REVISTA DE LA ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN
UNIVERSIDAD NACIONAL DE SAN AGUSTIN DE AREQUIPA
VOL. 1, NO. 1, NOVIEMBRE 2024

Laboratorio del curso de Computación Paralela y Distribuida: MPI - Programming assingments

Eduardo G. Ruiz[1]

[1] *Escuela de Ciencias de la Computación, Facultad de Ingeniería de Procesos, Universidad Nacional de San Agustín, Arequipa, Arequipa, Perú*

Resumen—En el siguiente laboratorio se resolvieron ejercicios propuestos del libro aplicando lo aprendido en el capítulo 3 a través de ejercicios de implementación, pero más que todo entendiendo la base teórica a través de la realización de cuadros, diagramas y clasificaciones.

Palabras clave—mpi, envio, recibimiento, distribución de datos

Abstract—In the following laboratory, proposed exercises from the book were solved by applying what was learned in chapter 3 through implementation exercises, but above all by understanding the theoretical basis through the creation of tables, diagrams and classifications.

Keywords—mpi, sending, receiving, data distribution

OBJETIVO

El objetivo de este laboratorio es implementar y comprender (a través de los ejercicios propuestos en el libro) el uso de funciones de MPI en un entorno de programación paralela.

Los diferentes ejercicios incluyen: Envío y recepción de datos entre procesos, cálculo de integrales mediante trapezoides, distribución de datos entre procesos y a la vez recojo de datos entre ellos.

Este laboratorio nos permite familiarizarnos con conceptos clave en la programación paralela en la biblioteca MPI a la vez que entendemos los conceptos explicados en el capítulo 3 del libro de Peter Pacheco. La implementación y ejecución de los ejercicios se encuentra en `https://github.com/EGRM23/CPD-2024/tree/main/MPI-programming`

EJERCICIOS

Los ejercicios propuestos para el siguiente laboratorio son 5 y todos requieren una explicación teórica de lo que se hizo para completarlo, sin embargo, son 2 los que requieren implementación (3.1 y 3.2), las explicaciones se apoyan en base a lo que se vio en el capítulo 3, a continuación detallaré que requiere cada ejercicio.

En el ejercicio 3.1 las preguntas son ¿Qué sucede en el programa "greetings"si, en lugar de strlen(greeting) + 1, usamos strlen(greeting) para la longitud del mensaje que envían los procesos 1, 2, . . . , comm sz1? ¿Qué sucede si usamos MAX STRING en lugar de strlen(greeting) + 1? ¿Puedes explicar estos resultados?. Este ejercicio requiere implementación debido a que es una manera de ver que pasa si se hace esos cambios al programa, y junto con la explicación se resuelve el problema.

El ejercicio 3.2 dice modifica la regla trapezoidal para que estime correctamente la integral incluso si comm_sz no divide a n de manera uniforme. (Aún se puede suponer que n comm_sz.) Este ejercicio obviamente requiere implementación y explicación de lo que se hizo.

EL ejercicio 3.3 pide determinar cuáles de las variables en el programa de la regla trapezoidal son locales y cuáles son globales. Este ejercicio no requiere implementación, pues es un análisis en base al código del ejercicio 3.2.

El ejercicio 3.6 dice supón que comm_sz = 4 y que x es un vector con n = 14 componentes ¿Cómo se distribuirían los componentes de x entre los procesos en un programa que usara una distribución en bloques? ¿Cómo se distribuirían los componentes de x entre los procesos de un programa que utilizara una distribución cíclica? ¿Cómo se distribuirían los componentes de x entre los procesos de un programa que utilizara una distribución cíclica por bloques con un tamaño de bloque b = 2? Esto se puede resolver en base a la tabla 3.4 del libro, y explicar posteriormente porque se hizo así, al ser una explicación teórica no requiere implementación.

Por último, el ejercicio 3.8 da 2 indicaciones en base a una consigna, suponga que comm_sz = 8 y n = 16, entonces: Dibuje un diagrama que muestre cómo se puede implementar MPI Scatter utilizando una comunicación estructurada en árbol con procesos comm_sz cuando el proceso 0 necesita distribuir una matriz que contiene n elementos, dibuje un diagrama que muestre cómo se puede implementar MPI Gather utilizando comunicación estructurada en árbol cuando una

matriz de n elementos que se ha distribuido entre procesos comm_sz necesita ser reunida en el proceso 0. Estos diagramas serán dibujados y explicados para la realización del ejercicio.

METODOLOGÍA

En el caso del ejercicio 3.1, se cambió el parámetro que indicaba del tamaño del mensaje en la función MPI_Send y se probó los resultados, adicional, aunque el problema no lo pidiera, también se hizo pruebas cambiando el mismo parámetro en la función MPI_Recv, por los mismos valores que indicaba el problema, los resultados se muestran en la siguiente sección.

En el ejercicio 3.2, sea añadió la variable remainder para poder controlar el número de trapecios sobrantes en caso de que no sea exacto al número de procesos, a la vez se cambio la asignación de local_a en base a esta nueva variable, ahora si el identificador del proceso es menor a remainder este obtiene un trapecio adicional que evaluar (esto se logra aumentando su local_n en 1), por otro lado, como los trapecios restantes se reparten entre los primeros procesos, la asignación del límite local_a cambia un poco para los procesos que no recibieron una parte del sobrante, tomando en cuenta que esta parte ya ha sido repartida y por lo tanto su límite inferior es ahora más alto.

Para el ejercicio 3.3 se analizó el programa ejecutado en 3.2 y se identificó cuáles eran las variables locales y globales, de acuerdo a cómo están declaradas y cómo se usan en los diferentes procesos.

Para la resolución del ejercicio 3.6 se usó como base la tabla 3.4 del libro, y se siguió el mismo esquema para dividir los componentes de acuerdo las especificaciones del problema.

RESULTADOS

Se comienza mostrando los resultados de la ejecución del ejercicio 3.1 y 3.2.

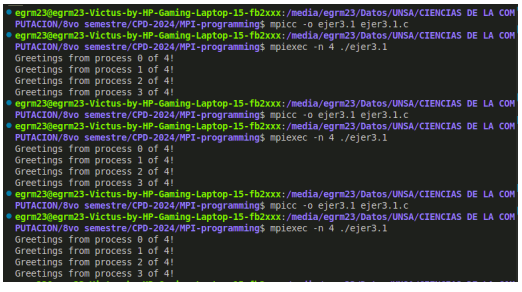

Fig. 1: Ejecución del ejercicio 3.1, según lo que pide el ejercicio

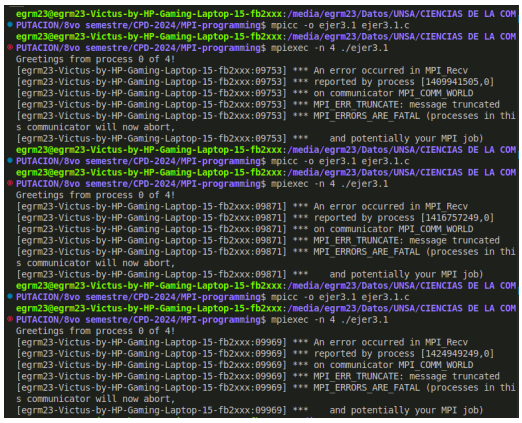

Fig. 2: Ejecución del ejercicio 3.1, añadiendo algunos cambios

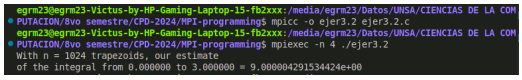

Fig. 3: Ejecución del ejercicio 3.2

En el caso del ejercicio 3.3, en sí dentro del programa no hay variables globales que todos usan del mismo punto, sin embargo, hay variables que cada uno tiene su propia copia y tienen el mismo valor en todos los procesos, como es el caso de comm_sz, n, a, b y h. Fuera de eso hay variables particulares que cada proceso tiene su propia interpretación, como es el caso de my_rank, local_n, local_a, local_b, local_int y total_int. Vale reaclarar que dentro de este programa no existen variables globales en el sentido de memoria compartida entre procesos en MPI.

Para el ejercicio 3.6, como se explicó se realizó una tabla indicando las distribución y el resultado es la figura 4.

Process	Components		
	Block	Cyclic	Block-Cyclic BlockSize = 2
0	0 1 2 3	0 4 8 12	0-1 8-9
1	4 5 6 7	1 5 9 13	2-3 10-11
2	8 9 10	2 6 10	4-5 12-13
3	11 12 13	3 7 11	6-7

Fig. 4: Resolución del ejercicio 3.6

En el ejercicio 3.8 los 2 diagramas se pueden representar de manera fácil con carácteres, en el caso del diagrama que representa la distribución de Scatter sería como el de la figura 5.

Fig. 5: Diagrama implementando MPI_Scatter del ejercicio 3.8

Y en el caso del diagrama que representa la distribución de Gather sería como el de la figura 6.

Fig. 6: Diagrama implementando MPI_Gather del ejercicio 3.8

ANÁLISIS

En el ejercicio 3.1 se puede observar que no hay diferencia al cambiar el parámetro de tamaño de envío en la función MPI_Send, no hay diferencia cuando se da un valor mayor, sin embargo, cuando es un valor menor es inseguro, aunque se vio que funcionó, el dar strlen(greeting) en ves de strlen(greeting) + 1, está calculando la longitud de la cadena sin incluir el carácter nulo, y esto a veces puede causar errores de interpretación en el lado receptor, pero como el lado receptor está esperando un mensaje de tamaño MAX_STRING es lo suficientemente grande para recibir el mensaje del emisor, caso contrario a la figura 2 donde si se varía el tamaño del mensaje de recibimiento en el función MPI_Recv, ahí si existen problemas porque ya no es compatible y crashea como se ve en el ejemplo.

En el ejercicio 3.2 ahora se puede tener una mayor precición de la eficiencia brindada por la integral, pues mientras n podía tomar valores muy grandes habían casos donde varios de esto trapecios se perdían, con la variable reminder se añade estos restantes a los procesos, como no puede pasar de n, máximo se añadirá un trapecio adicional a algunos procesos lo cuál no afectará la carga de trabajo ni en el equilibrio entre lo que hace cada 1.

En el ejercicio 3.3 como se explicó en los resultados, no hay variables gloables en el sentido de memoria compartida, sin embargo, si hay variables que cada uno tiene su propia copia y cada uno usa el mismo valor dentro de todos los procesos.

En la tabla del ejercicio 3.6 se pedía dividir 14 componentes entre 4 procesos usando diferentes técnicas, en el primer aparatado usando la repartición por bloques, sabemos que no se puede dividir equitativamente 14 elementos entre 4, por lo que, tratando de mantener al mínimo la diferencia entre los datos de cada proceso, se dieron 4 datos al primer y segundo proceso cada uno y 3 al tercer y cuarto proceso cada 1, se hizo por orden, del 0-3 al proceso 0, del 4-7 al proceso 1, del 8-10 al proceso 2 y del 11-13 al proceso 3. En el caso de la repartición cíclica el número de bloques por proceso fue igual que el primer ejercicio, sin embargo, es diferente los bloques que reciben cada uno, al ser ciclico se va repartiendo 1 bloque a cada 1 de los procesos hasta terminar, una vez se llega al último proceso, si aún quedan bloques, se vuelve a empezar, por eso el bloque 0 está en el primer proceso y el bloque 1 en el segundo proceso. Por último, la repartición cíclica por bloques de tamaño 2 varía bastante comparada a las otras distribuciones, se mantiene la entrega cíclica de la segunda distribución que va dando uno por uno hasta terminar, sin embargo, al ser bloques de tamaño 2, primero entrega el bloque 0 y 1 al proceso 0, luego el bloque 2 y 3 al proceso 1 y así sucesivamente, esta entrega doble hace que al final la distribución de bloques por proceso sea distinta a los otros casos, pues aunque el proceso 0 y el proceso 1 si temrinen con 4 datos, el proceso 2 también lo hará, a diferencia de las anteriores distribuciones donde terminaba con 3 datos, además consecuentemente el proceso 3 terminará solo con 2 datos, todo esto debido a que en la última entrega al no poder dividir la entrega en un número inferior a 2, le entrega los 2 datos finales al proceso 2 y el proceso 3 se queda sin nada más.

En los diagramas del ejercicio 3.8 se puede ver como se hace la distribución y el recibimiento de los datos en el caso de Scatter y Gather respectivamente, empezando con MPI_Scatter y apoyado con lo que dice el libro, una solución fácil hubiera sido hacer que el proceso 0 envíe los datos a todos los demás procesos, pero eso no sería equitativo y gastaría muchas iteraciones, por lo tanto la función Scatter lo hace como en el diagrama, siendo n = 16 y teniendo 8 procesos, cada uno de los procesos debe tener 2 datos, al comienzo todos los datos están en el proceso 0.

En la primera iteración el proceso 0 retiene sus 2 datos propios y envía 8 datos al proceso 1 y los 6 restantes al proceso 2. En la segunda iteración, el proceso 1 retiene sus 2 datos propios y envía 4 datos al proceso 3 y 2 datos al proceso 4, mientras que a su vez el proceso 2 retiene sus 2 datos propios y envía 2 datos al proceso 5 y 2 datos al proceso 6. En la última iteración, la mayoría tienen sus datos propios, aquí el proceso 3 retiene 2 datos y envía los 2 restantes al proceso 7, terminando la distribución.

En el caso de Gather es al revés, solo que ahora en envío empezando por el proceso 7 que devuelve los datos al proceso 3, este lo junto con lo que tiene y sigue el envío sucesivamente hasta llegar al proceso 0.

REFERENCIAS