Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
CC3069 Computación Paralela y Distribuida
Catedrático: Miguel Novella
Ciclo 1 de 2023

Laboratorio 3 - Vectores

Evelyn Amaya 19357
Josué Sagastume 18173

Guatemala, 19 de abril de 2023

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

   i. Check_for_error()
   Esta función se utiliza para verificar si hay algún error en el proceso local y luego combina la información de error de todos los procesos en el comunicador. La función utilizada para la comunicación grupal es **MPI_Allreduce()**, esta recopila los valores de la variable **local_ok** de todos los procesos y aplica la operación MIN a ellos y almacena el resultado en la variable **ok**. Si hay algún proceso que encontró un error, es decir que **local_ok == 0**, el resultado de la operación MIN será 0, y así esta función permite que todos los procesos sepan si hay algún error en el grupo.

   ii. Read_n()
   Esta función se utiliza para leer el tamaño del vector **n** en el proceso principal o **rank 0** y luego comparte este valor con todos los otros procesos en el grupo. La función utilizada para la comunicación grupal es **MPI_Bcast()**, esta envía un valor desde el proceso principal a todos los demás procesos en el comunicador.

   iii. Read_vector()
   Esta función se utiliza para leer y distribuir el vector entre todos los procesos. La función utilizada para la comunicación grupal es **MPI_Scatter()**, esta toma un array en el proceso principal o con **rank 0** y distribuye porciones iguales del array a todos los procesos en el comunicador.

   iv. Print_vector()
   Esta función se utiliza para recopilar el vector local en todos los procesos y luego imprimirlo en el proceso principal. La función utilizada para la comunicación grupal es **MPI_Gather()**, esta recopila porciones iguales de datos de todos los procesos del comunicador y los reune en un array en el proceso principal.

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.

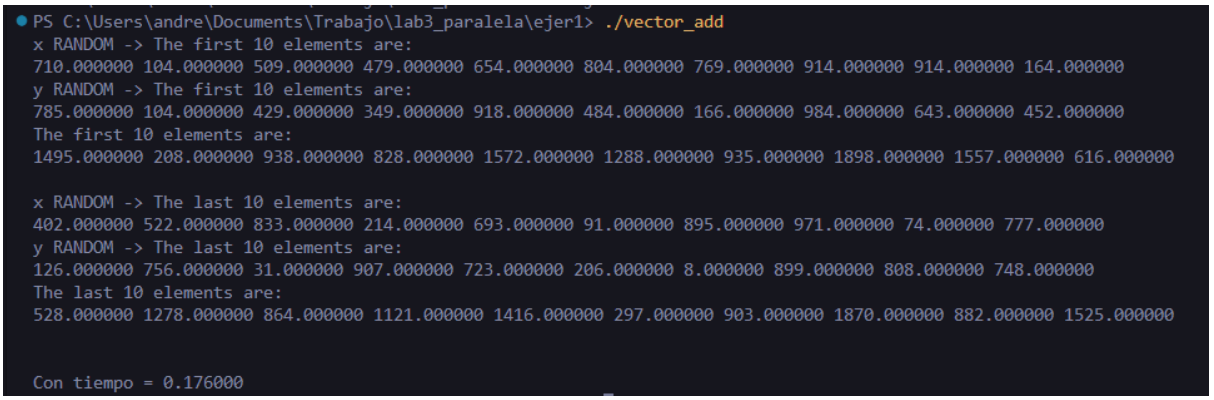

Imagen 1. Ejecución del programa **vector_add.c** con 100,000 elementos para cada vector.

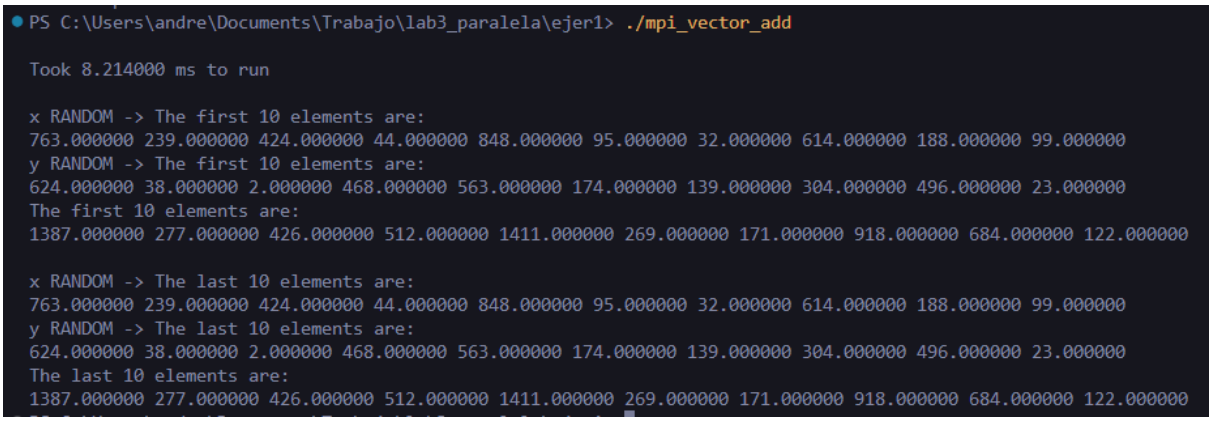

Imagen 2. Ejecución del programa **mpi_vector_add.c** con 100,000 elementos para cada vector.

c. 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.

Tiempo de ejecución secuencial (seg)	Tiempo de ejecución en paralelo (seg)	Speedup
6.377809	2.073722	3.075537126
6.279139	2.032676	3.089099788
6.238272	2.055554	3.034837324
6.227130	1.996549	3.118946743
6.284869	2.054637	3.05887074
6.262041	2.005079	3.123089414
6.278367	2.051502	3.060375764
6.587151	2.039971	3.229041491
6.262055	2.061234	3.038012666
6.260846	2.062730	3.035223224

Tabla 1. Resultados de ejecución de los programas **vector_add.c** y **mpi_vector_add.c** con 100,000,000 elementos para cada vector y el resultado del cálculo de speedup para cada resultado.

Tiempo de ejecución secuencial (seg)	Tiempo de ejecución en paralelo (seg)	Speedup
6.3057679 seg	2.0433654 seg	3.086303428

Tabla 2. Promedio de los resultados de la Tabla 1.

```
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ gcc -g -Wall -o vector_add vector_add.c -fopenmp
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ ./vector_add clear
Con tiempo = 6.377809 s
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ ./vector_add clear
Con tiempo = 6.279139 s
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ ./vector_add clear
Con tiempo = 6.238272 s
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ ./vector_add clear
Con tiempo = 6.227130 s
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ ./vector_add clear
Con tiempo = 6.284869 s
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ ./vector_add clear
Con tiempo = 6.262041 s
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ ./vector_add clear
Con tiempo = 6.278367 s
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ ./vector_add clear
Con tiempo = 6.587151 s
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ ./vector_add clear
Con tiempo = 6.262055 s
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ ./vector_add clear
Con tiempo = 6.260846 s
```

Imagen 3. Ejecución del programa **vector_add.c** con 100,000,000 elementos para cada vector.

```
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ mpicc -g -Wall -o mpi_vector_add mpi_vector_add.c
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ mpiexec -n 4 ./mpi_vector_add

Took 2.073722 s to run
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ mpiexec -n 4 ./mpi_vector_add

Took 2.032676 s to run
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ mpiexec -n 4 ./mpi_vector_add

Took 2.055554 s to run
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ mpiexec -n 4 ./mpi_vector_add

Took 1.996549 s to run
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ mpiexec -n 4 ./mpi_vector_add

Took 2.054637 s to run
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ mpiexec -n 4 ./mpi_vector_add

Took 2.005079 s to run
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ mpiexec -n 4 ./mpi_vector_add

Took 2.051502 s to run
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ mpiexec -n 4 ./mpi_vector_add

Took 2.039971 s to run
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ mpiexec -n 4 ./mpi_vector_add

Took 2.061234 s to run
josue@josue:~/Documents/GitHub/lab3_paralela/ejer1$ mpiexec -n 4 ./mpi_vector_add

Took 2.062730 s to run
```

Imagen 4. Ejecución del programa **mpi_vector_add.c** con 100,000,000 elementos para cada vector

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.

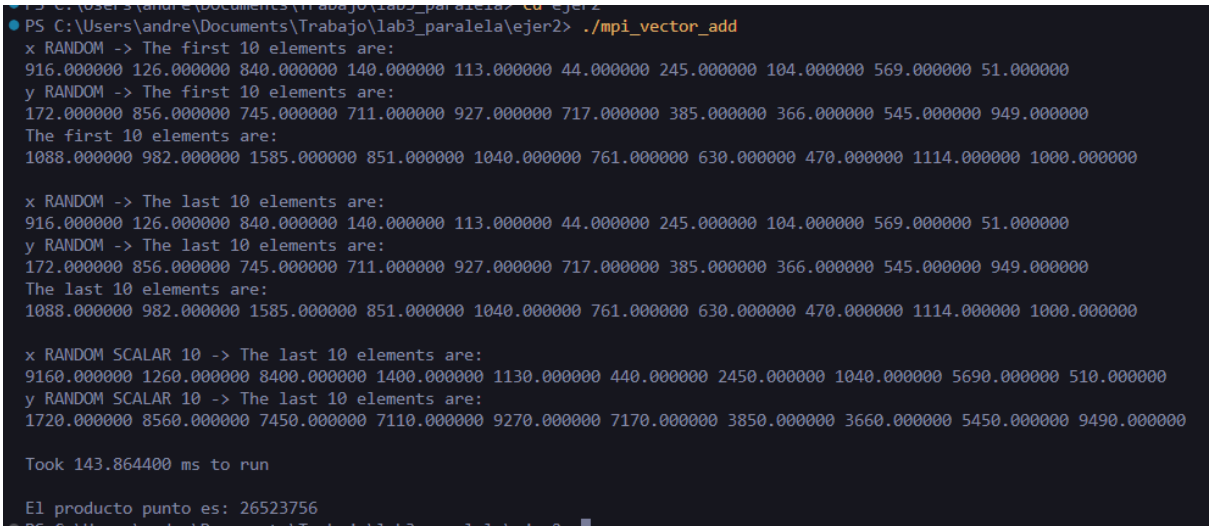

Imagen 5. Ejecución del programa **mpi_vector_add.c** para el cálculo del producto punto y producto escalar de ambos vectores.

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.

La computación de operaciones vectoriales utilizando MPI (Message Passing Interface) ha demostrado ser muy eficiente en la comunicación entre procesos. Varias técnicas se utilizan en el proceso para lograr esta eficiencia. Inicialmente, las funcionalidades propuestas por el laboratorio fueron analizadas exhaustivamente para determinar su potencial reutilización, optimizando así el tiempo de desarrollo y garantizando una implementación consistente.

La primera técnica aplicada fue entender las funciones que ya se nos proveía del laboratorio y así reutilizarlas de ser posible. La segunda técnica fue revisar el correcto funcionamiento de los programas, para el caso del programa paralelo se necesitó descomentar y editar algunas de las funciones ya que no estaba mostrando los resultados como esperado. La tercera técnica fue implementar las funciones que se nos requería de la forma más sencilla: relleno de los vectores de forma aleatoria, producto punto y producto por un escalar.

Los elementos que más nos llamaron la atención en MPI fueron las funciones de *reduce*, que nos permitió ir guardando el resultado local de cada función producto punto y

luego sumarlo todo en una variable global. Fue la primera vez que la implementamos por lo que nos pareció interesante su forma de uso. También nos gustó la función *print vector* por lo práctica que fue reutilizarla para mostrar el contenido de los demás vectores.

En general, el uso de MPI para la aritmética vectorial y la comunicación entre procesos demuestra ser una herramienta poderosa y versátil, pues como se puede observar en la Tabla 2, en promedio se obtuvo un speedup de 3.0863, lo cual es bastante bueno, logrando reducir el tiempo. Las implementaciones eficientes y efectivas se logran utilizando una variedad de métodos y funciones que facilitan el cómputo y la comunicación entre procesos. Esto, a su vez, ha llevado a una mejor comprensión de los beneficios que MPI aporta a la computación paralela y distribuida.