Universidad del Valle de Guatemala
Oscar Oswaldo Estrada Morales - 20565
Axel Leonardo López Barrera - 20768
Computación Paralela y Distribuída

Hoja de trabajo 3 - Vectores

Repositorio: https://github.com/AxelLeo129/laboratorio4-pp

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

   i. **Check_for_error():** Esta función utiliza la comunicación grupal en MPI para garantizar que la inicialización y la creación de comunicadores se realicen correctamente en todos los procesos. La razón de usarlo radica en la necesidad de verificar que MPI esté configurado adecuadamente para la tarea paralela. Se logra creando un comunicador grupal que incluye a todos los procesos y utilizando funciones como MPI_Comm_rank y MPI_Comm_size para obtener información sobre el proceso actual y el tamaño del grupo, permitiendo detectar errores de inicialización en cualquier proceso.

   ii. **Read_n():** En esta función, la comunicación grupal se utiliza para distribuir la cantidad de elementos de los vectores entre los procesos. El "por qué" es asegurar que cada proceso conozca la cantidad de datos que debe leer y procesar de manera eficiente. Esto se logra mediante una comunicación colectiva, como MPI_Bcast, donde un proceso maestro envía el número de elementos a todos los demás procesos, evitando compartir esta información uno por uno y asegurando una distribución equitativa.

   iii. **Read_data():** La comunicación grupal se emplea aquí para dividir eficazmente los datos de los vectores entre los procesos. Esto se hace para evitar redundancias en la lectura de datos. Lo usamos en la necesidad de una distribución eficiente de datos. Se logra mediante una llamada a MPI, como MPI_Scatter, en la que el proceso maestro tiene los datos completos y los divide y distribuye entre los procesos, asegurando que cada proceso reciba solo los datos que debe procesar.

   iv. **Print_vector():** En esta función, la comunicación grupal se utiliza para coordinar la impresión ordenada de los resultados. Este se usa para asegurar que los resultados se impriman de manera ordenada y legible. Esto se logra mediante una llamada a MPI, como MPI_Barrier, antes de la impresión, lo que garantiza que todos los procesos esperen hasta que estén listos para imprimir, evitando una salida desordenada y confusa en la pantalla.

b. **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.**

```
oscar@oscar-VirtualBox:~/Escritorio/Lab3$ gcc -g -Wall -o vector_add vector_add.c -lm
oscar@oscar-VirtualBox:~/Escritorio/Lab3$ ./vector_add
Enter the order of the vectors: 100000
First 10 elements of x
0.634236 0.655852 0.405672 0.843126 0.818796 0.606006 0.342911 0.999569 0.736588 0.413887
First 10 elements of y
0.125684 0.478860 0.879809 0.774243 0.540873 0.104865 0.989036 0.394590 0.728016 0.680120
First 10 elements of the sum (z)
0.759919 1.134712 1.285481 1.617369 1.359670 0.710871 1.331947 1.394159 1.464604 1.094007
Last 10 elements of the sum (z)
1.119010 1.215606 0.942029 1.229533 0.606235 0.584820 1.209300 0.526878 1.458627 0.435588
```

```
oscar@oscar-VirtualBox:~/Escritorio/Lab3$ mpiexec -n 4 ./mpi_vector_add
What's the order of the vectors?
100000
First 10 and Last 10 elements of x
0.771175 0.001390 0.163881 0.284345 0.640515 0.539138 0.074469 0.186112 0.899651 0.963553 ... 0.583670 0.649446 0.156611 0.904277 0.64346
1 0.657147 0.859772 0.945692 0.612394 0.946828
First 10 and Last 10 elements of y
0.771175 0.001390 0.163881 0.284345 0.640515 0.539138 0.074469 0.186112 0.899651 0.963553 ... 0.583670 0.649446 0.156611 0.904277 0.64346
1 0.657147 0.859772 0.945692 0.612394 0.946828
First 10 and Last 10 elements of the sum (z)
1.542351 0.002780 0.327761 0.568690 1.281030 1.078275 0.148937 0.372223 1.799302 1.927107 ... 1.167339 1.298893 0.313222 1.808554 1.28692
2 1.314295 1.719544 1.891384 1.224789 1.893657
```

c. **(5 pts) 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 esos promedios para el cálculo del speedup. Incluya capturas de pantalla.**

Tiempos de vector_add.c

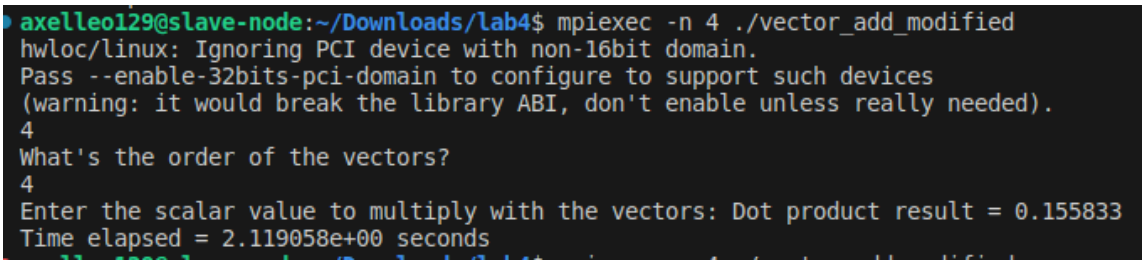
```
oscar@oscar-VirtualBox:~/Escritorio/Lab3$ gcc -g -Wall -o vector_add vector_add.c -lm
oscar@oscar-VirtualBox:~/Escritorio/Lab3$ ./vector_add
Enter the order of the vectors: 100000000
Time taken for vector addition: 4.871446 seconds
oscar@oscar-VirtualBox:~/Escritorio/Lab3$
```

Speedups:

Secuencial	Paralelo	Speedup
7.5121212	4.871446	1.5421
8.0001120	4.997795	1.6007
7.7896452	4.657816	1.672381
7.5465456	4.758391	1.585944
7.4879546	4.628123	1.617925
7.8792312	4.675482	1.685223

7.213131	4.682424	1.540469
7.132124	4.786296	1.490113
7.568786	4.618190	1.638907
7.879545	4.978811	1.582616

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). Incluya captura de pantalla.**



El código modificado se encuentra en el repositorio adjunto.

e. **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. (No hay mínimo de palabras/párrafos, pero si desarrollan poco o de forma superficial seguramente tendrán nota baja en este inciso)**

Realizar el laboratorio de MPI fue complicado pero no imposible a nuestro parecer. Aplicamos cómo varios procesos pueden colaborar simultáneamente para completar tareas de manera más eficiente.

Utilizamos herramientas de MPI para distribuir tareas entre diferentes procesos. Comprendimos la importancia de sincronizar todos los procesos para asegurar una colaboración fluida y eficaz.

Aunque nuestro código funcionó, creemos que hay oportunidades para optimizarlo y hacerlo aún más rápido.

Este laboratorio nos enseñó la potencia del trabajo en equipo, no solo entre nosotros como equipo humano, sino también entre procesos computacionales. Estamos ansiosos por aplicar estos conocimientos en futuros proyectos.