

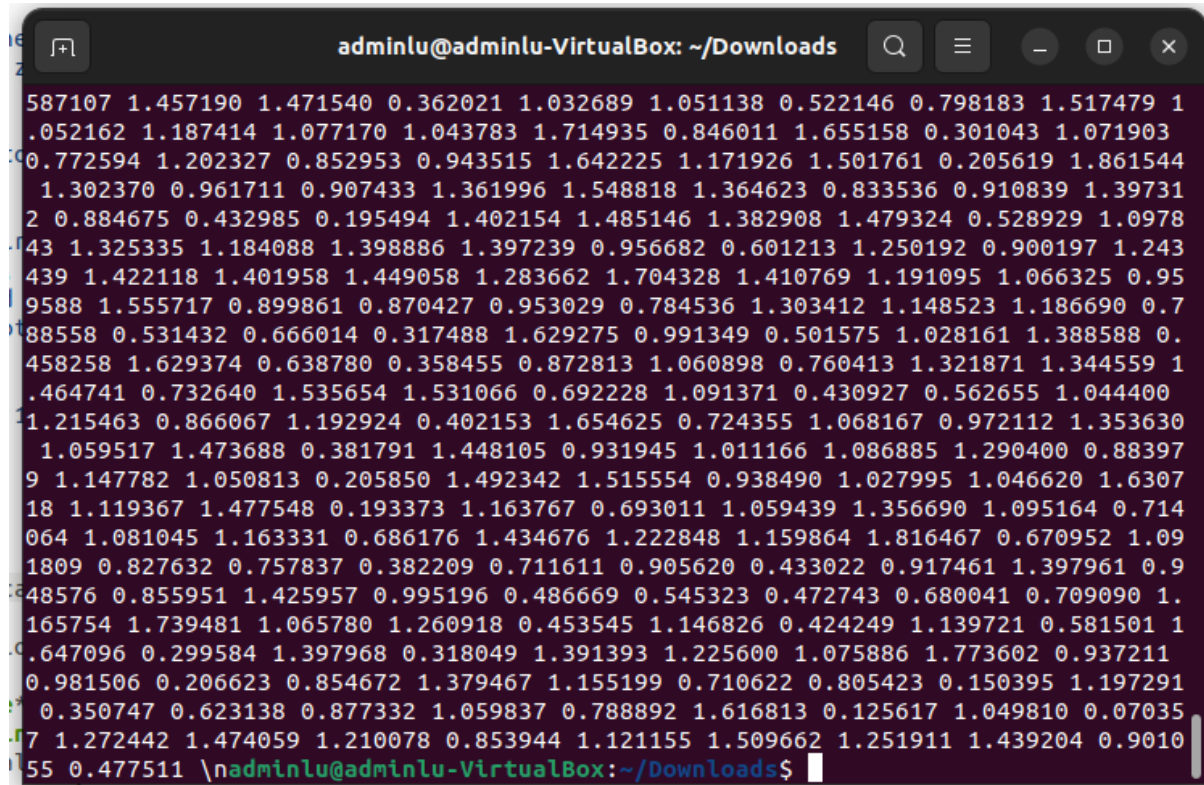
Universidad del Valle de Guatemala	Programación Paralela y Distribuida
Laboratorio 4 - MPI	28 / 7 / 2023

Laboratorio 3 - Vectores

1. Explique por qué y cómo usamos comunicación grupal en las siguientes funciones de `mpi_vector_add.c`:
 - **Check_for_error():** Verifica errores y proporciona mensajes de error descriptivos. Si detecta un error imprime los mensajes y detiene los procesos
 - **Read_n():** Lee el tamaño de los vectores y determina el tamaño local de cada subvector que se asignará a cada proceso.
 - **Read_data():** Se utiliza para leer un vector de entrada y distribuirlo entre los diferentes procesos usando “MPI_SCATTER”.
 - **Print_vector():** Imprime un vector.
 - **Comunicación Grupal:** Las funciones de comunicación grupal `MPI_Bcast()`, `MPI_Scatter()` y `MPI_Gather()` se utilizan para distribuir y recopilar datos entre procesos, mientras que `MPI_Allreduce()` podría ser utilizado en `Check_for_error()` para sincronizar y verificar errores en todos los procesos. Estas funciones facilitan la programación paralela al proporcionar operaciones de alto nivel para la comunicación entre procesos.
2. Descargue y modifique el programa `vector_add.c` para crear dos vectores de al menos 100,000 elementos generados de forma aleatoria. Haga lo mismo con `mpi_vector_add.c`. Imprima únicamente los primeros y últimos 10 elementos de cada vector (y el resultado) para validar. Incluya captura de pantalla.
 - `Vector_add.c` modificado:

```
adminlu@adminlu-VirtualBox:~/Downloads$ ./vectoradd
A part of the sum is
36.60 133.00 162.20 67.40 191.60 182.40 138.20 78.20 94.00 42.00
adminlu@adminlu-VirtualBox:~/Downloads$ ./vectoradd
A part of the sum is
36.60 133.00 162.20 67.40 191.60 182.40 138.20 78.20 94.00 42.00
adminlu@adminlu-VirtualBox:~/Downloads$ ./vectoradd
A part of the sum is
49.60 73.20 21.40 196.40 65.80 169.20 66.00 34.60 23.20 33.40
adminlu@adminlu-VirtualBox:~/Downloads$ ./vectoradd
A part of the sum is
1.00 190.20 166.00 184.40 37.80 48.00 22.00 47.80 7.20 127.20
adminlu@adminlu-VirtualBox:~/Downloads$
```

- mpi_vector_add modificado:



3. Mida los tiempos de ambos programas y calcule el speedup logrado con la versión paralelo. Realice al menos 10 mediciones de tiempo para cada programa y obtenga el promedio del tiempo de cada uno. Cada medición debe estar en el orden de los ~5 segundos para asegurar valores estables (utilice una cantidad de elementos adecuada para que a su maquina le tome por lo menos ~5 cada corrida). Utilice esos promedios para el calculo del speedup. Incluya capturas de pantalla.

# Corrida	Corrida MPI_vector_sum	Corrida vector_add
1	0.11422	0.000984
2	0.183279	0.000912
3	0.167783	0.000871
4	0.373297	0.001128

5	0.189118	0.00830
6	0.160378	0.00799
7	0.117595	0.000577
8	0.304715	0.000790
9	0.321036	0.000820
10	0.15655	0.000778

- Tiempo promedio para MPI_vector_sum: 0.2088 segundos.
- Tiempo promedio para vector_add (secuencial): 0.00232 segundos.

El speedup promedio, basado en estas mediciones, es aproximadamente 0.0111

Esto significa que la versión secuencial es alrededor de 90 veces más rápida que la versión paralela, lo cual es inusual. Es probable que la versión paralela tenga un overhead significativo debido a la comunicación entre procesos, inicialización de MPI y otras operaciones relacionadas. Además, la operación en sí (suma de vectores) es simple y puede no beneficiarse significativamente de la paralelización, especialmente si el tamaño del vector no es lo suficientemente grande.

d. Modifique el programa mpi_vector_add.c para que calcule de dos vectores 1) el producto punto 2) el producto de un escalar por cada vector (el mismo escalar para ambos). Verifique el correcto funcionamiento de su programa (para ello puede probar con pocos elementos para validar).

```
adminlu@adminlu-VirtualBox:~/Downloads$ mpiexec -n 2 ./mpi_vector_modified
4
Enter the scalar to multiply with the vectors: Scalar multiplied vector x:\n3.36
0751 1.577532 3.132397 3.193760 3.646589 0.790205 1.340891 3.072918 1.111099 2.2
15880 \nScalar multiplied vector y:\n1.909588 2.515484 1.459138 2.053604 3.80891
9 3.664780 2.542847 2.869188 0.566410 2.427876 \nDot product is: 2.212799\nThe s
um is\n0.401103 0.248016 0.285662 0.409920 0.868098 0.180996 0.213105 0.551049 0
adminlu@adminlu-VirtualBox:~/Downloads$ mpiexec -n 6 ./mpi_vector_modified
36
Enter the scalar to multiply with the vectors: Scalar multiplied vector x:\n30.2
46758 14.197785 28.191572 28.743841 32.819305 7.111849 0.635712 0.717297 0.14160
3 0.606969 \nScalar multiplied vector y:\n17.186294 22.639353 13.132241 18.48243
3 34.280270 32.983022 0.635712 0.717297 0.141603 0.606969 \nDot product is: 0.40
1103\nThe sum is\n0.401103 0.248016 0.285662 0.409920 0.868098 0.180996 0.635712
0.717297 0.141603 0.606969 \nadminlu@adminlu-VirtualBox:~/Downloads$
```

E. Reflexión

Se pudo profundizar en operaciones vectoriales paralelas mediante la exploración de programas como "mpi_vector_sum.c", "mpinew.c" y "mpi_vector_modified.c". Estos archivos fueron esenciales para introducirnos a conceptos como la distribución de bloques, lo que nos permitió comprender cómo se pueden dividir y procesar grandes conjuntos de datos de manera eficiente en paralelo. Además, mediante el uso de funciones como MPI_Scatter y MPI_Gather, aprendí a distribuir y recoger datos entre múltiples procesos, lo que comparo con dirigir una orquesta donde cada proceso juega su parte. Estas experiencias no solo ampliaron nuestra comprensión del paralelismo y la programación distribuida, sino que también subrayaron la importancia de la modularidad y la gestión de errores en la programación, preparándome para futuros desafíos académicos y profesionales.