Sistemas Paralelos

Entrega 3

Grupo 12; Juan Andrés Geido

Parte 1:

Ejercicio 2, Practica 4:

Pregunta: 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?

Respuesta: La versión no bloqueante vuelve a obtener el control inmediatamente, aunque obviamente sin haber recibido el mensaje.

Pregunta: 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?

Respuesta: Se imprime el contenido inicial del buffer (en este caso, explicitamente, "No deberia estar leyendo esta frase."). Se debe a que estoy continuando la ejecución sin haber primero esperado a recibir un dato necesario para finalizar la tarea.

Ejercicio 3, Practica 4:

El algoritmo non-blocking-ring.c requiere mucho menos tiempo de comunicación. Con las funciones Irecv MPI se encarga de recibir los datos y colocarlos en su buffer, "en el fondo", permitiendo que el programa continue haciendo cualquier otro tipo de ejecución que no requiera ese dato.

Parte 2:

Consideraciones en el testeo:

- Todas las pruebas se hicieron en el cluster a través de Slurm (scripts sbatch)
- Todas las pruebas se hicieron con -O2, ya que este había mostrado el mejor rendimiento con el algoritmo secuencial en la Entrega 2.
- Todas las pruebas se hicieron con BS=8, por la misma razón.

Consideraciones en el desarrollo:

Para el algoritmo MPI, tome la versión secuencial del algoritmo usado en la Entrega 2, y comense por evaluar la forma en cual dividir el problema. Lo que mas sentido termino teniendo fue mirarlo de forma parecida al pthreads, donde ciertas partes se divirian por “responsabilidad”, por ejemplo la primera matriz de cada multiplicación, y otras partes se compartirian por completo.

En su ejecución el algoritmo se podria trozar en 5 partes claras:

1. Etapa de comunicación, donde se dividen A y C en partes iguales, y se comparten B y D en su enteridad.
2. Etapa de ejecución, donde cada proceso primero calcula Pot2(D)*, y luego el minimo, maximo y promedio de A y B. En el caso de B cada hilo tiene su copia local de B. El caso de A, sin embargo, cada proceso tiene su chunk, lo cual requiere comunicación.
3. Etapa de comunicación, en la cual se hacen reducciones de minimo, maximo y suma total de A, lo cual permite que los procesos compartan y concluyan entre si esos 3 valores.
4. Etapa de ejecución final: Se calcula el escalar y se realiza la operación principal, cada proceso calculando su porción del resultado.
5. Etapa de comunicación final: Ya habiendo cada proceso finalizado su porción de R, finalmente todo se junta en MASTER, y se da por finalizada la ejecución.

* parece contra-intuitivo que cada hilo calcule esto, pero en mi testeo local, compartir D tenia un efecto medible en los tiempos de comunicación, mientras que hacerlo asi no tuvo cambios medibles en la ejecución

El algoritmo hibrido fue el resultado de aplicar los mismos fundamentos sobre el algoritmo OpenMP de la Entrega 2. Fue un proceso muy directo y sencillo del cual realmente no hay mucho que decir. Siento que una version Pthreads habria sido mucho, mucho mas complicado de implementar, ya que habria requerido considerar division de responsabilidades tanto a nivel de comunicación de procesos como a nivel de comunicación de hilos.

La conclusión es que OpenMPI y OpenMP son muy llevaderos de trabajar uno con el otro, y me es dificil ver una razón para no usar ambos a la vez en una situación de mundo real.

Pruebas de rendimiento:

Resultados de tiempo de ejecución total y comunicación.

Primero, para referencia, los resultados del agoritmo secuencial. Luego MPI, y luego el Hibrido.

SECUENCIAL	
N=512	0,56102
N=1024	4,60413
N=2048	37,00317
N=4096	294,13647

MPI						
P	8		16		32	
	total	comms	total	comms	total	comms
N=512	0,09831	0,02363	0,10427	0,06829	0,11252	0,1047
N=1024	0,71865	0,07198	0,60211	0,27218	0,48809	0,33587
N=2048	5,33576	0,27317	3,58952	1,02409	2,65346	1,31489
N=4096	40,74573	1,13347	24,02322	4,01555	15,24466	4,99844

MPI + OPENMP						
P	8		16		32	
	total	comms	total	comms	total	comms
N=512	/		0,09736	0,06233	0,10498	0,09259
N=1024			0,53523	0,23753	0,45872	0,30925
N=2048			3,29976	0,94608	2,38238	1,19739
N=4096			22,42354	3,78256	14,05184	4,71394

Los resultados me parecen esperados, donde la solución MPI que se ejecuta en un solo nodo presenta el minimo overhead, pero cuando se trabaja en mas de un nodo, la versión hibrida se convierte en la menos ociosa, ya que se ahorra hacer pasaje de mesajes dentro del mismo metal.

Esto se ve reflejado en la tabla de overhead.

OVERHEAD

MPI			
NUM_THRD	8	16	32
N=512	0,240	0,655	0,931
N=1024	0,100	0,452	0,688
N=2048	0,051	0,285	0,496
N=4096	0,028	0,167	0,328

MPI + OPENMP			
NUM_THRD	8	16	32
N=512	/	0,640	0,882
N=1024		0,444	0,674
N=2048		0,287	0,503
N=4096		0,169	0,335

Ya teniendo esta información, podemos calcular tambien el Speedup y Eficiencia de cada implementación comparada con la secuencial.

SPEEDUP

MPI			
NUM_THRD	8	16	32
N=512	5,707	5,380	4,986
N=1024	6,407	7,647	9,433
N=2048	6,935	10,309	13,945
N=4096	7,219	12,244	19,294

MPI + OPENMP			
NUM_THRD	8	16	32
N=512	/	5,762	5,344
N=1024		8,602	10,037
N=2048		11,214	15,532
N=4096		13,117	20,932

EFICIENCIA

MPI			
NUM_THRD	8	16	32
N=512	0,713	0,336	0,156
N=1024	0,801	0,478	0,295
N=2048	0,867	0,644	0,436
N=4096	0,902	0,765	0,603

MPI + OPENMP			
NUM_THRD	8	16	32
N=512	/	0,360	0,167
N=1024		0,538	0,314
N=2048		0,701	0,485
N=4096		0,820	0,654

Viendo estos datos, se vuelve a soportar lo mismo: Cuanto mayor es el tamaño del trabajo, mayor es la eficiencia. Y en el caso del algoritmo hibrido, se ve el mayor aprovechamiento de tiempo de CPU, posiblemente por la misma razon vista en la pagina anterior: Menor tiempo de comunicación.

Atención: Los graficos comparativos se encuentran en tres PDF separados: graficos_runtime.pdf, graficos_speedup_eficiencia_overhead.pdf, y mpi_vs_openmp.pdf respectivamente.

Esto se debe a unos problemas que tuve con LibreOffice que me llevaron a utilizar otro software para generar los graficos.