Universidad del Valle de Guatemala
Computación Paralela y Distribuida
Ciclo II, 2024
Fecha de entrega: 27 oct 2024 23:59

Andrea Ximena Ramírez Recinos	21874

Laboratorio 4

Riemann Using OpenMPI

Tal y como usted lo propuso durante el corto # 3, utilice su análisis y diseño sobre la posible implementación de un programa que calcula la aproximación de una integral mediante el uso de las sumas de Riemann y Open MPI. Deje constancia de cuál fue su propuesta inicial y trate de ejecutarla/implementarla. En dado caso su análisis tuviera fallas, deje constancia y detalle en un nuevo análisis cuáles son los cambios que deberían ser necesarios para que su implementación funcione correctamente. Luego de que su nuevo análisis sea corregido, termine la implementación de su programa.

Realice las iteraciones que considere necesarias en cuanto a análisis e implementación para tener una versión final y funcional de su código. Una vez tenga la versión final de su programa en Open MPI, realice una comparación y análisis del mismo programa en su versión secuencial y la versión paralela con OpenMP (estos 2 programas ya los deberían de tener hechos de laboratorios/hojas de trabajo anteriores). Mida speedups de sus programas y analice los resultados que obtenga a partir de la comparación de sus programas paralelos y la versión secuencial del algoritmo. Adjunte evidencia de las ejecuciones de sus programas y las pruebas que haya realizado para el cálculo de speedups.

[Parte I] Propuesta Original e Implementación

La propuesta inicial, presentada en la tercera evaluación corta del curso, se centra en implementar un modelo de comunicación punto a punto mediante el uso de las directivas de MPI: ***MPI_Send y MPI_Recv.*** En la evaluación se presentaron una serie de pasos bastante concisos y directos para describir cómo se podría implementar este tipo de comunicación.

En primer lugar, tras la inicialización de MPI, se obtienen tanto el identificador de cada proceso (rank) como el tamaño total del clúster. A continuación, se realiza un broadcast (con ***MPI_Bcast***) que envía a todos los procesos los parámetros de entrada, a y b, los cuales definen el intervalo donde inicia y termina el cálculo de las sumas de Riemann. Este paso es bastante importante, pues es una manera de garantizar que todos los procesos cuenten con la información necesaria para ejecutar sus tareas de forma sincronizada y correcta.

A continuación, el problema se divide entre el número total de procesos, asignando a cada uno un rango específico sobre el cual realizar su trabajo. Cada proceso, dentro de su rango asignado, calcula una suma local de Riemann, obteniendo una suma parcial de la integral definida en ese intervalo.

Una vez completada la suma local, cada proceso envía su resultado parcial al proceso raíz utilizando MPI_Send. El proceso raíz, a su vez, recibe y recopila todas las sumas parciales utilizando MPI_Recv. Al finalizar, el proceso raíz calcula la suma total de las sumas parciales, obteniendo así el resultado final de la integral, que se muestra en pantalla.

En la evaluación se incluyó un diagrama de flujo, aunque se cree que este no es tan fácil de interpretar, por lo que se decidió elaborarlo nuevamente para presentarlo de manera formal y ordenada. Rehacer el diagrama tiene como objetivo mejorar la comprensión del flujo de comunicación entre procesos y facilitar la visualización de todo el proceso al lector.

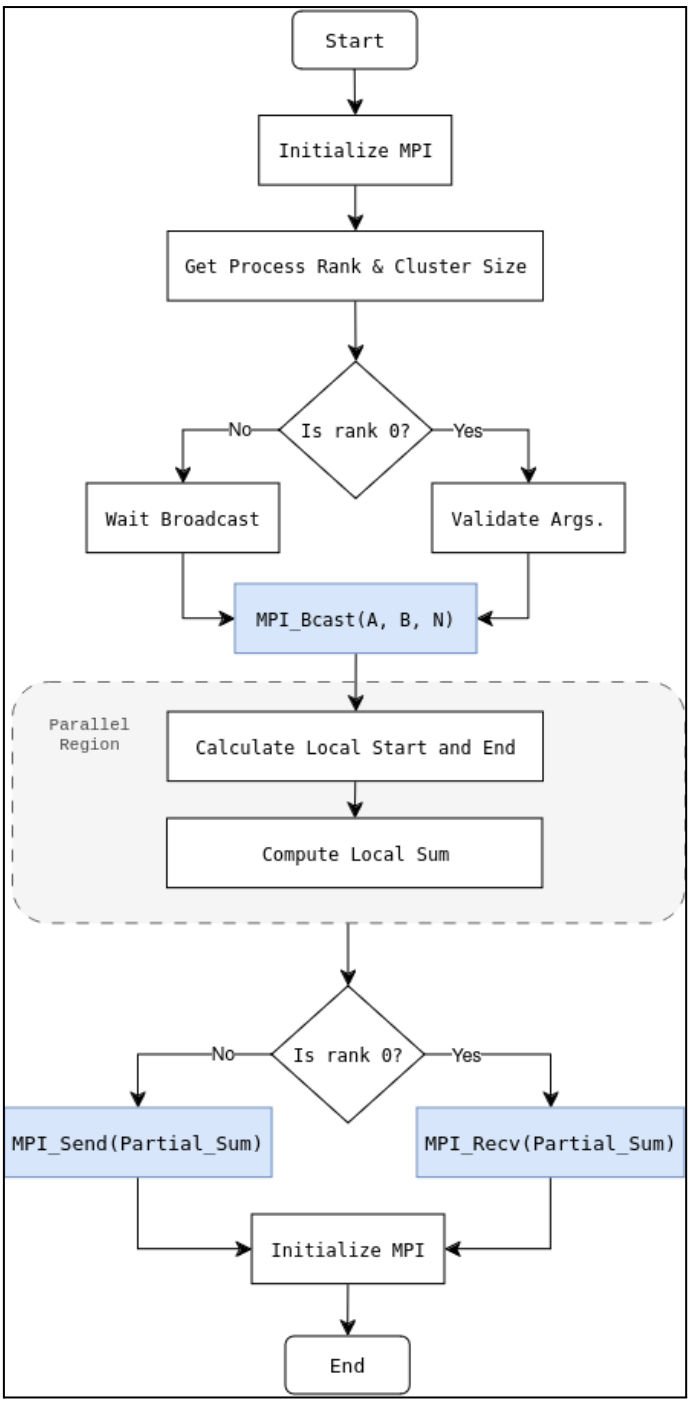

Figura 1 - Diagrama de Flujo de Propuesta Inicial.

```
ndrea@andrea:~/Documents/python-environment/Paralela/mpi-riemann$ ./riemann 2 1

ith n = 100000000, the approximation of the integral from point a = 2.00 to poi
t b = 10.00 is 330.666667
xecution time: 0.345569 seconds
ndrea@andrea:~/Documents/python-environment/Paralela/mpi-riemann$ ./riemann_omp
2 10 4
ith n = 100000000, the approximation of the integral from point a = 2.00 to poi
t b = 10.00 is 330.666667
xecution time: 0.105964 seconds
ndrea@andrea:~/Documents/python-environment/Paralela/mpi-riemann$ mpirun -np 4
/riemann_mpi 2 10
ith n = 100000000, the approximation of the integral from a = 2.00 to b = 10.00
is 330.666667
xecution time: 0.123885 seconds
```

Figura 2 - Prueba Realizada con Propuesta Inicial (N = 100,000,000)

La implementación de la comunicación punto a punto, como se presentó anteriormente, es un enfoque correcto y funcional. Sin embargo, aunque el método es válido y bien estructurado, presenta un tiempo de ejecución moderadamente mayor en comparación con la versión paralela que utiliza OpenMP. Aún así, se evidencia una mejora significativa en contraste con la versión secuencial. Se cree que este incremento sutil en el tiempo de ejecución, contra OpenMP, se debe principalmente a dos factores: el overhead generado por la constante comunicación entre procesos y la latencia que implica el intercambio continuo de datos, en contraste con el modelo de hilos utilizado en OpenMP.

La comunicación punto a punto en MPI, a través de MPI_Send y MPI_Recv, introduce una latencia adicional, ya que cada proceso debe esperar la recepción y envío de mensajes, lo cual se traduce en un mayor consumo de tiempo en cada paso. Además, este enfoque puede aumentar el consumo de recursos en el sistema, debido a la creación y gestión de múltiples procesos en lugar de hilos, que suelen ser menos costosos en términos de recursos y tiempos.

En base a lo antes mencionado, no se cree necesario realizar una iteración adicional del programa, pues este cumple con los objetivos.

[Parte II] Análisis Comparativo y Obtención de Speed Up

N	Tiempo Secuencial (s)	Tiempo OMP (s)	Tiempo MPI (s)
10,000	0.000060	0.000270	0.000233
	0.000375	0.000271	0.000246
	0.000388	0.000275	0.000355
	0.000388	0.000280	0.000384
	0.000415	0.000288	0.000484
100,000	0.000375	0.000110	0.000077
	0.000039	0.000110	0.000085
	0.000040	0.000117	0.000095
	0.000041	0.000145	0.000106
	0.000041	0.000182	0.000231
100,000,000	0.385067	0.148215	0.151248
	0.431924	0.162436	0.160738
	0.433437	0.168049	0.165900
	0.434643	0.171325	0.171235
	0.568279	0.183506	0.190334
Tiempo Promedio	0.150367	0.055705	0.056117

Cuadro 1. Registro de Tiempo de Ejecución de Programas (F(x) = (x)^2).

Figura 3 - Prueba de Ejecuciones de Programas (F(x) = (x)^2).

N	Tiempo Secuencial (s)	Tiempo OMP (s)	Tiempo MPI (s)
10,000	0.000046	0.000085	0.000115
	0.000047	0.000143	0.000116
	0.000047	0.000181	0.000124
	0.000047	0.000191	0.000129
	0.000047	0.000210	0.000149
100,000	0.000443	0.000315	0.000291
	0.000458	0.000337	0.000295
	0.000480	0.000375	0.000299
	0.000560	0.000413	0.000309
	0.000561	0.000452	0.000314
100,000,000	0.492699	0.181364	0.203854
	0.537510	0.198473	0.207635
	0.605846	0.207050	0.208237
	0.622335	0.214101	0.208650
	0.668724	0.253967	0.220648
Tiempo Promedio	0.195323	0.070510	0.070078

Cuadro 2. Registro de Tiempo de Ejecución de Programas (G(x) = 2(x)^3).

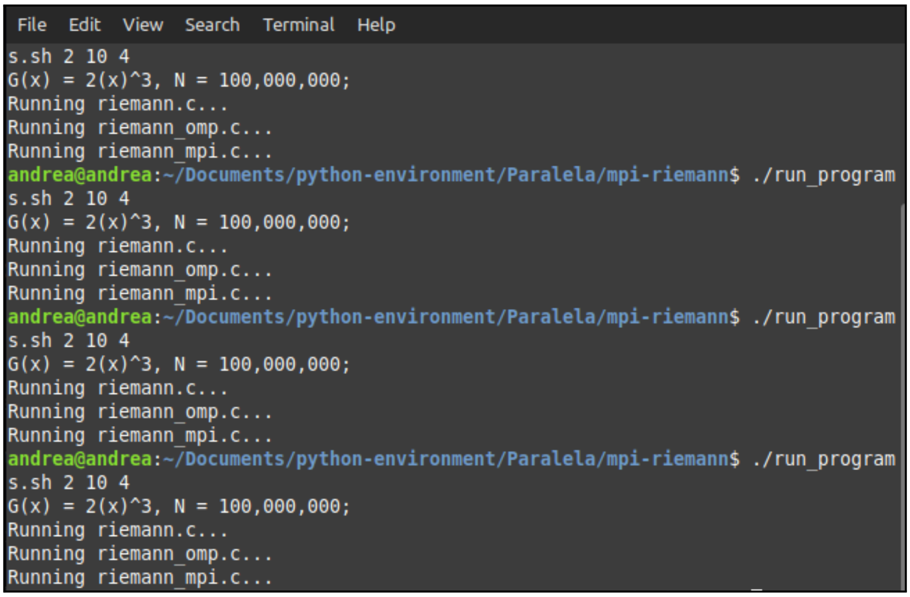

Figura 4 - Prueba de Ejecuciones de Programas (G(x) = 2(x)^3).

N	Tiempo Secuencial (s)	Tiempo OMP (s)	Tiempo MPI (s)
	0.000239	0.000159	0.000202
	0.000231	0.000210	0.000211
10,000	0.000235	0.000213	0.000218
	0.000235	0.000254	0.000220
	0.000243	0.000310	0.000251
	0.002155	0.001176	0.000905
	0.002181	0.001267	0.000914
100,000	0.002430	0.001306	0.000936
	0.002443	0.001337	0.000948
	0.002593	0.001451	0.001037
	2.391698	0.909338	0.894155
	2.445535	0.974627	0.910001
100,000,000	2.500654	1.005550	0.922544
	2.517777	1.048984	0.950431
	2.744485	1.102172	1.325808
Tiempo Promedio	0.840876	0.336557	0.333919

Cuadro 4. Registro de Tiempo de Ejecución de Programas (H(x) = sin(x)).

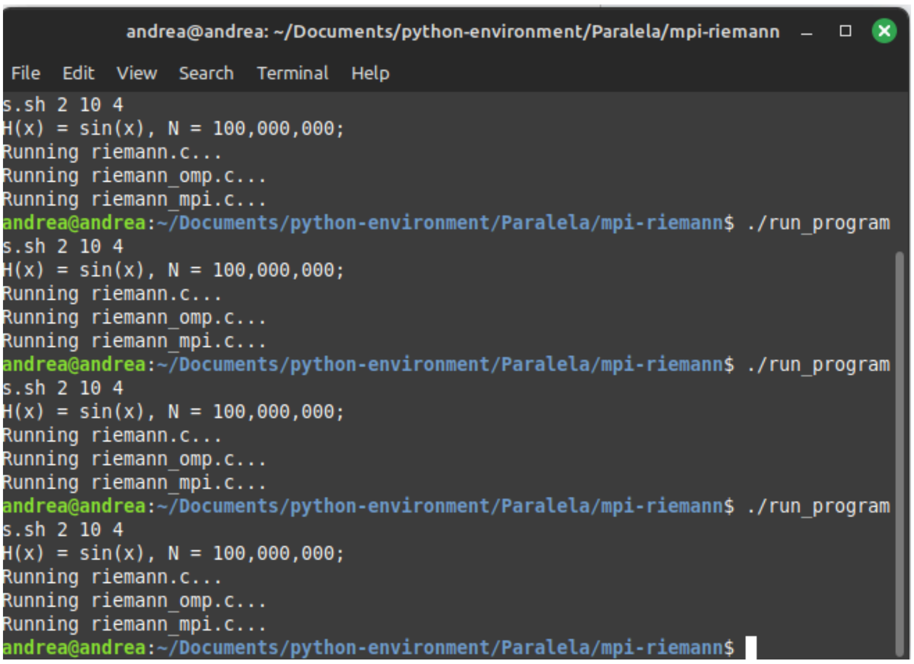

Figura 5 - Prueba de Ejecuciones de Programas (H(x) = sin(x)).

	Programa Secuencial	Paralelo OMP	Paralelo MPI
TIempo por Función (s)	0.150367	0.055705	0.056117
	0.195323	0.070510	0.070078
	0.840876	0.336557	0.333919
TIempo Promedio (s)	0.395522	0.154258	0.153371
Speed Up	N/A	2.564037	2.578858

Cuadro 5. Cálculo de Speed Up en Base a Tiempo Promedio de Programas.

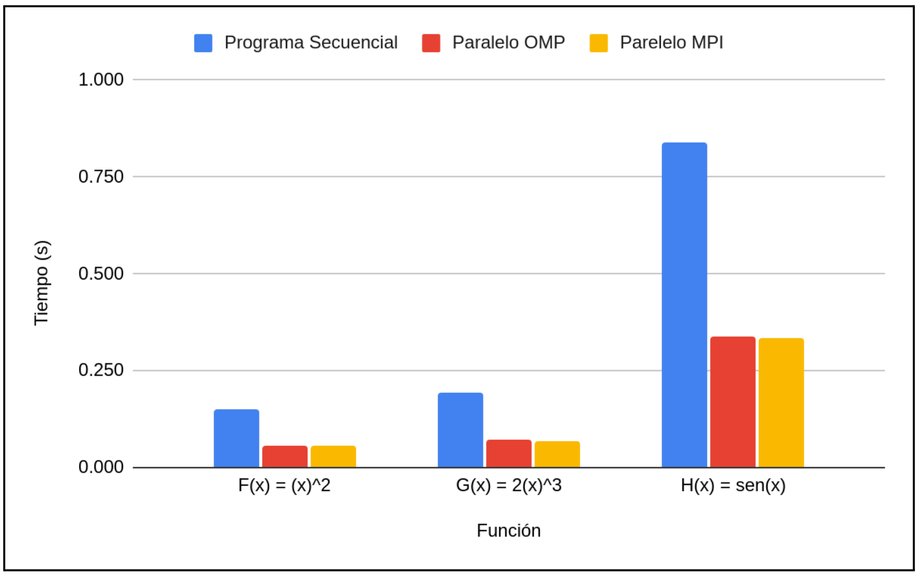

Figura 5 - Gráfico de Tiempo Promedio de Programa Contra Función.

Como se puede observar, ambas versiones paralelas logran mejorar el tiempo de ejecución en un factor de 2 en comparación con la versión secuencial. Sin embargo, la implementación paralela basada en MPI supera a la de OMP por un margen de 0.01 en términos de speed up.