



# Tecnológico de Monterrey

**Instituto Tecnológico y de Estudios Superiores de Monterrey**

**Maestría en Inteligencia Artificial Aplicada - MNA**

**Cómputo en la nube (Gpo 10)**

## **Tarea 1**

**Programación de una solución paralela**

**Profesor**

**Dr. Gilberto Echeverría Furió**

Ángel Eduardo Urueta Puello - A01796724

**1 de Febrero de 2026**

## 1. Introducción

OpenMP (Open Multi-Processing) es una interfaz de programación de aplicaciones (API) estandarizada para desarrollar programas paralelos en sistemas de memoria compartida [1]. En esencia, proporciona un conjunto de herramientas (directivas de compilador, funciones de biblioteca y variables de entorno) que facilitan la creación y gestión de threads (hilos) en aplicaciones C, C++ y Fortran [2]. Al utilizar directivas especiales en el código (por ejemplo, #pragma omp parallel for), el compilador y el runtime de OpenMP se encargan de distribuir el trabajo entre múltiples núcleos de la CPU, explotando el paralelismo disponible con mínimos cambios en el código original.

OpenMP se ha convertido en un estándar ampliamente adoptado para la programación paralela de memoria compartida en múltiples plataformas (Unix, Windows, macOS, etc.) [3]. Es valorado por su simplicidad y flexibilidad: permite a los desarrolladores escribir código paralelo de forma fácil y eficiente añadiendo directivas de compilador sencillas al código existente. Un programa escrito con OpenMP típicamente sigue un modelo de ejecución fork-join: inicia con un solo hilo maestro que, al entrar en una región paralela, “bifurca” varios hilos para trabajar concurrentemente y luego los “une” al final de dicha región. Todos los hilos comparten la misma memoria global del proceso, lo que evita la necesidad de pasar mensajes explícitos entre ellos.

Otro aspecto importante es que OpenMP es un modelo portable y escalable, soportado por diversos compiladores en una variedad de arquitecturas. Por ejemplo, compiladores como GCC (GNU Compiler Collection) y el Intel C/C++ Compiler implementan completamente el estándar OpenMP [4]. Esto significa que un mismo código con directivas OpenMP puede compilarse y ejecutarse en diferentes sistemas (desde una PC de escritorio hasta un clúster o supercomputadora) sin cambios significativos [5]. En resumen, OpenMP proporciona una manera sencilla de aprovechar los procesadores multinúcleo, permitiendo parallelizar secciones críticas (como bucles intensivos) para reducir los tiempos de ejecución y mejorar el rendimiento de las aplicaciones.

## 2. Enlace al Repositorio Github

Los programas que se muestran en el punto 3 se encuentran disponibles en el siguiente enlace de github:  
[https://github.com/AngelUruetaTec/SolucionSumaArreglosParalela\\_Angel\\_Urueta](https://github.com/AngelUruetaTec/SolucionSumaArreglosParalela_Angel_Urueta)

## 3. Secuencia de ejecución del programa

Teniendo en cuenta que la computadora disponible para este trabajo es una Mac con procesador M2 y Tahoe 26.2, se desarrolla el ejercicio instalando librerías y compiladores necesarios la línea de comandos usando Homebrew(<https://brew.sh/>).

El procedimiento realizado es el siguiente:

- Instalación de librería OPENMP, contenida en libomp(<https://openmp.llvm.org>), para Mac

```
base ~ (8.934s)
brew install libomp

==> Auto-updating Homebrew...
Adjust how often this is run with `'$HOMEBREW_AUTO_UPDATE_SECS` or disable with
`'$HOMEBREW_NO_AUTO_UPDATE=1` . Hide these hints with `'$HOMEBREW_NO_ENV_HINTS=1` (see `man brew`).
==> Auto-updated Homebrew!
Updated 3 taps (gromgit/fuse, homebrew/core and homebrew/cask).
==> New Formulae
cargo-features-manager: TUI like cli tool to manage the features of your rust-project dependencies
clawbot-cli: Your own personal AI assistant
codex-acp: Use Codex from ACP-compatible clients such as Zed!
dbcsrc: Distributed Block Compressed Sparse Row matrix library
gogcli: Google Suite CLI
hdrhistogram_c: C port of the HdrHistogram
litra: Control Logitech Litra lights from the command-line
llhttp: Port of http_parser to llparse
radicle: Sovereign code forge built on Git
tpix: Simple terminal image viewer using the Kitty graphics protocol
whothere: LAN discovery tool with a modern TUI written in Go
==> New Casks
infinidesk: Create multiple virtual desktops, each with unique files, wallpaper and widgets
ipaververse: Tool for downloading and managing iOS apps from the App Store
seam-app: Productivity-first Dynamic Island for your Notch
tritium: Integrated drafting environment for legal professionals
whyfi: Menu bar Wi-Fi monitor and diagnostics app

You have 11 outdated formulae installed.

==> Fetching downloads for: libomp
✓ Bottle Manifest libomp (21.1.8)                                                 Downloaded 11.9KB/ 11.9KB
✓ Bottle libomp (21.1.8)                                                 Downloaded 586.2KB/586.2KB
==> Pouring libomp--21.1.8.arm64-tahoe.bottle.tar.gz
==> Caveats
libomp is keg-only, which means it was not symlinked into /opt/homebrew,
because it can override GCC headers and result in broken builds.

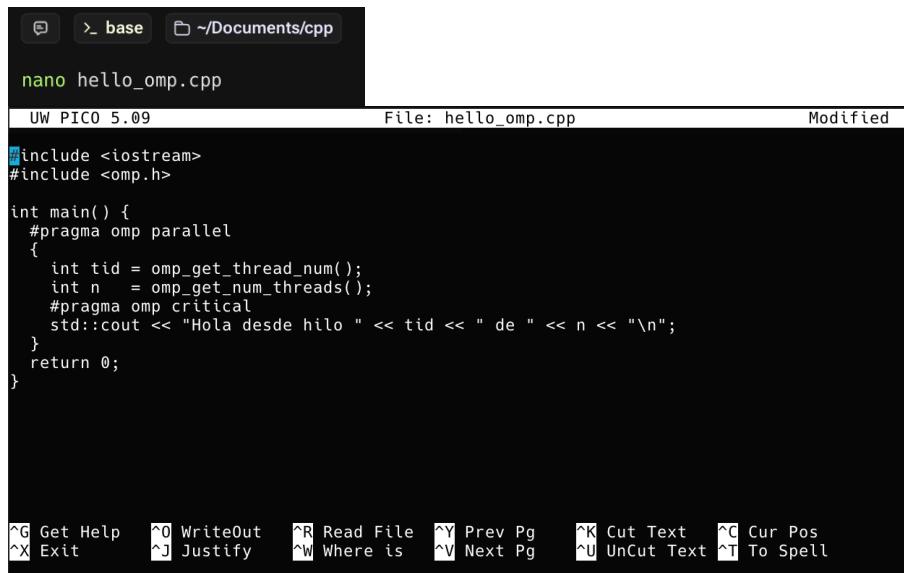
For compilers to find libomp you may need to set:
  export LDFLAGS="-L/opt/homebrew/opt/libomp/lib"
  export CPPFLAGS="-I/opt/homebrew/opt/libomp/include"
==> Summary
🍺 /opt/homebrew/Cellar/libomp/21.1.8: 9 files, 1.8MB
==> Running `brew cleanup libomp`...
Disable this behaviour by setting `'$HOMEBREW_NO_INSTALL_CLEANUP=1` .
Hide these hints with `'$HOMEBREW_NO_ENV_HINTS=1` (see `man brew` ).
```

- Instalación de compilador gcc(<https://gcc.gnu.org/>)

```
base ~ (1.398s)
brew install gcc

Warning: gcc 15.2.0 is already installed and up-to-date.
To reinstall 15.2.0, run:
  brew reinstall gcc
```

- Se crea un programa de prueba para verificar la correcta instalación de compilador y librería garantizando una ejecución paralela en los núcleos del procesador:



```
nano hello_omp.cpp
UW PICO 5.09          File: hello_omp.cpp          Modified

#include <iostream>
#include <omp.h>

int main() {
    #pragma omp parallel
    {
        int tid = omp_get_thread_num();
        int n   = omp_get_num_threads();
        #pragma omp critical
        std::cout << "Hello desde hilo " << tid << " de " << n << "\n";
    }
    return 0;
}

^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Pg  ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is   ^V Next Pg  ^U Uncut Text ^T To Spell
```

d. Compilación programa de prueba:

```
base ~/Documents/cpp (2.988s)
g++-15 hello_omp.cpp -o hello_omp -fopenmp
```

e. Ejecución programa de prueba. Al ejecutar en un Procesador M2 con 8 núcleos se tienen 8 hilos diferentes como resultado.

```
base ~/Documents/cpp  (0.818s)
./hello_omp
Hola desde hilo 3 de 8
Hola desde hilo 1 de 8
Hola desde hilo 2 de 8
Hola desde hilo 0 de 8
Hola desde hilo 6 de 8
Hola desde hilo 4 de 8
Hola desde hilo 5 de 8
Hola desde hilo 7 de 8
```

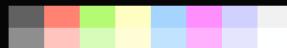
```
base ~/Documents/cpp  (0.372s)
fastfetch
```

```
...
,xNMM.
.0MMMo
lMM"
.;loddo... .olloddol;.
cKMMMMMMMMNNWMMMMMMMMMMWd:
.KMMMMMMMMMMMMMMMMMMMMMMMMMM:
XMMMMMMMMMMMMMMMMMMMMMMMMMM.
;MMMMMMMMMMMMMMMMMMMMMMMMMM:
:MMMMMMMMMMMMMMMMMMMMMMMMMM:
.MMMMMMMMMMMMMMMMMMMMMMMMMX.
KMMMMMMMMMMMMMMMMMMMMMMMMWd.
'XMMMMMMMMMMMMMMMMMMMMMMMMMM
'XMMMMMMMMMMMMMMMMMMMMMMMMMM.
kMMMMMMMMMMMMMMMMMMMMMMMMMd
;KMMMMMMMMWXWMMMMMMMMk.
"cooc*"    "*coo""
```

angeleduardouruetapuello@Angels-Mac-mini

-----

OS: macOS Tahoe 26.2 (25C56) arm64  
Host: Mac mini (M2, 2023, Two Thunderbolt 4 ports)  
Kernel: Darwin 25.2.0  
Uptime: 53 mins  
Packages: 153 (brew), 20 (brew-cask)  
Shell: zsh 5.9  
Display (LG HDR 4K): 1692x3008 in 27", 60 Hz [Exte\*  
Display (LG HDR 4K): 3384x6016 @ 2x in 27", 60 Hz ]  
WM: Quartz Compositor 1.600.0  
WM Theme: Multicolor (Light)  
Theme: Liquid Glass  
Font: .AppleSystemUIFont [System], Helvetica [User]  
Cursor: Fill - Black, Outline - White (32px)  
Terminal: Warp v0.2026.01.28.08.14.stable\_01  
Terminal Font: Hack (11.0pt)  
CPU: Apple M2 (8) @ 3.50 GHz  
GPU: Apple M2 (10) @ 1.40 GHz [Integrated]  
Memory: 6.73 GiB / 8.00 GiB (84%)  
Swap: 983.13 MiB / 2.00 GiB (48%)  
Disk (/): 207.16 GiB / 228.27 GiB (91%) - apfs [Re]  
Disk (/Volumes/FOTOGRAFIA): 340.67 GiB / 931.41 Gi]  
Disk (/Volumes/NMVE - MACMINI): 782.37 GiB / 931.4]  
Local IP (en0): 192.168.1.8/24  
Locale: C.UTF-8



f. Escritura del programa suma de arrays en acorde con lo especificado para la tarea:

```

nano arrays_rand_openMP.cpp
UW PICO 5.09
File: arrays_rand_openMP.cpp

#include <iostream>
#include <omp.h>
#include <cstdlib> // rand(), srand()
#include <ctime> // time()

#define N 1000
#define chunk 100
#define mostrar 10

void imprimeArreglo(float *d);

int main()
{
    std::cout << "Sumando Arreglos en Paralelo!\n";
    float a[N], b[N], c[N];
    int i;

    // Inicializar semilla para números aleatorios
    srand(time(NULL));

    // Llenar arreglos con valores aleatorios entre 0 y 100
    for (i = 0; i < N; i++)
    {
        a[i] = static_cast<float>(rand() % 101); // 0 a 100
        b[i] = static_cast<float>(rand() % 101); // 0 a 100
    }

    int pedazos = chunk;

#ifndef _OPENMP
    #pragma omp parallel for \
        shared(a, b, c, pedazos) private(i) \
        schedule(static, pedazos)
#endif

    for (i = 0; i < N; i++)
        c[i] = a[i] + b[i];

    std::cout << "Imprimiendo los primeros " << mostrar << " valores del arreglo a: " << std::endl;
    imprimeArreglo(a);

    std::cout << "Imprimiendo los primeros " << mostrar << " valores del arreglo b: " << std::endl;
    imprimeArreglo(b);

    std::cout << "Imprimiendo los primeros " << mostrar << " valores del arreglo c: " << std::endl;
    imprimeArreglo(c);
}

void imprimeArreglo(float *d)
{
    for (int x = 0; x < mostrar; x++)
        std::cout << d[x] << " - ";
    std::cout << std::endl;
}

```

^G Get Help      ^O WriteOut      ^R Read File      ^Y Prev Pg      ^K Cut Text      ^C Cur Pos  
 ^X Exit      ^J Justify      ^W Where is      ^V Next Pg      ^U Uncut Text      ^T To Spell

g. Compilación del programa de suma de arrays:

```

base ~/Documents/cpp (0.568s)
g++-15 arrays_rand_openMP.cpp -o arrays_rand_openMP -fopenmp

```

h. Prueba final de ejecución(3 pruebas en secuencia):

```
base ~/Documents/cpp  (0.426s)
./arrays_rand_openMP
Sumando Arreglos en Paralelo!
Imprimiendo los primeros 10 valores del arreglo a:
28 - 44 - 98 - 84 - 85 - 34 - 99 - 36 - 85 - 97 -
Imprimiendo los primeros 10 valores del arreglo b:
90 - 18 - 31 - 13 - 71 - 23 - 26 - 31 - 33 - 56 -
Imprimiendo los primeros 10 valores del arreglo c:
118 - 62 - 129 - 97 - 156 - 57 - 125 - 67 - 118 - 153 -

base ~/Documents/cpp  (0.045s)
./arrays_rand_openMP
Sumando Arreglos en Paralelo!
Imprimiendo los primeros 10 valores del arreglo a:
18 - 41 - 23 - 63 - 5 - 61 - 29 - 66 - 77 - 30 -
Imprimiendo los primeros 10 valores del arreglo b:
18 - 47 - 86 - 39 - 42 - 16 - 54 - 4 - 58 - 90 -
Imprimiendo los primeros 10 valores del arreglo c:
36 - 88 - 109 - 102 - 47 - 77 - 83 - 70 - 135 - 120 -

base ~/Documents/cpp  (0.033s)
./arrays_rand_openMP
Sumando Arreglos en Paralelo!
Imprimiendo los primeros 10 valores del arreglo a:
59 - 39 - 9 - 34 - 83 - 65 - 25 - 40 - 49 - 4 -
Imprimiendo los primeros 10 valores del arreglo b:
83 - 55 - 68 - 17 - 78 - 39 - 16 - 76 - 6 - 55 -
Imprimiendo los primeros 10 valores del arreglo c:
142 - 94 - 77 - 51 - 161 - 104 - 41 - 116 - 55 - 59 -
```

#### 4. Interpretación del código

El código presentado es un programa en C++ que suma dos arreglos de 1000 elementos en paralelo utilizando la biblioteca OpenMP para parallelizar el cálculo. En términos generales, el programa realiza lo siguiente:

- Define tres arreglos (a, b y c) de tamaño N = 1000.
- Inicializa los arreglos a y b con numeros aleatorios entre cero y 100 con una semilla basada en tiempo para que no se repita la misma secuencia.
- Utiliza una directiva de OpenMP #pragma omp parallel for para sumar en paralelo los elementos de a y b, almacenando el resultado en el arreglo c. La suma se reparte entre múltiples hilos de ejecución.
- Finalmente, imprime en pantalla los primeros 10 valores de cada arreglo (a, b y c) para verificar el resultado de la suma paralela.

En la siguiente tabla se da un mayor detalle de cada línea del código.

PARTE DEL CÓDIGO	QUÉ HACE	DETALLE
<code>#INCLUDE &lt;iostream&gt;</code>	Habilita entrada/salida estándar	Se usa para std::cout (imprimir en consola).
<code>#INCLUDE &lt;omp.h&gt;</code>	Activa soporte de OpenMP	Permite usar directivas #pragma omp ... para paralelismo (hilos).
<code>#INCLUDE &lt;cstdlib&gt;</code>	Funciones de C para aleatorios	Aporta rand() (número pseudoaleatorio) y srand() (semilla).
<code>#INCLUDE &lt;ctime&gt;</code>	Manejo de tiempo	Aporta time() para obtener el tiempo actual y usarlo como semilla.
<code>#DEFINE N 1000</code>	Constante del tamaño de arreglos	Los arreglos a, b, c tendrán 1000 elementos.
<code>#DEFINE CHUNK 100</code>	Tamaño de “bloques” (chunks) en el reparto	OpenMP usará bloques de 100 iteraciones por asignación (en schedule(static, pedazos)).
<code>#DEFINE MOSTRAR 10</code>	Cuántos elementos imprimir	La función imprimeArreglo() imprime solo los primeros 10 valores de cada arreglo.
<code>VOID IMPRIMEARREGLO(FLOAT *D);</code>	Declaración de función	Declara la función antes de main() para poder llamarla desde ahí.
<code>INT MAIN()</code>	Punto de entrada del programa	Todo el flujo principal ocurre dentro de main().
<code>STD::COUT &lt;&lt; "SUMANDO ARREGLOS EN PARALELO!\n";</code>	Mensaje inicial	Indica lo que hará el programa (sumar arreglos usando paralelismo).
<code>FLOAT A[N], B[N], C[N];</code>	Declara 3 arreglos de tipo float	a y b guardan datos; c guardará la suma elemento a elemento: $c[i] = a[i] + b[i]$ .
<code>INT i;</code>	Índice de iteración	Variable usada para recorrer los arreglos. En paralelo se marcará como privada (private(i)).
<code>SRAND(TIME(NULL));</code>	Inicializa semilla del generador pseudoaleatorio	time(NULL) devuelve el tiempo actual (segundos desde época). Así cada ejecución produce valores distintos (si no, rand() repetiría la misma secuencia).
<code>FOR (I=0; I&lt;N; I++) { A[I]=...; B[I]=...; }</code>	Llena a y b con valores aleatorios entre 0 y 100	rand() % 101 da un entero en [0,100]. Luego se convierte a float con static_cast<float>(...).
<code>INT PEDAZOS = CHUNK;</code>	Copia chunk en una variable	Se usa en OpenMP como tamaño de bloque para el schedule. Podrían haber usado chunk directamente, pero aquí lo pasan como variable compartida.
<code>#PRAGMA OMP PARALLEL FOR ...</code>	Paraleliza el for que viene justo después	OpenMP crea un equipo de hilos y reparte las iteraciones del for entre ellos. Cada hilo calcula una parte de c.
<code>SHARED(A, B, C, PEDAZOS)</code>	Declara variables <b>compartidas</b> en la región paralela	Todos los hilos ven los mismos arreglos a, b, c (misma memoria). Esto es correcto porque cada hilo escribe en índices distintos de c.
<code>PRIVATE(I)</code>	Declara i como <b>privada</b> por hilo	Cada hilo tiene su propia copia de i. Esto evita condiciones de carrera sobre el índice.
<code>SCHEDULE(STATIC, PEDAZOS)</code>	Política de asignación de iteraciones	<b>Static:</b> el reparto se hace al inicio. Con bloque de pedazos (=100), se asignan “paquetes” de 100 iteraciones a los hilos. Ejemplo: hilo 0 recibe 0–99, hilo 1 100–199, etc. (dependiendo del número de hilos). Es eficiente cuando cada iteración cuesta casi lo mismo.
<code>FOR (I = 0; I &lt; N; I++) C[I] = A[I] + B[I];</code>	Suma elemento a elemento en paralelo	Cada iteración calcula una suma independiente, así que es un caso ideal de paralelismo (“embarrassingly parallel”). No hay dependencia entre iteraciones.
<code>STD::COUT &lt;&lt; "IMPRIMIENDO..." (TRES VECES)</code>	Mensajes antes de imprimir arreglos	Indica qué arreglo se va a mostrar (a, b y c).

PARTE DEL CÓDIGO	QUÉ HACE	DETALLE
<code>IMPRIMEARREGLO(A); / IMPRIMEARREGLO(B); / IMPRIMEARREGLO(C);</code>	Llama a la función de impresión	Imprime los primeros mostrar (=10) valores de cada arreglo.
<code>VOID IMPRIMEARREGLO(FLOAT *D)</code>	Define la función de impresión	Recibe un puntero al primer elemento del arreglo (float*). En C++ un arreglo “decay” a puntero al pasarlo a función.
<code>FOR (INT X = 0; X &lt; MOSTRAR; X++) STD::COUT &lt;&lt; D[X] &lt;&lt; " - ";</code>	Imprime los primeros 10 elementos	Accede como arreglo d[x]. El separador es " - ".
<code>STD::COUT &lt;&lt; STD::ENDL;</code>	Salto de línea y flush	Termina la línea de salida después de imprimir los 10 elementos.

## 5. Reflexión y conclusiones

Al finalizar este trabajo se tienen las siguientes conclusiones:

- **OpenMP es una herramienta poderosa y accesible para parallelización**, permite convertir aplicaciones secuenciales en paralelas con mínimas modificaciones al código, facilitando el desarrollo de software de alto rendimiento.
- **El paralelismo a nivel de bucles es especialmente eficiente**, como se evidenció en el programa de suma de arreglos, los bucles con iteraciones independientes son ideales para ser parallelizados usando parallel for, obteniendo mejoras significativas en tiempo de ejecución.
- **macOS requiere una configuración adicional para usar OpenMP correctamente**, debido a la falta de soporte en el compilador por defecto, es necesario instalar GCC u otro compilador compatible. Sin embargo, una vez configurado, el uso de OpenMP es completamente funcional.
- **La gestión automática de hilos simplifica el desarrollo**, OpenMP abstrae detalles complejos como la creación de threads, sincronización y distribución de tareas, reduciendo errores comunes en programación concurrente.
- **El uso de planificación (schedule) mejora el control del rendimiento**, al dividir las iteraciones en bloques (chunks), se puede optimizar cómo se reparte el trabajo entre los núcleos del procesador, haciendo el programa más eficiente.
- **El programa desarrollado valida la aplicabilidad práctica de OpenMP**. la suma paralela de arreglos demuestra cómo tareas comunes en procesamiento numérico, simulaciones o análisis de datos pueden acelerarse de forma sencilla.

Como reflexión final, la utilización de OpenMP representa una solución práctica y eficiente para introducir paralelismo en programas escritos en C y C++, especialmente en arquitecturas modernas con múltiples núcleos de procesamiento. A lo largo de este trabajo se evidenció que, mediante simples directivas como #pragma omp parallel for, es posible transformar un algoritmo secuencial en uno concurrente sin necesidad de implementar manualmente la gestión de hilos, sincronización o comunicación entre procesos. La experiencia de implementación en macOS demuestra que, aunque el sistema no ofrece soporte nativo completo para OpenMP mediante el compilador por defecto (Apple Clang), y el uso de herramientas como GCC instaladas con Homebrew permiten superar esta limitación y trabajar de manera similar a sistemas Linux. Esto resalta la importancia de comprender tanto el entorno de desarrollo como las herramientas adecuadas para habilitar tecnologías de paralelización.

El programa analizado, que realiza la suma de dos arreglos de manera paralela, ejemplifica claramente uno de los escenarios ideales para OpenMP: operaciones independientes sobre grandes volúmenes de datos. Este tipo de problema se beneficia directamente del paralelismo, ya que cada iteración del ciclo puede ejecutarse simultáneamente sin interferencias, maximizando el aprovechamiento de los recursos de hardware disponibles. Además, el uso de políticas de planificación como schedule(static, chunk) permite controlar cómo se distribuye el trabajo entre los hilos, optimizando el rendimiento dependiendo de la carga computacional y la arquitectura del sistema.

## 6. Referencias:

- I. Dagum, L., & Menon, R. (1998). OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science & Engineering*, 5(1), 46-55.
- II. Chapman, B., Jost, G., & Van der Pas, R. (2007). *Using OpenMP: Portable Shared Memory Parallel Programming*. Cambridge, MA: MIT Press.
- III. OpenMP Architecture Review Board. (2021). OpenMP Application Programming Interface Version 5.2. Última actualización Nov 2021. Recuperado de OpenMP.org
- IV. GeeksforGeeks. (2023, 19 de marzo). Introduction to Parallel Programming with OpenMP in C++. Recuperado de <https://www.geeksforgeeks.org>
- V. OpenMP Architecture Review Board. (2024). About OpenMP – The OpenMP API specification for parallel programming. (Sección “About OpenMP” en OpenMP Tutorials & Articles). Recuperado de <https://www.openmp.org>