

Adrián Ricardo Flores Trujillo	21500
Andrea Ximena Ramírez Recinos	21874

*Laboratorio 3*  
**OpenMPI**

---

**[Ejercicio I] (10 puntos)**

Explique por qué y cómo usamos comunicación grupal en las siguientes funciones de `mpi_vector_add.c`:

- Check\_for\_error():** El propósito de esta función es verificar si alguno de los procesos ha encontrado un error. En caso afirmativo, se imprime un mensaje y se terminan todos los procesos. De lo contrario, se continúa con la ejecución. Su implementación es importante, porque permite detectar errores en un proceso y notificar a los demás, previniendo así la propagación de datos incorrectos. Además, facilita enormemente la depuración del código al proporcionar información sobre el origen del error.

En este caso, la comunicación grupal se lleva a cabo mediante el uso de `MPI_Allreduce`, que combina los valores de todos los procesos y distribuye el resultado a todos ellos. Aquí, se evalúa si algún proceso ha encontrado un error.

- Read\_n():** Esta función se utiliza para recibir el orden (o tamaño) de los vectores. Es ingresado por el usuario mediante la línea de comando y, posteriormente, este resultado se difunde a los demás procesos. La comunicación grupal en este caso en particular se implementa a través de la llamada a `MPI_Bcast`, donde un proceso (en este caso, el proceso 0) envía los mismos datos a todos los procesos dentro del comunicador.
- Read\_vector()** → Sirve para la lectura de un vector desde la línea de comandos para distribuirlo entre los procesos. En este apartado se utiliza la comunicación grupal mediante la función `MPI_Scatter`. Al implementarla, se envían los datos a todos los procesos dentro del comunicador. Es similar a `MPI_Bcast`, pero con la diferencia muy importante de que `MPI_Scatter` NO manda la misma información a los procesos, sino que distribuye y manda bloques del array.
- Print\_vector()** → A partir de esta función, se puede realizar la impresión de un vector que tiene una distribución en bloques en la salida estándar. Aquí se utiliza la comunicación grupal con `MPI_Gather`, que de manera inversa a `MPI_Scatter`, reúne los bloques que tienen los procesos y que componen al vector y los coloca en un único proceso.

**[Ejercicio II] (15 puntos)**

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.

```

andrea@andrea:~/Documents/python-environment/Paralela/mpi-vector-add$ ./vector_add.c
dd
What's the order of the vectors?
100000

Elements of x are:
38.018787 68.341846 18.734561 2.493814 86.836528 91.493481 36.699760 56.050509 6
6.109909 67.166452
...
22.545324 93.119292 44.039555 70.780508 38.754893 69.104273 49.389447 37.512299
49.648037 30.249650

Elements of y are:
0.163354 48.052010 34.775301 21.839748 42.197236 99.823516 18.571949 27.569751 4
2.754585 66.658916
...
15.272916 29.033867 42.559061 2.161593 14.027396 36.382016 11.482000 93.292264 6
3.360905 20.552077

The sum is:
38.182141 116.393856 53.509862 24.333562 129.033763 191.316996 55.271709 83.6202
59 108.864494 133.825368
...
37.818240 122.153159 86.598616 72.942101 52.782288 105.486289 60.871447 130.8045

```

Figura 1 - Primeros y últimos 10 elementos de vector aleatorio (vector\_add.c).

```

andrea@andrea: ~/Documents/python-environment/Paralela/mpi-vector-add
File Edit View Search Terminal Help
What's the order of the vectors?
100000

Elements of x are:
74.512336 56.361717 76.671927 84.153216 66.291198 41.194818 26.680106 83.772808
23.054794 84.934933
...
42.822951 4.339746 0.288649 6.571483 12.215506 38.011512 81.163763 9.628413 37.0
41027 82.583189

Elements of y are:
40.785084 61.805613 6.766031 66.988049 23.964122 46.972168 41.930917 82.032586 7
3.407471 65.653330
...
17.924714 37.846828 18.596792 71.781875 73.621449 2.908253 59.867406 70.264642 9
7.613846 7.563947

The sum is:
115.297420 118.167330 83.437959 151.141265 90.255321 88.166985 68.611023 165.805
395 96.462265 150.588264
...
60.747665 42.186575 18.885441 78.353359 85.836955 40.919764 141.031169 79.893054
134.654874 90.147136

```

Figura 2 - Primeros y últimos 10 elementos de vector aleatorio (mpi\_vector\_add.c).

### [Ejercicio III] (15 puntos)

Mida los tiempos de ambos programas y calcule el speedup logrado con la versión paralela. 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 máquina le tome por lo menos ~5 cada corrida). Utilice estos promedios para el cálculo del speedup. Incluya capturas de pantalla.

```

andrea@andrea: ~/Documents/python-environment/Paralela/mpi-vector-add
File Edit View Search Terminal Help

Elements of x are:
40.040892 90.690186 40.664255 39.820699 91.500971 16.294361 2.421895 71.734576 5
4.178055 42.239098
...
25.007246 86.400612 5.637443 47.848275 9.037718 68.961853 84.570225 36.656497 81
.869918 46.017746

Elements of y are:
66.031264 35.471618 66.192209 58.489033 47.769418 6.829614 15.036733 95.018497 1
0.694435 11.200851
...
63.906686 42.324236 9.047388 74.157417 69.656583 87.002025 21.935024 58.500558 2
8.844281 52.037381

The sum is:
106.072157 126.161804 106.856464 98.309732 139.270389 23.123976 17.458628 166.75
3073 64.872490 53.439949
...
88.913932 128.724848 14.684831 122.005692 78.694301 155.963878 106.505249 95.157
055 110.714199 98.055126

Took 3.820053 seconds to run

```

Figura 3 - Prueba de obtención de tiempos de ejecución (mpi\_vector\_add.c).

```

andrea@andrea: ~/Documents/python-environment/Paralela/mpi-vector-add
File Edit View Search Terminal Help

Elements of x are:
3.671324 59.274448 1.649918 50.581676 19.693161 79.709329 36.947601 33.473940 45
.478073 70.045035
...
49.150604 90.888424 78.975073 57.871034 53.210138 27.406867 16.867344 47.858672
79.153824 78.427965

Elements of y are:
19.317734 42.428727 19.732028 28.872686 34.403876 83.466081 83.207741 91.863052
54.899467 87.559807
...
71.065498 26.728594 81.231260 36.654088 10.401119 68.547295 10.408554 25.814994
85.040209 55.902357

The sum is:
22.989058 101.703176 21.381947 79.454362 54.097037 163.175410 120.155342 125.336
992 100.377540 157.604841
...
120.216102 117.617018 160.206333 94.525122 63.611256 95.954162 27.275898 73.6736
66 164.194033 134.330321

Took 4.807966 seconds to run

```

Figura 4 - Prueba de obtención de tiempos de ejecución (vector\_add.c).

N	Tiempo Secuencial (s)	Tiempo Paralelo (s)
100,000,000	4.53	3.92
	4.89	3.88
	4.82	3.16
	5.38	3.48
	4.87	3.40
	4.84	3.53
	4.84	3.23
	4.76	3.80
	5.15	4.06
	4.80	3.82
Promedio	4,888	3,628

Cuadro 1 - Tiempo de ejecución de programas.

$$SpeedUp = \frac{Tiempo\ secuencial}{Tiempo\ paralelo} = \frac{4,888}{3,628}$$

$$SpeedUp = 1.35$$

#### [Ejercicio IV] (55 puntos)

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). Incluya captura de pantalla.

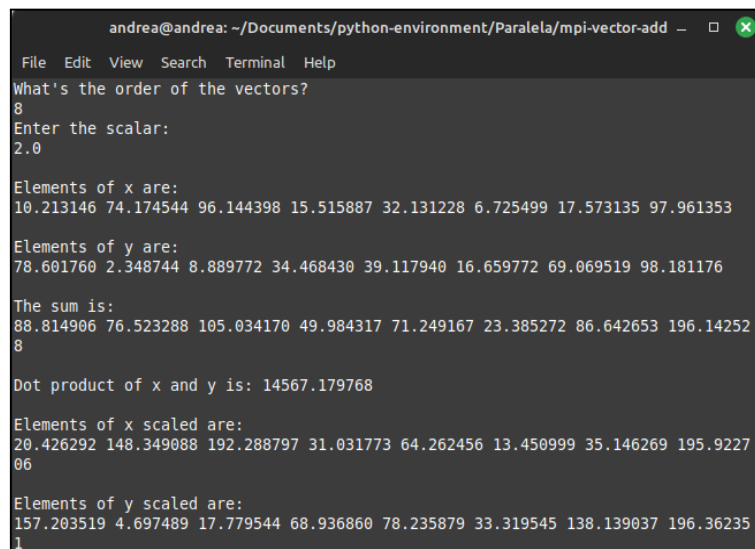


```

andrea@andrea: ~/Documents/python-environment/Paralela/mpi-vector-add
File Edit View Search Terminal Help
andrea@andrea:~/Documents/python-environment/Paralela/mpi-vector-add$ mpiexec -n
4 ./mpi_vector_add
What's the order of the vectors?
8
Elements of x are:
98.046895 53.340967 83.976689 44.484723 69.562752 85.613922 55.990790 77.440790
Elements of y are:
11.712710 68.734042 91.870135 50.857106 72.088253 82.581593 52.759126 65.281348
The sum is:
109.759605 122.075009 175.846824 95.341829 141.651005 168.195515 108.749917 142.
722138
Dot product of x and y is: 34886.304901
Took 0.000098 seconds to run

```

Figura 5 - Ejecución de obtención de producto punto.



```

andrea@andrea: ~/Documents/python-environment/Paralela/mpi-vector-add
File Edit View Search Terminal Help
What's the order of the vectors?
8
Enter the scalar:
2.0
Elements of x are:
10.213146 74.174544 96.144398 15.515887 32.131228 6.725499 17.573135 97.961353
Elements of y are:
78.601760 2.348744 8.889772 34.468430 39.117940 16.659772 69.069519 98.181176
The sum is:
88.814906 76.523288 105.034170 49.984317 71.249167 23.385272 86.642653 196.14252
8
Dot product of x and y is: 14567.179768
Elements of x scaled are:
20.426292 148.349088 192.288797 31.031773 64.262456 13.450999 35.146269 195.9227
06
Elements of y scaled are:
157.203519 4.697489 17.779544 68.936860 78.235879 33.319545 138.139037 196.36235
1

```

Figura 6 - Elementos escalados (Valor = 2.0).

#### [Ejercicio V] (15 puntos)

Finalmente, escriba una reflexión del laboratorio realizado en donde hable de las técnicas aplicadas, lo que se aprendió y pudo repasar, elementos que le llamaron la atención, ediciones/mejoras que considera que son posibles y cualquier otra cosa relevante que tengan en mente.

- En general, considero que este laboratorio nos permitió profundizar en los conocimientos adquiridos en clase, sobre MPI. Un aspecto que destacó durante la realización de este laboratorio fue el manejo adecuado de la memoria, así como el uso efectivo de las directivas de MPI, en particular `MPI_Scatter` y `MPI_Gather`. Como se sabe, son funciones para la

distribución y recolección de datos entre los procesos, y su correcta implementación podría llegar a representar un reto en algunas ocasiones.

Además, se evidenciaron claramente los principios de comunicación colectiva entre procesos. Se observó que estos principios no solo facilitan la cooperación entre los distintos procesos, sino que también optimizan la eficiencia en el manejo de grandes volúmenes de datos, tal como se evidencia en los resultados de tiempo de ejecución de los programas.

En términos de mejoras, considero que la presentación de los vectores podría mejorarse para ofrecer una interfaz más amigable al usuario. En lo que respecta a aspectos más técnicos, creo que sería beneficioso optimizar el manejo de los arreglos. Actualmente, uno de ellos (denominado *z*) se sobrescribe con los resultados de cada vector, lo que podría generar inconsistencias en ciertas situaciones, aunque no se observaron problemas durante las pruebas realizadas. De igual forma se sugiere implementar un manejo más robusto.

Finalmente, creo que se podría explorar la paralelización de la generación aleatoria de números. Sin embargo, como se ha observado en otros laboratorios, esta tarea puede ser compleja debido a las limitaciones de la función `rand()`, que no es adecuada para entornos paralelos debido a su naturaleza no concurrente. Una posible solución sería utilizar librerías más adecuadas para la generación de números aleatorios.