Practico Final
Sistemas distribuidos & paralelos

Hernán Flores Leyes – 2013

El objetivo de este informe es analizar y comparar el desempeño de una solución que será desarrollada de manera secuencial y luego la misma solucion implementada como un programa paralelo híbrido.

El algoritmo con el que se trabaja es el que sigue:

1. Dadas dos imagenes BMP de 24 bits:
2. Combinarlas ambas imagenes en una sola imagen BMP resultado de la mezcla de los componentes de color de cada una de ellas.
3. Convertir la imagen resultante de esta combinacion en una imagen en escala de grises.

El paralelismo se implementará mediante un enfoque híbrido:

- Paralelismo a nivel de procesos a traves de MPI (pasaje de mensajes)
- Paralelismo a nivel de threads a traves de OpenMP (memoria compartida)

¿Cual es la justificacion para el uso de este modelo hibrido?

Es una solucion elegante en concepto y a nivel arquitectonico; usamos MPI como mecanismo entre los nodos y openMP dentro de los nodos en sí.
Ademas permite un mejor uso de los recursos de sistema compartidos (memoria y ancho de banda), ya que al utilizar OpenMP para la comunicacion intra-nodo evitamos el overhead de comunicacion de MPI. Sumado a esto OpenMP nos permite tener una granularidad mas fina (el tamaño de los mensajes puede ser mayor), lo que logra un balanceo de cargas mas dinamico.

¿Cuales serian los pros y contras de usar cualquier de las dos metodologias exclusivamente?

MPI puro

Pros	Contras
Portable	Dificil de desarrollar y hacer debugging
Es posible alcanzar alta performance	Alta latencia, bajo ancho de banda

Escala a mas de un nodo	Comunicacion explicita
No exsitesn problemas de ubicacion de datos	Granularidad grande
Sincronizacion implicita	Se dificulta el balance de carga.

OpenMP Puro

Pros	**Contras**
Paralelismo facil de implementar	Solo aplicable a maquinas de memoria compartida
Baja latencia, gran ancho de banda	Performance promedio
Comunicacion Implicita	Escala dentro de un nodo
Granularidad fina y gruesa	Posible problema de ubicacion de datos
Balance de carga dinamico	No hay order especifico de threads

Probaremos dentro de este esquema hibrido dividir la imagen segun la cantidad de procesadores para ver si es existe alguna mejora en el desempeño. Tanto la solución secuencial como la paralela, fueron desarrolladas en C sin librerías destinadas al procesamiento de imágenes para poder tener pleno acceso al código que maneja las imágenes.

Antes debemos tener en cuenta dos cosas:
Por la naturaleza del problema en sí, tenemos una region paralelizable–a saber, el calculo de la mezcla de imagenes; la iteracion sobre el arreglo de bits de cada imagen y su correspondiente suma, para luego pasarlo a escala de grises– y varias regiones criticas que solo pueden ejecutarse de manera secuencial–la carga de cada archivo debe hacerse de manera secuencial por el proceso “maestro”, asi como la posterior escritura en el archivo del resultado.
Las imagenes a procesar tendran una resolucion de 1024x768 pixeles.

Solucion secuencial

Tiempo de corrida (MS)
26.356
34.402
30.52
26.04
28.41
30.212
29.986
28.162

26.106
25.694
33.378
26.596
26.096
32.99
25.9
26.174
34.09
25.44
27.538
28.72
26.39
25.486
28.828
27.692
26.212
28.99
25.88
25.562
27.822
25.746

De 30 ejecuciones, la solucion secuencial demoró un promedio de 28.05 milisegundos.

Solucion con Paralelismo

Tiempo p/ 2 (ms)	Tiempo p/ 4 (ms)	Tiempo p/ 8 (ms)
17.1314	13.70512	12.91444
22.3613	17.88904	16.85698
19.838	15.8704	14.9548
16.926	13.5408	12.7596
18.4665	14.7732	13.9209
19.6378	15.71024	14.80388
19.4909	15.59272	14.69314
18.3053	14.64424	13.79938
16.9689	13.57512	12.79194
16.7011	13.36088	12.59006
21.6957	17.35656	16.35522
17.2874	13.82992	13.03204
16.9624	13.56992	12.78704
21.4435	17.1548	16.1651
16.835	13.468	12.691
17.0131	13.61048	12.82526
22.1585	17.7268	16.7041

16.536	13.2288	12.4656
17.8997	14.31976	13.49362
18.668	14.9344	14.0728
17.1535	13.7228	12.9311
16.5659	13.25272	12.48814
18.7382	14.99056	14.12572
17.9998	14.39984	13.56908
17.0378	13.63024	12.84388
18.8435	15.0748	14.2051
16.822	13.4576	12.6812
16.6153	13.29224	12.52538
18.0843	14.46744	13.63278
16.7349	13.38792	12.61554

De 30 ejecuciones los resultados son:

- 2 procesadores: 18.24ms
    - Speed-up promedio: 1.53846153846154
- 4 procesadores: 14.59ms
    - Speed-up promedio: 1.92307692307692
- 8 procesadores: 13.74ms
    - Speed-up promedio: 2.04081632653061

Conclusiones

Si bien en este caso no se dió, a veces esta aproximacion mezclada puede ser peor que la implementacion secuencial por el overhead de comunicacion (por MPI) o el overhead de creacion de threads (por openMP). En referencia a la escalabilidad, los resultados obtenidos muestran que la solución híbrida es escalable y que el aumento en cantidad de procesadores lleva a tiempos de ejecución menores y a un speed-up que se incrementa en relacion al agregado de procesadores.