Trabajo en Grupo

Sistemas Operativos Avanzados
0341405

Pablo Alonso Serrano - 137557
Jacob Altenburger Villar - 146659
Jorge Alejandro Cadrecha Del Rey – 139846
Rubén Rodríguez García - 143780
Diego Villena Macarron - 139301

Introducción

El principal objetivo de esta práctica es desarrollar un sistema para calcular la suma de los elementos de una matriz de 10 x 10 elementos empleando Py-MPI (Message Passing Interface). Este enfoque permite explorar conceptos de programación paralela y comunicación entre procesos en entornos distribuidos

En concreto, se busca que el sistema cumpla con las siguientes tareas:

- En nodo 0 definirá una matriz de 10 x 10 y calculará la suma de los elementos de la fila 0.

0, 0, 0, ..., 0

1, 1, 1, ..., 1

... ...

9, 9, 9, ..., 9

- Tres nodos serán los encargarán de calcular de manera independiente la suma de tres filas cada uno.

- En nodo 0 mostrará el resultado de la suma de todos los elementos de la matriz

El entorno utilizado fue una máquina virtual configurada con un sistema operativo Linux concretamente Kali-Linux. Dicha máquina virtual cuenta con cuatro procesadores del procesador nativo Intel I7-9700F 3.00GHz y 8192 MB de memoria RAM, características que permiten ejecutar las pruebas necesarias para abordar la práctica.

Instalación

El paso principal fue la instalación de varias librerías necesarias para la ejecución de Py-MPI, las principales dependencias utilizadas fueron:

- **openmpi-bin:** Paquete base para el uso de MPI
- **libopenmpi-dev :** Herramientas de desarrollo para OpenMPI.

Estas librerías son necesarias para sistemas operativos basados en Linux como Ubuntu o Debian.

La instalación se realizó utilizando el comando:

“sudo apt-get install openmpi-bin libopenmpi-dev”

Durante este paso inicial se presentó una incidencia relacionada con archivos faltantes. Este inconveniente quedo pendiente para el futuro.

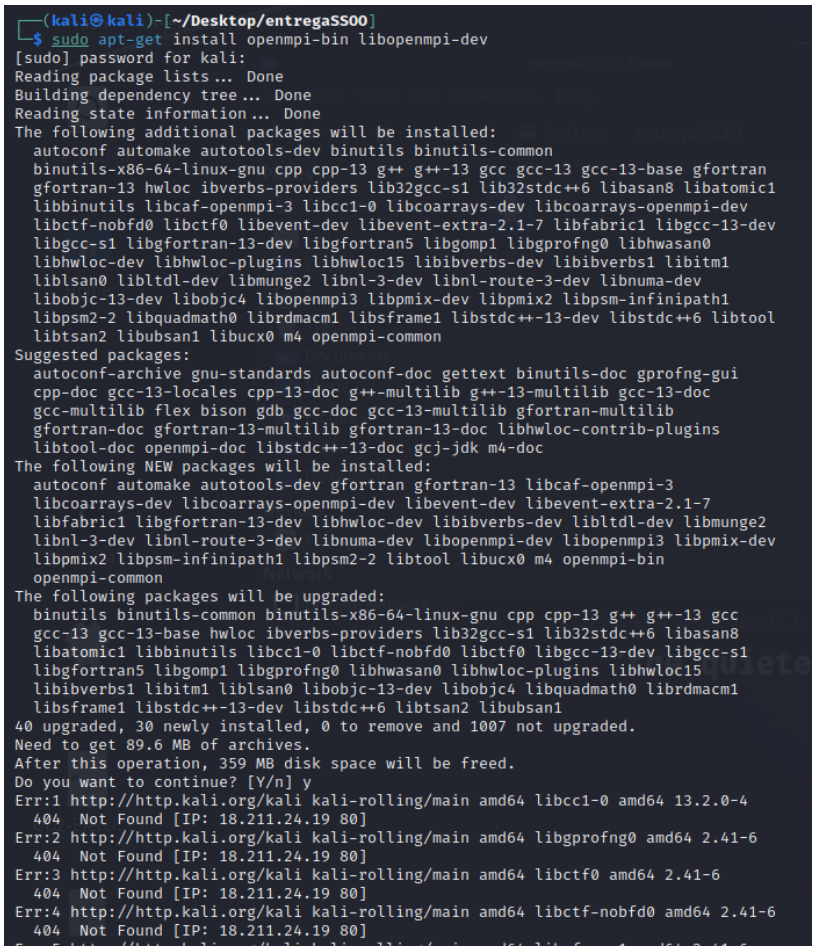

Ilustración 1. Instalación de openmpi-bin y libopenmpi-dev

Posteriormente, se realizó la instalación de librerías de Python que son:

- **mpi4py:** Biblioteca que facilita el uso de MPI desde Python, proporcionando una interfaz sencilla para la programación paralela.

- **numpy:** Esta herramienta fue seleccionada para realizar las operaciones de cálculo de manera eficiente, debido a que es utilizado para tipos de ámbitos parecidos. Simplifica bastante las operaciones matemáticas gracias a su amplia gama de funciones específicas, como la función “sum”, que permite calcular la suma de los elementos de una fila de la matriz , devolviendo el resultado de manera rápida y precisa.

El proceso de instalación de estas librerías se realizó con el gestor de paquetes pip mediante el siguiente comando:

“pip install mpi4py numpy”

Sin embargo, al igual que con las librerías de OpenMPI, el sistema notifico que hay archivos faltantes, asique se realizó una investigación sobre los problemas encontrados.

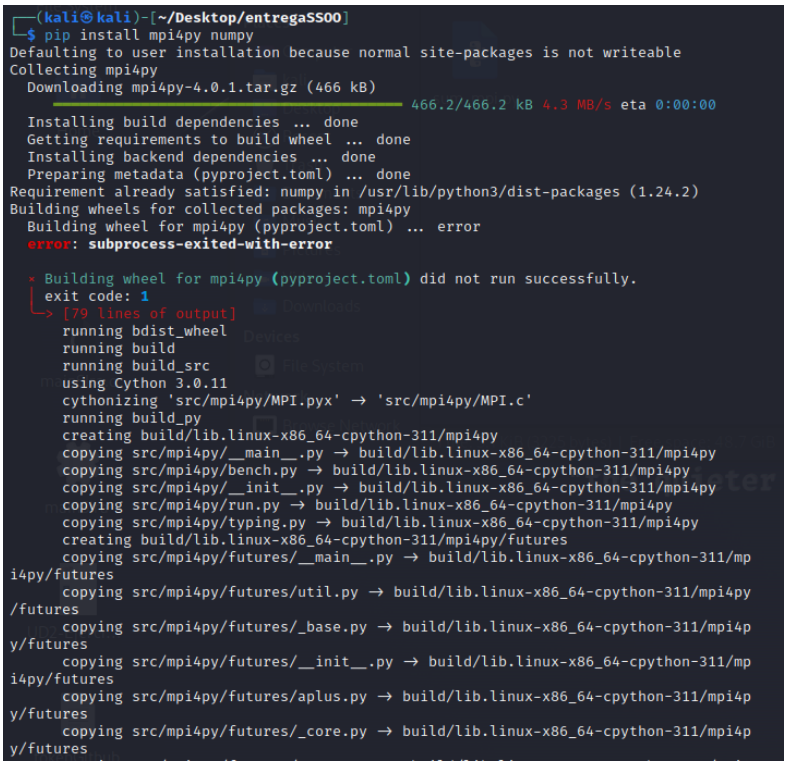

Ilustración 2. Instalación de mpi4py y numpy

Tras investigar los posibles orígenes del problema, se identificó que las listas de paquetes del sistema no estaban actualizadas.

Para resolver este problema, se ejecutó el comando

- **“sudo apt-get update”:** Este comando se encarga de actualizar las listas de paquetes disponibles, permitiendo una correcta instalación de las dependencias.

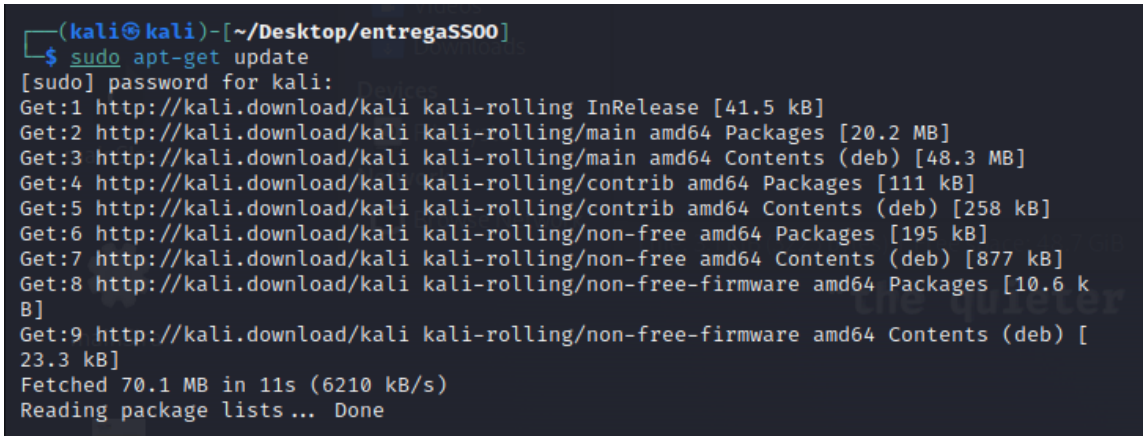

Ilustración 3. Actualización de las listas de paquetes del sistema

Una vez actualizadas las listas de paquetes del sistema, se repitió la instalación de las librerías, logrando finalmente la instalación correcta de mpi4py, numpy, openmpi-bin y libopenmpi-dev , confirmando que el error se ubicaba por no a ver actualizado las listas previamente.

Verificación de la instalación

Para comprobar que la instalación de "mpirun" fue exitosa, se utilizó el comando

- **"which mpirun":** Este comando permite visualizar la ruta donde está instalado.

Además, comprobamos la versión instalada haciendo uso del comando

- **"mpitun – versión"** : En este caso, la versión es la 5.0.6, lo que asegura la compatibilidad con el resto de las herramientas y librerías utilizadas en el proyecto.

Implementación del código

El desarrollo del programa se realizó en Python, empleando las librerías mpi4py y numpy para implementar la lógica del cálculo distribuido. El código cuenta con comentarios detallados para facilitar la comprensión de su funcionamiento. Además, este código está disponible en el repositorio de GitHub para su acceso y revisión (Link al repositorio).

Para mejor comprensión a continuación se explican las funciones claves utilizadas en MPI

- **send y recv:** Estas funciones, invocadas mediante el comunicador comn, son fundamentales para la comunicación punto a punto entre los procesos.
    - **send:** Permite que un proceso envíe datos a otro proceso específico dentro del comunicador.
    - **recv:** Facilita la recepción de datos enviados por otro proceso.
- **gather:** Esta función permite que el proceso maestro recoja una variable concreta calculada por cada uno de los procesos trabajados.

Código

```python
from mpi4py import MPI
import numpy as np

def main():
    # aqui se inicializan las variables de MPI
    comm = MPI.COMM_WORLD # se crea el comunicador de procesos
    rank = comm.Get_rank()  # esta variable sirve para identificar al proceso actual
    size = comm.Get_size()  # para saber la cantidad de procesos totales

    # checkeo que hay 4 procesos exactos
    if size != 4:
        if rank == 0: # si no pones esto todos los procesos sacan el mensaje y solo debe hacerlo el proceso maestro
            print("Este programa requiere exactamente 4 procesos")
        MPI.Finalize() # se ciera MPI
        return

    # el proceso 0 crea la matriz
    if rank == 0:
        # crea matriz 10x10 donde cada fila contiene su número de indice
        matriz = np.array([[i] * 10 for i in range(10)]) # esta linea esta sacada de internet, se usa una funcion de numpy para agilizar el proceso de creacion de la matriz y sus contenidos
        print("Matriz original:\n")
        print(f"{matriz}\n")
        print("Distribuyendo trabajo entre procesos...\n")

        # el proceso 0 suma la priemra fila
        suma_parcial = np.sum(matriz[0]) # sum tambien es una funcion de numpy que suma todos los elementos de la fila
        print(f"Proceso {rank} suma fila 0\n")
        print(f"Suma parcial del proceso {rank}: {suma_parcial}\n")
```

Ilustración 4. Código

```
    # se distribuyen las fials entre los procesos restantes
    for i in range(1, size):
        start_row = 1 + (i-1) * 3 # se decide la fila inicial en funcion del proceso
        end_row = start_row + 3 if i < size-1 else 10 # se decide la fila final
        rows_to_send = matriz[start_row:end_row]
        comm.send(rows_to_send, dest=i) # se usa la funcion send para mandar las filas al proceso correspondiente
        print(f"Enviando filas {start_row} a {end_row-1} al proceso {i}\n")

# aqui se reciben las filas y se suman
else:
    # los procesos trabajadores reciben desde el proceso maestro
    filas = comm.recv(source=0) # recv recoge la informacion mandada por el proceso maestro, source es el proceso desde el cual se recibe por eso es 0
    # calcula la suma de las fials recibidas
    suma_parcial = np.sum(filas)
    print(f"Proceso {rank} suma filas {1+(rank-1)*3} a {min(1+(rank-1)*3+2, 9)}")
    print(f"Suma parcial del proceso {rank}: {suma_parcial}\n")


# se recogen todas las sumas parciales en el proceso maestro
sumas_parciales = comm.gather(suma_parcial, root=0) # gather recoge la variable definida entre los parentesis de cada proceso, y los alamcena en un array
```

Ilustración 5. Código 2

```
    # el proceso maestro calcula y muestra la suma total
    if rank == 0:
        suma_total = sum(sumas_parciales)
        print("Resultados finales:")
        print(f"Sumas parciales de cada proceso: {sumas_parciales}")
        print(f"Suma total de la matriz: {suma_total}\n")

        # verificación de que los calculos son correctos
        expected_sum = np.sum(matriz)
        print(f"Verificación:")
        print(f"Suma calculada de forma distribuida: {suma_total}")
        print(f"Suma calculada directamente: {expected_sum}")
        print(f"¿Resultados coinciden? {suma_total == expected_sum}\n")
if __name__ == "__main__":
    main()
```

Ilustración 6. Código 3

Una vez completado el código se procedió con la ejecución del código para comprobar su funcionamiento mediante el comando “mpirun -n 4 sum_mpi.py” donde:

- **“mpirun”:** Es el comando encargado iniciar la ejecución de programas que utilizan MPI. Además, gestiona la comunicación entre los procesos involucrados coordinando el intercambio de datos.

- **-n 4:** Especifica el número de procesos a lanzar, en este caso, 4.

```
┌──(kali㉿kali)-[~/Desktop/entregaSSOO]
└─$ mpirun -n 4 python sum_mpi.py
Matriz original:

[[0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 1 1]
 [2 2 2 2 2 2 2 2 2 2]
 [3 3 3 3 3 3 3 3 3 3]
 [4 4 4 4 4 4 4 4 4 4]
 [5 5 5 5 5 5 5 5 5 5]
 [6 6 6 6 6 6 6 6 6 6]
 [7 7 7 7 7 7 7 7 7 7]
 [8 8 8 8 8 8 8 8 8 8]
 [9 9 9 9 9 9 9 9 9 9]]

Distribuyendo trabajo entre procesos ...

Proceso 0 suma fila 0

Suma parcial del proceso 0: 0

Enviando filas 1 a 3 al proceso 1

Enviando filas 4 a 6 al proceso 2

Enviando filas 7 a 9 al proceso 3

Proceso 2 suma filas 4 a 6
Suma parcial del proceso 2: 150

Proceso 1 suma filas 1 a 3
Suma parcial del proceso 1: 60

Proceso 3 suma filas 7 a 9
Suma parcial del proceso 3: 240

Resultados finales:
Sumas parciales de cada proceso: [0, 60, 150, 240]
Suma total de la matriz: 450

Verificación:
Suma calculada de forma distribuida: 450
Suma calculada directamente: 450
¿Resultados coinciden? True
```

Ilustración 4. Ejecución del programa

Conclusión

Durante el desarrollo de este trabajo, tuvimos la oportunidad de implementar un sistema distribuido utilizando Py-MPI y exponer nuestros conocimientos de programación paralela. A través de la resolución del programa de suma de la matriz, abordamos conceptos clave como la comunicación entre procesos y la integración de nuevas librerías como mpi4py y numpy.

Asimismo, enfrentamos desafíos como la instalación y configuración de librerías en un entorno Linux, destacando la importancia de mantener actualizadas las listas de paquetes del sistema. Este proceso también fomento la búsqueda de documentación técnica, explorar ejemplos prácticos en foros como Stack Overflow, y revisar contenido digital en plataformas como Youtube.

En conclusión, el proyecto no solo nos ha parecido interesante, sino que también nos ha permitido mejorar nuestros conocimientos sobre programación paralela, además de mejorar nuestras habilidades en planificación, análisis de errores y la generación de documentación. Aprendizajes que serán de gran utilizad una vez finalizados los estudios universitarios.