INFORME

ENTREGA 2

Algoritmos Paralelos

Nehuen Pereyra 878/6

Soledad Rocha 773/7

Grupo 9

7 de Julio de 2021

Sistemas Distribuidos y Paralelos

ENUNCIADO

Ejercicio 1

Resolver mediante una *algoritmo híbrido MPI-OpenMP* la siguiente ecuación:

$$R = PromB \cdot AC + PromA \cdot BD$$

Donde:

- A, B, C y D son matrices de NxN.
- PromA y PromB son los promedios de los valores de los elementos de las matrices A y B, respectivamente.
- Evaluar la expresión para los siguientes tamaños de matrices potencias de 2: 512, 1024 y 2048.

Ejercicio 2

Resolver mediante un *algoritmo híbrido MPI-Pthreads* el problema de encontrar los 100 elementos más frecuentes en un arreglo de E números enteros.

Se debe elegir E en potencias de 2, a partir del millón de elementos. Considerar que el tiempo de ejecución secuencial debe superar 1 segundo de ejecución y la cantidad de elementos no debe superar la cantidad de memoria disponible.

Resolución ejercicio 1

Para la resolución secuencial del problema, primero se realizó la multiplicación optimizada de matrices AC y DB, luego se calculó el promedio de la matriz A y B. Por último, se calculó la matriz resultado.

Para la resolución paralela del problema, se utilizó MPI para distribuir las filas de las matrices A y B entre el número de procesos y se les pasó la matrices C y D a todos los procesos, para que puedan a través de Open MP realizar la multiplicación parcial de las matrices (utilizando hilos). Al mismo tiempo se calcula el promedio de las matrices A y B de forma parcial. Para optimizar esta parte, realizamos un solo bloque de Open MP, que gracias a la cláusula nowait permite evitar la barrera implícita y que los hilos que terminaron de hacer los cálculos puedan seguir trabajando, evitando que se tengan que poner a dormir y después volver a despertarlos. Luego se reduce el promedio de A y B (con MPI), de forma siguiente utilizando Open MP, se calcula el resultado parcial de R y

por último se reduce utilizando MPI para obtener la matriz resultado R.

Para la validación en ambos casos se tomó en cuenta que las matrices A, B, C, D toman todos sus valores 1 por lo tanto la matriz resultante R que es NxN con valores N+N.

Para probar este algoritmo, para la versión secuencial:

- sbatch ./p1_script_secuencial.sh N

Mientras que:

- sbatch ./p1_script_paralelo.sh N T

Donde N es la dimensión de la matriz y T es la cantidad de hilos. En este caso, además en el script se agregó el parámetro --bind-to none, para que los procesos se distribuyan correctamente en cada nodo.

Tabla 1. Análisis de Speedup y eficiencia de los algoritmos respecto al secuencial del ejercicio 1 (tiempo en segundos)

		Tamaño de problemas (N)			Ideal
Procesadores		512	1024	2048	
Secuencial	Tiempo	1.971546	16.14016	128.678543	
P0 Cores 4 2 Procesos 2 Hilos	Tiempo	0.655732	4.814881	37.326091	
	Speedup	**3.006633808**	**3.352140998**	**3.447415455**	**4**
	Eficiencia	0.7516584519	0.8380352495	0.8618538638	1
P1 Cores 8 2 Procesos 4 Hilos	Tiempo	0.370179	2.562568	19.349147	
	Speedup	**5.325926106**	**6.298431886**	**6.650347067**	**8**
	Eficiencia	0.6657407633	0.7873039857	0.8312933834	1
P2 Cores 16 2 Procesos 8 Hilos	Tiempo	0.231497	1.439091	10.373099	
	Speedup	**8.516507773**	**11.21552424**	**12.40502409**	**16**
	Eficiencia	0.5322817358	0.7009702653	0.7753140057	1
P3 Cores 32 4 Procesos 8 Hilos	Tiempo	0.191871	0.943677	5.951893	
	Speedup	**10.27537252**	**17.10347926**	**21.61976753**	**32**
	Eficiencia	0.3211053911	0.534483727	0.6756177352	1

Un programa paralelo puede ser:

- **Fuertemente escalable**: mantiene la eficiencia constante sin incrementar el tamaño del problema.
- **Débilmente escalable**: mantiene la eficiencia constante al incrementar el tamaño de problema al mismo tiempo que se incrementa el número de procesadores.
- **No escalable**: no se cumplen los casos anteriores.

Para realizar el análisis de escalabilidad, veremos si cumple las condiciones definidas previamente.

Para que sea fuertemente escalable tenemos que ver cómo se comporta la eficiencia sin incrementar el tamaño del problema. Si vemos la tabla 2 se puede ver que la eficiencia disminuye al incrementar la cantidad de cores (lo mismo ocurre para N más grandes) por lo tanto no es fuertemente escalable.

Tabla 2. Eficiencia para un tamaño del problema específico.

		Tamaño de problemas (N)			Ideal
Procesadores		512	1024	2048	
Secuencial	Tiempo	1.971546	16.14016	128.678543	
P0 Core 4 2 Procesos 2 Hilos	Tiempo	0.655732	4.814881	37.326091	
	Speedup	3.006633808	3.352140998	3.447415455	4
	Eficiencia	**0.7516584519**	**0.8380352495**	**0.8618538638**	**1**
P1 Core 8 2 Procesos 4 Hilos	Tiempo	0.370179	2.562568	19.349147	
	Speedup	5.325926106	6.298431886	6.650347067	8
	Eficiencia	**0.6657407633**	**0.7873039857**	**0.8312933834**	**1**
P2 Cores 16 2 Procesos 8 Hilos	Tiempo	0.231497	1.439091	10.373099	
	Speedup	8.516507773	11.21552424	12.40502409	16
	Eficiencia	**0.5322817358**	**0.7009702653**	**0.7753140057**	**1**
P3 Cores 32 4 Procesos 8 Hilos	Tiempo	0.191871	0.943677	5.951893	
	Speedup	10.27537252	17.10347926	21.61976753	32
	Eficiencia	**0.3211053911**	**0.534483727**	**0.6756177352**	**1**

Para analizar si es débilmente escalable, analizamos si la eficiencia se mantiene relativamente constante incrementando al mismo tiempo el tamaño de problema y el número de procesadores. Si vemos la tabla 3 podemos observar que si cumple, por lo tanto es débilmente escalable.

Tabla 3. Eficiencia al incrementar el problema y la cantidad de cores.

	Tamaño de problemas (N)				Ideal
Procesadores		512	1024	2048	
Secuencial	Tiempo	1.971546	16.14016	128.678543	
P0 Core 4 2 Procesos 2 Hilos	Tiempo	0.655732	4.814881	37.326091	
	Speedup	3.006633808	3.352140998	3.447415455	4
	Eficiencia	**0.7516584519**	**0.8380352495**	**0.8618538638**	1
P1 Core 8 2 Procesos 4 Hilos	Tiempo	0.370179	2.562568	19.349147	
	Speedup	5.325926106	6.298431886	6.650347067	8
	Eficiencia	**0.6657407633**	**0.7873039857**	**0.8312933834**	1
P2 Cores 16 2 Procesos 8 Hilos	Tiempo	0.231497	1.439091	10.373099	
	Speedup	8.516507773	11.21552424	12.40502409	16
	Eficiencia	**0.5322817358**	**0.7009702653**	**0.7753140057**	1
P3 Cores 32 4 Procesos 8 Hilos	Tiempo	0.191871	0.943677	5.951893	
	Speedup	10.27537252	17.10347926	21.61976753	32
	Eficiencia	**0.3211053911**	**0.534483727**	**0.6756177352**	1

Resolución ejercicio 2

Para la resolución secuencial como paralela se utilizó un registro que almacena el número y la cantidad de ocurrencias.

Para la resolución secuencial del problema, primero se definió un vector de 2^N donde N es el parámetro ingresado por el usuario. Para cumplir con más de un millón de elementos se eligieron los N con valores de 20, 21 y 22 para realizar el trabajo y el análisis de escalabilidad. Los valores de este vector se generan de forma aleatoria. Lo primero que se realiza es totalizar en un vector, los elementos con la cantidad de veces que se repite. Luego se aplica el algoritmo de merge sort de forma iterativa, para que sea más eficiente (a comparación de la forma recursiva, ya que utiliza la pila). Por último se seleccionan en un vector R de resultado, los primeros 100 elementos, los cuales son los más frecuentes.

Para la resolución paralela del problema, se utiliza MPI para distribuir el arreglo de forma equitativa entre los procesos, luego utilizando Pthreads se calcula de forma parcial la totalización de los elementos y luego se reduce, lo siguiente es realizar la reducción con MPI, para este punto ya se tienen todos los elementos TOTALIZADOS **con respecto al arreglo global inicial**. Luego el proceso root distribuye las porciones del arreglo reducido que ya está totalizado (cada elemento aparece una sola vez), a cada uno de los procesos. Cada proceso ordena su porción, para este punto ya tenemos cada porción TOTALIZADA y ORDENADA. Ahora comienza la reducción, en la reducción lo que se hace es ir tomando del vector local y el vector pasado por el “proceso par” el registro que posea la ocurrencia más grande y se incrementa el índice de la porción que tenga el registro más grande, esto se realiza hasta tener los 100 primeros elementos (que van a ser los más grandes) y luego se los pasa a su proceso par, hasta finalizar la reducción, donde se tienen los 100 elementos más occurrentes. Esto se puede hacer ya que se tiene las porciones ordenas y totalizadas.

Para la validación, en ambos casos se imprimen los primeros 100 elementos con su correspondiente cantidad de ocurrencias, para así poder validar que son los más frecuentes.

Para probar este algoritmo, para la versión secuencial:

- sbatch ./p2_script_secuencial.sh N

Para la versión paralela:

- sbatch ./p2_script_paralelo.sh N T

Donde N es el valor al cual se va elevar con base 2 (2^N), esto indica el tamaño del vector y T es la cantidad de hilos. En en caso paralelo, además en el script se agregó el parámetro --bind-to none, para que los procesos se distribuyan correctamente en cada nodo.

Tabla 4. Análisis de Speedup y eficiencia de los algoritmos respecto al secuencial del ejercicio 2 (tiempo en segundos)

	Tamaño de problemas (N)				Ideal
Procesadores		2^20	2^21	2^22	
Secuencial	Tiempo	15.921851	31.862383	63.795083	
P0 Cores 4 2 Procesos 2 Hilos	Tiempo	5.45228	10.655289	21.100191	
	Speedup	**2.920218881**	**2.990288016**	**3.023436281**	**4**
	Eficiencia	0.7300547202	0.7475720039	0.7558590702	1
P1 Cores 8 2 Procesos 4 Hilos	Tiempo	3.011453	5.62148	10.863164	
	Speedup	**5.287099284**	**5.667970534**	**5.872606084**	**8**
	Eficiencia	0.6608874105	0.7084963168	0.7340757605	1
P2 Cores 16 2 Procesos 8 Hilos	Tiempo	1.878103	3.193612	5.83705	
	Speedup	**8.477623964**	**9.976911096**	**10.92933639**	**16**
	Eficiencia	0.5298514978	0.6235569435	0.6830835246	1
P3 Cores 32 4 Procesos 8 Hilos	Tiempo	1.35286	2.034105	3.392778	
	Speedup	**11.76903079**	**15.66407978**	**18.80319991**	**32**
	Eficiencia	0.3677822123	0.4895024931	0.5875999973	1

Utilizando los valores obtenidos y reflejados en la tabla 4, realizaremos el análisis de escalabilidad de igual forma que hicimos en el ejercicio 1.

Para que sea fuertemente escalable tenemos que ver cómo se comporta la eficiencia sin incrementar el tamaño del problema. Si vemos la tabla 5 se puede ver que la eficiencia disminuye al incrementar la cantidad de cores (lo mismo ocurre para N más grandes)

por lo tanto no es fuertemente escalable.

Tabla 5. Eficiencia para un tamaño del problema específico.

		Tamaño de problemas (N)			Ideal
Procesadores		2^20	2^21	2^22	
Secuencial	Tiempo	15.921851	31.862383	63.795083	
P0 Core 4 2 Procesos 2 Hilos	Tiempo	5.45228	10.655289	21.100191	
	Speedup	2.920218881	2.990288016	3.023436281	4
	Eficiencia	**0.7300547202**	**0.7475720039**	**0.7558590702**	**1**
P1 Core 8 2 Procesos 4 Hilos	Tiempo	3.011453	5.62148	10.863164	
	Speedup	5.287099284	5.667970534	5.872606084	8
	Eficiencia	**0.6608874105**	**0.7084963168**	**0.7340757605**	**1**
P2 Core 16 2 Procesos 8 Hilos	Tiempo	1.878103	3.193612	5.83705	
	Speedup	8.477623964	9.976911096	10.92933639	16
	Eficiencia	**0.5298514978**	**0.6235569435**	**0.6830835246**	**1**
P3 Core 32 4 Procesos 8 Hilos	Tiempo	1.35286	2.034105	3.392778	
	Speedup	11.76903079	15.66407978	18.80319991	32
	Eficiencia	**0.3677822123**	**0.4895024931**	**0.5875999973**	**1**

Para analizar si es débilmente escalable, analizamos si la eficiencia se mantiene relativamente constante incrementando al mismo tiempo el tamaño de problema y el número de procesadores. Si vemos la tabla 6 podemos observar que si cumple, por lo tanto es débilmente escalable.

Tabla 6. Eficiencia al incrementar el problema y la cantidad de cores.

		Tamaño de problemas (N)			Ideal
Procesadores		2^20	2^21	2^22	
Secuencial	Tiempo	15.921851	31.862383	63.795083	
P0 Core 4 2 Procesos 2 Hilos	Tiempo	5.45228	10.655289	21.100191	
	Speedup	2.920218881	2.990288016	3.023436281	4
	Eficiencia	**0.7300547202**	**0.7475720039**	**0.7558590702**	**1**
P1 Core 8 2 Procesos 4 Hilos	Tiempo	3.011453	5.62148	10.863164	
	Speedup	5.287099284	5.667970534	5.872606084	8
	Eficiencia	**0.6608874105**	**0.7084963168**	**0.7340757605**	**1**
P2 Core 16 2 Procesos 8 Hilos	Tiempo	1.878103	3.193612	5.83705	
	Speedup	8.477623964	9.976911096	10.92933639	16
	Eficiencia	**0.5298514978**	**0.6235569435**	**0.6830835246**	**1**
P3 Core 32 4 Procesos 8 Hilos	Tiempo	1.35286	2.034105	3.392778	
	Speedup	11.76903079	15.66407978	18.80319991	32
	Eficiencia	**0.3677822123**	**0.4895024931**	**0.5875999973**	**1**