálculo de componentes disjuntos aprovechando que el cálculo de cada matriz es independiente en el tiempo. Notas sobre el hardware: intel i7 2,6 GHz 3720QM 4 cores + hyperthreading 8GB RAM 1600

Configuración inicial de Julia en paralelo

In [1]:
```
CPU_CORES
```

Out[1]:
```
8
```

In [1]:
```
addprocs(CPU_CORES)
```

Out[1]:
```
8-element Array{Any,1}:
 2
 3
 4
 5
 6
 7
 8
 9
```

In [2]:
```
cd("/Users/Nes/Desktop/NeuroCiencias/JuliaExperiments/Parallel-Neuroscience/ComponentesDisjuntos");
```

Carga los datos, aplica los filtros, laplacianos, etc.

In [3]:
```
include("init_parallel.jl")
```

```
INFO: Loading help data...
INFO: Loading help data...
INFO: Loading help data...
INFO: Loading help data...
INFO: Loading help data...
INFO: Loading help data...
INFO: Loading help data...
INFO: Loading help data...
INFO: Loading help data...
```

Out[3]:
```
0
```

Prueba de la versión paralela

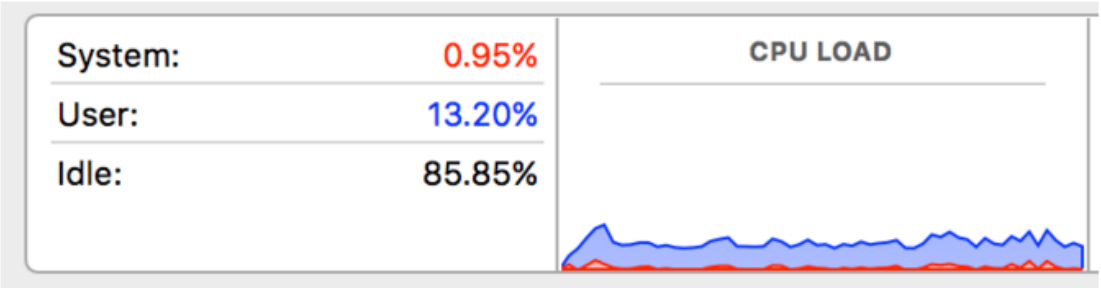
La versión paralela, que se encuentra dentro del archivo obtenComponentesDisjuntosParallel.jl, básicamente consiste en usar las macros @Parallel que mandan llamar a @spawn automáticamente para distribuir las rebanadas de tiempo entre los distintos procesos. Es importante usar @sync en el for para que el código espere que todos los workers terminen el loop. También se debe declarar cada función en cada worker con @everywhere.

La versión paralela tiene un speedup de 10x con respecto a la versión secuencial. Además de que los 8 workers procesan en paralelo cada rebanada de tiempo. Además, comando @Parallel, configura la rutina mapreduce para usar todos los núcleos, lo cual ayuda en las búsquedas de los índices de los fuentes y pozos.

```
In [59]: @time ObtenComponentesYEscribe(CSD,101,120)
```

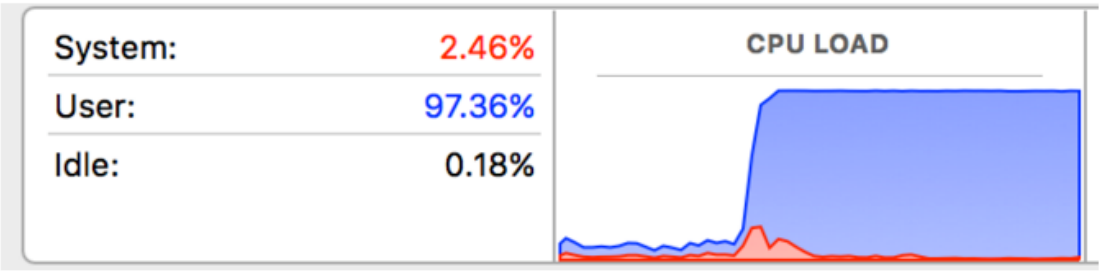
```
elapsed time: 401.756996716 seconds (37209500428 bytes allocated, 69.73% gc time)
```

```
Out[59]: 0
```




```
In [62]: @time ObtenComponentesYEscribeP(CSD,101,120)
```

```
elapsed time: 38.634555157 seconds (1998970576 bytes allocated, 12.00% gc time)
```



Pruebas con menos cores

Uno de los procesos se encarga de administrar, por lo que originalmente hay 9 procesos aunque hay 8 cores.

Quitamos 4 procesos => nos quedan 5 procesos

```
In [4]: rmprocs(6,7,8,9)
```

```
Out[4]: :ok
```

```
In [7]: @time ObtenComponentesYEscribeP(CSD,101,120)
```

```
elapsed time: 44.176137892 seconds (1005908740 bytes allocated, 3.11% gc time)
```

Quitamos otros 2 procesos => nos quedan 3 procesos

```
In [8]: rmprocs(4,5)
```

```
Out[8]: :ok
```

```
In [9]: @time ObtenComponentesYEscribeP(CSD,101,120)
```

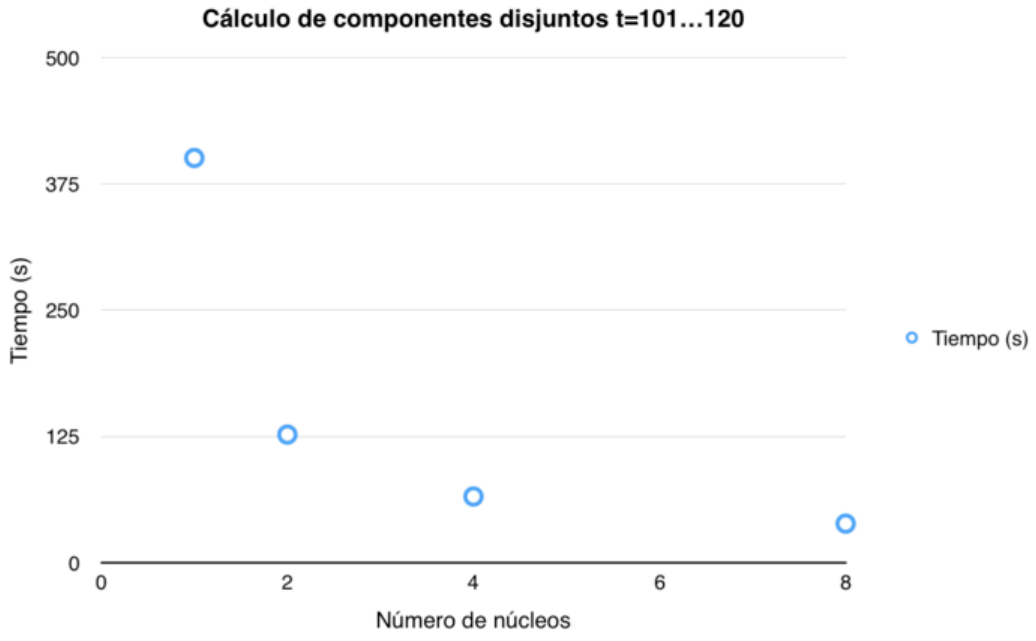
```
elapsed time: 65.453253076 seconds (499742456 bytes allocated, 1.05% gc time)
```

```
In [10]: workers()
```

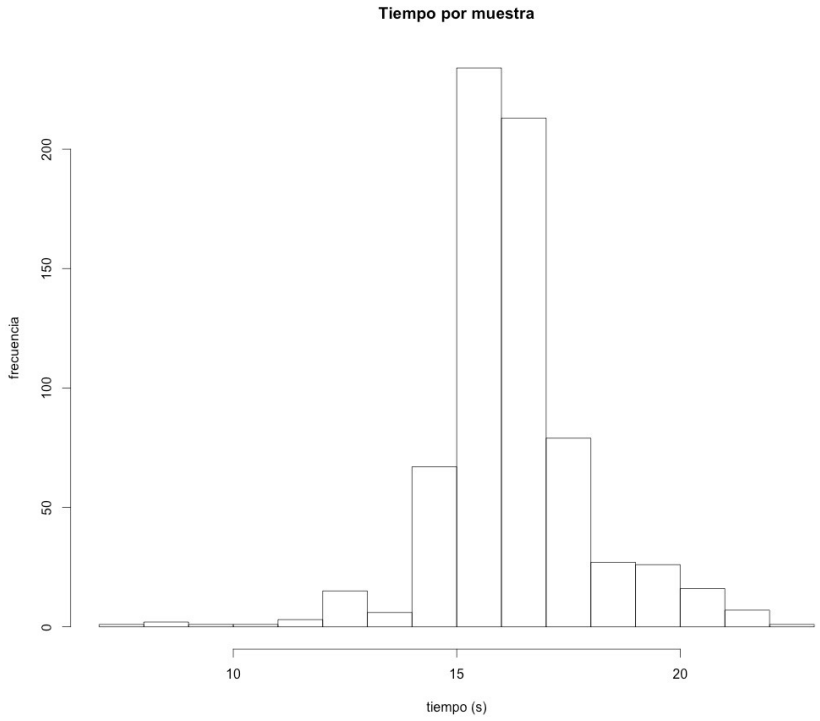
```
Out[10]: 2-element Array{Int64,1}:
 2
 3
```

Quitamos otro proceso más => nos quedan 2 procesos

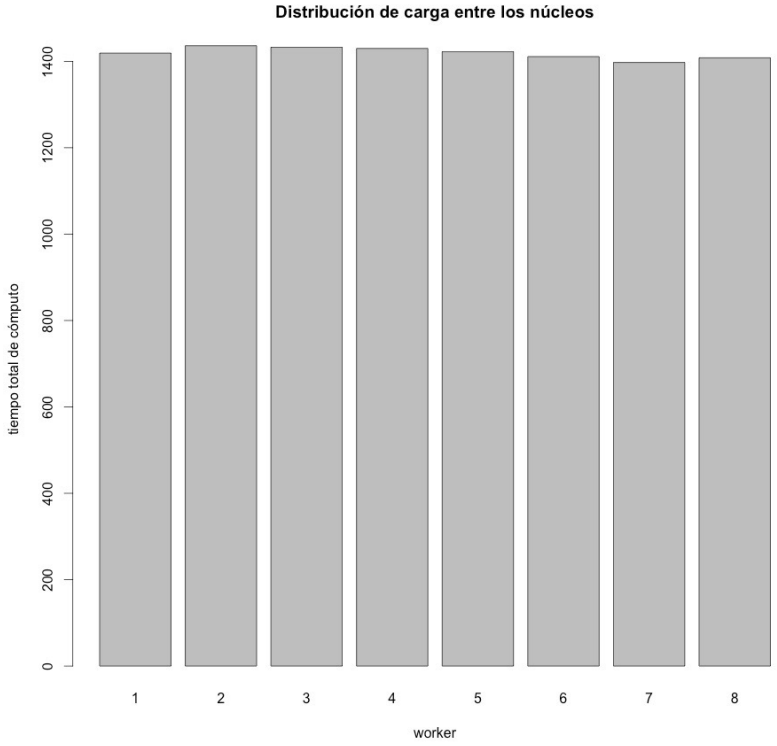
```
In [12]: rmprocs(3)
Out[12]: :ok

In [14]: workers()
Out[14]: 1-element Array{Int64,1}:
          2

In [15]: @time ObtenComponentesYEscribeP(CSD,101,120)
         elapsed time: 126.744695375 seconds (249871380 bytes allocated, 0.28% gc time)

In [16]: nprocs()
Out[16]: 2

In [17]: rmprocs(2)
Out[17]: :ok

In [18]: @time ObtenComponentesYEscribeP(CSD,101,120)
         elapsed time: 401.523635118 seconds (37295192276 bytes allocated, 69.81% gc time)
```



Cada muestra tiene un tiempo de cómputo distinto. A continuación se muestra un histograma en el que se observa la distribución del tiempo por muestra en 700 muestras.



Sin embargo, a pesar de que el tiempo de cómputo es distinto entre las diferentes muestras, la carga se distribuye equitativamente entre los núcleos.



La posible explicación de la caída exponencial del tiempo de cómputo de uno a dos núcleos, se debe a que se aprovechan dos cosas: el cómputo paralelo de muestras y las búsquedas usando map reduce, que como el programa tiene muchas entradas/salidas, permite usar ahí ese tiempo de CPU. Luego, de dos a cuatro núcleos parece tener un decremento lineal y de 4 a 8 observamos un comportamiento asintótico que podría ser causado por las escrituras a disco, ya que por cada muestra en paralelo se guarda un archivo de datos de los componentes.

En caso de que el problema fuera mucho mayor y se tuviera una enorme cantidad de datos la solución que vería viable, sería una BD distribuida, ya que se podría escribir en paralelo. Sin embargo, en este caso, debido a la cantidad de datos a analizar y el tiempo de análisis, no nos afecta esto.