CC3069 Computación Paralela y Distribuida
Sección 11
Erick Stiv Junior Guerra Muñoz - 21781

Laboratorio 3 - Computación Paralela y Distribuida

Repositorio de GitHub: erickguerra22/Lab3_Paralela

1. Explique por qué y cómo usamos comunicación grupal en las siguientes funciones de `mpi_vector_add.c`:
    a. Check_for_error()
    R: Utiliza comunicación grupal al informar a todos los procesos la existencia de un error detectado y que así puedan detener su ejecución. La razón de por qué se utiliza es, precisamente, el poder terminar la ejecución de forma correcta, para evitar la propagación del error y que el programa pueda incurrir en problemas más críticos. Utiliza `MPI_Allreduce` para cumplir con este propósito, ya que esta función se encarga de combinar los resultados de todos los procesos y distribuye el resultado a todos los procesos.

    b. Read_n()
    R: Esta función se encarga de que solamente un proceso deba obtener el tamaño de los vectores a trabajar, para luego transmitir esta información a todo el resto de procesos. Se utiliza comunicación grupal a través de la función `MPI_Bcast`, la cual envía un único mensaje a todos los procesos de un mismo grupo a partir de un proceso “principal”. El motivo de utilizarla es evitar la redundancia al calcular el tamaño de los vectores una única vez y asegurarse que el resto de procesos trabajen con esta misma información.

    c. Read_vector()
    R: En este caso, nuevamente es solo el vector principal (o maestro) el encargado de ingresar el contenido de cada vector para luego compartir la información con el resto de procesos. Sin embargo, esta función utiliza la operación `MPI_Scatter`, con la cual la información no se transmite por completo, sino que es dividida equitativamente y repartida entre todos los procesos (incluido él mismo), para asegurar la división de trabajo.

    d. Print_vector()
    R: Proceso inverso de `Read_vector`. Esta función utiliza la operación `MPI_Gather` para reunir los resultados parciales de cada proceso (resultados obtenidos con el segmento de información otorgado en `Read_vector`), para que luego el proceso maestro sea el encargado de imprimir el resultado final. Aquí se hace uso de la comunicación grupal para reunir toda la información individual obtenida en cada proceso y así llegar al resultado final esperado.

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.

Imagen 2.1: Resultado vector_add.c

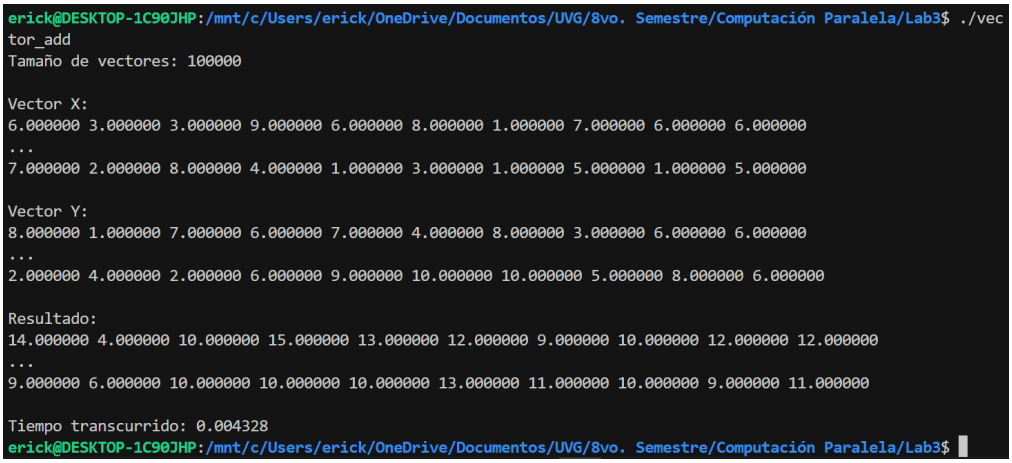
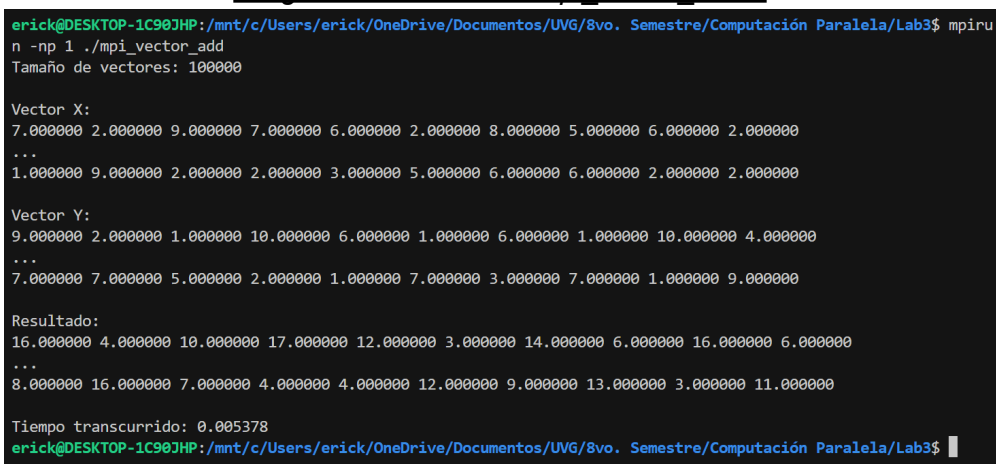


Imagen 2.2: Resultado mpi_vector_add.c



3. 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.

   ***NOTA:** Se realizaron todas las corridas con un tamaño de vectores de 100000000.

Imagen 3.1: Ejemplo de resultado secuencial

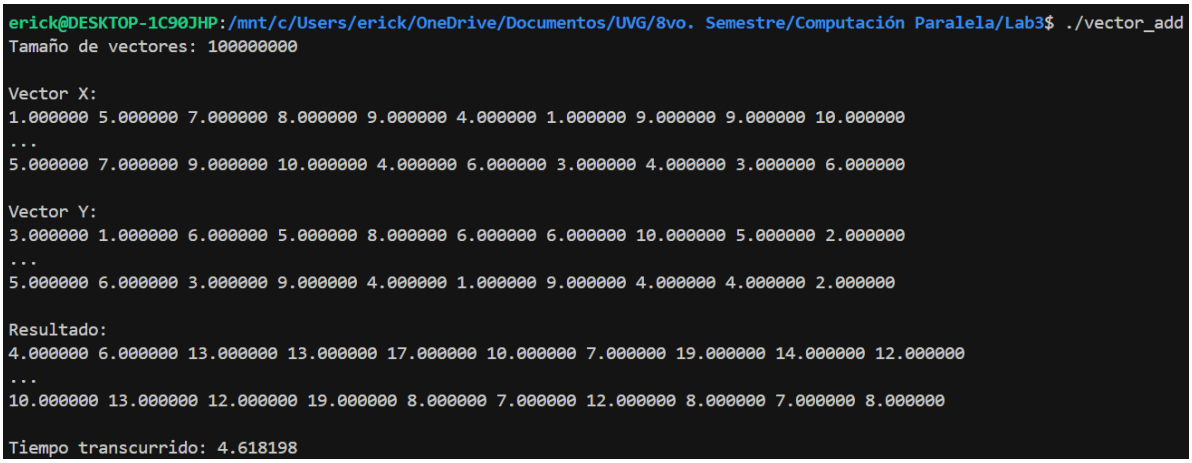


Imagen 3.2: Ejemplo de resultado paralelo (MPI -np 2)

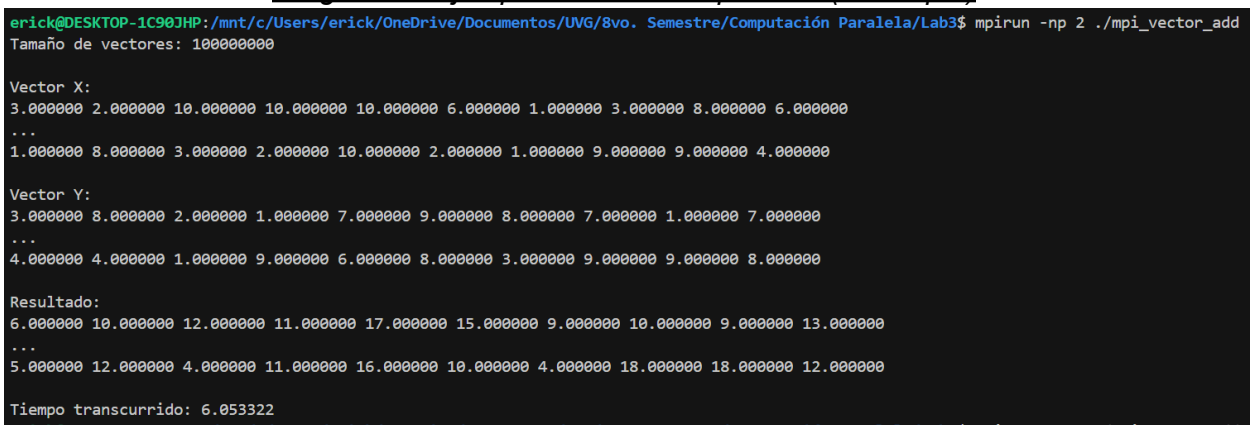


Imagen 3.3: Ejemplo de resultado paralelo (MPI -np 4)

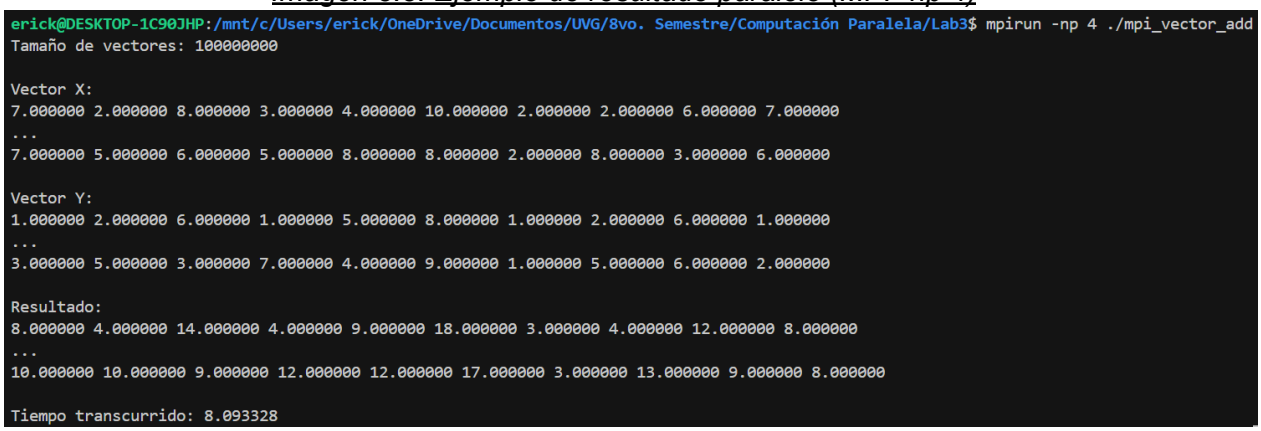


Tabla 3.1: Resultados obtenidos

Corrida	Secuencial	MPI (-np 2)	MPI (-np 4)
1	3.852322	4.474826	5.042033
2	4.076966	5.530646	7.640176
3	4.109191	6.155507	8.071075
4	4.828904	6.055211	8.25799
5	4.618198	6.053322	8.093328
6	4.467263	6.619779	7.077383
7	4.714195	5.95598	8.208145
8	4.803722	6.449832	8.164701
9	4.794683	5.915637	9.070929
10	4.898485	6.249165	8.109498
Promedios	**4.516393**	**5.9459905**	**7.7735258**
Speedup		**0.759569478**	**0.5809967081**

4. 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.

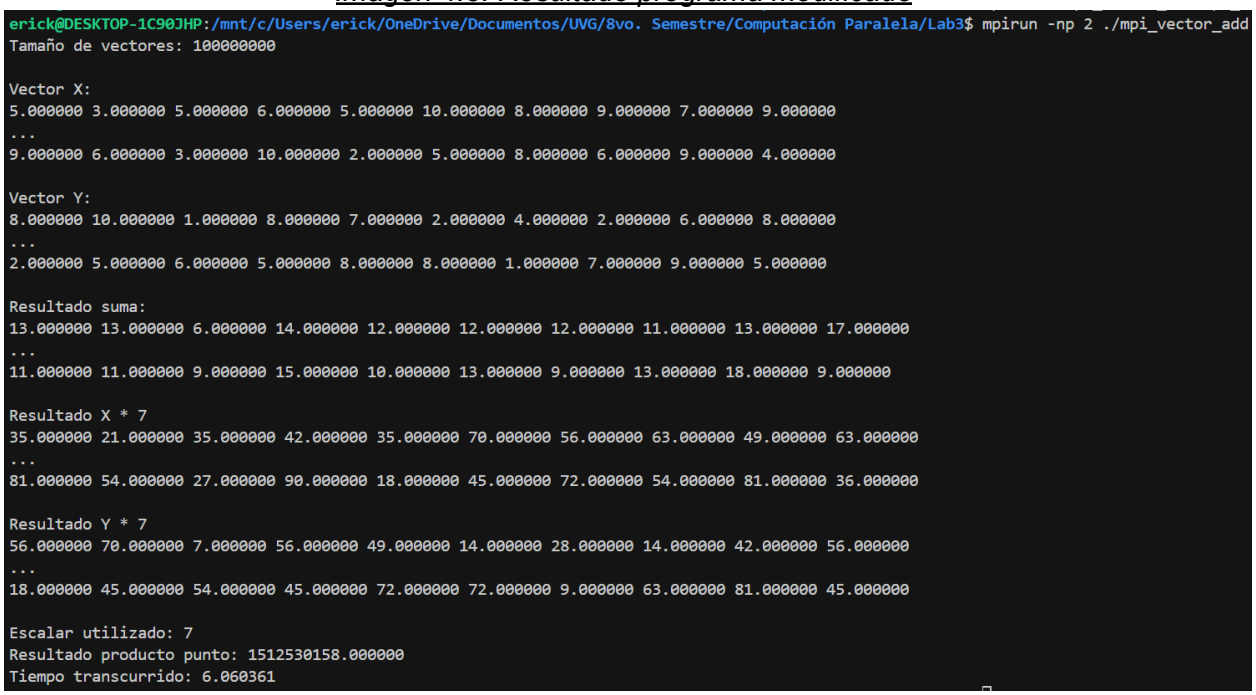
Imagen 4.1: Función `Parallel_vector_dot()`

```
void Parallel_vector_dot(
    double local_x[] /* in  */,
    double local_y[] /* in  */,
    double *dotP /* out */,
    int local_n /* in  */)
{
      int local_i;
      *dotP = 0.0;

      for (local_i = 0; local_i < local_n; local_i++)
            *dotP += local_x[local_i] * local_y[local_i];
} /* Parallel_vector_dot */
```

Imagen 4.2: Función `Parallel_vector_scalar()`

```
void Parallel_vector_scalar(
    double local_x[] /* in  */,
    double local_y[] /* in  */,
    double local_a[] /* out */,
    double local_b[] /* out */,
    int local_n /* in  */,
    int scalar)
{
      int local_i;

      for (local_i = 0; local_i < local_n; local_i++)
      {
            local_a[local_i] = scalar * local_x[local_i];
            local_b[local_i] = scalar * local_y[local_i];
      }
} /* Parallel_vector_scalar */
```

Imagen 4.3: Resultado programa modificado



5. 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.
   R: Durante el laboratorio se implementaron diferentes técnicas de comunicación grupal utilizando MPI, en donde cada una mantiene un enfoque y propósito distinto según su uso; por ejemplo, se utilizó `MPI_Bcast` para el envío de un mismo mensaje hacia todos los procesos disponibles, asi´como `MPI_Scatter` y `MPI_Gather` para el envío de un mensaje dividido equitativamente y para la recepción de los resultados parciales de cada proceso, respectivamente. El propósito principal del ejercicio fue realizar distintas operaciones vectoriales, tales como la suma, cálculo de producto punto y cálculo de producto vector-escalar, lo que permitió un mejor entendimiento de cómo se maneja la sincronización entre procesos la distribución de tareas en sistemas paralelos con MPI.

   De lo que más destaco de este laboratorio, podría mencionar que, a pesar de la paralelización, no siempre se logra una mejora en el tiempo de ejecución del programa, como se puede observar en la tabla 3.1, en donde se obtuvo que el speedup realmente no refleja una mejora significativa, sino más bien una caída en el rendimiento para ambos casos: utilizando 2 y 4 procesos. Adicionalmente, destaco que este laboratorio me ayudó a repasar la importancia de medir correctamente los tiempos y analizar el impacto de diferentes configuraciones. Creo que se podría mejorar el programa optimizando la forma en que se dividen las tareas entre procesos, especialmente al trabajar con vectores de gran tamaño, como el utilizado en este laboratorio para cumplir el requerimiento de que los programas tardaran aproximadamente 5 segundos en ejecutarse.