|  |  |
| --- | --- |
| **GUÍA N° 01** | |
| **Escuela:** Ingeniería de Software. | **Asignatura:** PROGRAMACIÓN CONCURRENTE Y PARALELA |
| **Semestre Académico:** 2025-II | **Nombre:** Bruno Pumapillo Sarmiento |

**Procesamiento secuencial y paralelo de la multiplicación de matrices**

**Tabla de contenido**

* Definición de la multiplicación de matrices
* Versión secuencial en Python
* Versión secuencial en C
* Versión paralela de la multiplicación de matrices

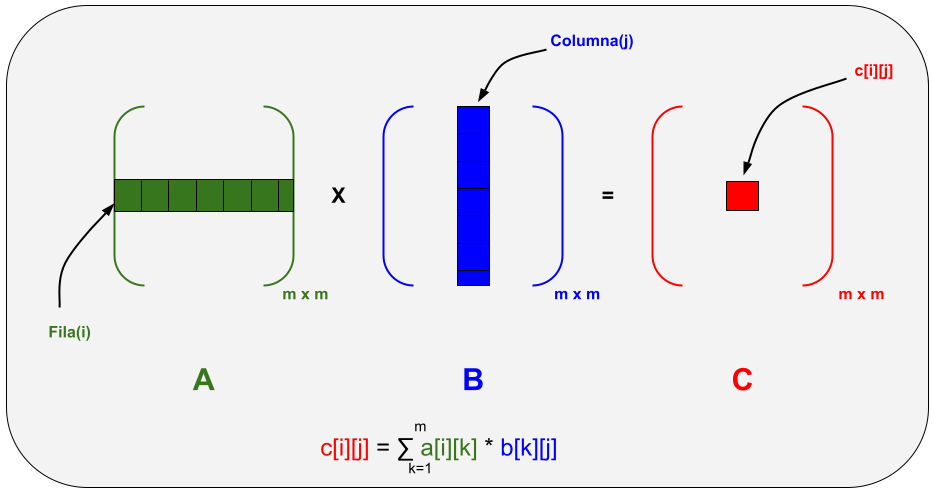
**Definición de la multiplicación de matrices**

Sean A y B dos matrices de dimensiones m×m. La multiplicación de estas dos matrices da como resultado la matriz C de dimensiones m×m y donde

c[i,j]=  ∀ i,j ∈ [1…m].

En otras palabras, la posición c[i,j] es el resultado de multiplicar la fila i de A por la columna j de B.

Gráficamente la multiplicación de matrices la podríamos ver entonces como:



Las siguientes implementaciones en Python y C, se hicieron en un entorno Linux.

**Versión en Python**

A continuación, veremos la implementación de la multiplicación de matrices en Python. El código consta de las siguientes partes:

* **Creación de variables** en esta sección se definen MAX, a, b, y c. Estas tres últimas son matrices de dimensión MAX×MAX.
* **Inicialización de matrices A y B**
* **Cálculo del producto de A x B**
* **Impresión de la matriz C = A x B**

$nano multimat.py

import numpy as np  
#  
# Creación de variables  
#  
MAX=500  
a = np.arange(MAX \* MAX).reshape(MAX,MAX)  
b = np.arange(MAX \* MAX).reshape(MAX,MAX)  
c = np.arange(MAX \* MAX).reshape(MAX,MAX)  
  
#  
# Inicialización de las matrices. Observe que A será la matriz identidad  
#  
for i in range(0,MAX):  
    for j in range(0,MAX):  
        a[i][j] = 0  
        b[i][j] = 100 + i  
    a[i][i] = 1  
  
#  
# Multiplicación efectivamente de las matrices  
#  
for i in range(0,MAX):  
    for j in range(0,MAX):  
        c[i][j] = 0  
        for k in range(0,MAX):  
            c[i][j] = c[i][j] + a[i][k] \* b[k][j]  
  
#  
# Impresión de la matriz (C) resultado de multiplicar A x B  
#  
for i in range(0,MAX):  
    for j in range(0,MAX):  
        print(f"{c[i][j]}  ",end="")  
    print()

Ejecutamos el código midiendo su tiempo de ejecución.

$time python3 multimat.py

…

599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599

python3 multimat.py 112.87s user 0.08s system 99% cpu 1:53.38 total

**Versión en C**

A continuación, se mostrará el código en C de la multiplicación de matrices. El código tiene una estructura igual al de Python (definición de variables, inicialización, multiplicación y visualización) solo que en sintaxis del lenguaje C.

$ nano multimat.c

#include <stdio.h>  
  
#define MAX 500  
  
int main() {  
  /\*\*   
  Definicion de las matrices: a, b, y c.  
  \*/  
  int a[MAX][MAX],  
      b[MAX][MAX],  
      c[MAX][MAX];  
  int i,j,k;  
    
  /\*\*   
  Inicializacion de las matrices a y b. a sera la matriz identidad.  
  \*/  
  for (i = 0; i < MAX; i++){  
    for (j = 0; j < MAX; j++) {  
         a[i][j] = 0;  
         b[i][j] = 100 + i;  
    }  
    a[i][i] = 1;  
  }  
  
  /\*\*  
  Multiplicacion de las matrices  
  \*/  
  for (i = 0; i < MAX; i++){  
    for (j = 0; j < MAX; j++) {  
       c[i][j] = 0;  
       for (k = 0; k < MAX; k++){  
         c[i][j] += a[i][k] \* b[k][j];  
     }  
 }

}  
  /\*\*  
  Impresion de la matriz C  
  \*/  
  for (i = 0; i < MAX; i++) {  
    for (j = 0; j < MAX; j++)  
      printf("%5d ", c[i][j]);  
    printf("\n");  
  }  
  return 0;  
}

Se compila y seguidamente se ejecuta, midiendo su tiempo de ejecución.

$ gcc multimat.c -o multimat && time ./multimat

…

599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 ….

./multimat 0.48s user 0.04s system 96% cpu 0.540 total

**Pregunta de reflexión: ¿Por qué cree Ud, que C es más rápido que Python?**

C es más rápido porque es un lenguaje compilado de más bajo nivel, lo que se traduce en que para ejecutarse necesita traducir todas las instrucciones de código en lenguaje máquina y esto permite que el programa sea mucho más eficiente. En el lenguaje Python sucede distinto ya que es un lenguaje interpretado, lo que significa que su código traduce y se ejecuta instrucción por instrucción y añade una capa más de procesamiento lo que lo vuelve más lento.

**Ejercicio: Implemente el mismo programa en Java y obtenga su tiempo de ejecución. Reflexione acerca de los resultados.**

|  |
| --- |
| public class Multimat {  public static void main(String[] args) {  long startTime = System.currentTimeMillis();  final int MAX = 500;  int a[][] = new int[MAX][MAX];  int b[][] = new int[MAX][MAX];  int c[][] = new int[MAX][MAX];  int i, j, k;  for (i = 0; i < MAX; i++) {  for (j = 0; j < MAX; j++) {  a[i][j] = 0;  b[i][j] = 100 + i;  }  a[i][i] = 1;  }  /\*\*  \* Multiplicación de las matrices  \*/  for (i = 0; i < MAX; i++) {  for (j = 0; j < MAX; j++) {  c[i][j] = 0;  for (k = 0; k < MAX; k++) {  c[i][j] += a[i][k] \* b[k][j];  }  }  }  /\*\*  \* Impresión de la matriz c  \*/  for (i = 0; i < MAX; i++) {  for (j = 0; j < MAX; j++)  System.out.printf("%5d ", c[i][j]);  System.out.printf("\n");  }  long endTime = System.currentTimeMillis();  System.out.println("Execution time: " + (endTime - startTime) + " ms");  }  }  java Multimat.java 5.61s user 3.77s system 149% cpu 6.259 total |

Reflexión:

Luego de haberse compilado el código en java, se puede apreciar que para ejecutarse no se compila directamente a lenguaje máquina, sino que se traducen en bytecode, es decir, que se ejecuta sobre la maquina virtual de java, lo que significa que se interpreta o se compila en el tiempo de ejecución. Esto da como resultado una sobrecarga adicional a comparación con C. En los resultados se obtuvieron que C tardó 0.490 s, sin embargo, en el caso de Java se tardó 5.61 s, que son 11 veces más de lo que tardó el anterior.

**Versión paralela de la multiplicación de matrices**

Ahora vamos a implementar la multiplicación con programación paralela, presentaremos una versión en C y Uds investigarán una versión paralela en Python.

**Versión paralela en C**

Se utilizará la API OpenMP, esta API será desarrollada en el curso. Se muestra el código a continuación, el cual tiene cambios mínimos:

$nano multimat\_parallel.c

#include <stdio.h>  
#include <omp.h>

#define MAX 500  
  
int main() {  
  /\*\*   
  Definicion de las matrices: a, b, y c.  
  \*/  
  int a[MAX][MAX],  
      b[MAX][MAX],  
      c[MAX][MAX];  
  int i,j,k,tid,nthreads;

  /\*\*   
  Inicializacion de las matrices a y b. a sera la matriz identidad.  
  \*/  
#pragma omp parallel shared(a,b,c,nthreads) private(tid,i,j,k)

{

tid=omp\_get\_thread\_num();

#pragma omp for

  for (i = 0; i < MAX; i++){  
    for (j = 0; j < MAX; j++) {  
         a[i][j] = 0;  
         b[i][j] = 100 + i;  
    }  
    a[i][i] = 1;  
  }  
  
  /\*\*  
  Multiplicacion de las matrices  
  \*/

#pragma omp for

  for (i = 0; i < MAX; i++){  
    for (j = 0; j < MAX; j++) {  
       c[i][j] = 0;  
       for (k = 0; k < MAX; k++){  
         c[i][j] += a[i][k] \* b[k][j];  
      }

}  
 }

}  
  /\*\*  
  Impresion de la matriz C  
  \*/  
  for (i = 0; i < MAX; i++) {  
    for (j = 0; j < MAX; j++)  
      printf("%5d ", c[i][j]);  
    printf("\n");  
  }  
  return 0;  
}

Se compila y seguidamente se ejecuta, midiendo su tiempo de ejecución.

$ gcc -fopenmp multimat\_parallel.c -o multimat\_parallel && time ./multimat\_parallel

…

599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599 599

./multimat\_parallel 2.40s user 0.03s system 868% cpu 0.280 total

**Preguntas a desarrollar:**

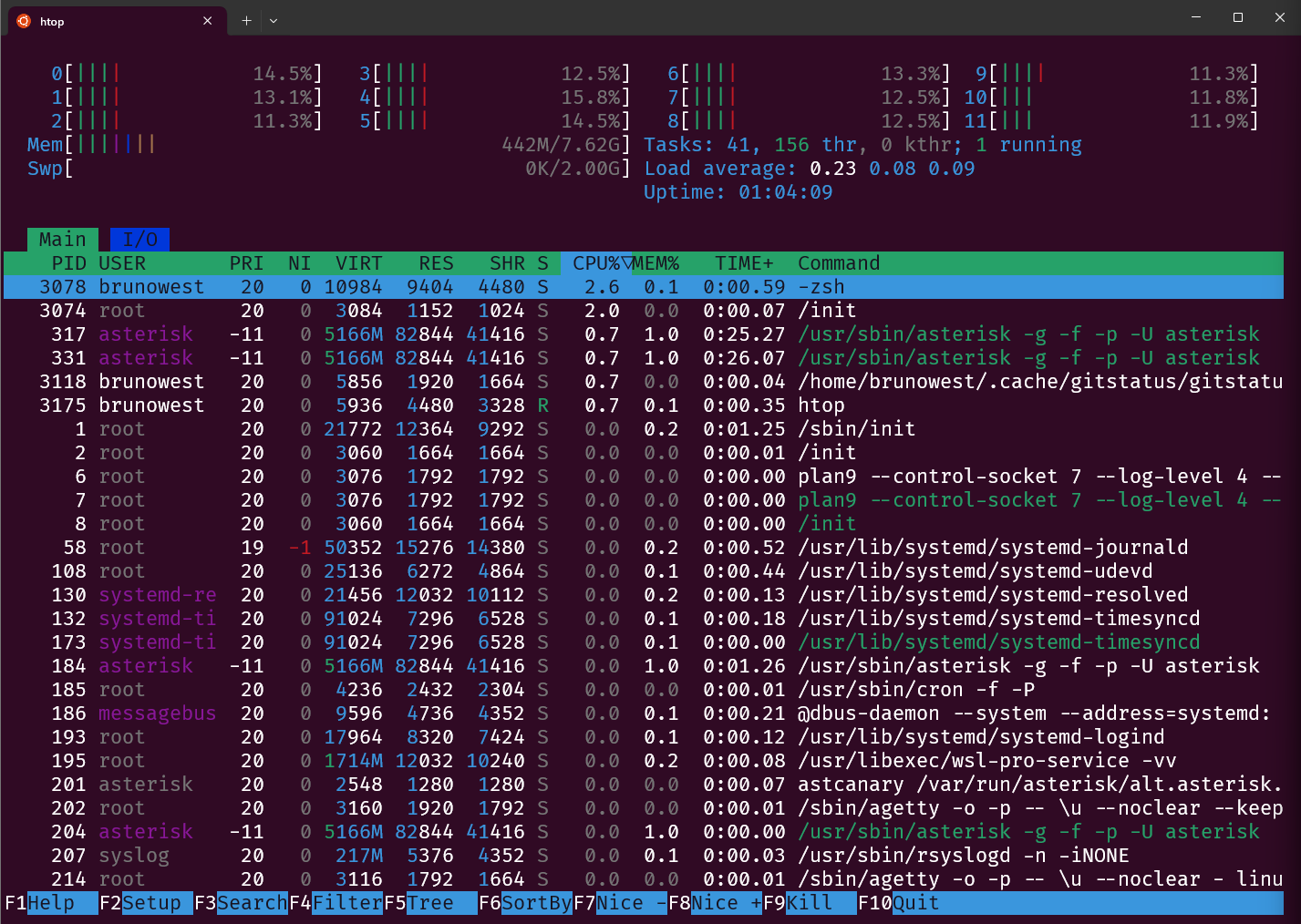
1. ¿Cuánto es la mejora del código paralelo versus el código secuencia o serial en el Lenguaje C?

La mejora del código paralelo respecto al secuencial es casi 2 veces más rápida que la secuencial.

1. ¿Qué partes aún siguen siendo secuenciales en ambos códigos del Lenguaje C?

Las partes que siguen siendo secuenciales son la inicialización de variables como las matrices y las variables para la iteración de bucles, impresión por consola de las matrices y por último todo lo que esta fuera de la sección de #pragma omp hasta el return 0.

1. En Linux, mediante el programa htop, compruebe el uso de todos los núcleos del procesador (Adjunte imagen)

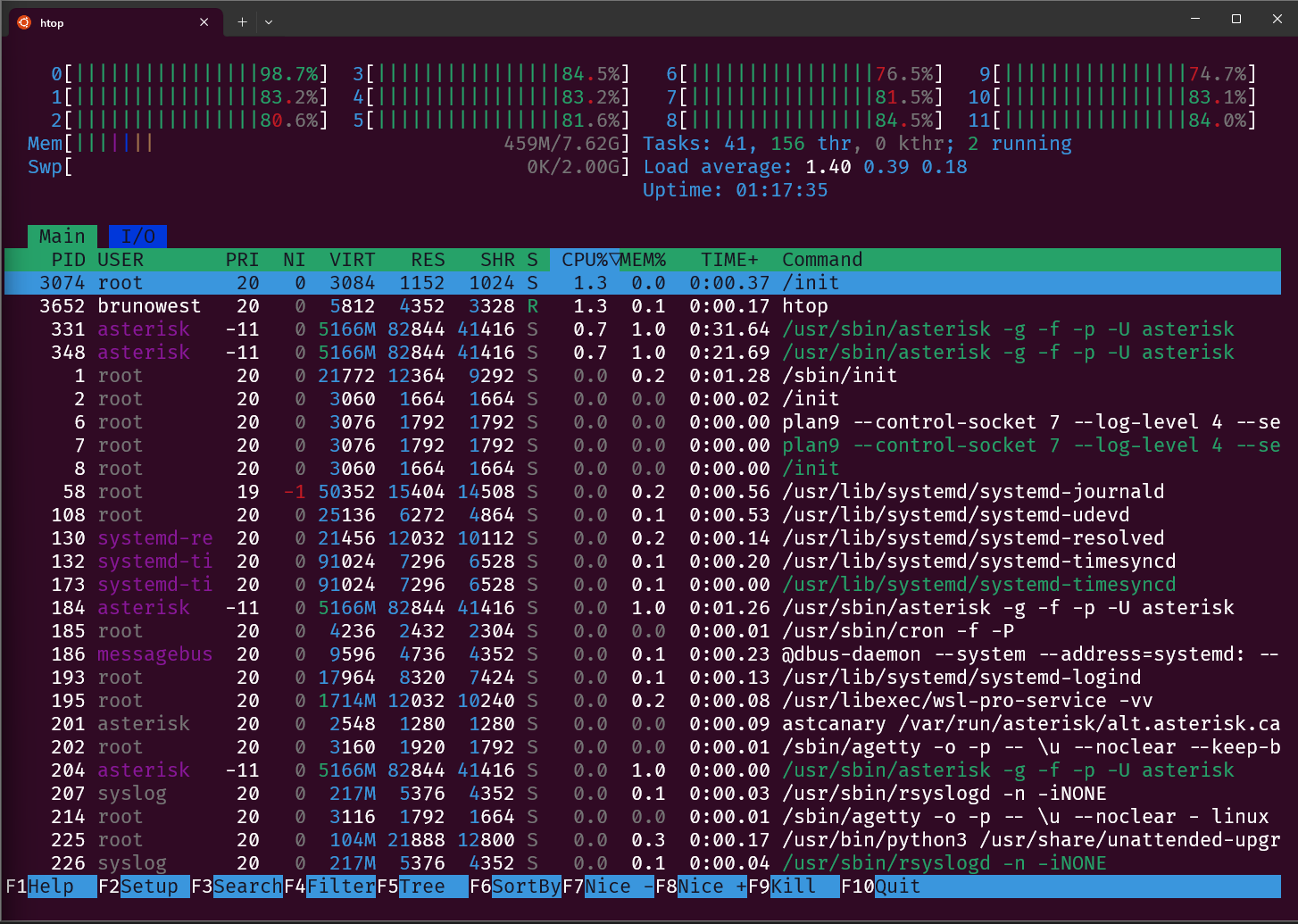


1. Investigue como se implementaría el código Python de forma paralela.

Para implementar en Python se necesita importar una librería llamada multiprocessing en la cual va a dividir el trabajo en procesos e hilos manualmente.

|  |
| --- |
| import multiprocessing as mp  MAX = 500  # Inicialización de matrices  a = [[0 for \_ in range(MAX)] for \_ in range(MAX)]  b = [[100 + i for \_ in range(MAX)] for i in range(MAX)]  c = [[0 for \_ in range(MAX)] for \_ in range(MAX)]  for i in range(MAX):  a[i][i] = 1 # matriz identidad  def multiply\_row(i):  """Multiplica la fila i de A por la matriz B."""  row\_result = [0] \* MAX  for j in range(MAX):  s = 0  for k in range(MAX):  s += a[i][k] \* b[k][j]  row\_result[j] = s  return i, row\_result  if \_\_name\_\_ == "\_\_main\_\_":  with mp.Pool(processes=mp.cpu\_count()) as pool:  results = pool.map(multiply\_row, range(MAX))  # Insertar resultados en C en el orden correcto  for i, row in results:  c[i] = row  # Impresión de la matriz C  for i in range(MAX):  for j in range(MAX):  print(f"{c[i][j]:5d}", end=" ")  print() |

python3 multimat\_parallel.py 17.34s user 0.34s system 856% cpu 2.065 total



1. También implemente el programa paralelo en Lenguaje C y Python en un entorno Windows.

En el caso de Powershell de Windows se tiene la cantidad de tiempo de ejecución para C

|  |
| --- |
| Measure-Command { ./multimat\_parallel.exe }  Days : 0  Hours : 0  Minutes : 0  Seconds : 4  Milliseconds : 164  Ticks : 41648315  TotalDays : 4.8204068287037E-05  TotalHours : 0.00115689763888889  TotalMinutes : 0.0694138583333333  TotalSeconds : 4.1648315  TotalMilliseconds : 4164.8315 |

Para Python

|  |
| --- |
| Measure-Command { python multimat\_parallel.py }  Days : 0  Hours : 0  Minutes : 0  Seconds : 3  Milliseconds : 583  Ticks : 35833798  TotalDays : 4.14743032407407E-05  TotalHours : 0.000995383277777778  TotalMinutes : 0.0597229966666667  TotalSeconds : 3.5833798  TotalMilliseconds : 3583.3798 |

Reflexión:

En el caso de Ubuntu, la librería OpenMP funciona de manera más eficiente porque el kernel de Linux tiene un mejor soporte para la programación paralela y la asignación de hilos. En cambio, en Windows, la sobre carga del manejo de threads suele penalizar el rendimiento OpenMP sobre todo si el trabajo en paralelo no está balanceado de manera óptima. Y para el lenguaje Python la cantidad de tiempo de ejecución en Windows se tardó apróximamente 1 segundo más que en la distro de Linux.