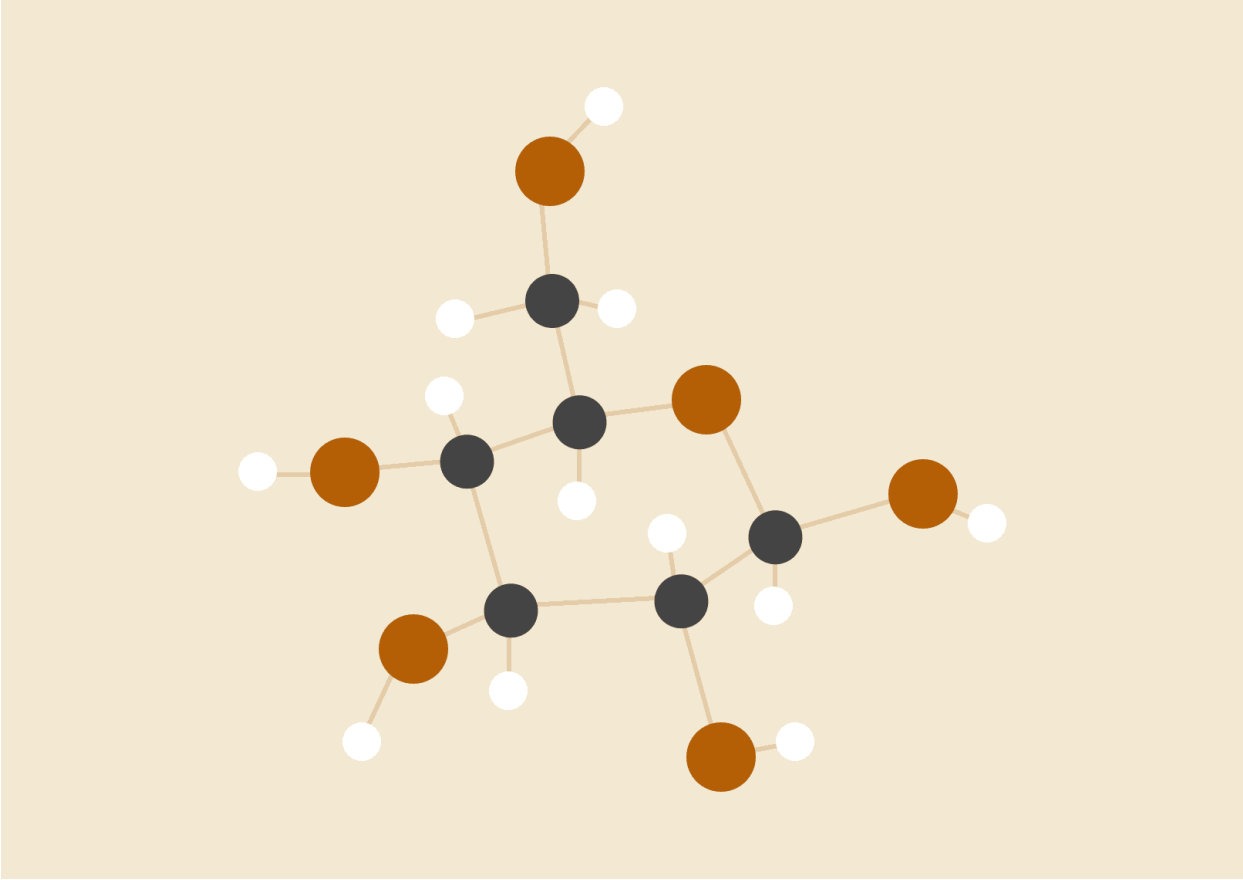
Entrega 3

Programación en pasaje de mensajes / Programación híbrida



Grupo 13

Del Grosso, Augusto

Sánchez Magariños, Juan Pablo

11.06.2021

UNLP

TECNOLOGÍAS UTILIZADAS

```
# Para compilar el código
    - gcc version 10.2.0 (Ubuntu 10.2.0-13ubuntu1)
    - NPTL 2.32
# Para editar y compartir el código
    - Visual Studio Code 1.55.2
    - git 2.27.0
# Cluster:
    - Ocho (8) nodos de procesamiento, cada uno con:
        ■ 8GB de memoria RAM
        ■ Dos procesadores Intel Xeon E5405 a 2.0GHz de 4 cores
    - 1Gbit Ethernet
```

EJERCICIOS PRÁCTICA

2. Los códigos blocking.c y non-blocking.c siguen el patrón master-worker, donde los procesos worker le envían un mensaje de texto al master empleando operaciones de comunicación bloqueantes y no bloqueantes, respectivamente.

> **Compile y ejecute ambos códigos usando P={4,8,16} (no importa que el número de núcleos sea menor que la cantidad de procesos). ¿Cuál de los dos retorna antes el control?**
>
> El no bloqueante devuelve el control inmediatamente, aunque se queda esperando con la instrucción MPI_Wait.
>
> **En el caso de la versión no bloqueante, ¿qué sucede si se elimina la operación MPI_Wait() (línea 52)? ¿Se imprimen correctamente los mensajes enviados?**
>
> **¿Por qué?**
>
> No, ya que al no haber recibido el mensaje, cuando intenta imprimir la variable **`mensaje`** tiene el valor de inicialización "No debería estar leyendo esta frase."

3. Los códigos blocking-ring.c y non-blocking-ring.c comunican a los procesos en forma de anillo empleando operaciones bloqueantes y no bloqueantes, respectivamente. Compile y ejecute ambos códigos empleando P={4,8,16} (no importa que el número de núcleos sea menor que la cantidad de procesos) y N={10000000, 20000000, 40000000, ...}. ¿Cuál de los dos algoritmos requiere menos tiempo de comunicación? ¿Por qué?

El no bloqueante requiere menos tiempo de comunicación. El "recorrido" del anillo es "secuencial" en el bloqueante. El último proceso destraba el 0, que destraba el 1, y así hasta completar todos. Eso lleva tiempo porque se hacen de a uno los pasajes de mensajes. En el no bloqueante todos los pasajes se hacen simultáneamente.

	P	N		
		10000000	20000000	40000000
Blocking	2	0.120265	0.239247	0.473480
	4	0.247791	0.494161	0.981727
	8	0.549699	1.095003	2.175230
	16	2.336014	4.653252	9.624439
Non-Blocking	2	0.114120	0.228095	0.453958
	4	0.219891	0.442885	0.886108
	8	0.299114	0.565364	1.186413
	16	0.937696	1.924614	3.854657

DESARROLLO

Algoritmo MPI

A la hora de desarrollar este algoritmo lo primero que nos planteamos es la forma en la que se iba a comportar el algoritmo. Tuvimos presente que este algoritmo el código se ejecutará en procesos que va a tener su propia memoria. También que la comunicación

de información entre ellos se realiza mediante pasaje de mensajes, lo cual cuesta tiempo y genera esperas de sincronización.

Teniendo esto en cuenta, optamos por utilizar comunicaciones colectivas como, MPI_Scatter, MPI_Gather, MPI_Allreduce, MPI_Bcast.
Lo primero a realizar es repartir las matrices a cada proceso, para esto debemos identificar cuales matrices se requieren completas y cuales parciales.

Para las matrices que se requieren parciales utilizamos MPI_Scatter, el cual reparte la matriz en partes iguales según la cantidad de procesos. Por otro lado, para las matrices que se requieren completas utilizamos MPI_Bcast, el cual envía a todos los procesos los mismos datos.

Posterior a esto comenzamos a ejecutar el cálculo para R1 , R2 y sus promedios. Lo importante a destacar de este paso es la necesidad de que todos los procesos obtengan la suma de los valores de R1 y R2. Utilizamos MPI_Allreduce, el cual podemos indicarle que operación de reducción lleve a cabo y hacerle llegar el valor total a todos los procesos. Luego, se debe realizar la multiplicación de R1*A y R2*B, donde cada proceso se dedica a procesar su parte de cada multiplicación para posteriormente realizar su parte del cálculo de C. Finalizado el cálculo de C de cada proceso debemos reensamblar cada parte de C en una sola matriz, entonces con MPI_Gather, se envían todas las partes hacia el Coordinador, con el objetivo de realizar la comprobación del resultado final.

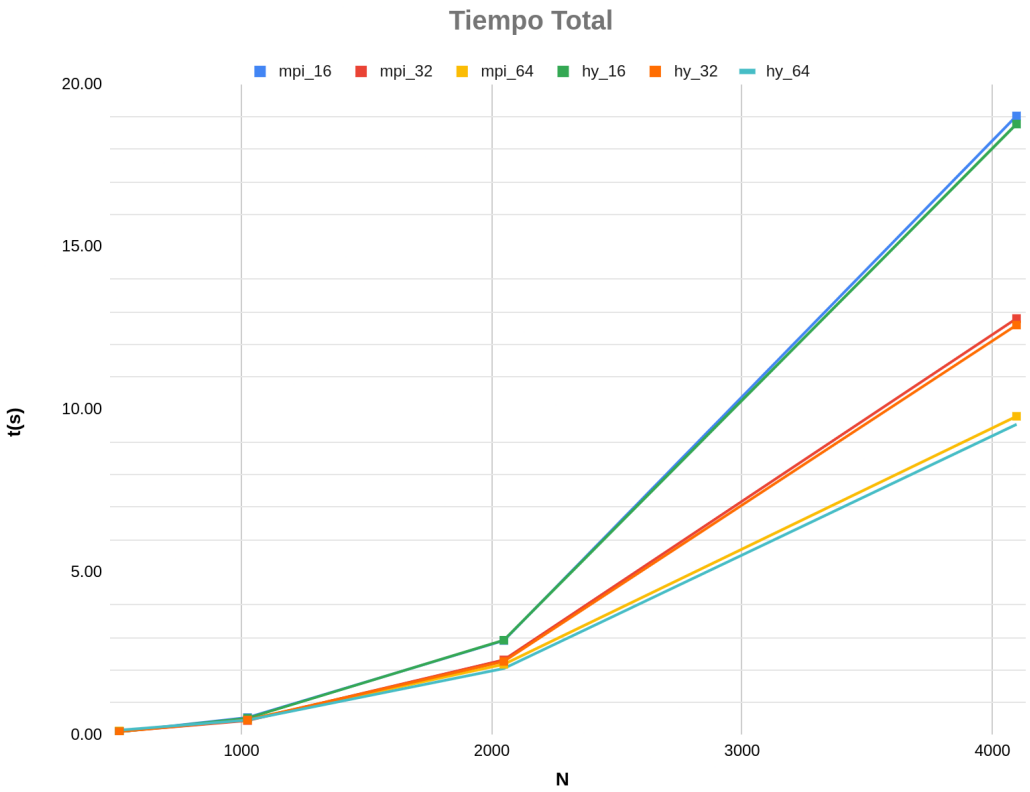
Algoritmo Hybrid (MPI + OpenMP)

Este algoritmo sigue la misma lógica que el anterior. Salvo que al incorporar OpenMP buscamos disminuir la comunicación inter-nodo utilizando comunicaciones intra-nodo (entre los hilos). Por lo tanto ahora dividimos la matriz entre los distintos procesos, y a su vez estos procesos dividen su porción de matriz entre sus hilos (8 por cada proceso). Cabe mencionar que en algunos casos se requiere una redimensión del tamaño de bloque, si bien esto podría afectar el aprovechamiento de los principios de localidad de la caché es realizado con el fin de maximizar uso de recursos.

Validación de los resultados

Para realizar la verificación simplemente se reinicializan las matrices calculadas, excepto C que se calculará sobre otra variable C_CHECK, y se realizan el procesamiento secuencialmente, habiendo detenido la medida del tiempo anteriormente. Luego se comparan las dos matrices resultado, elemento a elemento

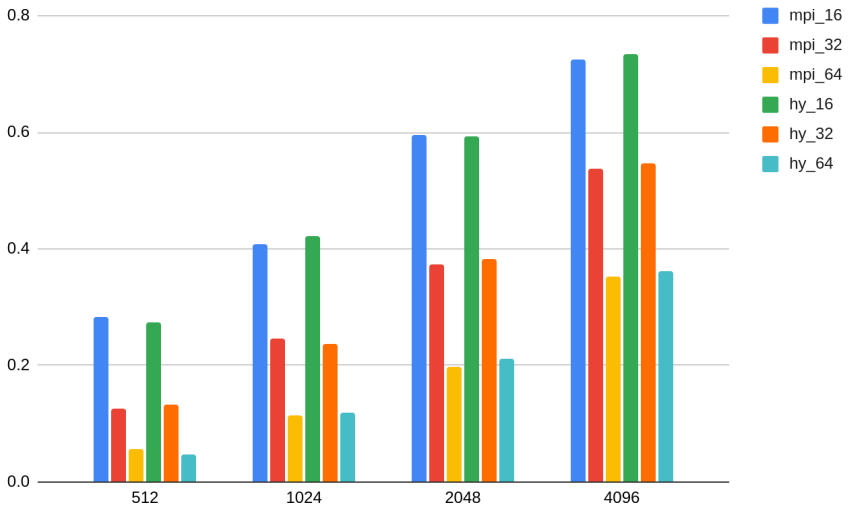
Comparación MPI e Híbrido



TIEMPO					
Algoritmo	P	N			
		512	1024	2048	4096
best sec		0.46	3.53	27.65	220.42
MPI	16	0.10	0.54	2.91	19.02
	32	0.11	0.45	2.31	12.80
	64	0.13	0.48	2.18	9.79
Hibrido	16	0.10	0.52	2.92	18.77
	32	0.11	0.47	2.26	12.60
	64	0.15	0.46	2.05	9.55

SPEEDUP					
Algoritmo	P	N			
		512	1024	2048	4096
MPI	8	5.30	6.65	7.18	7.33
	16	4.54	6.53	9.52	11.59
	32	4.02	7.88	11.96	17.22
	64	3.55	7.39	12.71	22.50
Hibrido	16	4.40	6.77	9.48	11.74
	32	4.27	7.56	12.21	17.50
	64	3.03	7.66	13.51	23.08

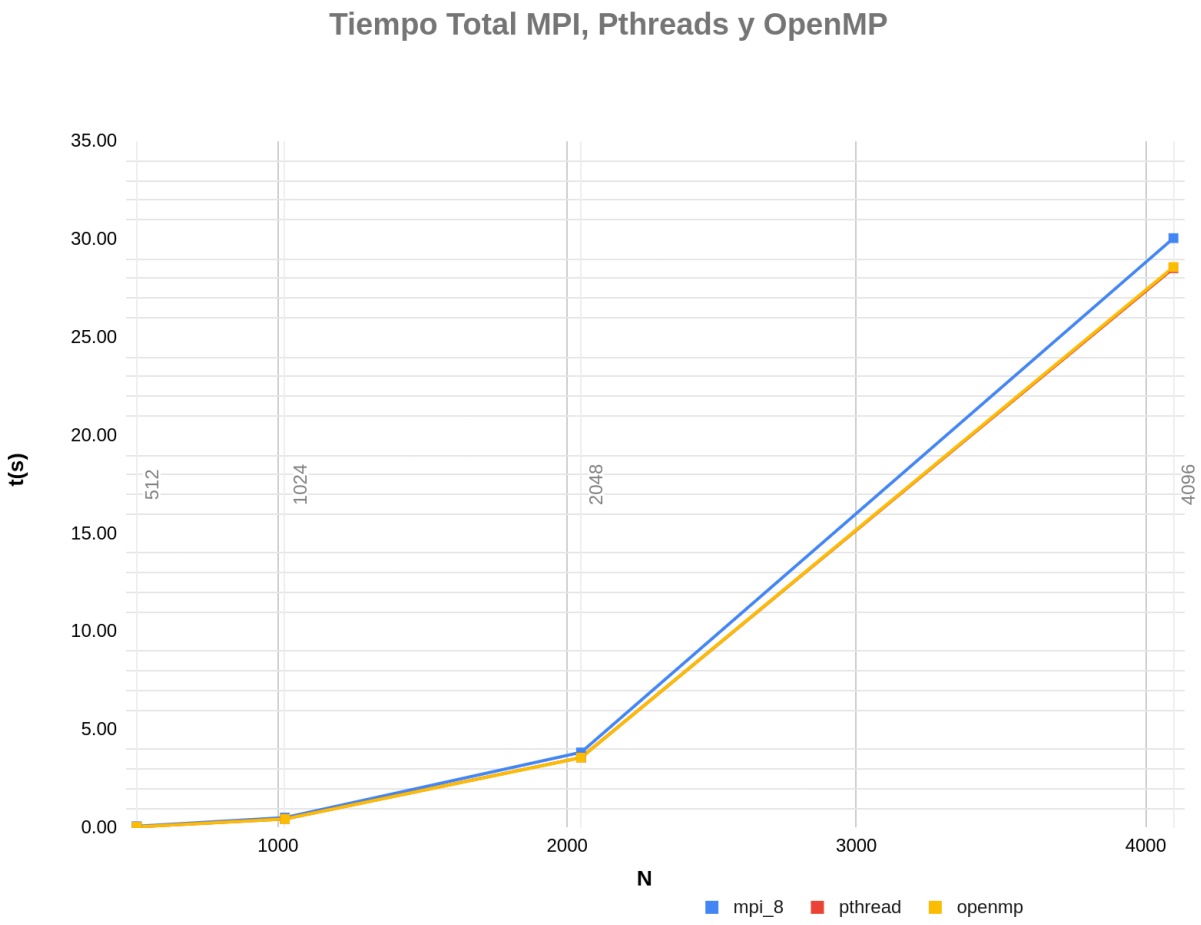
EFICIENCIA					
Algoritmo	P	N			
		512	1024	2048	4096
MPI	16	0.28	0.41	0.59	0.72
	32	0.13	0.25	0.37	0.54
	64	0.06	0.12	0.20	0.35
Hibrido	16	0.27	0.42	0.59	0.73
	32	0.13	0.24	0.38	0.55
	64	0.05	0.12	0.21	0.36



Tiempo Comunicación					
Algoritmo	P	N			
		512	1024	2048	4096
MPI	16	0.07	0.32	1.15	4.56
	32	0.10	0.32	1.43	5.60
	64	0.12	0.41	1.72	6.20
Hibrido	16	0.07	0.28	1.12	4.44
	32	0.09	0.34	1.36	5.44
	64	0.14	0.39	1.57	5.42

Overhead Comunicación					
Algoritmo	P	N			
		512	1024	2048	4096
MPI	16	70%	58%	39%	24%
	32	84%	72%	62%	44%
	64	92%	86%	79%	63%
Hibrido	16	68%	55%	38%	24%
	32	82%	74%	60%	43%
	64	92%	85%	77%	57%

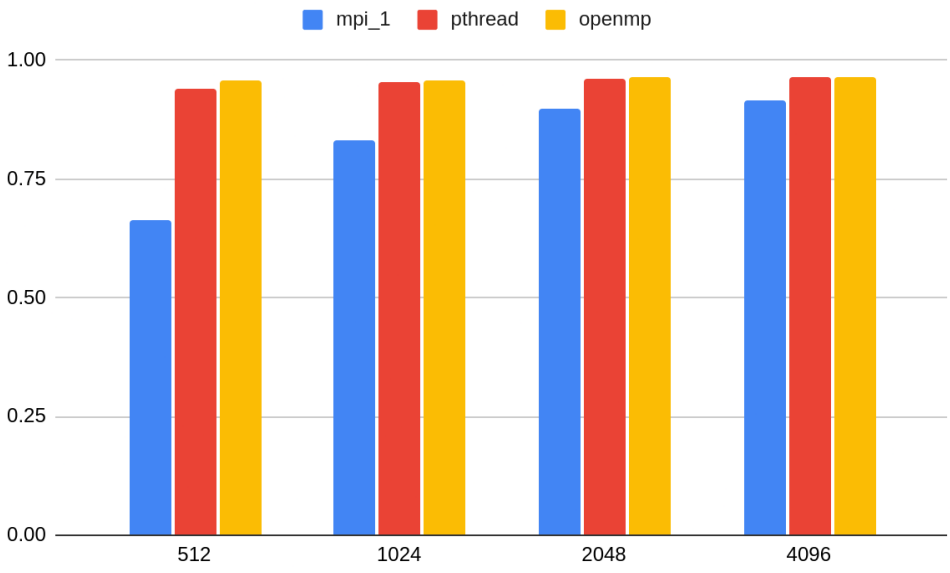
COMPARACIÓN MPI con Pthreads/OpenMP



Tiempo de Ejecución					
Algoritmo	P	N			
		512	1024	2048	4096
Best sec	1	0.46	3.53	27.65	220.42
MPI	8	0.09	0.53	3.85	30.06
Pthreads	8	0.06	0.46	3.59	28.53
OpenMP	8	0.06	0.46	3.58	28.59

SPEEDUP					
Algoritmo	P	N			
		512	1024	2048	4096
MPI	8	5.30	6.65	7.18	7.33
Pthreads	8	7.53	7.64	7.70	7.73
OpenMP	8	7.65	7.66	7.72	7.71

EFICIENCIA					
Algoritmo	P	N			
		512	1024	2048	4096
MPI	8	0.66	0.83	0.90	0.92
Pthreads	8	0.94	0.96	0.96	0.97
OpenMP	8	0.96	0.96	0.96	0.96



Overhead Comunicación					
Algoritmo	P	N			
		512	1024	2048	4096
MPI	8	32%	16%	9%	4%

CONCLUSIÓN

Dados los resultados que se pueden apreciar en las tablas, podemos concluir que el uso de hilos, en este algoritmo, representa una ventaja mínima. Lo más costoso es realizar la comunicación entre los procesos. Un ejemplo de esto es que en las matrices de 512 se emplea más del 90% del tiempo en comunicaciones en el punto más extremo.

Vemos que el rendimiento de los algoritmos se agrupa por la cantidad de nodos en los que corren. Cuantos más procesos deben sincronizarse, crece el overhead tanto en MPI como en el híbrido, dado que si bien el híbrido aprovecharía un poco más la memoria que comparte en el nodo, tiene que esperar las que se dan por Ethernet desde y hacia los procesos de otros nodos, por ejemplo para sincronizar los promedios.

Por otro lado también se ve como el algoritmo no escala ya que las eficiencia decae al incrementar el número de unidades de procesamiento.

En el algoritmo de MPI con 8 procesos, que se hace en un nodo, y como las comunicaciones intra-nodo son más rápidas vemos como mejora el rendimiento, respecto de los de 16, 32 y 64.

De todas formas es un poco más ineficiente que los algoritmos de la Entrega anterior, ya que aquellos no tienen overhead de comunicación.