Universidad Nacional Mayor de San Marcos
Facultad de Ingeniería de Sistemas e Informática
Escuela de Ingeniería de Sistemas

LABORATORIO

CURSO	:	***Programación Paralela.***
No.	:	*Laboratorio No.07*
TEMA	:	*Suma paralela con C++ y Open MPI en Linux.*
DURACIÓN ESTIMADA	:	*01:40 horas.*

I. OBJETIVOS

El presente laboratorio tiene por objetivos:

- Utilizar el editor de texto para elaborar un código fuente en C++
- Realizar una aplicación de suma de números enteros en paralelo usando OpenMP
- Empleo de C++ con MP para programación paralela
- Crear un proyecto para un caso de estudio.

II. RESUMEN

El objetivo de este programa es demostrar cómo dividir un problema (sumar los elementos de un vector) en partes más pequeñas y resolverlas en paralelo usando OpenMP. Esto es esencial para aprovechar la potencia de múltiples procesadores o computadoras en un clúster.

III. PROBLEMÁTICA

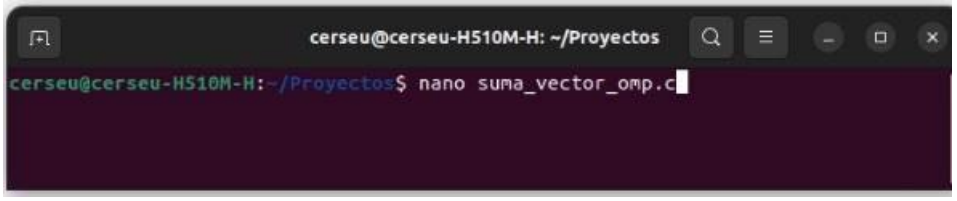
Deseamos realizar la suma de N números aleatorios de forma secuencial y realizar la misma suma de los N números aleatorios en forma paralelo para utilizar múltiples hilos de ejecución y obtener un mejor tiempo de respuesta.

IV. PLANTEAMIENTO DE LA SOLUCIÓN

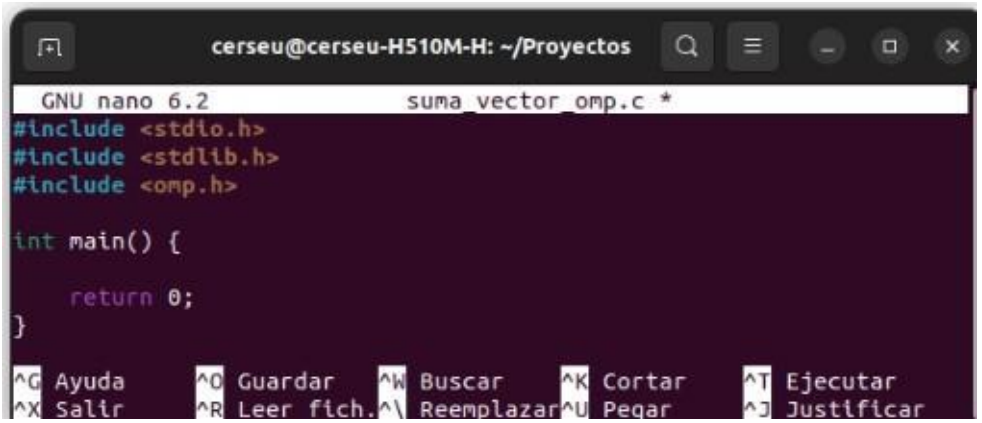
- Un ejemplo de suma en paralelo utilizando C y OpenMP implica dividir el conjunto de datos en porciones más pequeñas y asignar cada porción a un hilo diferente para calcular la suma parcial. Luego, se combinan las sumas parciales para obtener la suma total.

V. SOLUCIÓN

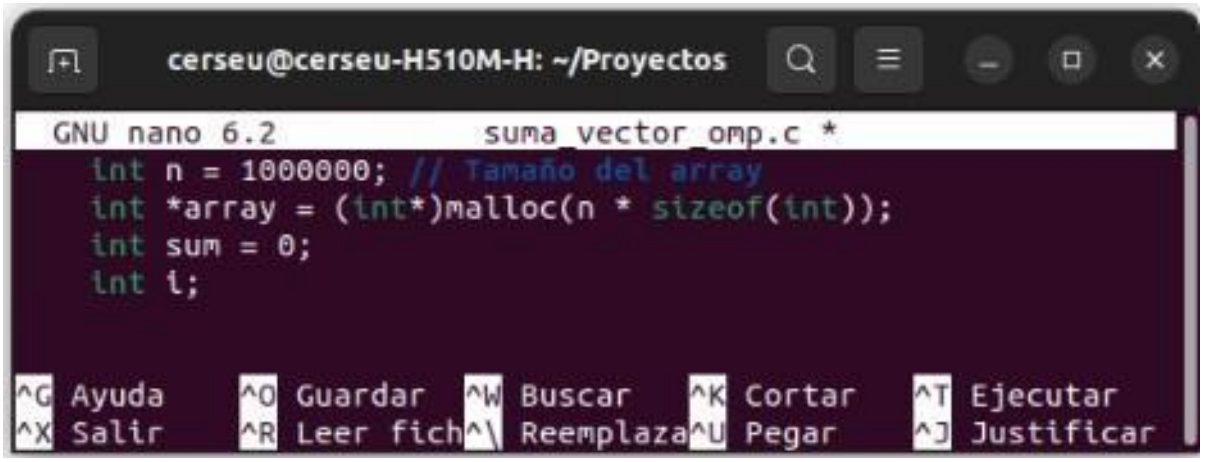
- Iniciamos, cuando vamos a abrir una consola para crear el archivo suma_vector_omp en c



- En el editor, creamos la estructura básica de una aplicación en C con el siguiente código



- Definimos las variables de la aplicación de suma que vamos a utilizar:



- Inicializamos el arreglo con valores para efectuar la suma en paralelo

- A continuación, agregamos el siguiente código para realizar la creación de hilos que sumen en paralelo

- Continuaremos, con imprimir el resultado de la suma global y a liberar la memoria de los hilos, del vector principal y procedemos a finalizar la aplicación OpenMP

VI. EXPLICACIÓN DEL ALGORITMO

1. **#include <omp.h>**: Incluye la biblioteca OpenMP para usar sus funciones y directivas.
2. **#pragma omp parallel for reduction(+:sum)**: Esta directiva de OpenMP crea un bucle paralelo.
   - **parallel for**: Indica que el bucle for siguiente se ejecutará en paralelo.
   - **reduction(+:sum)**: Especifica que la variable sum se reducirá (es decir, se combinarán los resultados de cada hilo).
     - La **+** indica que la operación de reducción es una suma.
3. **for (i = 0; i < n; i++) { ... }**: Este bucle for itera sobre el array.
4. **sum += array[i];**: Dentro del bucle, cada hilo calcula la suma parcial de su porción del array.
   - La **cláusula reduction** asegura que la suma global se actualice correctamente con la suma parcial de cada hilo.

VII. IMPLEMENTACIÓN PROPIA

Se implementa el desarrollo del ejercicio de suma paralela de elementos de un vector usando OpenMP, tomando un valor de 10000 en el tamaño del vector:

```
 GNU nano 7.2                          suma_vector_omp.c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main() {
        int n = 10000; // Tamano del array
        int *array = (int*)malloc(n*sizeof(int));
        int sum = 0;
        int i;

// Inicializar el array con valores
        for (i = 0; i < n; i++){
                array[i] = i + 1;
        }
        //Suma en paralela con OpenMP
        #pragma omp parallel for reduction(+:sum)
        for(i = 0; i < n; i++) {
                sum += array[i];
        }

        printf("La suma total es: %d\n", sum);

        free(array);

        return 0;
}
```

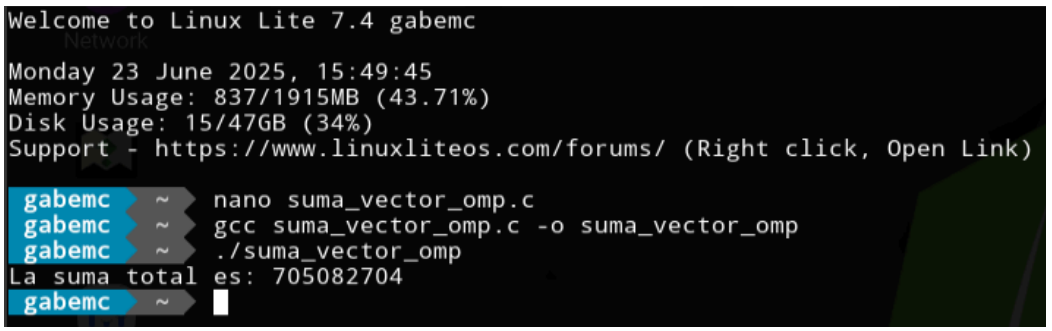
Al compilar y ejecutar el programa en la terminal de Linux, se obtiene un resultado correcto:

```
File  Edit  View  Terminal  Tabs  Help
Welcome to Linux Lite 7.4 gabemc

Monday 23 June 2025, 15:46:05
Memory Usage: 828/1915MB (43.24%)
Disk Usage: 15/47GB (34%)
Support - https://www.linuxliteos.com/forums/ (Right click, Open Link)

 gabemc  ~  nano suma_vector_omp.c
 gabemc  ~  gcc suma_vector_omp.c -o suma_vector_omp
 gabemc  ~  ./suma_vector_omp
La suma total es: 50005000
 gabemc  ~
```

Ahora, qué sucede si cambiamos el tamaño del vector a 100000?

```
 GNU nano 7.2                          suma_vector_omp.c *
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main() {
        int n = 100000; // Tamano del array
        int *array = (int*)malloc(n*sizeof(int));
        int sum = 0;
        int i;

// Inicializar el array con valores
        for (i = 0; i < n; i++){
                array[i] = i + 1;
        }
        //Suma en paralela con OpenMP
        #pragma omp parallel for reduction(+:sum)
        for(i = 0; i < n; i++) {
                sum += array[i];
        }

        printf("La suma total es: %d\n", sum);

        free(array);

        return 0;
}
```

Podemos observar que la suma final no es la esperada, o mejor dicho, no es correcta.



VIII. ANÁLISIS DEL PROBLEMA

Cuando intentamos realizar la suma de los primeros 100,000 números naturales en paralelo utilizando OpenMP, todo parecía estar funcionando correctamente en cuanto a la paralelización del código. Sin embargo, al observar los resultados, notamos que el valor obtenido era completamente erróneo. En lugar de obtener **5,000,050,000**, que es la suma correcta de los primeros 100,000 números naturales, el programa devolvía un número **705,082,704**.

Este error se debe a un **desbordamiento de la variable `sum`**. El tipo de dato **`int`** utilizado para la variable `sum` no tiene suficiente capacidad para almacenar el resultado de la suma total. En sistemas de 32 bits, un `int` tiene un límite de **2,147,483,647**, y al tratar de almacenar un valor más grande, como el de **5,000,050,000**, se produce un desbordamiento.

El error era claro: el tipo de dato no era el adecuado para manejar sumas de valores grandes, lo cual es un problema común cuando se trabajan con grandes volúmenes de datos en operaciones como esta.

IX. SOLUCIÓN AL PROBLEMA

La solución al problema fue sencilla, pero muy eficaz: **cambiar el tipo de la variable `sum` de `int` a `long`**. El tipo `long` tiene un rango mucho mayor y es capaz de manejar números mucho más grandes, lo que evita el desbordamiento que ocurría al usar `int`.

Además, para garantizar que la paralelización de la suma fuera correcta, mantuvimos la directiva **`reduction(+:sum)`**, que le indica a OpenMP que cada hilo debe tener su propia copia de la variable `sum`, y que al final de la ejecución, se debe combinar la suma de todas las copias parciales en una sola.

X. IMPLEMENTACIÓN FINAL

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main() {
   int n = 100000; // Tamano del array
   long *array = (long*)malloc(n * sizeof(long)); //Cambiamos int por long
   long sum = 0; //Cambiamos int por long para sumas mas grandes
   int i;

   // Inicializamos el array con valores
   for (i = 0; i < n; i++){
      array[i] = i + 1;
   }

   //Suma en paralela con OpenMP
   #pragma omp parallel for reduction(+:sum)
   for(i = 0; i < n; i++) {
      sum += array[i];
   }

   //Mostramos el resultado
   printf("La suma total es: %ld\n", sum);

   //LIberamos memoria al terminar con la suma paralela
   free(array);

   return 0;
}
```

- **Tipo de dato `long`**:
  Se cambió el tipo de la variable `sum` a `long`, que tiene un rango más amplio que el tipo `int`, permitiendo manejar números mucho mayores sin desbordarse.

- **Directiva `reduction(+:sum)`**:
  Usamos la directiva de OpenMP `reduction(+:sum)` para asegurarnos de que cada hilo tenga su propia copia de la variable `sum`, y luego se combine correctamente al final.

XI. RESULTADOS

- **Con un arreglo de 10.000 elementos**: El programa devolvió el resultado correcto de **5.000.000**, que es la suma de los primeros 10.000 números naturales. No hubo desbordamiento y todo funcionó correctamente.
- **Con un arreglo de 100.000 elementos**: El cambio al tipo `long` permitió que el programa devolviera el valor esperado de **5.000.050.000**, sin desbordarse ni generar errores. La suma total fue correcta, demostrando que ahora el programa puede manejar arreglos más grandes de manera efectiva.

```
Welcome to Linux Lite 7.4 gabemc

lunes 23 junio 2025, 17:47:15
Memory Usage: 726/1915MB (37.91%)
Disk Usage: 15/47GB (34%)
Support - https://www.linuxliteos.com/forums/ (Right click, Open Link)

gabemc ~ nano suma_vector_omp.c
gabemc ~ gcc -fopenmp suma_vector_omp.c -o suma_vector_omp
gabemc ~ ./suma_vector_omp
La suma total es: 5000050000
gabemc ~
```