


# R en el CCAD - Clase 3

Lic. Marcos Mazzini - CCAD / UNC  
CPA CONICET

Lic. Juan Cruz Rodriguez - FAMAF / UNC  
CIDIE - CONICET



# Conceptos - Programación Paralela

## **Paralelismo de Datos**

Los procesos realizan las mismas operaciones sobre conjuntos particionados de datos.

## **Memoria Compartida**

Los procesos se comunican mediante un área de memoria accesible por todos.

## **Paralelismo de Tareas**

Cada proceso realiza tareas diferentes y debe comunicarse con el resto.

## **Memoria Distribuida**

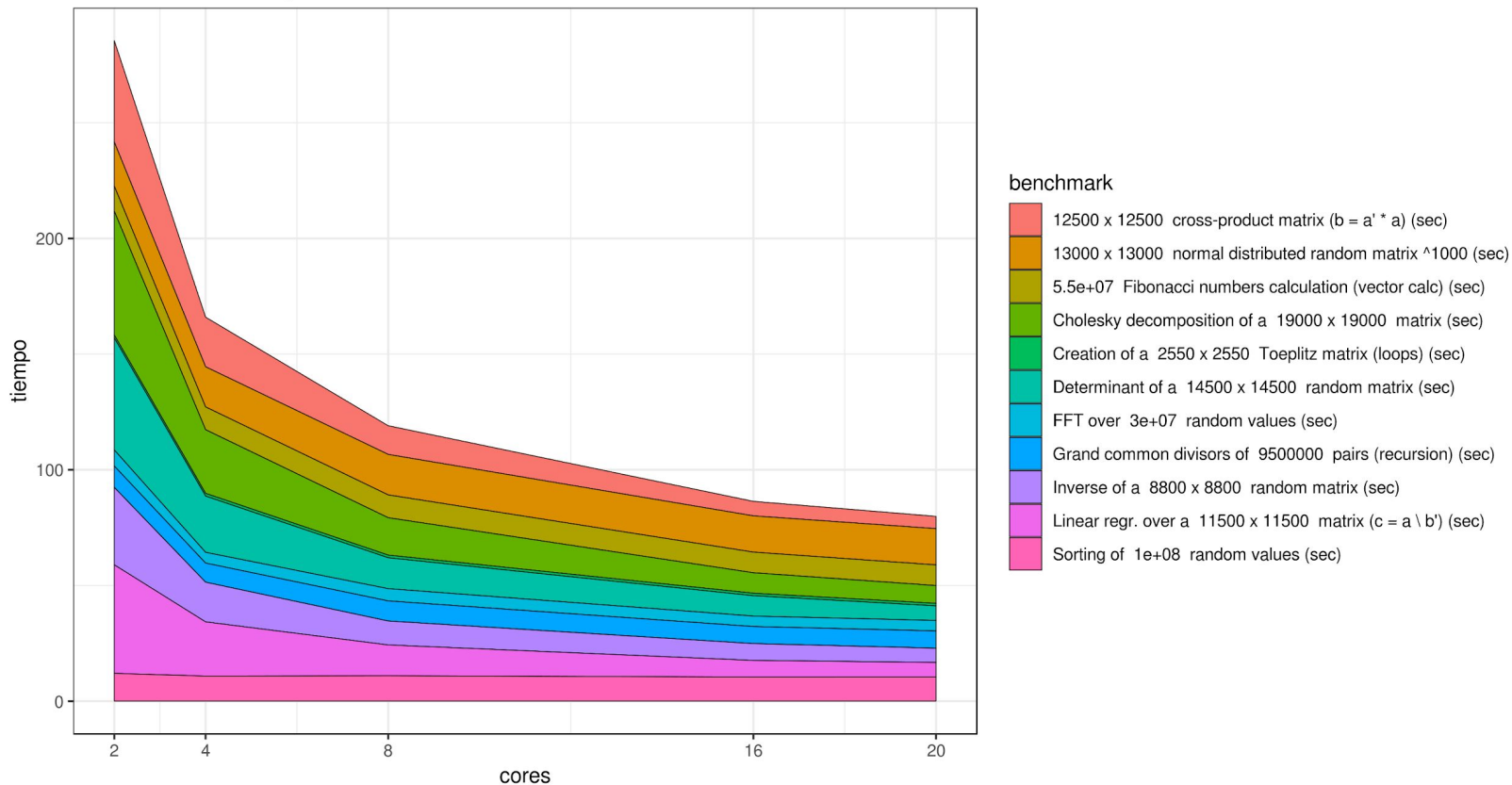
Los procesos se comunican enviando y recibiendo mensajes por un canal de comunicación.

# Niveles de paralelización

- Aceleración gracias BLAS/MKL (ventajas en uno y más cores)
- Librerías ya paralelizadas
- Parallel
- Rmpi

# Compilando R con soporte para librerías Intel MKL

Reducción de tiempo utilizando librerías MKL



# Librerías ya paralelizadas

- En general se requiere analizar qué parámetros funcionan mejor con nuestro problema.
- Internamente aplican alguna técnica de programación paralela.
- En muchos casos importan `parallel` que ya está en el base de R.

# Librería parallel

Nosotros somos responsables de particionar adecuadamente el problema de forma que cada core tenga una cantidad de trabajo acorde.

Estamos limitados a correr en un solo nodo (threads).

# Librería Rmpi

Debemos manejar **todos** los aspectos de la comunicación entre los cores:

- Código que ejecutará cada core
- Cómo se distribuirán los datos
- La sincronización

Rmpi provee funciones “wrapper” sobre la implementación MPI ya existente.

Se puede usar para correr en uno o más nodos.

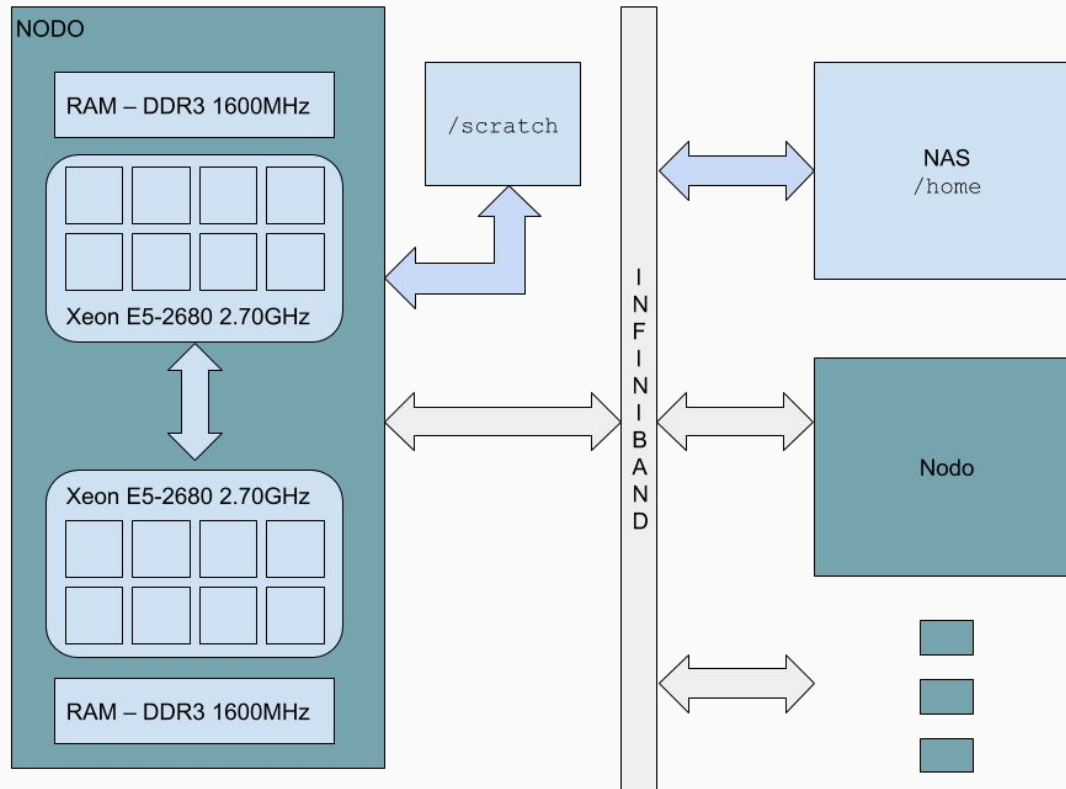
# Arquitectura de un nodo

Si bien hay programas que se pueden paralelizar de forma inmediata, en algún momento tendremos que conocer **detalles del hardware** sobre el que van a ejecutarse

- Número de cores
- Cantidad de memoria
- Optimizar la entrada/salida

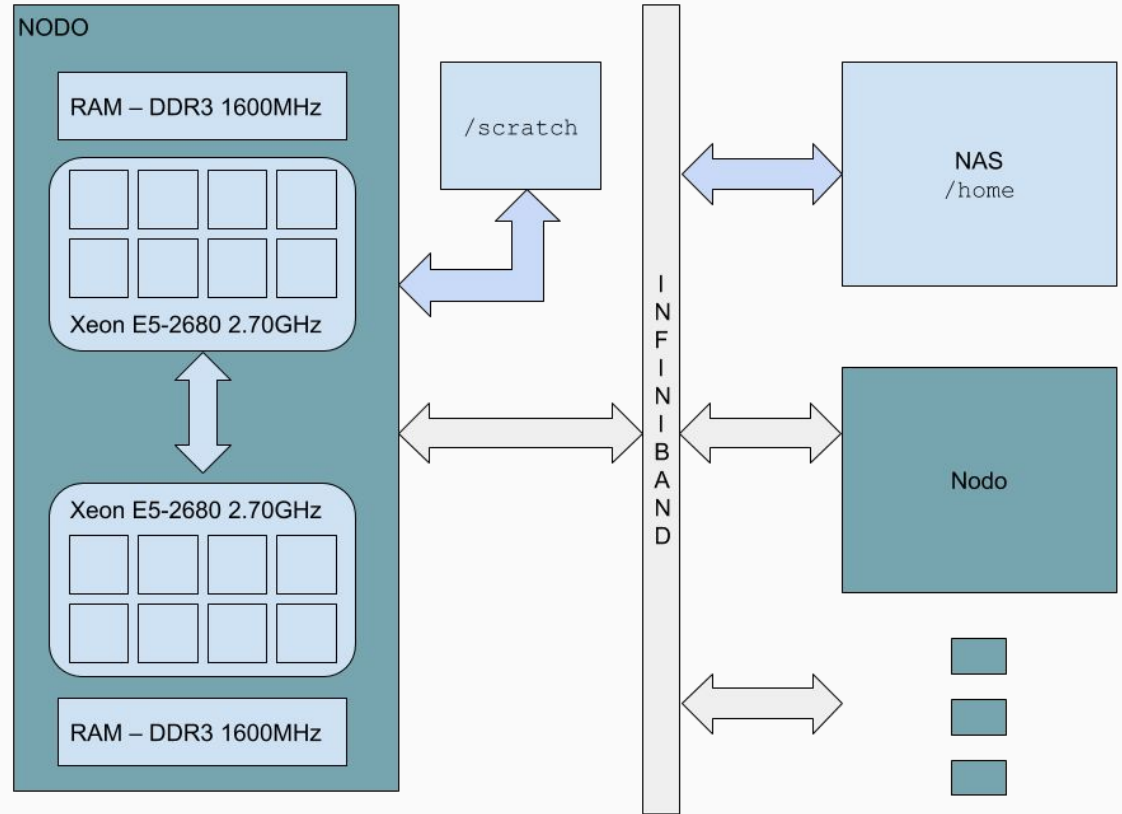


# Arquitectura de un nodo de Mendieta: 2x 8 cores - 64Gb de RAM



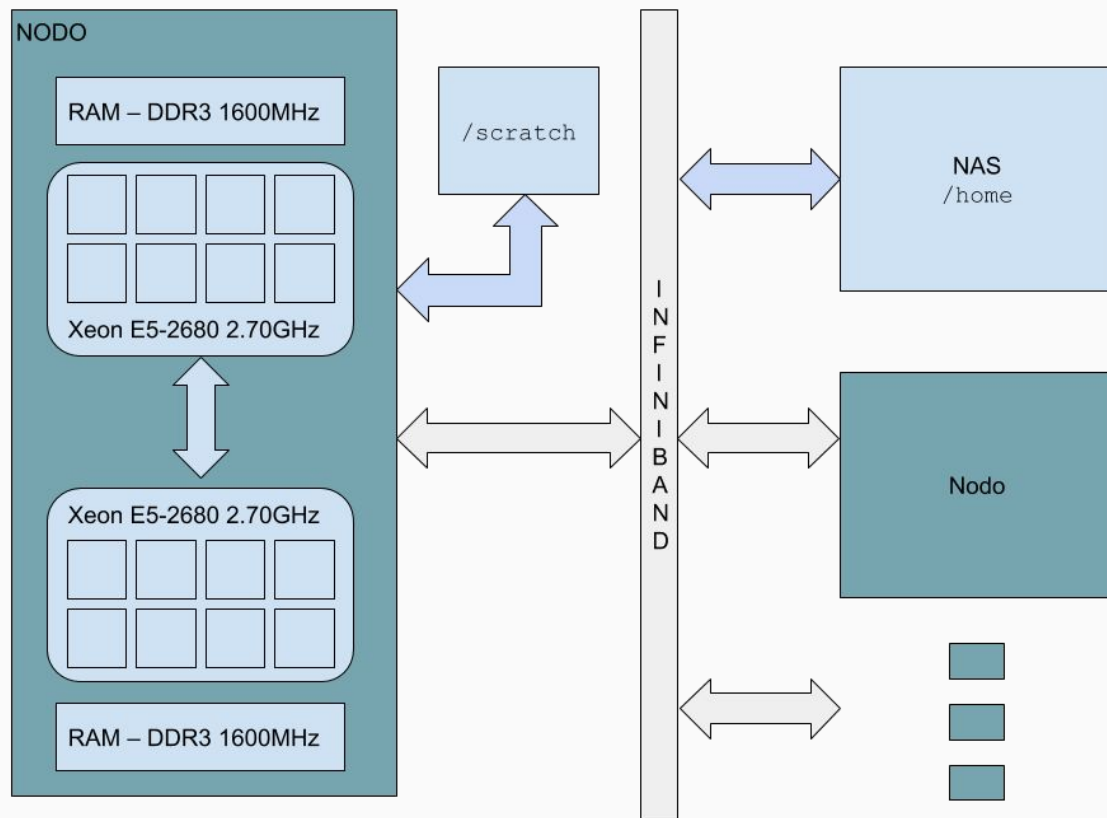
# Memoria Compartida (parallel)

- No puede exceder un nodo
- Oculta el mecanismo de comunicación



# Memoria distribuida (Rmpi)

- Mono/Multi-nodo
- Oculta la latencia de la comunicación



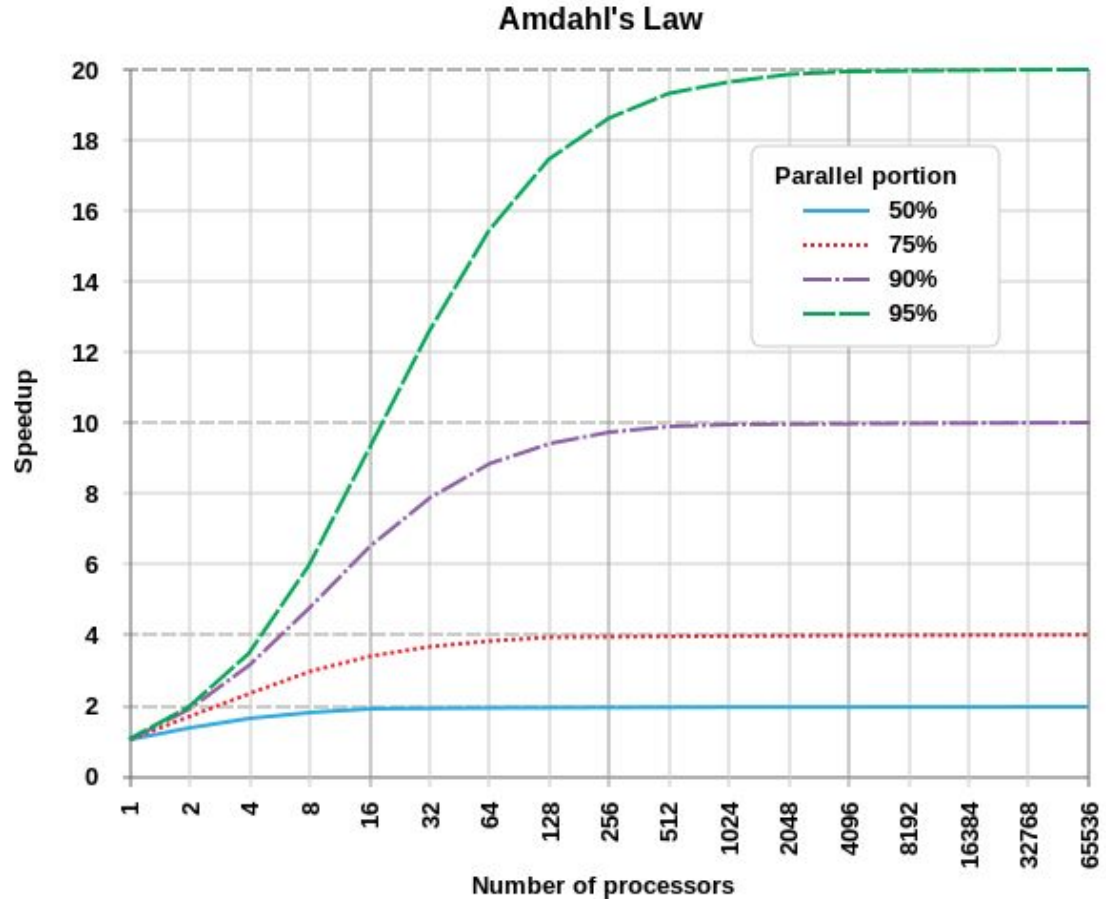
# Paralelizar

Un modelo simple podría correr **N** veces más rápido cuando se reparte entre **N procesadores**.

Pero todo problema tiene una fracción **S** que es **inherentemente secuencial** y otra fracción **1 - S** que puede ser paralelizada:

$$speedUp = \frac{1}{s + \frac{1-s}{n}}$$

- Los problemas con  $S \approx 0$  son inherentemente paralelos
- Puede haber casos inherentemente secuenciales



# SLURM: Particiones

# Particiones

Los nodos del cluster se encuentran agrupados en **particiones**.

Cada partición o cola tiene restricciones y valores por defecto.

Para correr un trabajo debemos indicar a qué partición lo enviaremos:

```
#SBATCH --partition=multi
```

# SLURM: Particiones

En nuestros clusters contamos con las siguientes particiones:

- **DEBUG:** Máximo **2 min.**
- **MONO:** Trabajos de **1 nodo**, hasta **8 cores** por nodo, hasta **7 días** de ejecución.
- **MULTI:** Trabajos de **2 a 8 nodos**, hasta **20 cores** por nodo, hasta **4 días** de ejecución.

También tenemos **GPU** y **PHI**.



# SLURM: Reserva

Hemos reservado **2 nodos durante un mes** para este curso, para utilizar la reserva deben incluirla en su script de submit.

Esta reserva se realizó sobre la partición **cursoR**, habrá que utilizar ésta por más que sólo corramos en un nodo.

```
#SBATCH --partition=cursoR  
#SBATCH --reservation=cursoR
```

Cuando finalice la reserva quitaremos la línea **reservation** y utilizaremos la partición **mono**. multi sólo para correr con Rmpi

Script parametrizado

# Script parametrizado

Recordemos con un ejemplo cómo habíamos particionado una simulación que recibía parámetros.

Fuera del cluster ejecutábamos con

```
$ Rscript hola.R 10 40
```

En el cluster debemos integrar esos parámetros a nuestro script de submit.

# Script parametrizado

```
[mmazzini@mendieta clase3]$ cat hola.R
sim_start <- 1
sim_end <- 40
args <- commandArgs(TRUE)
if (length(args) == 2) {
  sim_start <- as.numeric(args[[1]])
  sim_end <- as.numeric(args[[2]])
}

cat ("Corriendo en ", Sys.info()["nodename"],
     "desde: ", sim_start, "hasta: ", sim_end, "\n")
```

# Script parametrizado

Para que éste código R pueda recibir los parámetros deberemos adaptar nuestro script de submit.

**Método 1:** El script llama a Rscript **con todos los parámetros** que recibió.

**Método 2:** **Dentro del script** explicitamos los parámetros al llamar a Rscript.

## Metodo 1: Parametrizar el submit

```
[mmazzini@mendieta clase3]$ cat submit.sh
#!/bin/bash
#SBATCH --partition=cursor
#SBATCH --reservation=cursor
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --time=0-00:02

. /etc/profile

module load R

srun Rscript hola.R $*
```

## Metodo 1: Parametrizar el submit

```
[mmazzini@mendieta clase3]$ sbatch submit.sh 1 10
[mmazzini@mendieta clase3]$ sbatch submit.sh 11 20
[mmazzini@mendieta clase3]$ sbatch submit.sh 21 30
[mmazzini@mendieta clase3]$ sbatch submit.sh 31 40
[mmazzini@mendieta clase3]$ ls
hola.R  slurm-1.out  slurm-2.out  slurm-3.out  slurm-4.out  submit.sh
[mmazzini@mendieta clase3]$ cat slurm-*
Corriendo en  mendieta23.mendieta.ccad.unc.edu.ar desde:  1 hasta:  10
Corriendo en  mendieta01.mendieta.ccad.unc.edu.ar desde: 11 hasta:  20
Corriendo en  mendieta23.mendieta.ccad.unc.edu.ar desde: 21 hasta:  30
Corriendo en  mendieta07.mendieta.ccad.unc.edu.ar desde: 31 hasta:  40
```

## Metodo 2: Todos en el mismo submit

```
[mmazzini@mendieta clase3]$ cat submit.sh
#!/bin/bash
#SBATCH --partition=cursor
#SBATCH --reservation=cursor
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --time=0-00:02

. /etc/profile
module load R
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export MKL_NUM_THREADS=$SLURM_CPUS_PER_TASK

srun Rscript hola.R 1 10 &
srun Rscript hola.R 11 20 &
srun Rscript hola.R 21 30 &
srun Rscript hola.R 31 40
```



## Metodo 2: Todos en el mismo submit

```
[mmazzini@mendieta clase3]$ sbatch submit.sh
[mmazzini@mendieta clase3]$ cat slurm-1.out
Corriendo en   mendieta23.mendieta.ccad.unc.edu.ar desde:  1 hasta:  10
Corriendo en   mendieta23.mendieta.ccad.unc.edu.ar desde: 31 hasta:  40
Corriendo en   mendieta23.mendieta.ccad.unc.edu.ar desde: 11 hasta:  20
Corriendo en   mendieta23.mendieta.ccad.unc.edu.ar desde: 21 hasta:  30
```

# Extra RAM

El nodo mendieta23 es un FAT-Node que cuenta con **256 GB** de memoria RAM

Si la simulación que vamos a correr necesita más de 64GB podemos correr en este nodo simplemente agregando el requerimiento a nuestro script de submit.

```
#SBATCH --mem=96G
```

Nota: 252 GB máximo disponibles para las simulaciones

# Validar un script de submit

Finalmente podemos **verificar** si hay errores en el script, estimar **cuando correría** y con **qué recursos** agregando la opción **--test-only** al comando sbatch:

```
$ sbatch --test-only submit.sh
```

Sólo informa, NO realiza el submit.

[soporte@ccad.unc.edu.ar](mailto:soporte@ccad.unc.edu.ar)

Muchas Gracias!

