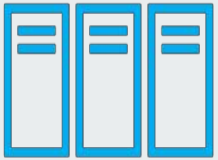
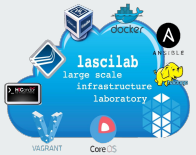


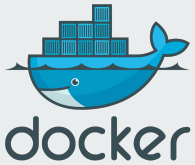
2 Seminario de Plataformas Computacionales

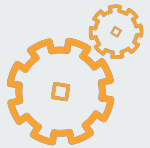
Computación de Alto Rendimiento y Reproducibilidad Experimental

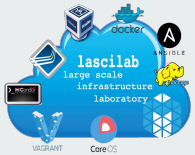


Introdución a OpenMP

Aurelio Vivas - aurelio.vivas@correounivalle.edu.co





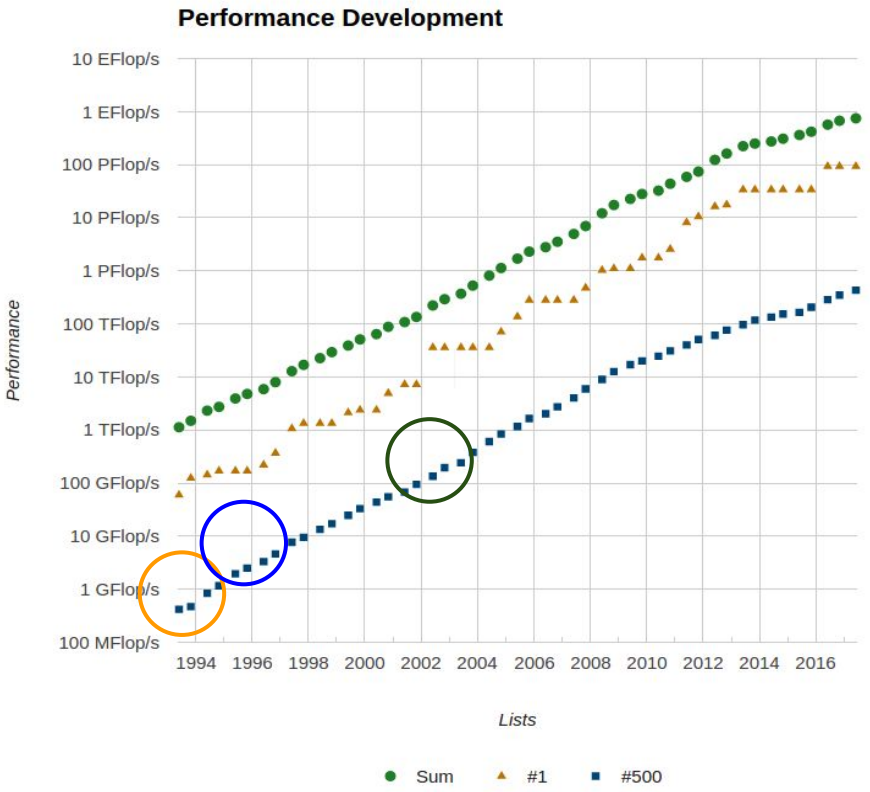


Lo que aprenderás hoy

¿Cómo OpenMP puede mejorar el rendimiento de tus experimentos?

- Resumen de la sección anterior
  - ¿Por qué la Computación Paralela es importante?
  - ¿Qué es la Computación Paralela?
  - Clasificación de los Computadores paralelos
- Uso del Compilador GCC/G++
- OpenMP

Resumen



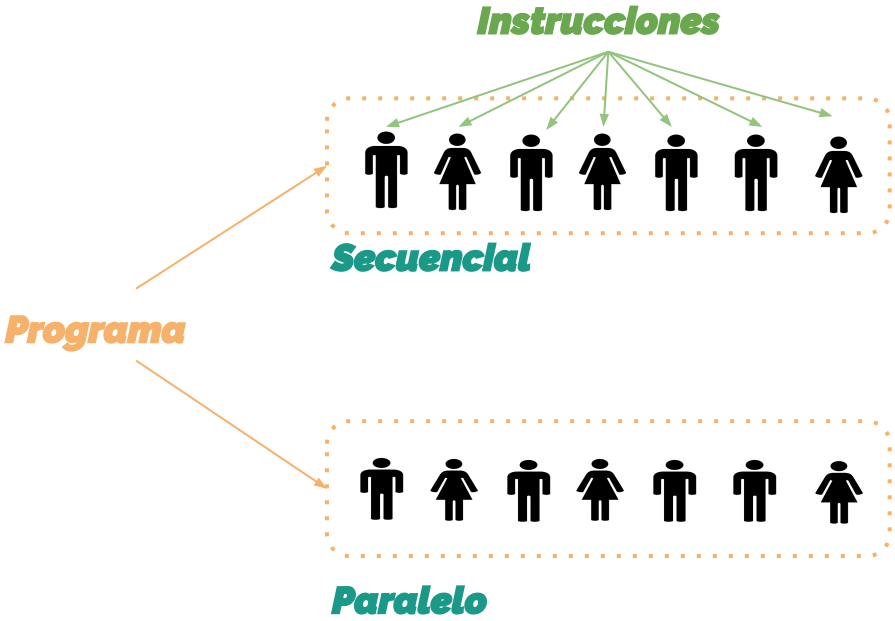
~2.79 GFLOPs/core
Secuencial

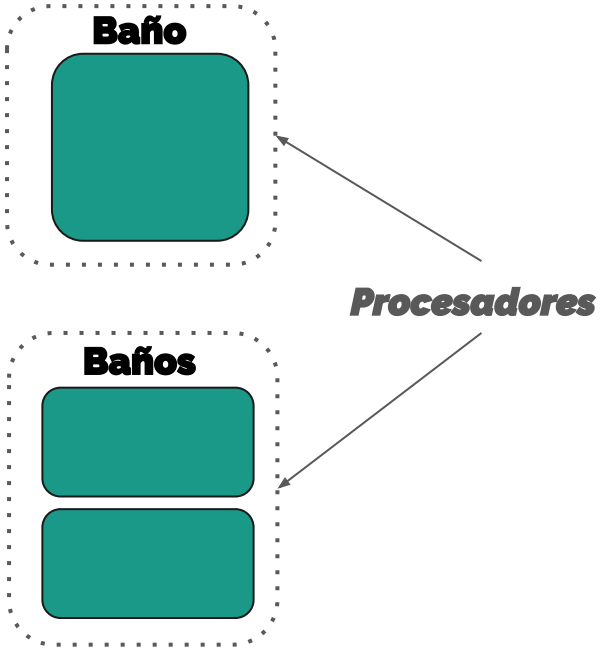
~13.69 GFLOPs/computador cpus
Paralelismo

~641.3 GFLOPs/computador gpus
Paralelismo

¿Por qué la computación paralela es importante?

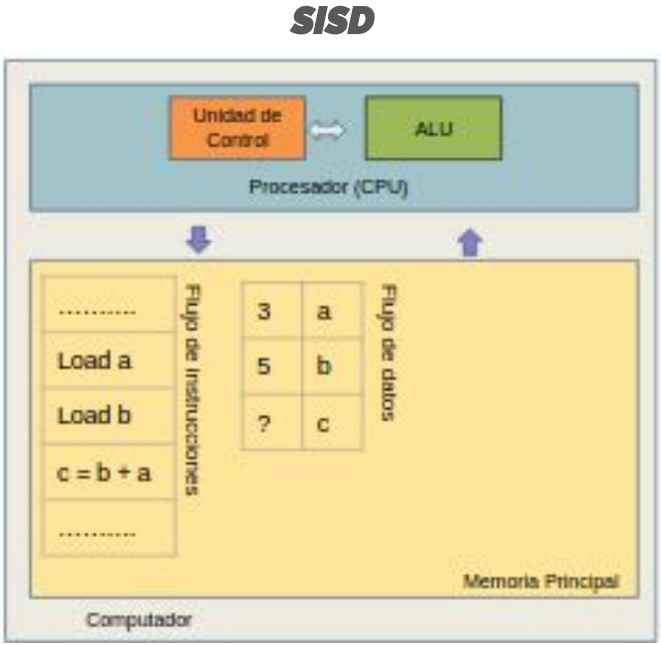
Resumen

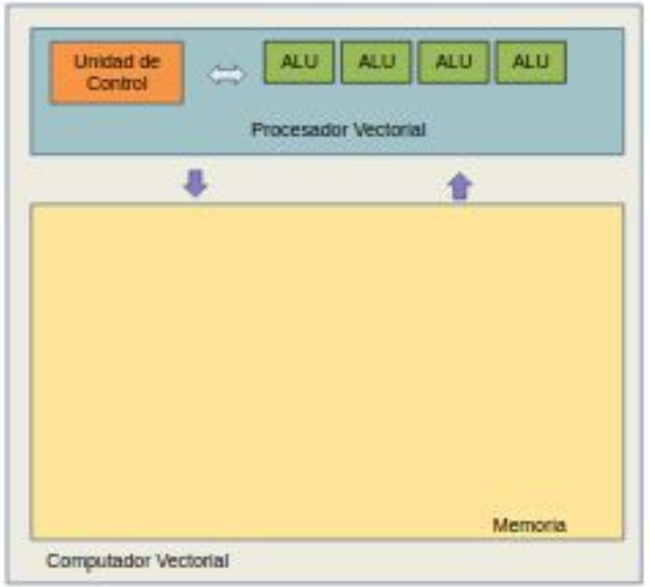




¿Qué es la Computación Paralela?

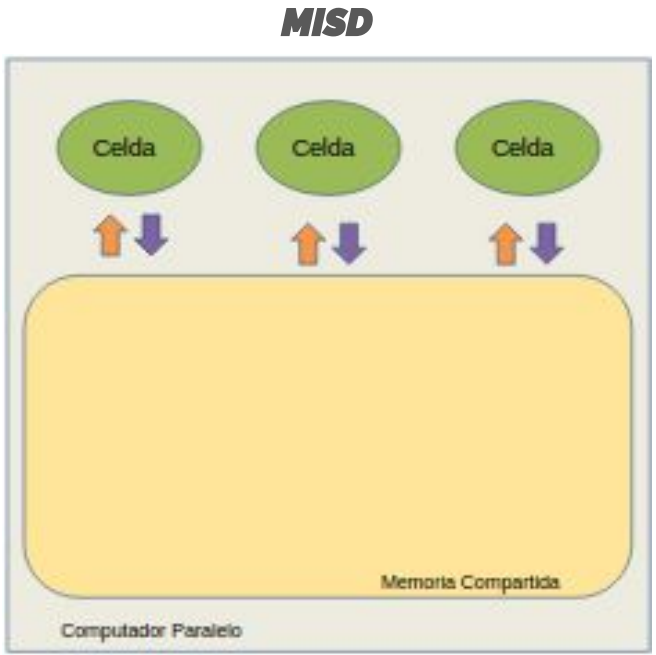
Resumen

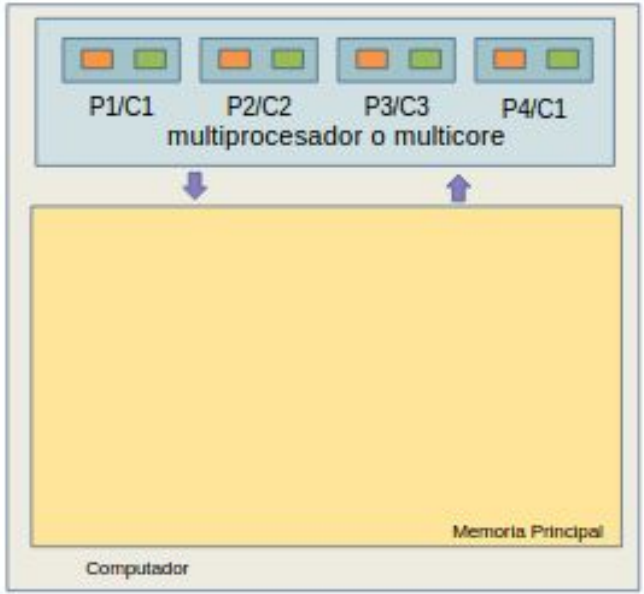




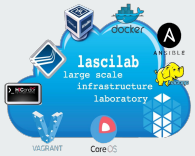
Clasificación de los computadores paralelos

Resumen





Clasificación de los computadores paralelos

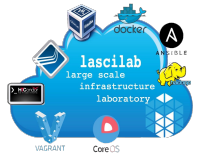


Lo que aprenderás hoy

¿Cómo OpenMP puede mejorar el rendimiento de tus experimentos?

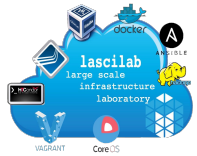
- Resumen de la sección anterior
- Uso del Compilador GCC/G++
  - ¿Que es?
  - ¿Qué puede hacer?
  - ¿Cómo habilitar optimizaciones?
- OpenMP

Las CPUs poseen características que puede ser habilitadas por medio del compilador y nuestra habilidad para programar



¿Qué es un Compilador?

- El compilador es una herramienta que permite **traducir** nuestro código (en un lenguaje de programación) en **código máquina optimizado**.
- El compilador hace su mejor esfuerzo por optimizar el código. Sin embargo, no siempre encuentra la mejor forma de hacerlo ya que no cuenta con mucha información de nuestro código.



¿Qué puede hacer?

El compilador se basa en **heurísticas** para determinar la forma más adecuada de optimizar el código, este puede llevar a cabo tareas como:

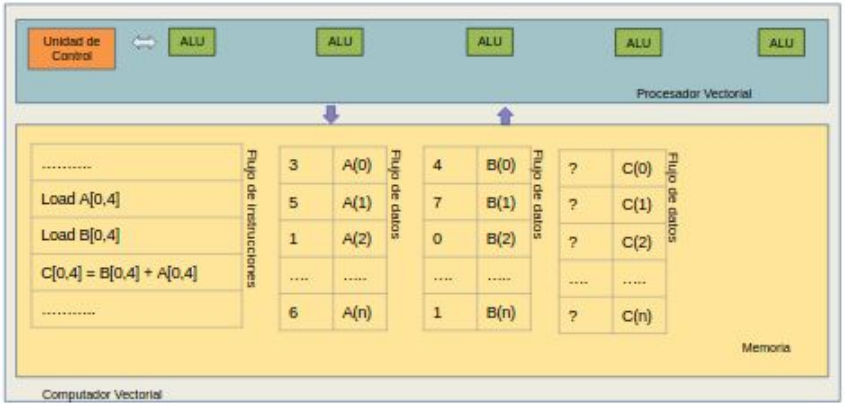
- Determinar si hay paralelismo a nivel de instrucción.

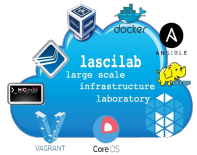
Sequential Execution	Instruction-Level Parallelism
1. a = 10 + 5 2. b = 12 + 7 3. c = a + b Instructions: 3 Cycles: 3	1.A. a = 10 + 5 1.B. b = 12 + 7 2. c = a + b Instructions: 3 Cycles: 2 (-33%)

¿Qué puede hacer?

El compilador se basa en **heurísticas** para determinar la forma más adecuada de optimizar el código, este puede llevar a cabo tareas como:

- Determinar si se puede **vectorizar** los *loops*

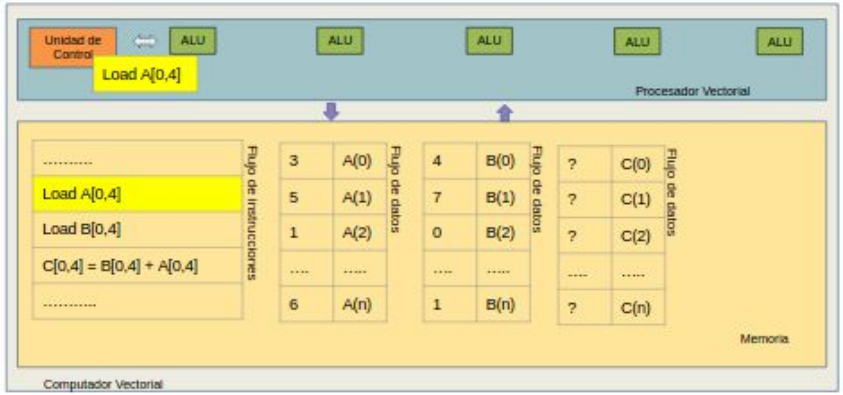




¿Qué puede hacer?

El compilador se basa en **heurísticas** para determinar la forma más adecuada de optimizar el código, este puede llevar a cabo tareas como:

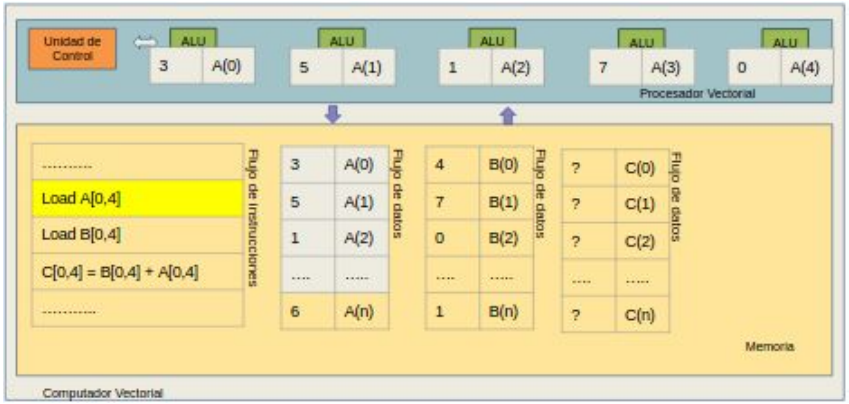
- Determinar si se puede **vectorizar** los *loops*

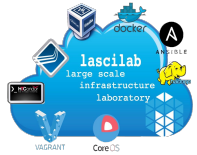


¿Qué puede hacer?

El compilador se basa en **heurísticas** para determinar la forma más adecuada de optimizar el código, este puede llevar a cabo tareas como:

- Determinar si se puede **vectorizar** los *loops*

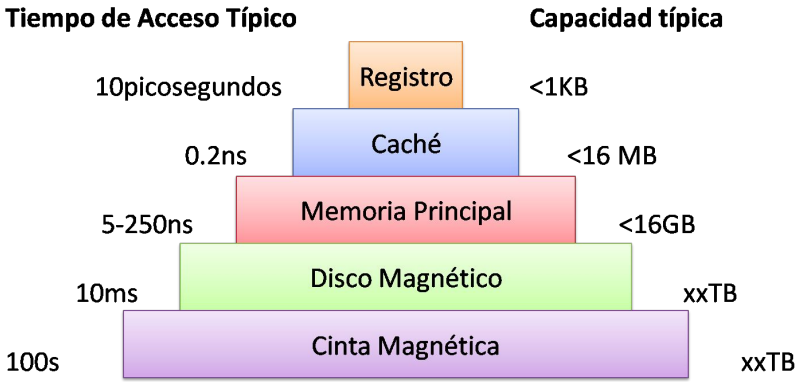


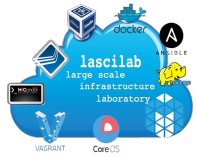


¿Qué puede hacer?

El compilador se basa en **heurísticas** para determinar la forma más adecuada de optimizar el código, este puede llevar a cabo tareas como:

- Reordenar las variables para reducir los accesos a memoria.





¿Qué puede hacer?

El compilador se basa en **heurísticas** para determinar la forma más adecuada de optimizar el código, este puede llevar a cabo tareas como:

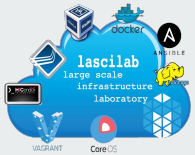
- Eliminar el cálculo de variables que no son usadas.
- etc.

```
int foo(void)
{
  int a = 24;
  int b = 25; /* Assignment to dead variable */
  int c;
  c = a * 4;
  return c;
  b = 24; /* Unreachable code */
  return 0;
}
```

Veamos cómo habilitar las optimizaciones del compilador

~~`git clone git@github.com:DonAurelio/openmp-workshop.git`~~

```
cd openmp-workshop/1.compiler
```

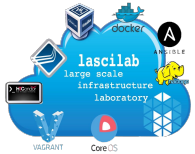


Lo que aprenderás hoy

¿Cómo OpenMP puede mejorar el rendimiento de tus experimentos?

- Resumen de la sección anterior
- Uso del Compilador GCC/G++
- **OpenMP**
  - **¿Qué es?**
  - **¿Qué problema intenta solucionar?**
  - **¿Cómo intenta solucionarlo?**
  - **Directivas OpenMP - Taller**

OpenMP



¿Qué es?

OpenMP (Open Multi-Processing) es un **modelo de programación** y un **lenguaje basado en directivas de compilación**.

Permite expresar el paralelismo en las aplicaciones por medio de **anotaciones en el código.**

El programa paralelizado toma ventaja de los múltiples núcleos del computador.

¿Cómo se usa?

```
#include <omp.h>


int main(void){
  #pragma omp parallel
  { // Start Parallel Region
    // some code .. to be executed in parallel
  } // End Parallel Region
  return 0;
}
```

OpenMP

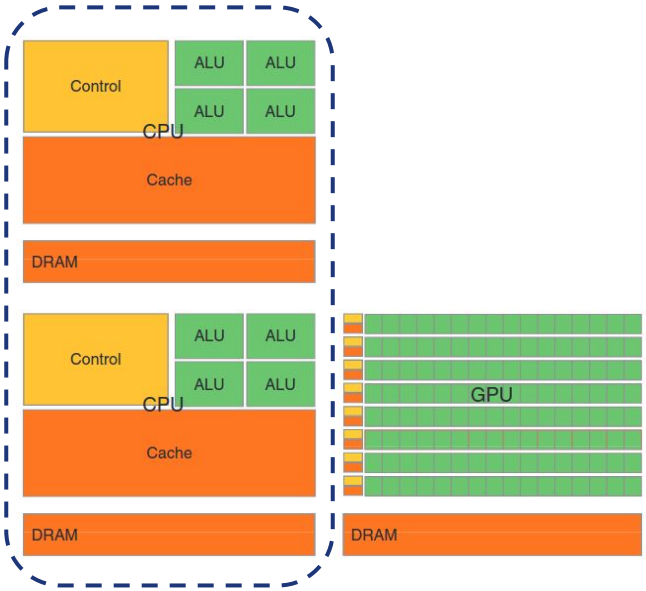
¿Qué es?

OpenMP (Open Multi-Processing) es un **modelo de programación** y un **lenguaje basado en directivas de compilación**.

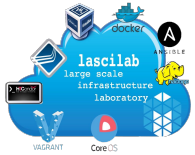
Permite expresar el paralelismo en las aplicaciones por medio de **anotaciones en el código.**

El programa paralelizado toma ventaja de los múltiples núcleos del computador.

¿Donde se usa?



OpenMP



¿Qué es?

OpenMP (Open Multi-Processing) es un **modelo de programación** y un **lenguaje basado en directivas de compilación**.

Permite expresar el paralelismo en las aplicaciones por medio de **anotaciones en el código.**

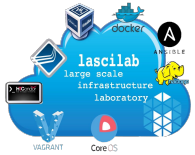
El programa paralelizado toma ventaja de los **múltiples núcleos del computador**.

¿Donde lo puedo encontrar?

Es una característica presente en los compiladores, puede ser habilitada usando banderas de compilación:

- **GNU gcc/g++**, -fopenmp
- **PGI compiler**, -mp
- Otros

OpenMP



¿Existen otras herramientas para paralelizar nuestras aplicaciones?

Cada lenguaje de programación posee un conjunto de herramientas para paralelizar nuestras aplicaciones.

Para el lenguaje de programación C/C++ vamos a usar **OpenMP** aunque existen más herramientas.

Octave

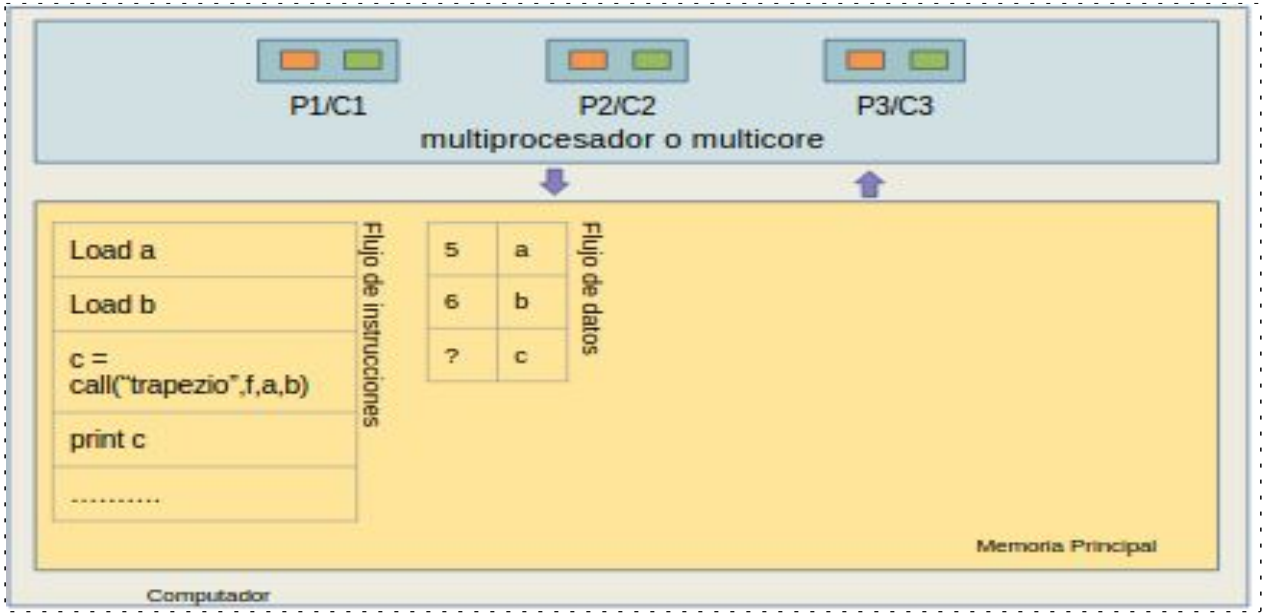
Wolfram Mathematica 10

MATLAB
The Language of Technical Computing

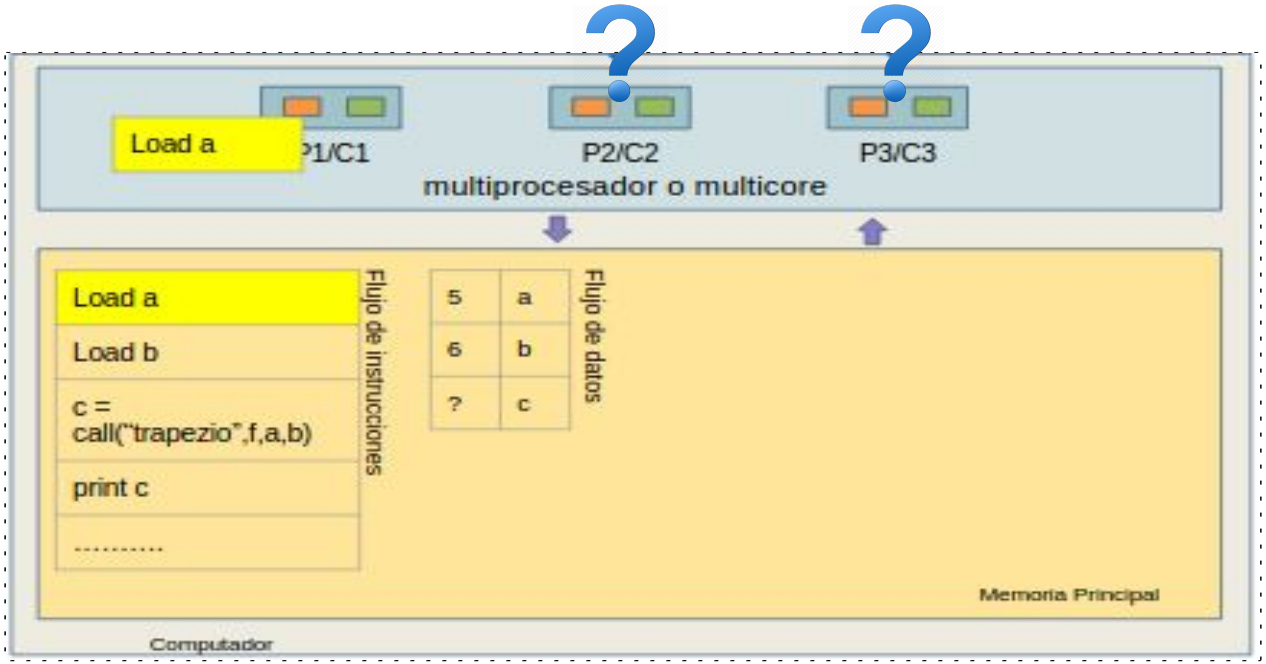
Scilab

python™

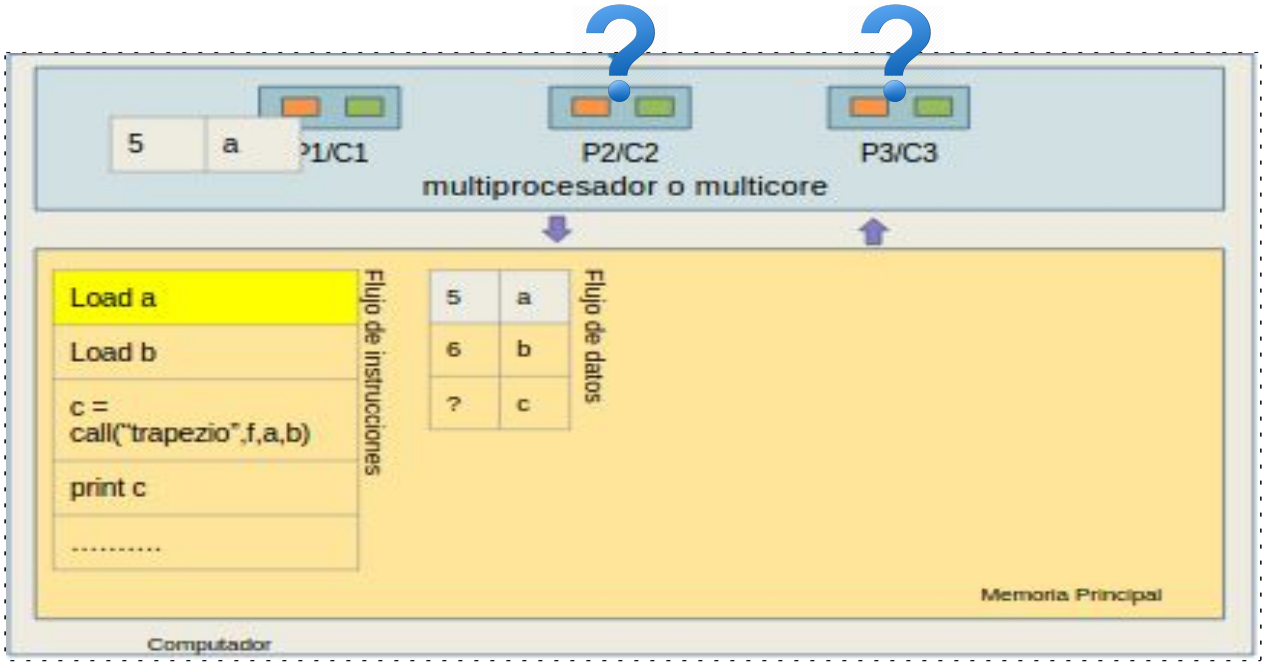
¿Que problema intenta solucionar OpenMP?



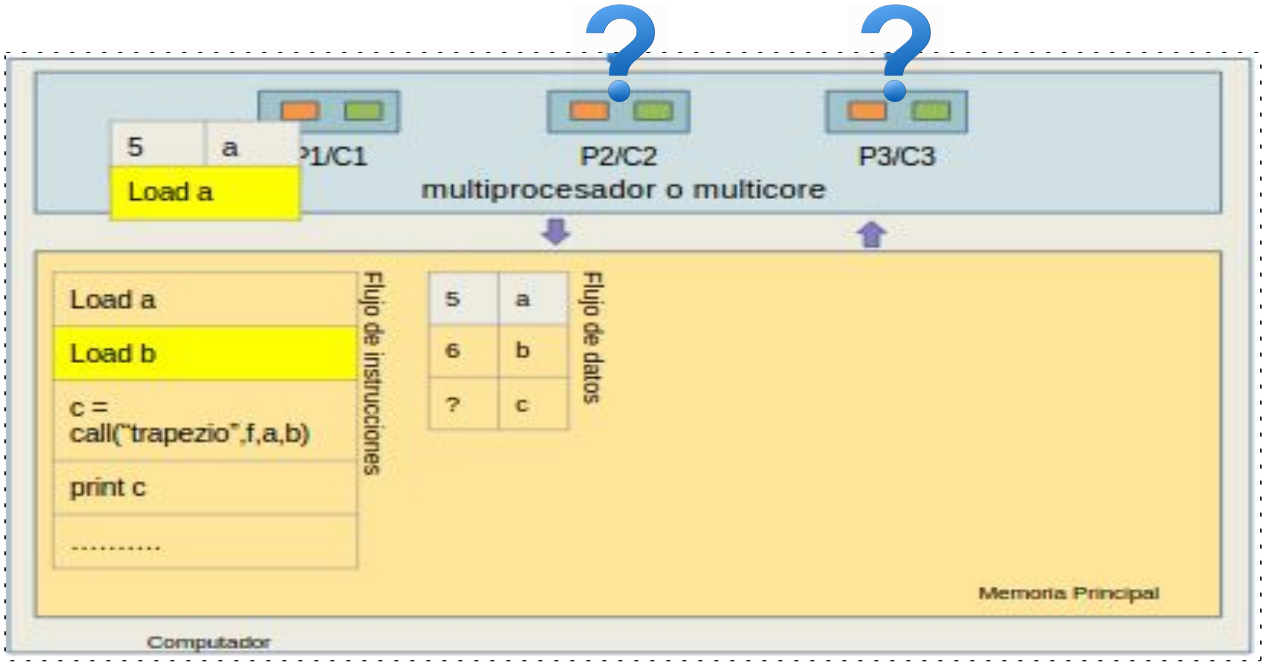
¿Que problema intenta solucionar?



¿Que problema intenta solucionar?



¿Que problema intenta solucionar?

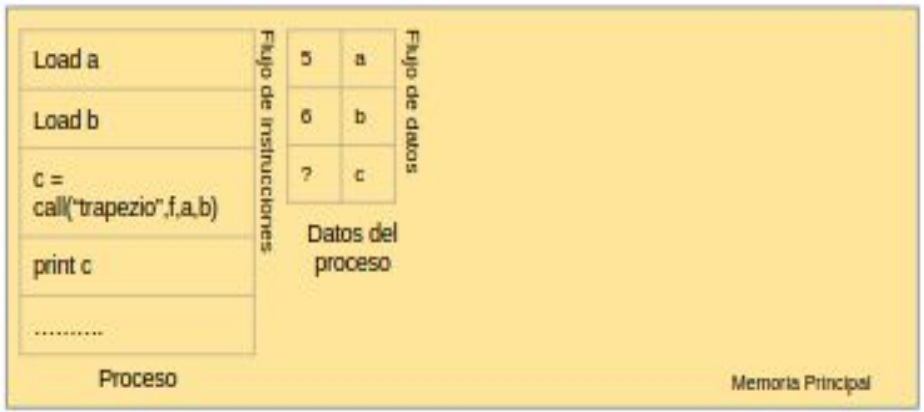


¿Que problema intenta solucionar?

¿Cómo intenta solucionarlo?

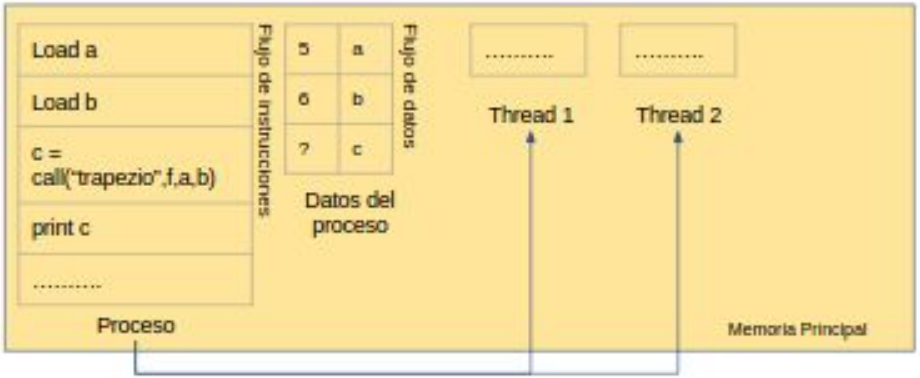
¿Cómo intenta solucionarlo?

Un **proceso** es un programa en ejecución, los programas en ejecución son cargados en la **memoria principal**.



¿Cómo intenta solucionarlo?

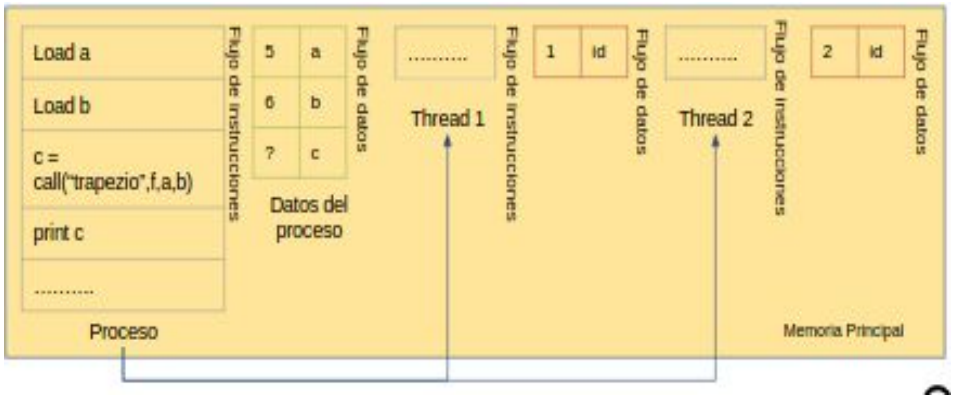
Un **proceso** puede tener **hijos** (**hilos**) le pueden ayudar al padre a realizar el trabajo pesado.



¿Cómo intenta solucionarlo?

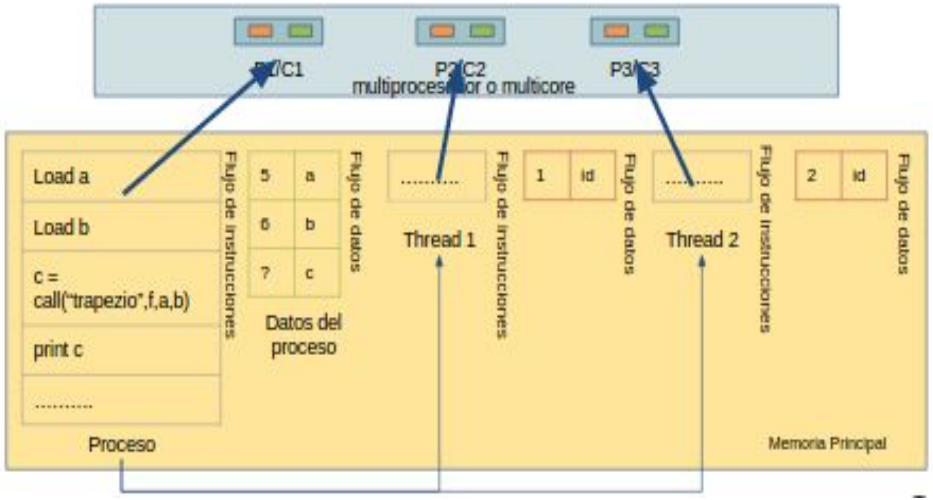
Un **hilo**, es un proceso ligero.

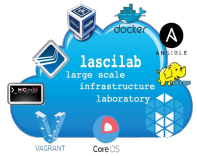
- Los hilos pueden acceder a los datos del proceso padre.
- Los hilos pueden tener su propios datos privados.



¿Cómo intenta solucionarlo?

De esta forma, teniendo **hilos o procesos ligeros**, cada proceso puede ejecutarse en un procesador diferente (**paralelismo**) o compartir procesador con otros procesos (**concurrencia**).





Directivas OpenMP

Una **directiva OpenMP** es una anotación llevada a cabo sobre el código secuencial, para paralelizar o crear un equipo de hilos, los cuales pueden ser orquestados para hacer la **misma** o **distintas tareas** en **paralelo**.

```
#pragma omp directive-name [clause ...]
```

```
#include <omp.h>

int main(void){
  // sequential region
  #pragma omp parallel
  { // Start Parallel Region
    print("Hello World");
  } // End Parallel Region
  // sequential region
  return 0;
}
```

Se debe importar la librería de OpenMP

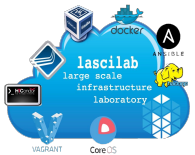
El código corre de forma secuencial **antes** de la región paralela.

Creación de la región paralela

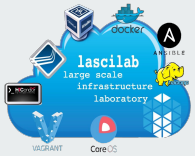
Código paralelizado

El código corre de forma secuencial **después** de la región paralela.

```
#pragma omp parallel
```



La directiva "*Parallel*"



Click Taller

- El taller es guiado, después de cada ejercicio se hará una retroalimentación.
- Los que terminan primero deben ayudar a sus compañeros.
- El taller sugiere preguntas para discutir durante la presentación.
- **IMPORTANTE: Si no entiende háganoslo saber**

Directiva “*Parallel*” - ejercicio

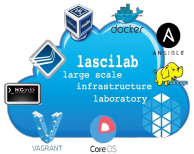
```
git clone git@github.com:DonAurelio/openmp-workshop.git

cd openmp-workshop/4.parallel_directive
```

Master Thread

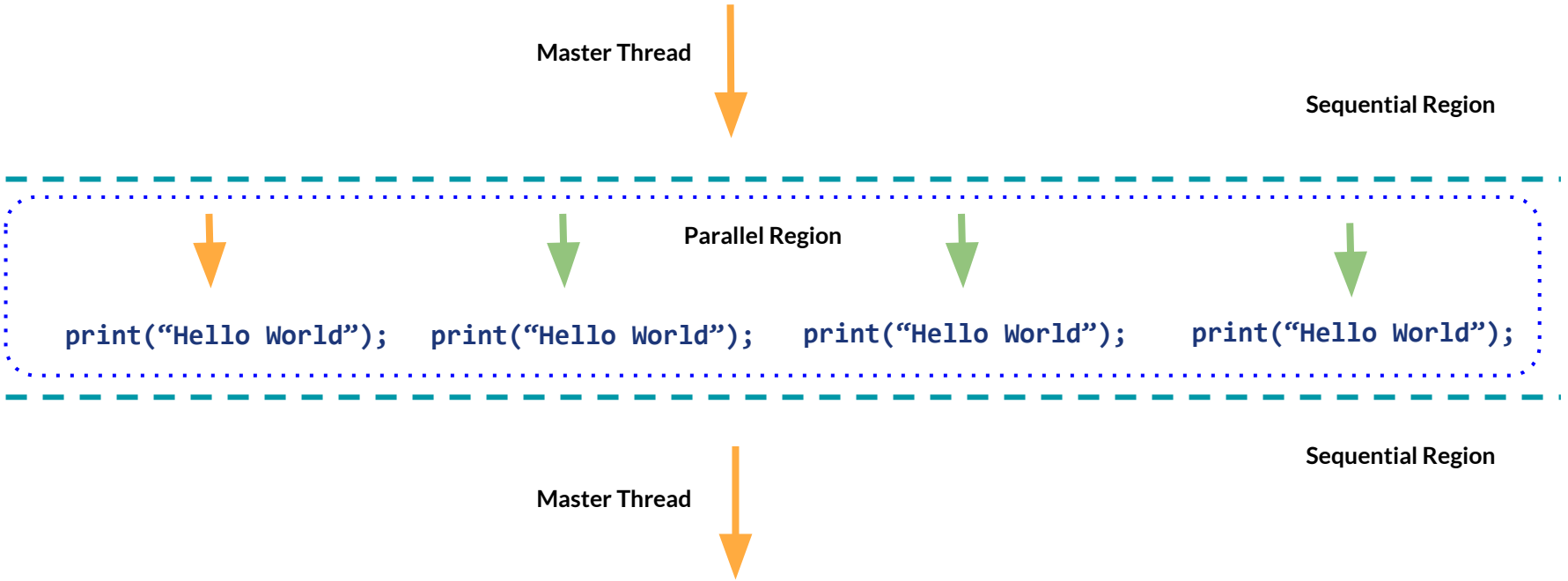
```
print(“Hello World”);
```

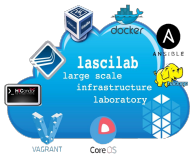
Master Thread



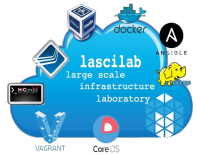
La directiva “*Parallel*”

```
#pragma omp parallel
```





La directiva "*Parallel*"

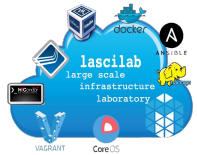


Trabajo Compartido en OpenMP

Para **dividir el trabajo** entre el equipo de hilos creados por la región paralela, OpenMP.

```
#pragma omp for [clause ...]
```

```
#pragma omp sections [clause ...]
```



Trabajo Compartido en OpenMP

Para **dividir el trabajo** entre el equipo de hilos creados por la región paralela, OpenMP.

```
#pragma omp for [clause ...]
```

Permite dividir las iteraciones de un **ciclo *for*** entre el número de hilos creados por la región paralela.

- Es útil cuando se desea procesar vectores o matrices muy grandes.

Directiva “*for*” para procesar arreglos/vectores- ejercicio

```
git clone git@github.com:DonAurelio/openmp-workshop.git

cd openmp-workshop/5.parallel_for
```

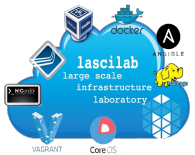
```
#include <omp.h>
```

Se debe importar la librería de OpenMP

```
int main(void){
  // sequential region
  #pragma omp parallel shared(...)
  { // Start Parallel Region
    #pragma omp for
    for(int i = 0; i < N; ++i){
      ........
    }
  } // End Parallel Region
  // sequential region
  return 0;
}
```

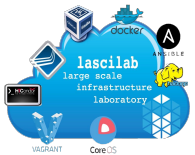
Creación de la región paralela + Variables que compartirán los hilos dentro de la región paralela

¿Cómo los hilos se dividen el trabajo?

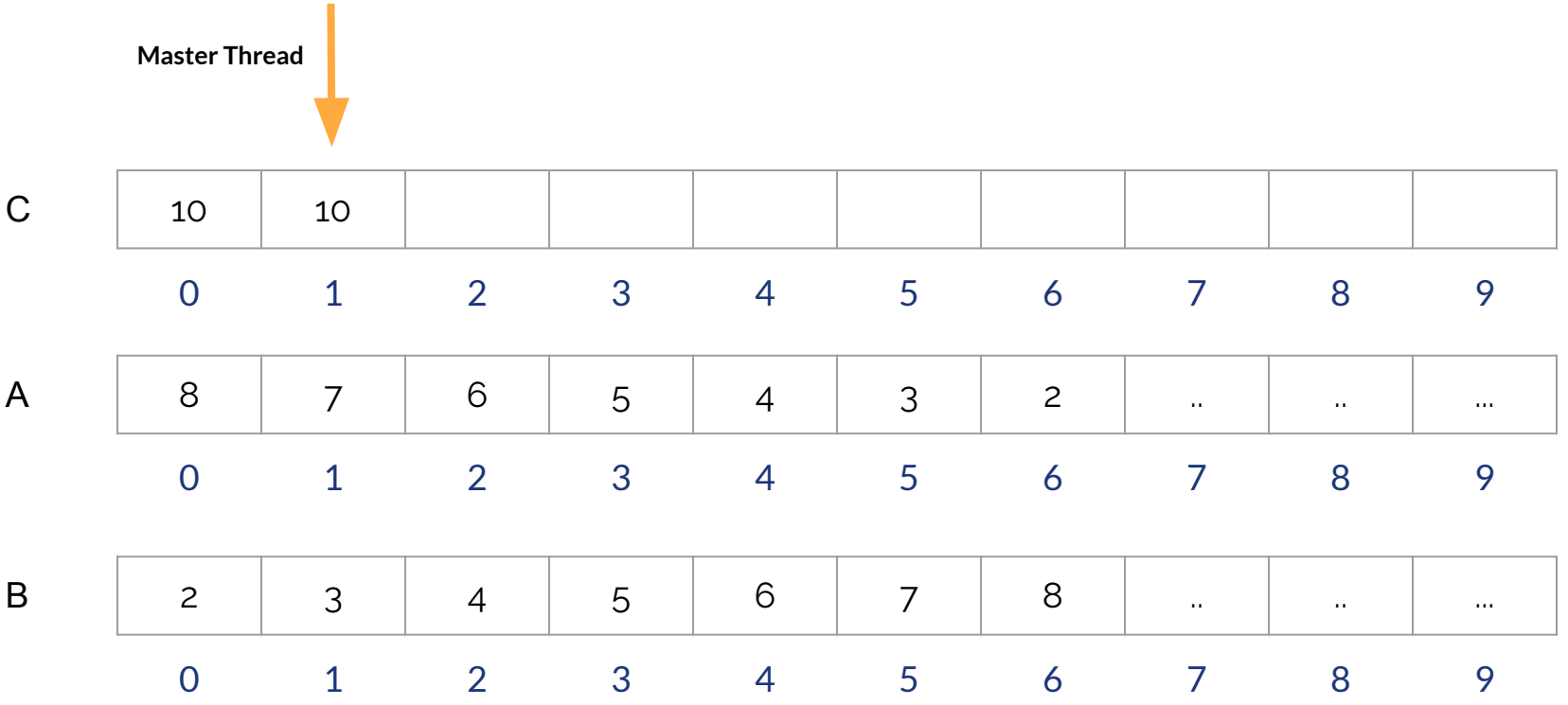


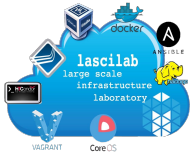
La directiva “*for*”

C	10									
	0	1	2	3	4	5	6	7	8	9
A	8	7	6	5	4	3	2	..	..	...
	0	1	2	3	4	5	6	7	8	9
B	2	3	4	5	6	7	8	..	..	...
	0	1	2	3	4	5	6	7	8	9

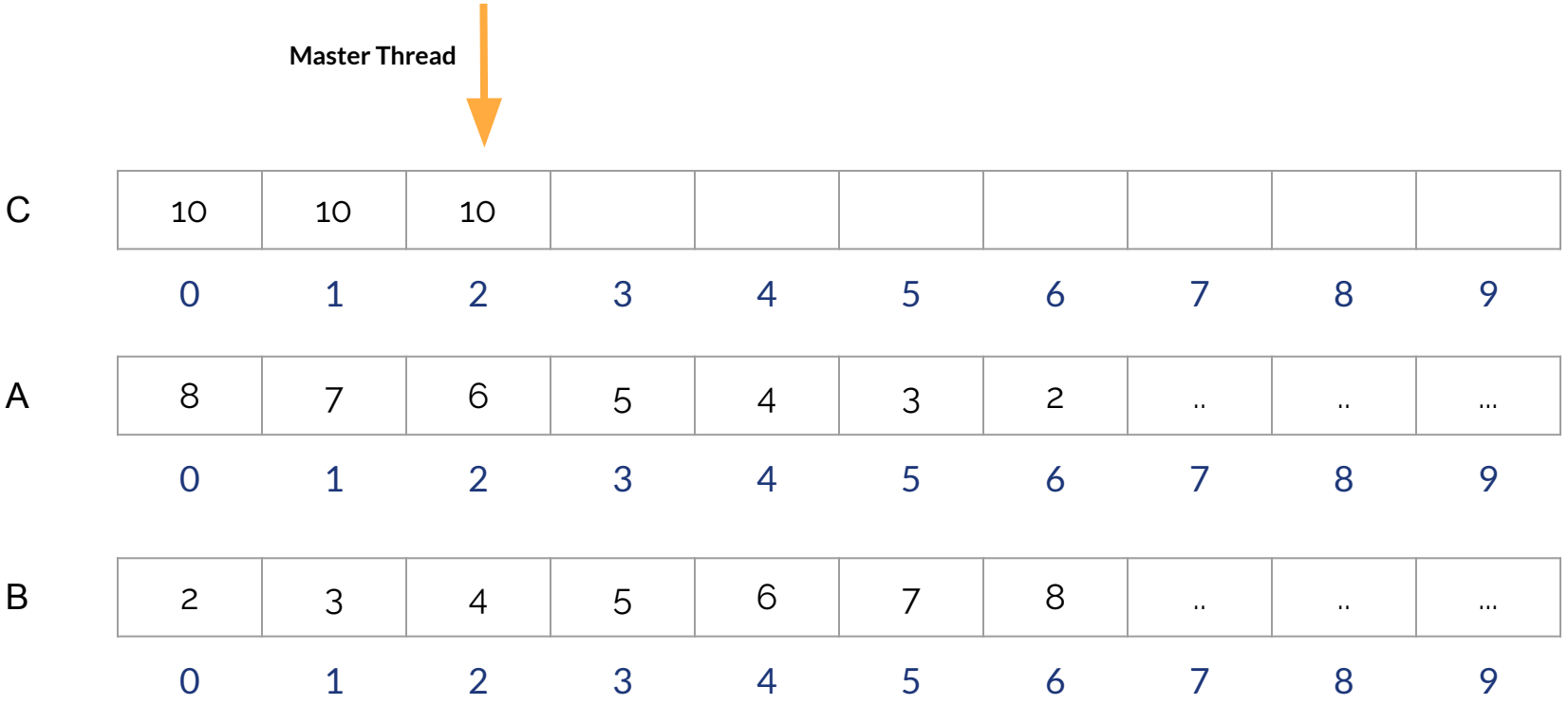


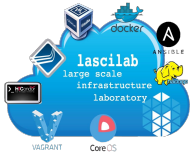
La directiva “*for*”





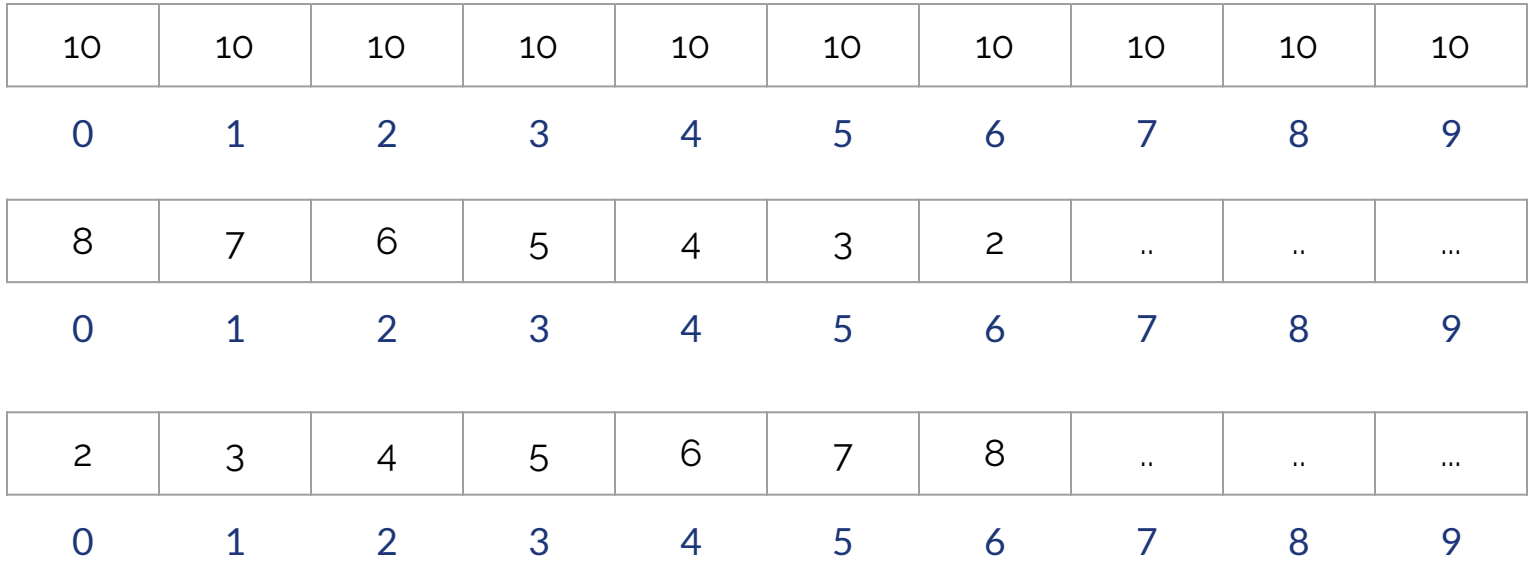
La directiva “*for*”

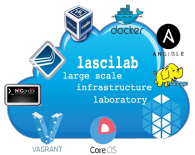




La directiva “*for*”

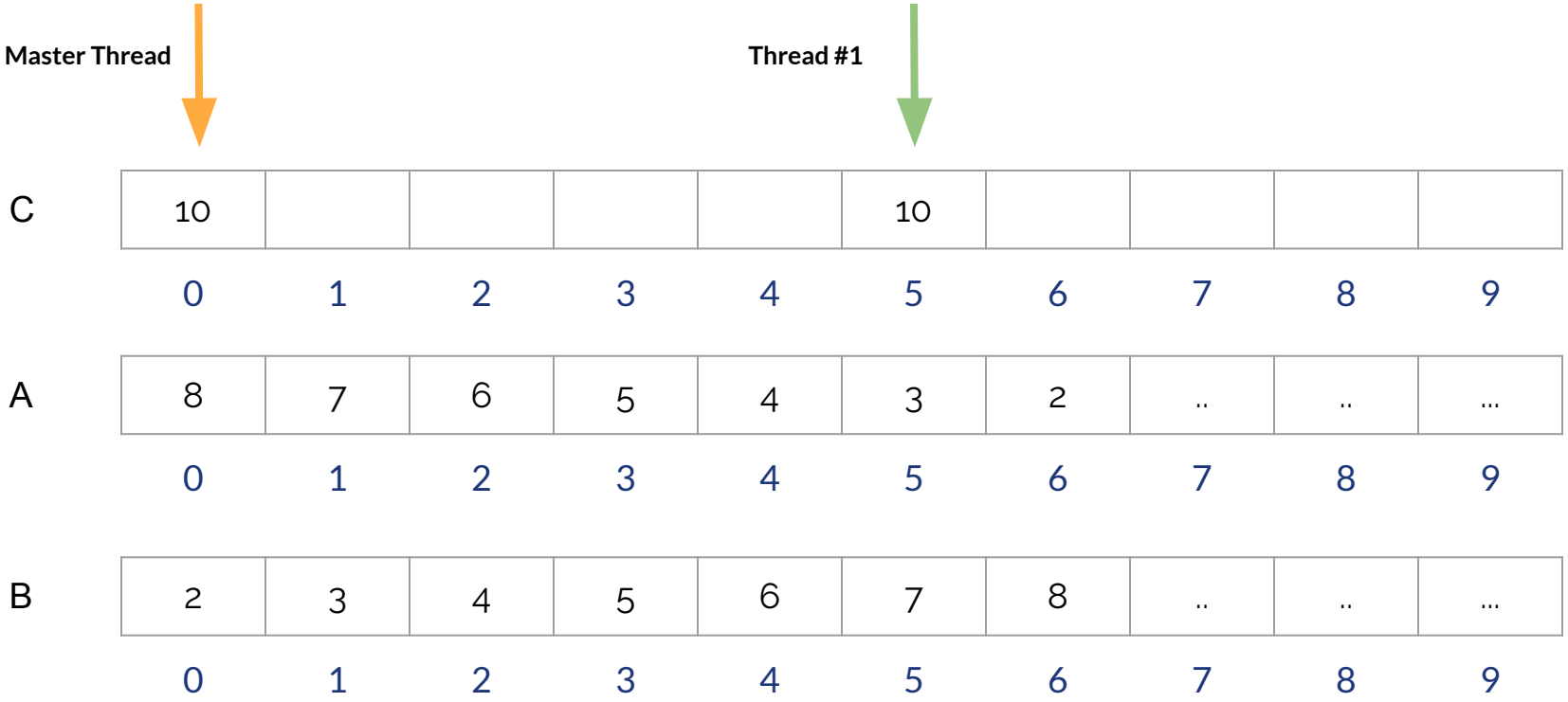
Master Thread

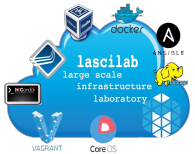




La directiva “*for*”

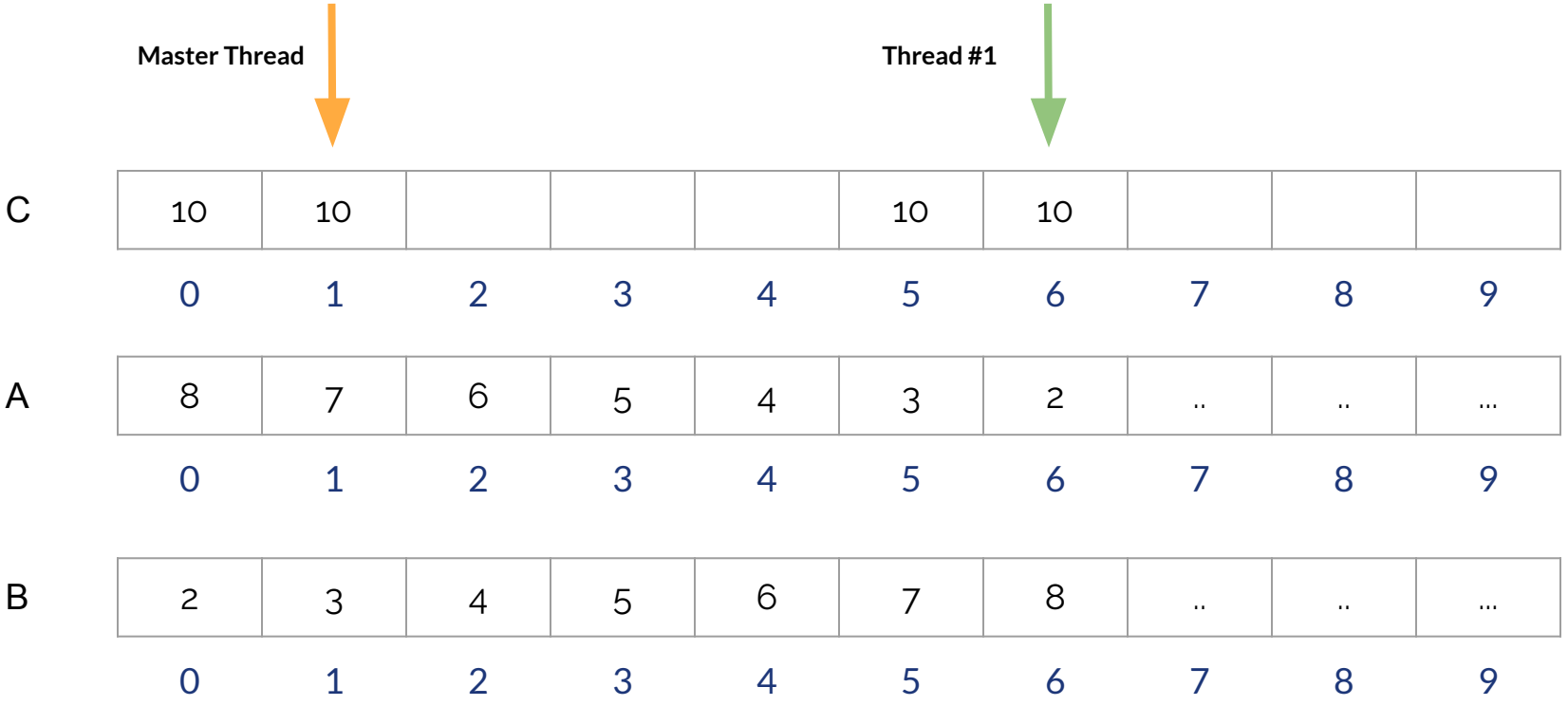
#pragma omp for

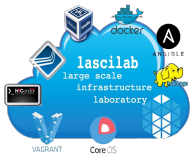




La directiva “*for*”

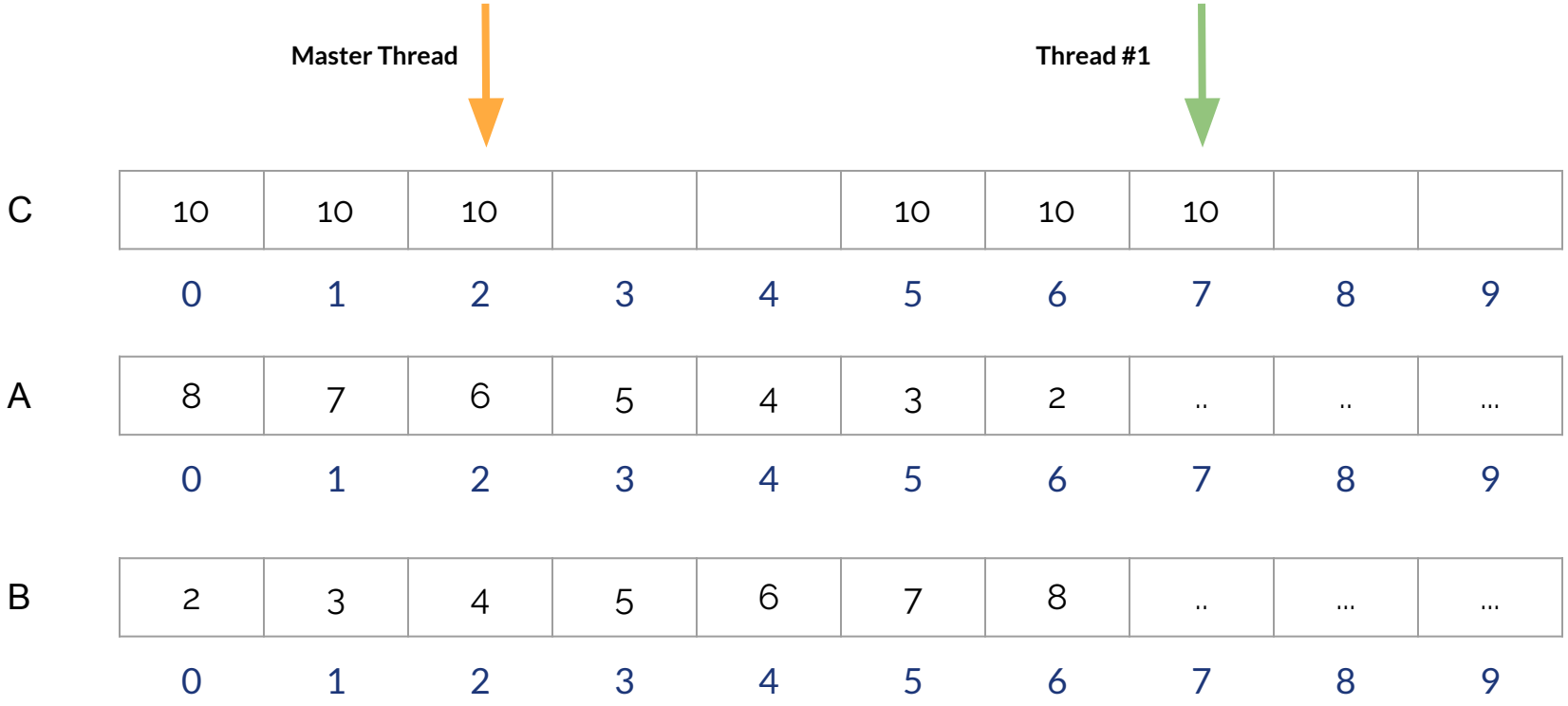
```
#pragma omp for
```

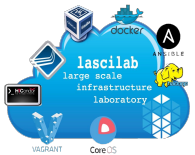




La directiva “*for*”

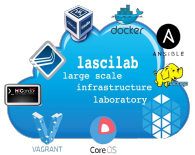
#pragma omp for





La directiva “*for*”

#pragma omp for



La directiva “*for*”

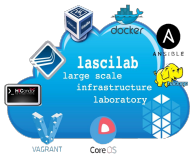
Directiva “*for*” para procesar matrices- actividad

```
git clone git@github.com:DonAurelio/openmp-workshop.git

cd openmp-workshop/5.parallel_for
```

```
#include <omp.h>

int main(void){
 for(int i = 0; i < N; ++i){
    for(int j = 0; j < N; ++j){
    ........
    }
  }
  return 0;
}
```



2					

C

=

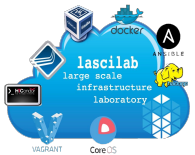
1	2	3	4	5	6
7	8	9	10	11	12
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..

A

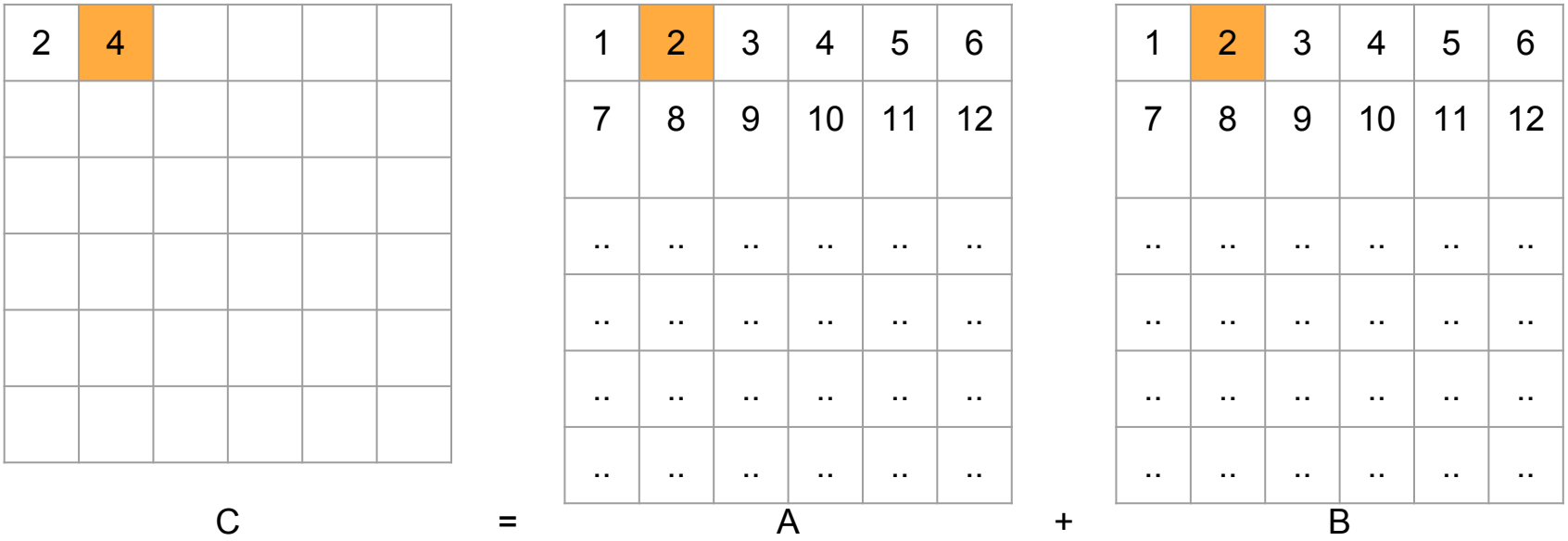
+

1	2	3	4	5	6
7	8	9	10	11	12
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..

B



La directiva “for” para procesar matrices



La directiva “for” para procesar matrices

2	4	6			

C

=

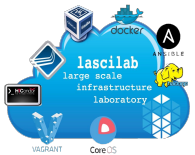
1	2	3	4	5	6
7	8	9	10	11	12
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..

A

+

1	2	3	4	5	6
7	8	9	10	11	12
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..

B



La directiva “for” para procesar matrices

2	4	9	8	10	12
14	16	18	20	22	24
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..

C

=

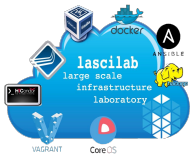
1	2	3	4	5	6
7	8	9	10	11	12
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..

A

+

1	2	3	4	5	6
7	8	9	10	11	12
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..

B



La directiva “for” para procesar matrices

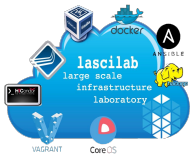
```
#include <omp.h>
```

Se debe importar la librería de OpenMP

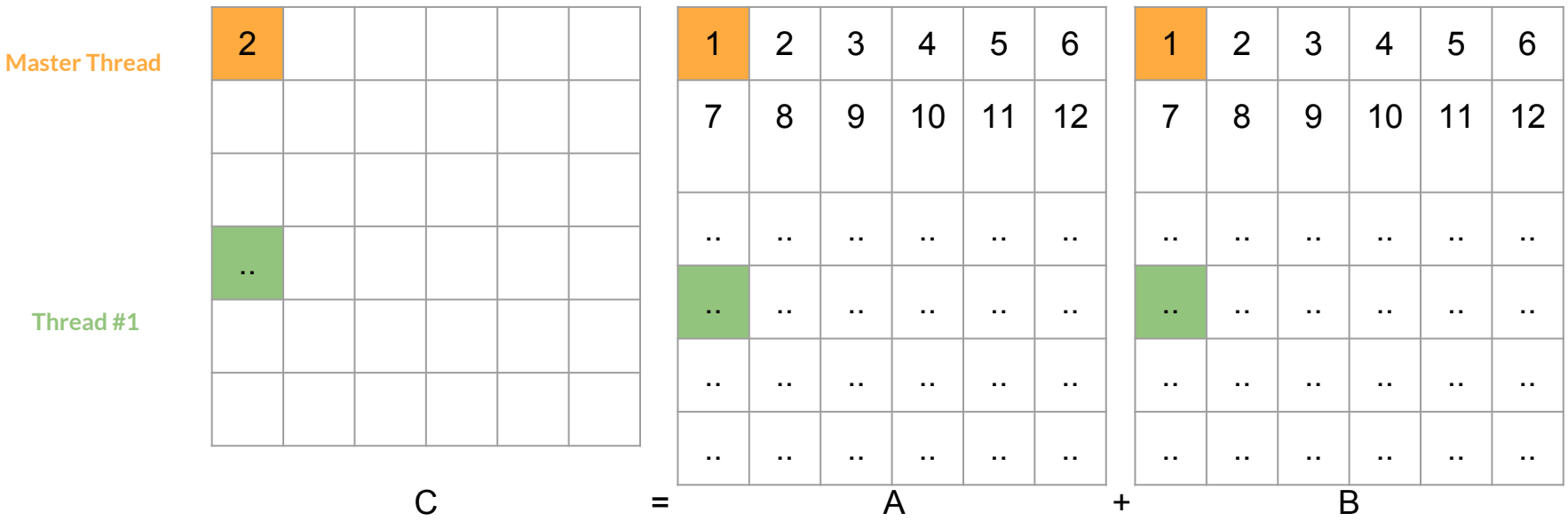
```
int main(void){
  #pragma omp parallel shared(...)
  {
   #pragma omp for
   for(int i = 0; i < N; ++i){
     for(int j = 0; j < N; ++j){
     ........
     }
   }
  }
  return 0;
}
```

Creación de la región paralela + Variables que compartirán los hilos dentro de la región paralela

Paralelize las **filas** o el **primer ciclo**



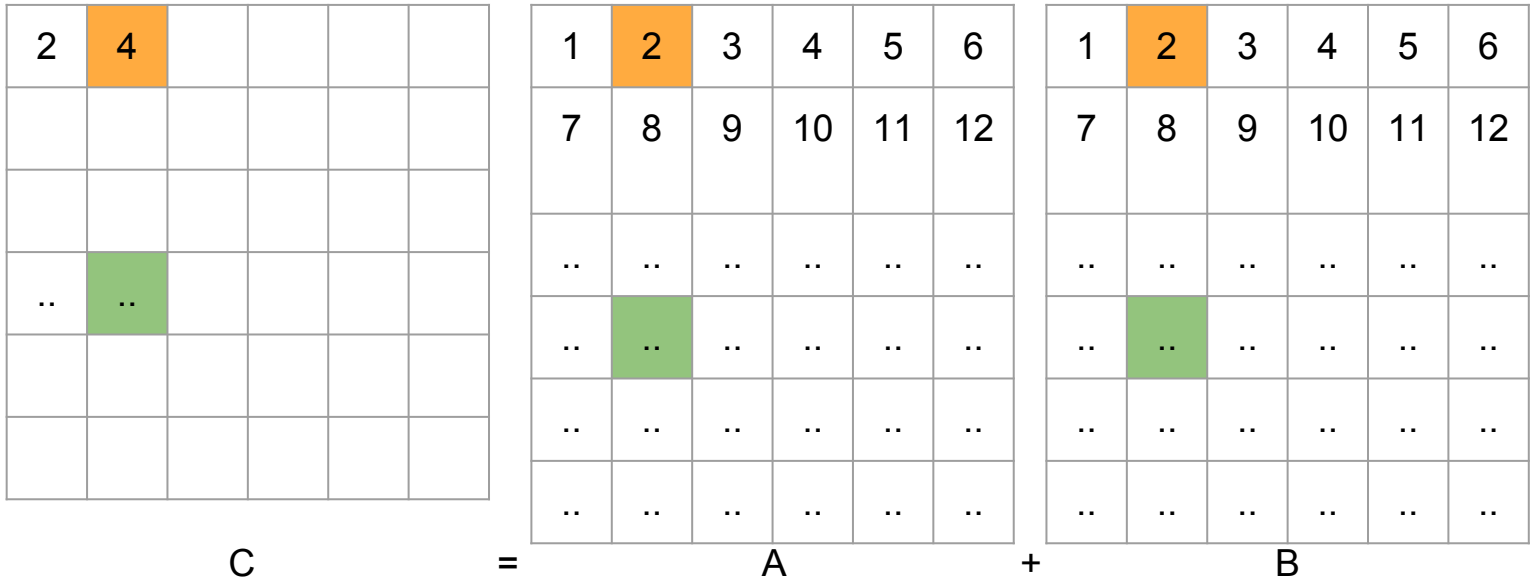
La directiva "*for*" para procesar matrices

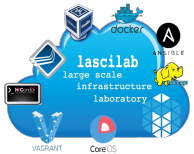


La directiva “*for*” para procesar matrices

Master Thread

Thread #1

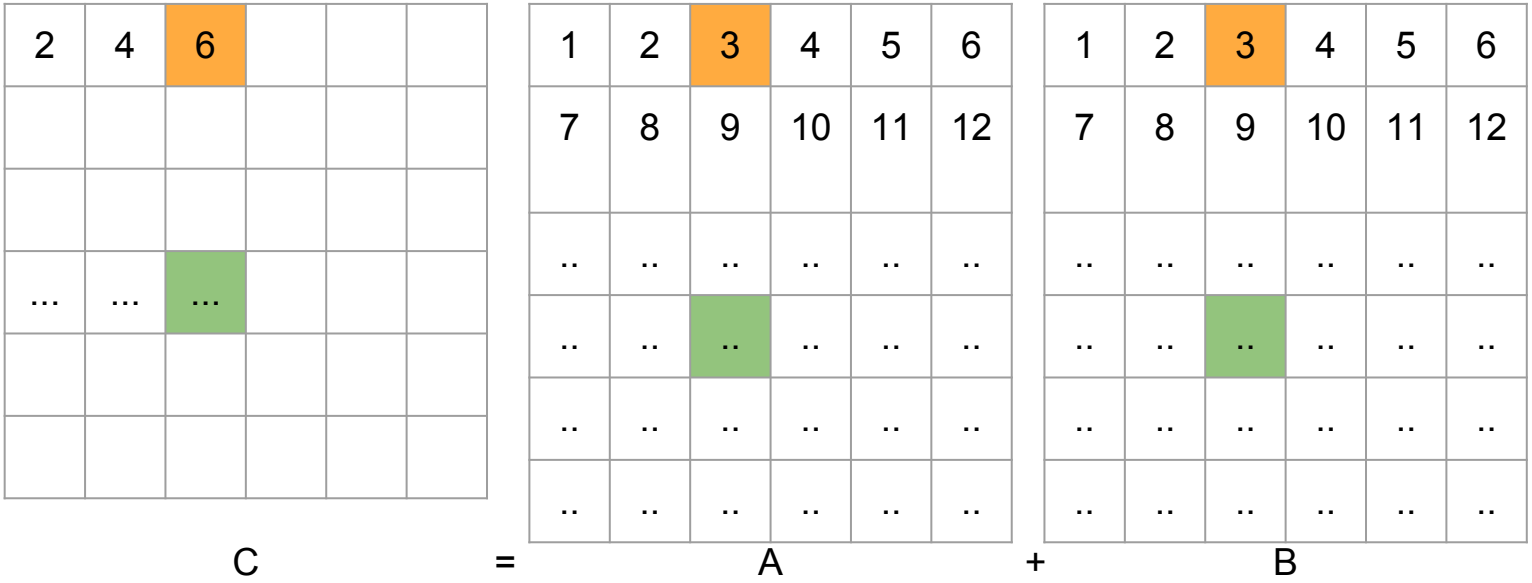


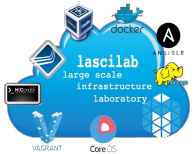


La directiva “*for*” para procesar matrices

Master Thread

Thread #1





La directiva “*for*” para procesar matrices

Master Thread

Thread #1

2	4	9	8	10	12
14	16	18	20	22	24
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..

C

=

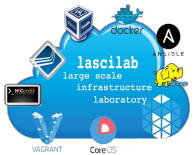
1	2	3	4	5	6
7	8	9	10	11	12
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..

A

+

1	2	3	4	5	6
7	8	9	10	11	12
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..

B



La directiva “*for*” para procesar matrices

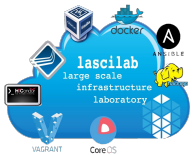
```
#include <omp.h>
```

Se debe importar la librería de OpenMP

```
int main(void){
  #pragma omp parallel shared(...)
  {
   for(int i = 0; i < N; ++i){
     #pragma omp for
     for(int j = 0; j < N; ++j){
     ........
     }
   }
  }
  return 0;
}
```

Creación de la región paralela + Variables que compartirán los hilos dentro de la región paralela

Paralelize las **columnas** de cada fila



La directiva “*for*” para procesar matrices

Master Thread Thread #1

C

=

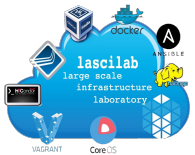
1	2	3	4	5	6
7	8	9	10	11	12
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..

A

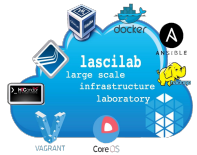
+

1	2	3	4	5	6
7	8	9	10	11	12
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..
..	..	..	..	..	..

B



La directiva “*for*” para procesar matrices - **¿Donde empezaria cada hilo? ¿Donde terminarian?**



Trabajo Compartido en OpenMP

Para **dividir el trabajo** entre el equipo de hilos creados por la región paralela, OpenMP.

```
#pragma omp sections [clause ...]
```

Permite ejecutar **varias secciones** de código en paralelo.

- Es útil cuando tenemos varias funciones que son independientes y toman mucho tiempo en ejecutarse.

Directiva “*sections*” - ejercicio

```
git clone git@github.com:DonAurelio/openmp-workshop.git

cd openmp-workshop/6.sections
```

A continuación ...

Aurelio Vivas

Introducción a la Programación Paralela en OpenACC