

Procesamiento Masivo de datos: OpenMP II and MPI I

Alex Di Genova

22/09/2022

OpenMP

Repaso

OpenMP

Introduction

- **Definición : OpenMP**

- OpenMP es una interfaz de programación de aplicaciones (API) para la paralelización de sistemas de memoria compartida, utilizando C, C++ o Fortran.
- La API consta de directivas de compilador para especificar y controlar la paralelización, aumentada con funciones de tiempo de ejecución y variables de entorno.
- Corresponde al usuario identificar el paralelismo e insertar las estructuras de control apropiadas en el programa (directivas).
- En C/C++, la directiva se basa en la construcción **#pragma omp**.

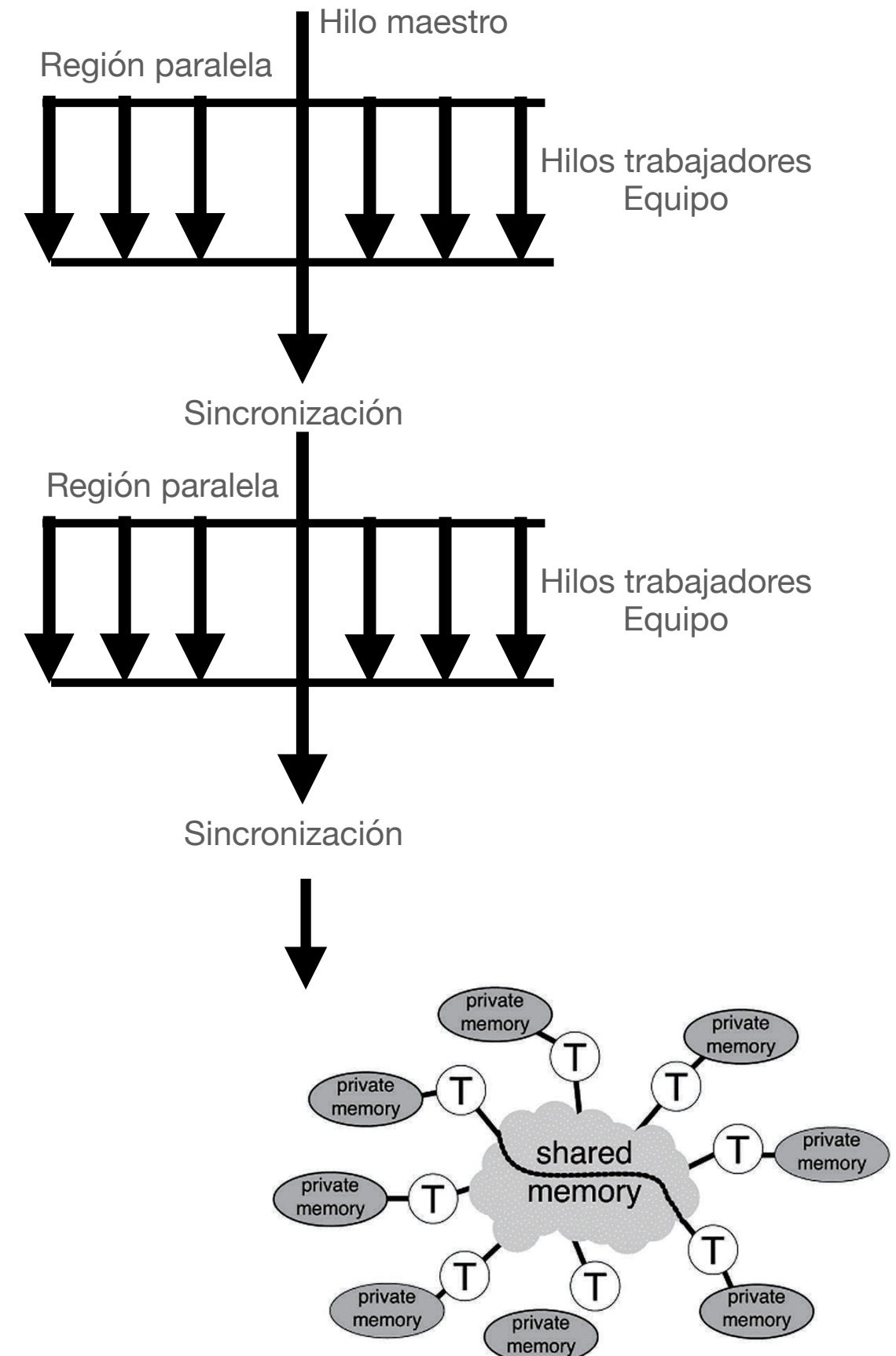
- **Syntax**

- **#pragma omp parallel** [clause[clause], ...] new-line

Structured block

- Es responsabilidad del programador identificar qué parte(s) del código se selecciona(n) para ejecutar en paralelo y usar las diversas construcciones para garantizar resultados correctos.
- También se debe especificar la naturaleza (privada o compartida) o el "alcance" de las variables.

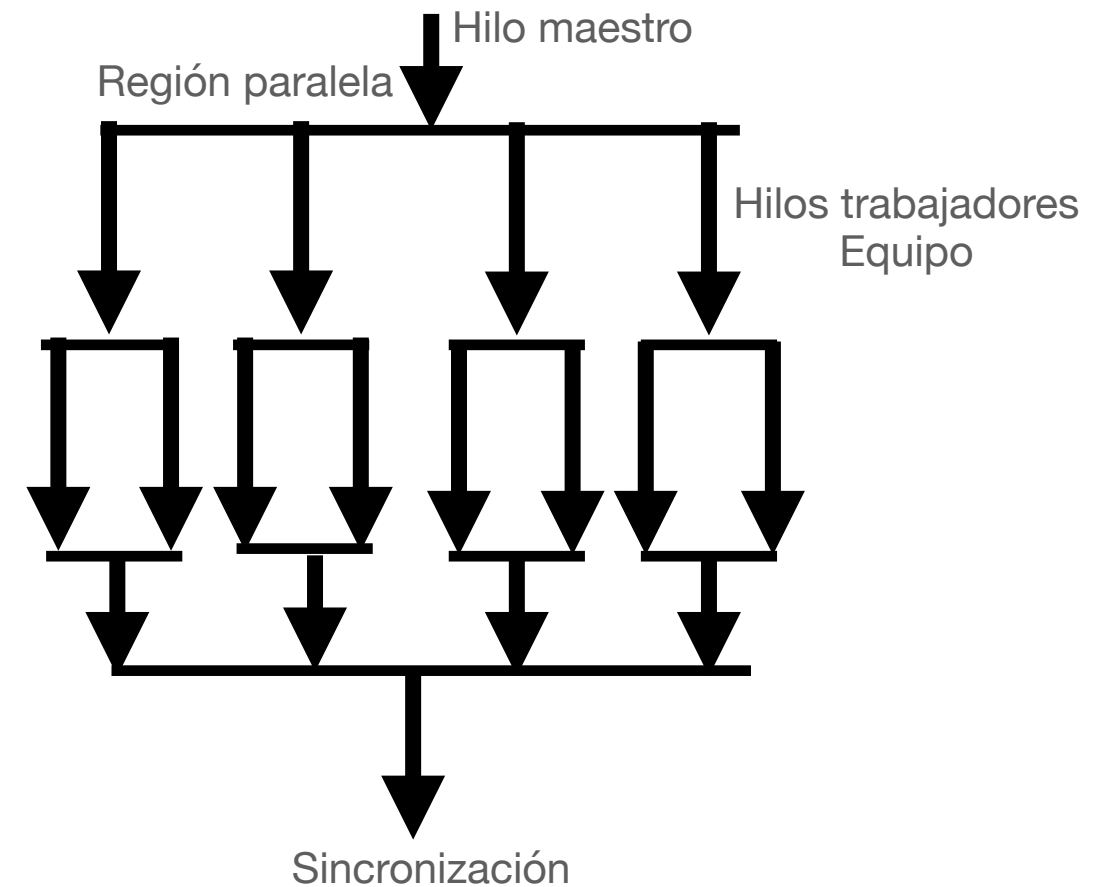
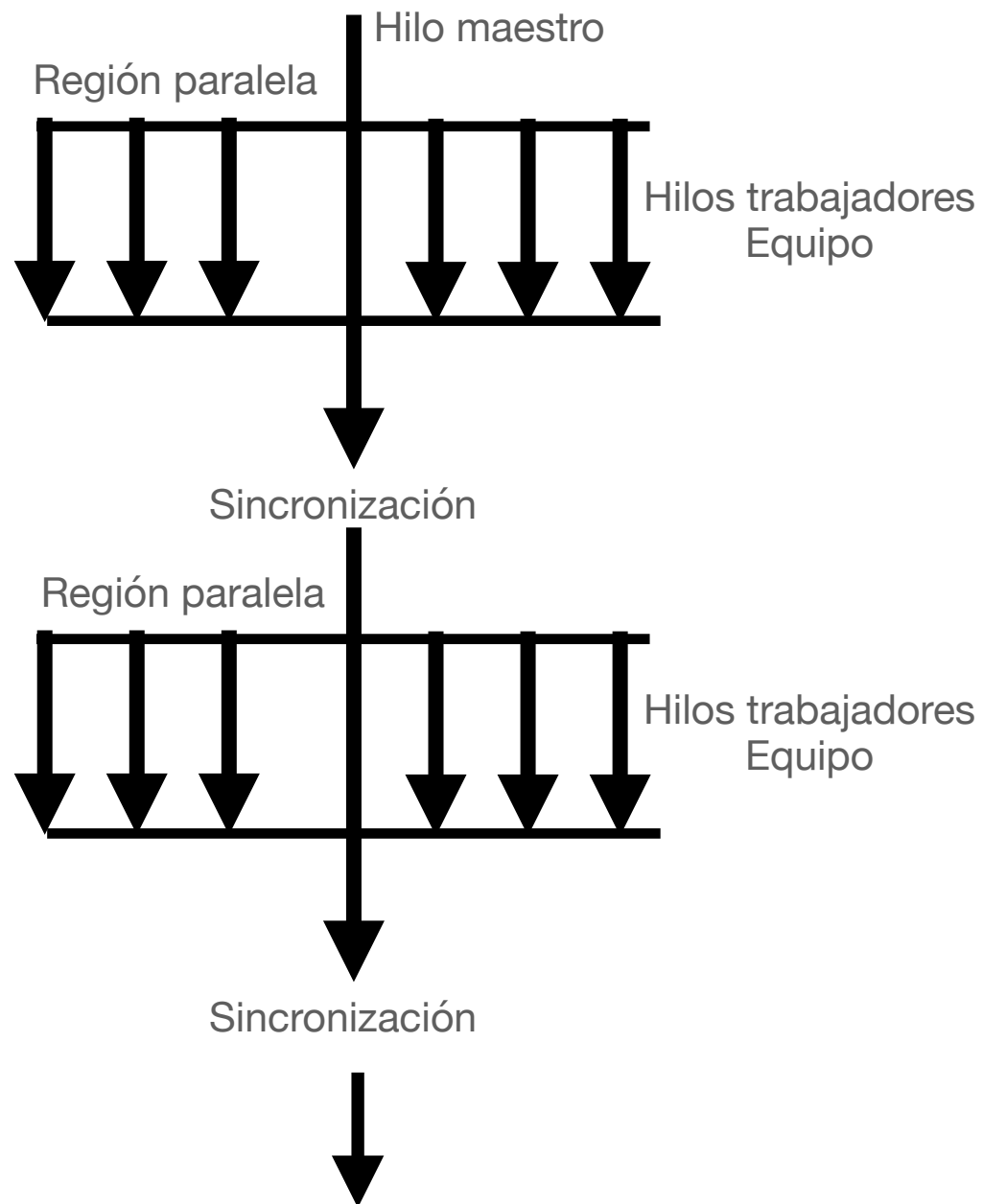
Modelo de Ejecución OpenMP (fork-join)



OpenMP

Paralelismo anidado

Modelo de Ejecución OpenMP (fork-join)



```
#pragma omp parallel num_threads(1)
{
    Work1();

    #pragma omp parallel num_threads(5)
    { //1 x 5 = 5 threads
        Work2();
    }
}
```

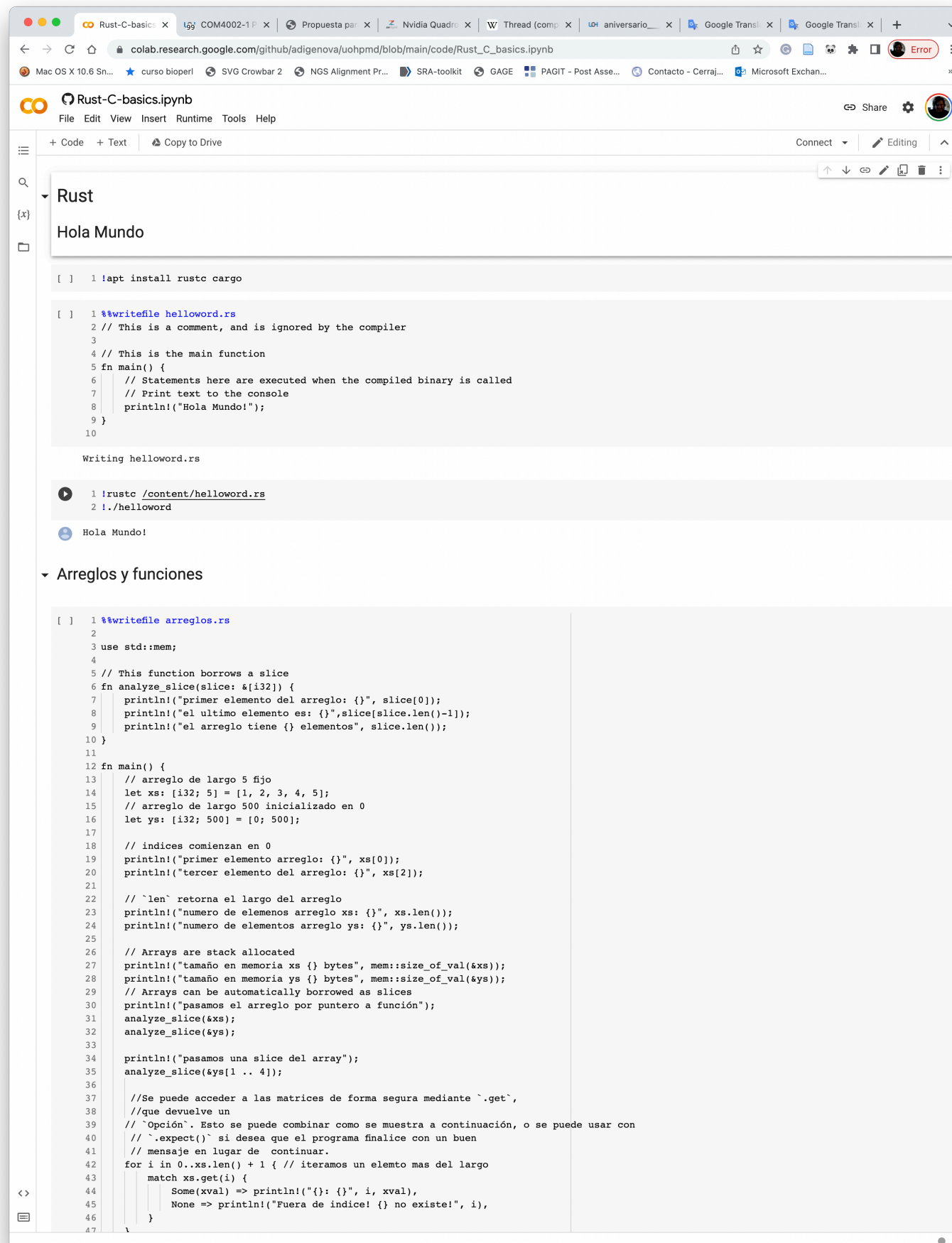
OpenMP

Funciones de tiempo de ejecución

- Estas funciones se pueden usar para consultar la configuración y también sobrescribir los valores iniciales, ya sea establecidos de forma predeterminada o especificados a través de variables de ambiente definidas antes del inicio del programa.
- Un ejemplo es el número de hilos utilizados para ejecutar una región paralela. El valor inicial depende de la implementación, pero a través de la variable de entorno OMP_NUM_THREADS, este número puede establecerse explícitamente antes de que se inicie el programa. Durante la ejecución del programa, la función **omp_set_num_threads()** se puede usar para aumentar o disminuir el número de hilos que se usarán en las siguientes regiones paralelas.

Función	Descripción
omp_set_num_threads	Establece el número de hilos.
omp_get_num_threads	Número de hilos en el equipo actual.
omp_get_max_threads	número de hilos en la siguiente región paralela.
omp_get_num_procs	Número de procesadores disponibles para el programa.
omp_get_thread_num	Número de hilo dentro de la región paralela.
omp_in_parallel	Comprueba si está dentro de una región paralela.
omp_get_dynamic	Comprueba si el ajuste del hilo está habilitado.
omp_set_dynamic	Habilita o deshabilita el ajuste del hilos.
omp_get_nested	Comprueba si el paralelismo anidado está habilitado.
omp_set_nested	Habilita o deshabilita el paralelismo anidado.

Practicar las funciones OpenMP en Google Colab



The screenshot shows a Google Colab notebook interface. The browser tabs at the top include 'Rust-C-basics', 'COM4002-1 F', 'Propuesta pa...', 'Nvidia Quad...', 'W Thread (comp...', 'aniversario...', 'Google Transl...', 'Google Transl...', and a '+' icon. The address bar shows the URL 'colab.research.google.com/github/adigenova/uohpmd/blob/main/code/Rust_C_basics.ipynb'. The notebook title is 'Rust-C-basics.ipynb' with a share icon and a settings icon. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The toolbar has '+ Code', '+ Text', 'Copy to Drive', 'Connect', 'Editing', and a vertical ellipsis. The left sidebar shows a file explorer with 'Rust' and 'Hola Mundo'. The main code area contains two code blocks. The first block, titled 'Rust', contains Rust code for a 'Hello World' program. The second block, titled 'Arreglos y funciones', contains Rust code for array operations. The output of the first block shows 'Hola Mundo!'. The output of the second block is empty.

```
[ ] 1 !apt install rustc cargo

[ ] 1 %%writefile helloworld.rs
2 // This is a comment, and is ignored by the compiler
3
4 // This is the main function
5 fn main() {
6     // Statements here are executed when the compiled binary is called
7     // Print text to the console
8     println!("Hola Mundo!");
9 }
10

Writing helloworld.rs

1 !rustc /content/helloworld.rs
2 !./helloworld

Hola Mundo!
```

Arreglos y funciones

```
[ ] 1 %%writefile arreglos.rs
2
3 use std::mem;
4
5 // This function borrows a slice
6 fn analyze_slice(slice: &[i32]) {
7     println!("primer elemento del arreglo: {}", slice[0]);
8     println!("el ultimo elemento es: {}", slice[slice.len()-1]);
9     println!("el arreglo tiene {} elementos", slice.len());
10 }
11
12 fn main() {
13     // arreglo de largo 5 fijo
14     let xs: [i32; 5] = [1, 2, 3, 4, 5];
15     // arreglo de largo 500 inicializado en 0
16     let ys: [i32; 500] = [0; 500];
17
18     // indices comienzan en 0
19     println!("primer elemento arreglo: {}", xs[0]);
20     println!("tercer elemento del arreglo: {}", xs[2]);
21
22     // `len` retorna el largo del arreglo
23     println!("numero de elemenos arreglo xs: {}", xs.len());
24     println!("numero de elementos arreglo ys: {}", ys.len());
25
26     // Arrays are stack allocated
27     println!("tamaño en memoria xs {} bytes", mem::size_of_val(&xs));
28     println!("tamaño en memoria ys {} bytes", mem::size_of_val(&ys));
29     // Arrays can be automatically borrowed as slices
30     println!("pasamos el arreglo por puntero a función");
31     analyze_slice(&xs);
32     analyze_slice(&ys);
33
34     println!("pasamos una slice del array");
35     analyze_slice(&ys[1..4]);
36
37     //Se puede acceder a las matrices de forma segura mediante `.get`,
38     //que devuelve un
39     // `Opción`. Esto se puede combinar como se muestra a continuación, o se puede usar con
40     // `.expect()` si desea que el programa finalice con un buen
41     // mensaje en lugar de continuar.
42     for i in 0..xs.len() + 1 { // iteramos un elemento mas del largo
43         match xs.get(i) {
44             Some(xval) => println!("{}", i, xval),
45             None => println!("Fuera de indice! {} no existe!", i),
46         }
47     }
```

<https://github.com/adigenova/uohpmd/blob/main/code/OpenMP.ipynb>

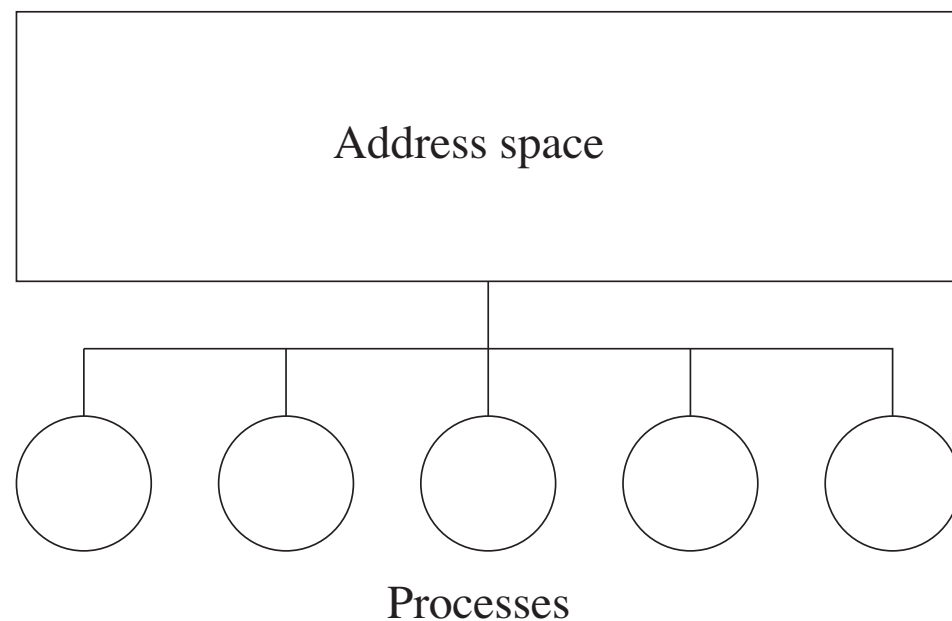
MPI

Message Passage Interface

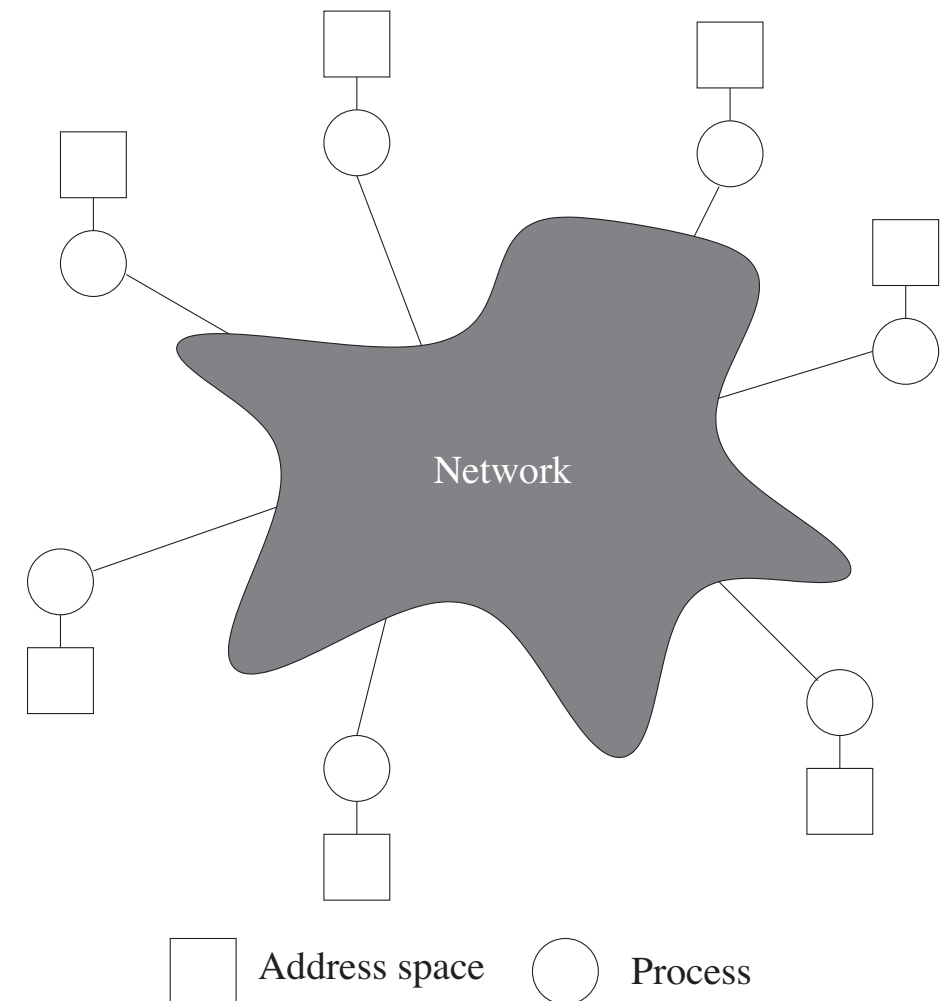
MPI

Modelo de paralelismo

Modelo de memoria compartida

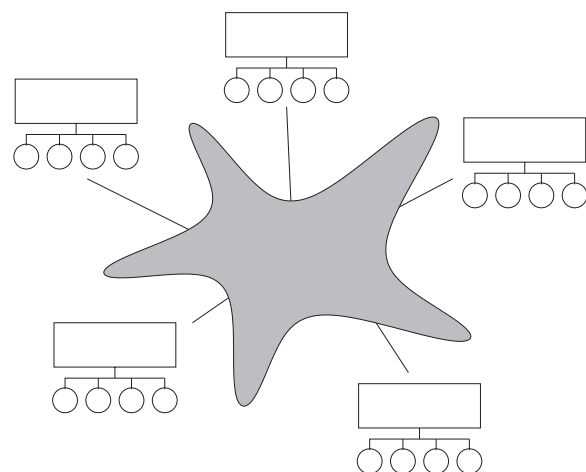


Modelo de paso de mensajes



Modelo Híbrido

PThread/ OpenMP / Fork



- El modelo de paso de mensajes postula un conjunto de procesos que solo tienen memoria local pero que pueden comunicarse con otros procesos enviando y recibiendo mensajes.
- Es una característica definitoria del modelo de paso de mensajes que la transferencia de datos desde la memoria local de un proceso a la memoria local de otro requiere que ambos procesos realicen operaciones.
- MPI es una implementación específica del modelo de paso de mensajes

MPI

Ventajas del modelo de paso de mensajes

- **Universalidad.** El modelo de paso de mensajes encaja bien en procesadores separados conectados por una red de comunicación (rápida o lenta). Por lo tanto, coincide con el nivel más alto del hardware de la mayoría de las supercomputadoras paralelas actuales.
- **Expresividad.** Se ha encontrado que el paso de mensajes es un modelo útil y completo para expresar algoritmos paralelos.
- **Facilidad de depuración.** La depuración de programas paralelos sigue siendo un área de investigación desafiante. La razón es que una de las causas más comunes de error es la sobrescritura inesperada de la memoria. El modelo de paso de mensajes, al controlar las referencias a la memoria de manera más explícita que cualquiera de los otros modelos, facilita la localización de lecturas y escrituras de memoria erróneas.
- **Rendimiento.** La razón más convincente por la que el paso de mensajes seguirá siendo una parte permanente del entorno informático paralelo es el rendimiento. A medida que las CPU modernas se han vuelto más rápidas, la gestión de sus cachés y la jerarquía de la memoria en general se ha convertido en la clave para aprovechar al máximo estas máquinas.

MPI

Que es?

- **MPI es una libreria, no un lenguaje.**
 - Especifica los nombres, las secuencias de llamada y los resultados de las funciones que se llamarán desde los programas C.
 - Los programas que los usuarios escriben en C se compilan con compiladores ordinarios y se vinculan con la libreria MPI.
- **MPI es una especificación, no una implementación particular.**
 - Un programa MPI correcto debería poder ejecutarse en todas las implementaciones de MP sin cambios.
- **MPI aborda el modelo de paso de mensajes.**
 - Enfoque en el movimiento de datos entre espacios de direcciones separados.

MPI

Conceptos basicos

- La comunicación se produce cuando una parte del espacio de direcciones de un proceso se copia en el espacio de direcciones de otro proceso.
- Esta operación es cooperativa y ocurre solo cuando el primer proceso ejecuta una operación de **envío (send)** y el segundo proceso ejecuta una operación de recepción (**receive**).
 - **MPI_Send**(address, count, datatype, destination, tag, comm)
 - **MPI_Recv**(address, maxcount, datatype, source, tag, comm, status)
 - **tag** es un número entero que se utiliza para la coincidencia de mensajes.
 - **comm** identifica un grupo de procesos y un contexto de comunicación.
 - **destination/source** es el ranking del destino/fuente en el grupo asociado con el comunicador **comm**.

MPI

comunicaciones colectivas

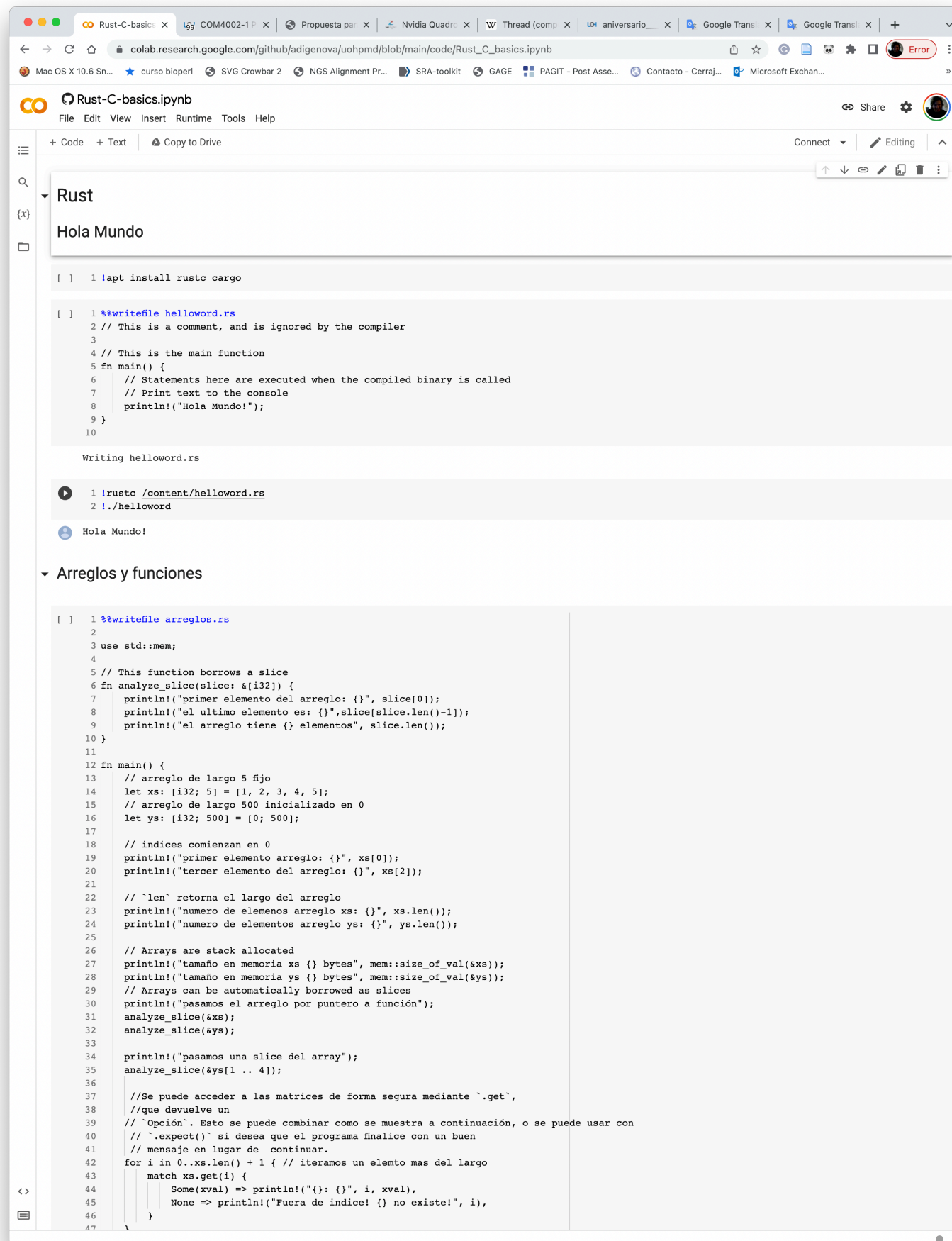
- Operaciones realizada por todos los procesos en un cómputo.
- Las operaciones colectivas son de dos tipos:
 - Las operaciones de **movimiento de datos** se utilizan para reorganizar los datos entre los procesos. La más simple de ellas es **broadcast**, pero se pueden definir muchas operaciones elaboradas de dispersión (**scattering**) y recopilación (**gathering**).
 - Operaciones de **cálculo colectivo** (mínimo, máximo, suma, OR lógico, etc., así como operaciones definidas por el usuario).

MPI

Funciones basicas

Función	Descripción
MPI_Init	Inicializar MPI
MPI_Comm_size	Determina cuántos procesos hay
MPI_Comm_rank	qué proceso soy
MPI_Send	Envio un mensaje
MPI_Recv	Recibo un mensaje
MPI_Finalize	Termina MPI

Google Colab



The screenshot shows a Google Colab notebook interface. The browser tabs at the top include 'Rust-C-basics', 'COM4002-1 F...', 'Propuesta pa...', 'Nvidia Quad...', 'W Thread (comp...', 'aniversario...', 'Google Transl...', and another 'Google Transl...'. The address bar shows the URL 'colab.research.google.com/github/adigenova/uohpmd/blob/main/code/Rust_C_basics.ipynb'. The notebook title is 'Rust-C-basics.ipynb'.

The notebook content is as follows:

```
File Edit View Insert Runtime Tools Help
+ Code + Text + Copy to Drive
Connect Editing
Rust
Hola Mundo

[ ] 1 !apt install rustc cargo

[ ] 1 %%writefile helloworld.rs
2 // This is a comment, and is ignored by the compiler
3
4 // This is the main function
5 fn main() {
6     // Statements here are executed when the compiled binary is called
7     // Print text to the console
8     println!("Hola Mundo!");
9 }
10

Writing helloworld.rs

[ ] 1 !rustc /content/helloworld.rs
2 !./helloworld

Hola Mundo!
```

Below the first code block, there is a section titled 'Arreglos y funciones' (Arrays and functions) with the following code:

```
[ ] 1 %%writefile arreglos.rs
2
3 use std::mem;
4
5 // This function borrows a slice
6 fn analyze_slice(slice: &[i32]) {
7     println!("primer elemento del arreglo: {}", slice[0]);
8     println!("el ultimo elemento es: {}", slice[slice.len()-1]);
9     println!("el arreglo tiene {} elementos", slice.len());
10 }
11
12 fn main() {
13     // arreglo de largo 5 fijo
14     let xs: [i32; 5] = [1, 2, 3, 4, 5];
15     // arreglo de largo 500 inicializado en 0
16     let ys: [i32; 500] = [0; 500];
17
18     // indices comienzan en 0
19     println!("primer elemento arreglo: {}", xs[0]);
20     println!("tercer elemento del arreglo: {}", xs[2]);
21
22     // `len` retorna el largo del arreglo
23     println!("numero de elemenos arreglo xs: {}", xs.len());
24     println!("numero de elementos arreglo ys: {}", ys.len());
25
26     // Arrays are stack allocated
27     println!("tamaño en memoria xs {} bytes", mem::size_of_val(&xs));
28     println!("tamaño en memoria ys {} bytes", mem::size_of_val(&ys));
29     // Arrays can be automatically borrowed as slices
30     println!("pasamos el arreglo por puntero a función");
31     analyze_slice(&xs);
32     analyze_slice(&ys);
33
34     println!("pasamos una slice del array");
35     analyze_slice(&ys[1..4]);
36
37     //Se puede acceder a las matrices de forma segura mediante `.get`,
38     //que devuelve un
39     // `Opción`. Esto se puede combinar como se muestra a continuación, o se puede usar con
40     // `.expect()` si desea que el programa finalice con un buen
41     // mensaje en lugar de continuar.
42     for i in 0..xs.len() + 1 { // iteramos un elemento mas del largo
43         match xs.get(i) {
44             Some(xval) => println!("{}", i, xval),
45             None => println!("Fuera de indice! {} no existe!", i),
46         }
47     }
```

https://github.com/adigenova/uohpmd/blob/main/code/MPI_I.ipynb

Consultas?

Consultas o comentarios?

Muchas gracias