Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Guadalajara

Ingeniería en Robótica y Sistemas Digitales

Clave: TE2004B.501

Parallel Programming

Matrix multiplication in open MP

Alumno:

Jorge Carrillo Castro - A01634630

Profesor:

Manuel Rodriguez Bahena

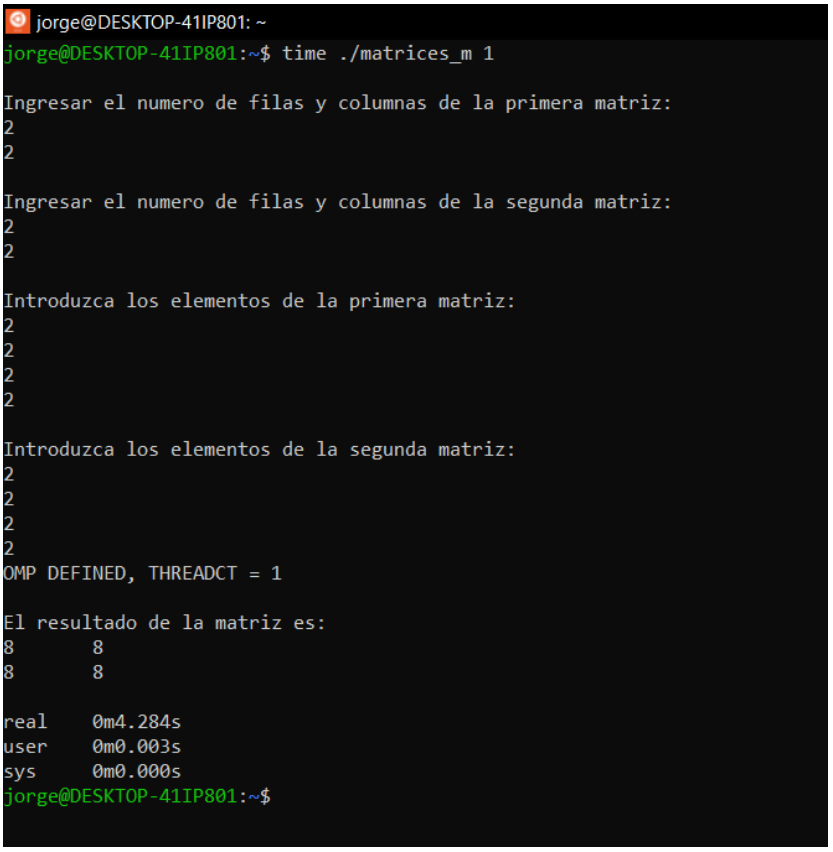
Fecha y lugar entrega:
5 de Noviembre de 2022
Zapopan Jalisco

Link github: https://github.com/jorgais1234/parallel-programming-ITESM.git

Para esta actividad tuvimos que hacer un programa que fuera capaz de multiplicar dos matrices y dar el resultado en otra. El programa también debía ser capaz de detectar si el tamaño de las matrices no permitía la multiplicación, es decir el número de columnas de la matriz A debe ser igual al número de filas de la matriz B. Una vez con este script, implementamos la programación en paralelo para ver cómo cambiaban los tiempos dependiendo de la cantidad de cores que utilizabamos. La parte en donde implementamos la programación paralela fue en los ciclos for que ejecutan la multiplicación.

El primer código que desarrolló necesitaba los siguientes inputs: Tamaño de la matriz A, tamaño de la matriz B y los valores de cada uno de los elementos de las matrices. Hice varias pruebas para poder demostrar el cambio en los tiempos de ejecución. Is

Prueba con 1 core



Prueba con 2 cores

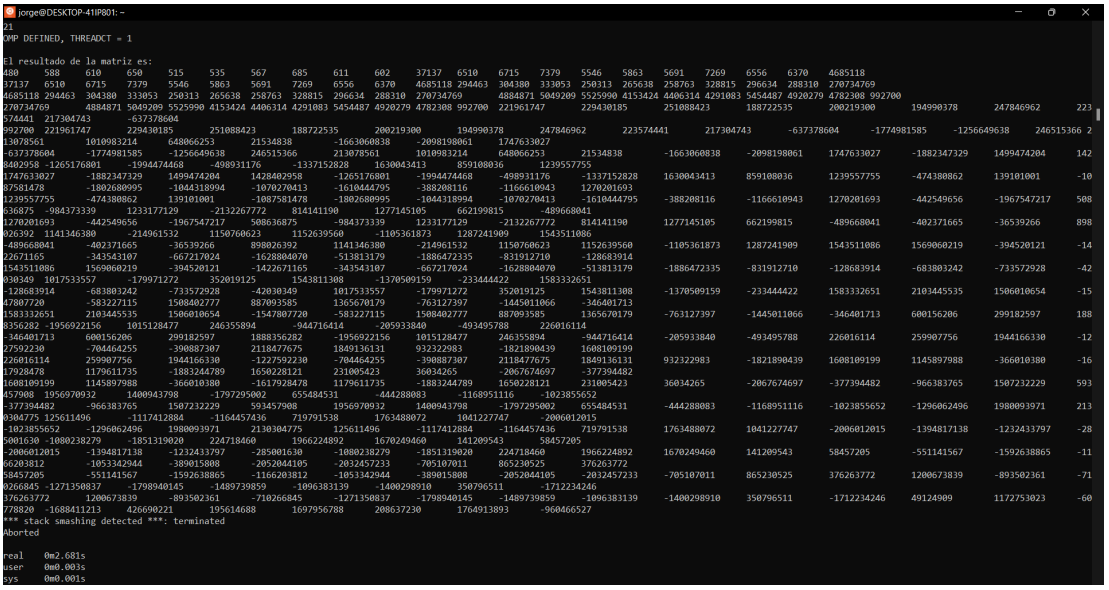
```
jorge@DESKTOP-41IP801:~$ time ./matrices_m 2

Ingresar el numero de filas y columnas de la primera matriz:
2
2

Ingresar el numero de filas y columnas de la segunda matriz:
2
2

Introduzca los elementos de la primera matriz:
2
2
22
2

Introduzca los elementos de la segunda matriz:
2
2
2
2
OMP DEFINED, THREADCT = 2

El resultado de la matriz es:
8       8
48      48

real    0m2.732s
user    0m0.003s
sys     0m0.000s
jorge@DESKTOP-41IP801:~$
```

Prueba con 15 cores

```
jorge@DESKTOP-41IP801:~$ time ./matrices_m 15

Ingresar el numero de filas y columnas de la primera matriz:
2
2

Ingresar el numero de filas y columnas de la segunda matriz:
2
2

Introduzca los elementos de la primera matriz:
2
2
2
2

Introduzca los elementos de la segunda matriz:
2
2
2
2
OMP DEFINED, THREADCT = 15

El resultado de la matriz es:
8       8
8       8

real    0m2.687s
user    0m0.006s
```

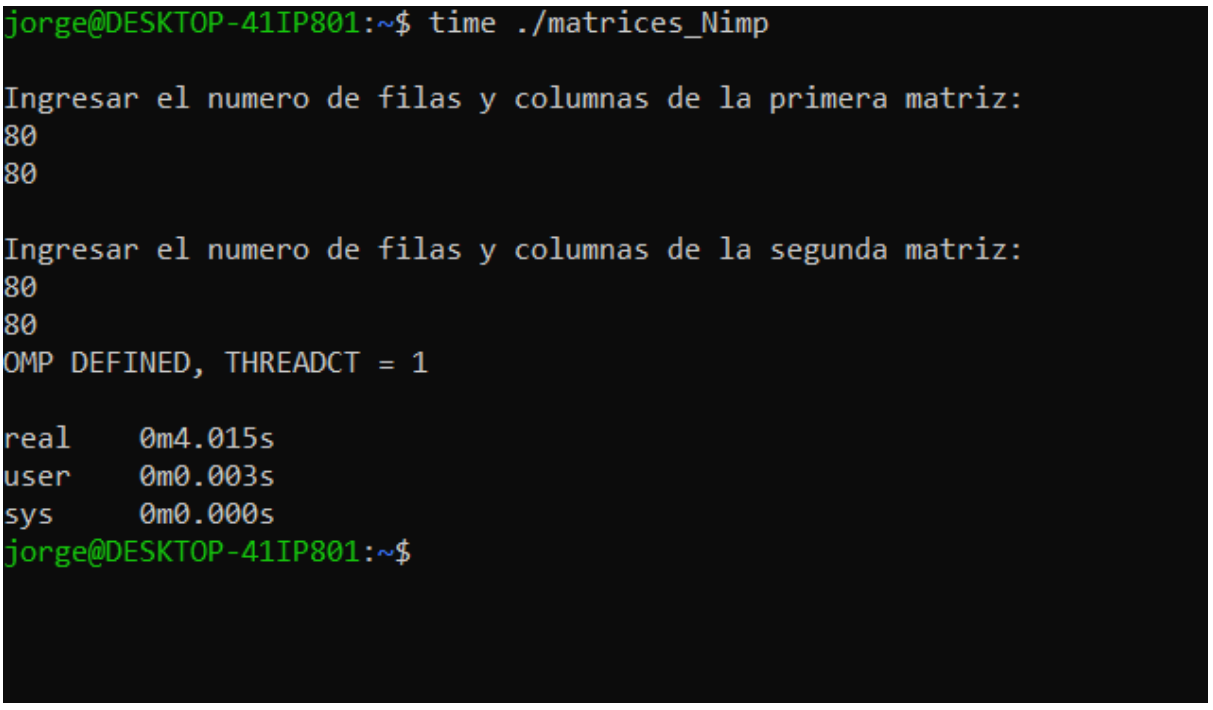
Justo antes de empezar a hacer las gráficas de tiempo de todos los experimentos, me percaté que a pesar de que el tiempo de ejercicio si se reduce conforme incrementa la cantidad de threads, este no es el único factor que altera el tiempo. A pesar de que intente introducir los datos a todos los experimentos con la misma velocidad, es imposible que tarde exactamente el mismo tiempo, por lo cual decidí modificar mi código y hacer que las matrices se llenen de números aleatorios.

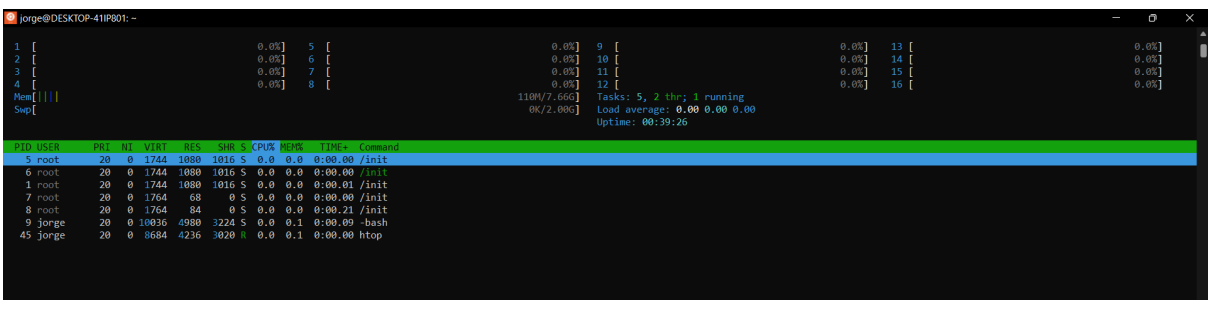
Mi código imprimió la matriz resultante de la multiplicación pero al momento de insertar números grandes de matrices, se aborta la ejecución y se detenía el programa.



Debido a esto tuve que comentar la parte en la que se imprime la matriz resultante para poder comparar los tiempos de manera correcta.

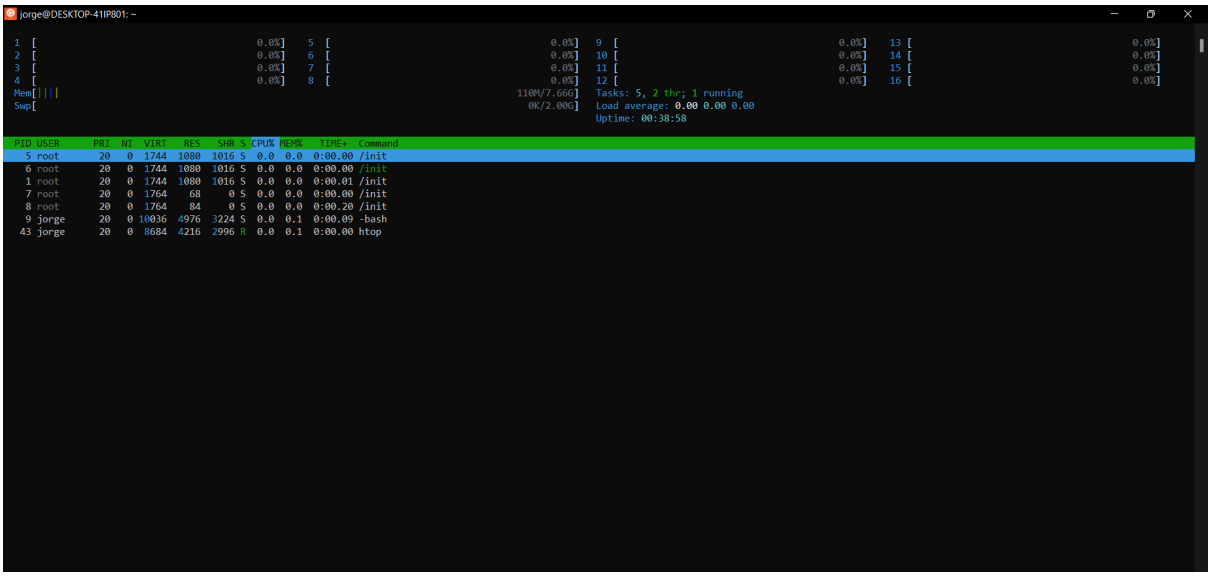
Prueba con 1 core





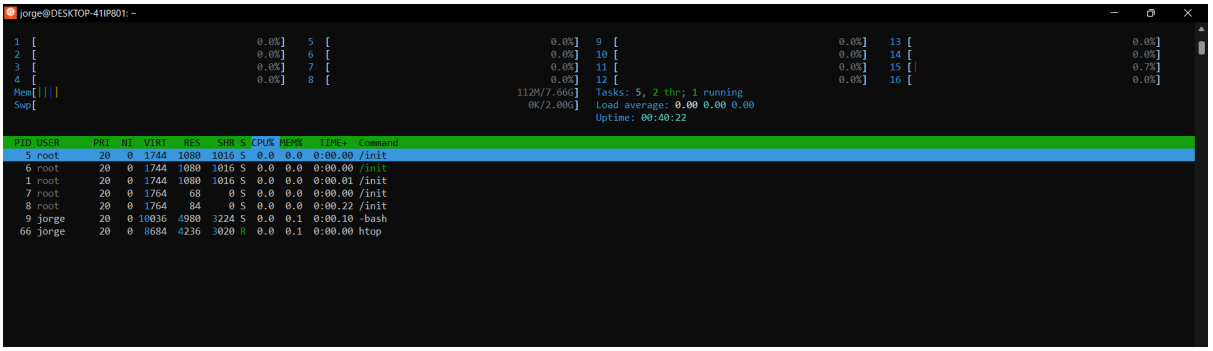
Prueba con 2 core

```
jorge@DESKTOP-41IP801:~$ time ./matrices_Nimp 2

Ingresar el numero de filas y columnas de la primera matriz:
80
80

Ingresar el numero de filas y columnas de la segunda matriz:
80
80
OMP DEFINED, THREADCT = 2

real    0m2.447s
user    0m0.008s
sys     0m0.000s
jorge@DESKTOP-41IP801:~$
```



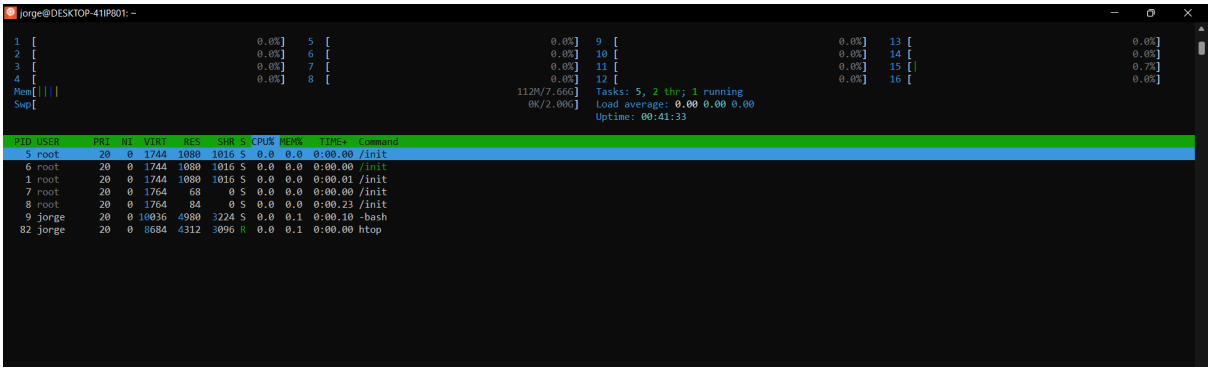
Prueba con 10 cores

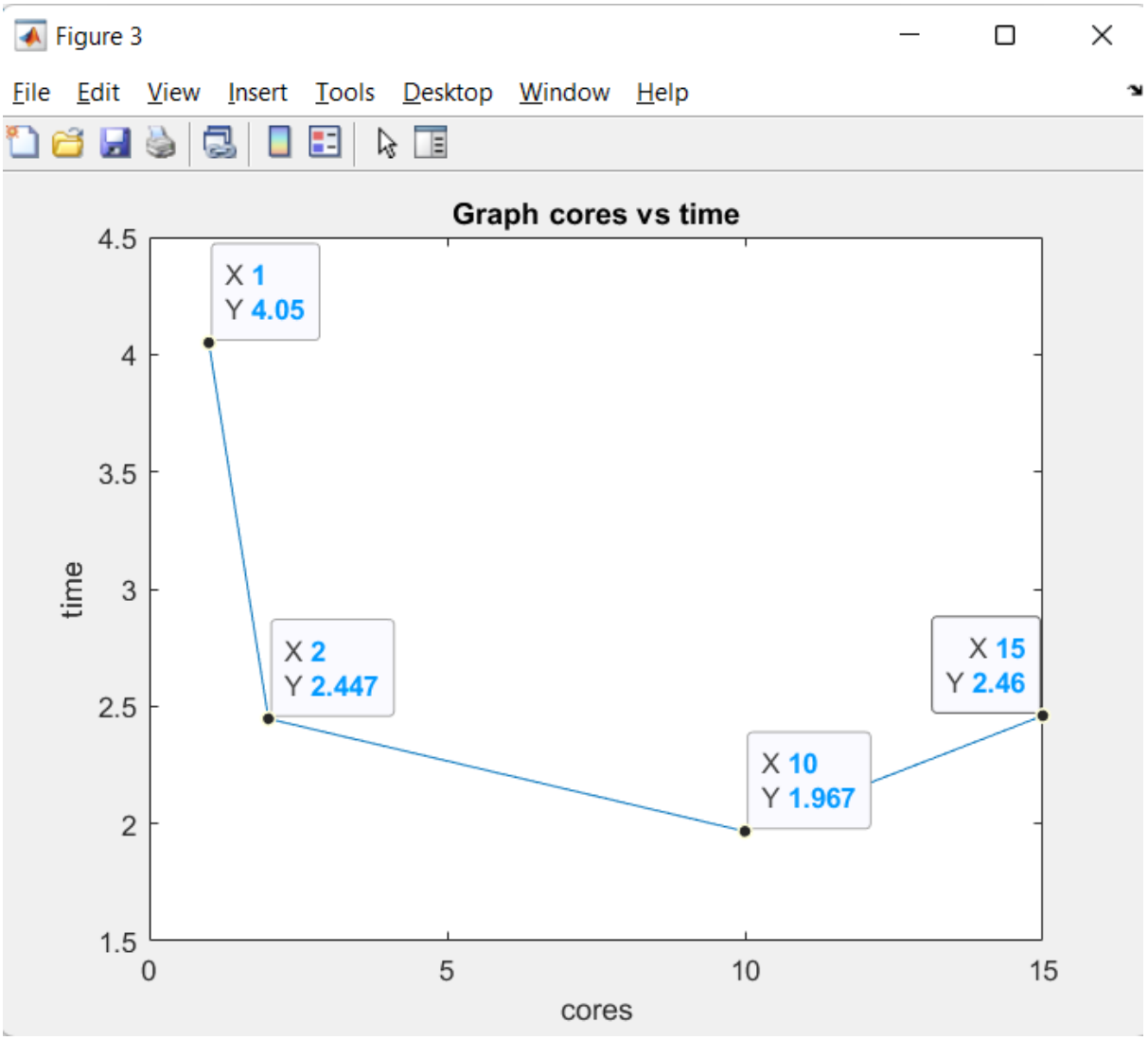
```
jorge@DESKTOP-41IP801:~$ time ./matrices_Nimp 10

Ingresar el numero de filas y columnas de la primera matriz:
80
80

Ingresar el numero de filas y columnas de la segunda matriz:
80
80
OMP DEFINED, THREADCT = 10

real    0m1.967s
user    0m0.005s
sys     0m0.000s
jorge@DESKTOP-41IP801:~$
```



Prueba con 15 cores

```
jorge@DESKTOP-41IP801:~$ time ./matrices_Nimp 15

Ingresar el numero de filas y columnas de la primera matriz:
80
80

Ingresar el numero de filas y columnas de la segunda matriz:
80
80
OMP DEFINED, THREADCT = 15

real    0m2.460s
user    0m0.005s
sys     0m0.000s
jorge@DESKTOP-41IP801:~$
```





Conclusión

En esta actividad aprendí a implementar la programación en paralelo en mis códigos. En los primeros experimentos que hice los resultados no eran confiables porque dependían mucho de la velocidad con la que ingresaba los datos. La segunda parte (donde solo inserto el tamaño de la matriz) es la más confiable por lo cual la utilice para mi análisis. En la gráfica se puede ver que al inicio se sigue la regla que entre más cores pongamos, menor es el

tiempo de ejecución. En la última prueba donde puse la mayor cantidad de colores, el tiempo subió. Esto puede ser por dos razones:

- Tarde más en introducir el tamaño de las matrices
- LLega un punto en el que seguir agregando cores perjudica el rendimiento ya que es complicado hacer la coordinación de esa cantidad de cores

Esta actividad me fue de mucha utilidad ya que en clase todo es sencillo por que vamos de la mano del profesor, pero el verdadero reto empieza cuando es momento de hacerlo solo. Tardé un poco en definir la línea de #Pragma por que no sabia que variables debían de ser privadas, hasta que recordé que en clase mencionaron que los contadores siempre son los privados.