Sistemas Distribuidos y Paralelos

Entregable 3

Juan Manuel Ramallo 954/1

- Ejercicio 1

1.1 Enunciado

Realizar un algoritmo MPI que resuelva la expresión:

$$M = \bar{d}ABC + \bar{b}DEF \quad ,$$

donde A, B, C, D, E y F son matrices de NxN. $\bar{d}$ y $\bar{b}$ son los promedios de los valores de los elementos de las matrices D y B, respectivamente. Evaluar N=512, 1024 y 2048.

1.2 Resolución

- Primero que nada para mayor legibilidad y poder entender mejor el flujo del programa, se separan las acciones que deben ejecutar el proceso root, o maestro, de los procesos workers.
- El programa empieza enviando los datos necesarios a todos los procesos disponibles para realizar los cálculos necesarios.
- Las multiplicaciones de matrices se realizan en paralelos con todos los procesos disponibles, para lo cual antes se debió haber enviado una matriz completa y la segunda matriz en porciones usando el `MPI_Scatter`.

1.3 Métricas

- A continuación se pueden observar las mediciones tomadas en las computadoras del aula de postgrado.

Ejercicio 1						
Una máquina						
Procesos	Tamaño	Overhead	Tiempo	Total	Speedup	Eficiencia
4	512	0.01	0.77	0.77	3.43	0.86
	1024	0.03	6.10	6.13	3.33	0.83
	2048	0.08	48.60	48.68	3.35	0.84
Dos máquinas						
Procesos/Máquina	Tamaño	Overhead	Tiempo	Total	Speedup	Eficiencia
2	512	0.11	0.73	0.83	3.18	0.79
	1024	0.22	5.85	6.07	3.36	0.84
	2048	0.85	46.84	47.69	3.42	0.86
4	512	0.15	0.38	0.52	5.05	0.63
	1024	0.23	3.04	3.26	6.26	0.78
	2048	0.86	24.26	25.13	6.50	0.81

- El principio de localidad hace que la ejecución en dos máquinas con la misma cantidad de procesos sea mejor que con una sola, cada máquina poseerá los accesos a memoria requeridos por el programa, en posiciones contiguas de memoria.

- Ejercicio 2

2.1 Enunciado

Paralelizar con MPI un algoritmo que ordene un vector de N elementos por mezcla.

2.2 Resolución

- Para mantener el codigo de la función principal más limpio, al igual que en el ejercicio 1 se separan las instrucciones que ejecutan el root y los workers en distintas funciones.
- El root se encarga de generar el arreglo de N elementos con M como número máximo entre los numeros que contendrá el arreglo (N y M son parámetros del main)
- Lo primero que hace el root es distribuir el arreglo entre todos los procesos
- Luego cada proceso (incluyendo el root) ordena su porción del arreglo
- Luego cada proceso con ID impar y el último, envía su porción ordenada y libera la memoria

- A su vez, todos los procesos con ID par reciben los datos enviados de los impares para hacer de nuevo una ordenación por mezcla.
- El root, al final de la ejecución, recibe todas las porciones y las ordena

2.3 Métricas

- Según la eficiencia vista, para una sola máquina los procesadores pasan más del 80% del tiempo realizando trabajo útil.
- Para dos máquinas, se ve como decrece la eficiencia, debido a que se introduce demasiado tiempo de comunicación.

Ejercicio 2							
Una máquina							
Procesos	Tamaño	Num. Max	Overhead	Tiempo	Total	Speedup	Eficiencia
4	5 millones	100	0.01	0.31	0.32	3.19	0.80
		1000	0.01	0.33	0.34	3.24	0.81
	50 millones	100	0.07	3.37	3.44	3.26	0.81
		1000	0.06	3.56	3.63	3.29	0.82
Dos máquinas							
Procesos/Máquina	Tamaño	Num. Max	Overhead	Tiempo	Total	Speedup	Eficiencia
2	5 millones	100	0.18	0.30	0.48	2.12	0.53
		1000	0.18	0.32	0.50	2.20	0.55
	50 millones	100	1.74	3.23	4.97	2.26	0.56
		1000	1.73	3.42	5.16	2.31	0.58
4	5 millones	100	0.19	0.19	0.38	2.72	0.34
		1000	0.18	0.20	0.38	2.91	0.36
	50 millones	100	1.76	1.98	3.74	3.00	0.37
		1000	1.75	2.08	3.83	3.12	0.39

- Ejercicio 3

3.1 Enunciado

Paralelizar con MPI un algoritmo que dado un vector de números enteros encuentre los 100 elementos más frecuentes.

3.2 Resolución

- Se hace uso de una estructura para almacenar las cuentas de cada número que aparece en el arreglo a buscar.

- La estructura (Counter) está compuesta por dos números enteros, uno que representa el número encontrado y el otro que representa la cantidad de veces que apareció ese número en el arreglo.
- Se implementó una función `agregar_ordenado` que agrega la cuenta de un número en un arreglo pasado como parámetro por referencia y ordena el arreglo luego de agregar, al final retorna el tamaño del arreglo; recibe los parámetros:
  - Arreglo de `Counters`
  - Numero a agregar
  - Cuenta a incrementar
  - Tamaño del arreglo de cuentas
- El root inicializa el arreglo con N elementos con M como número máximo dentro del arreglo
- El root reparte porciones iguales del arreglo entre todos los procesos
- Cada proceso (incluyendo el root) cuenta las apariciones de cada número en su porción y las guarda en un arreglo de `Counters`
- Luego se une las cuentas con el root de la misma forma que con el ejercicio anterior se unieron las ordenaciones.
- Los procesos con ID impares envían sus cuentas y terminan, mientras los procesos pares agregan las cuentas hasta que al final envían sus cuentas al root, el cual las espera y las agrega.

3.3 Métricas

Al observar como varía el speedup para un mismo `número máximo` variando el tamaño del arreglo se puede observar que el speedup aumenta, por lo tanto se puede afirmar que al aumentar el tamaño del problema el rendimiento de la ejecución del programa mejora.

Ejercicio 3							
Una máquina							
Procesos	Tamaño	Num. Max	Overhead	Tiempo	Total	Speedup	Eficiencia
4	5 millones	100	0.005	0.38	0.39	3.62	0.90
		1000	0.08	3.62	3.7	3.61	0.90
	50 millones	100	0.04	3.75	3.79	3.70	0.92
		1000	0.04	35.93	35.97	3.69	0.92
Dos máquinas							
Procesos/Máquina	Tamaño	Num. Max	Overhead	Tiempo	Total	Speedup	Eficiencia
2	5 millones	100	0.005	0.38	0.38	3.68	0.92
		1000	0.005	3.63	3.63	3.67	0.92
	50 millones	100	0.04	3.76	3.80	3.69	0.92
		1000	0.05	35.93	35.98	3.69	0.92
4	5 millones	100	0.10	0.19	0.29	4.84	0.60
		1000	0.10	1.83	1.92	6.95	0.87
	50 millones	100	0.88	1.88	2.76	5.08	0.63
		1000	0.88	18.00	18.88	7.03	0.88