Departamento de [Ingeniería Civil Informática]
Facultad de Ingeniería
Universidad de Santiago de Chile

[Sistemas Distribuidos y Paralelos]

LAB 3: Paralelismo con OpenMP

Estudiante: Nícolas Farfán Cheneaux
Docente: Fernando Rannou

Resumen

En el presente informe de rendimiento computacional, se estudia el desempeño de una solución en usando el estándar de programación paralela OpenMP, se utilizan múltiples tareas/procesadores grideadoras con mecanismos de sincronización y exclusión mutua como **taskwait** para garantizar la sincronización y **critical** para garantizar la exclusión mutua.

Estrategia de paralelización

Se realizan dos métodos de paralelización y concurrencia, en primer lugar el de las matrices propias/privadas donde cada tarea tiene sus propias matrices y al finalizar deben ser acumuladas en una absoluta en el main o tarea principal, la otra solución consiste en que exista solo una matríz absoluta y sea escrita por exclusión mútua usando crítical. Al fin y al cabo ambas soluciones son muy similaresen cuanto a código, solo cambia en la acumulación de resultados.

Optimización la solución

Si se utiliza un buffer que almacena las visibilidades en la sección críctica estático, es decir que lo almacene en el stack se obtiene una mejora significativa en los tiempos entre 0.5s- 0.7s, llegando a ser la mitad de los tiempos en la solución planteada, sin embargo está restringido a usar un chunk máximo de 30.000 líneas (depende de la memoria del ordenador). Por lo que se optó por usar un arreglo dinámico **malloc** compartido que se usará de buffer para almacenar los strings/visibilidades y luego obtener sus componentes y proceder al procesamiento.

Resultados

Especificaciones:

- CPU: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx (8) @ 2.100GHz
- Memory: 5864MiB
- OS: Arch Linux x86-64

0.1. Speedup

Usando la formula

$$\frac{T_s}{T_p(n)}$$

En donde **Ts** representa el valor obtenido promedio en la ejecución secuencial (se adjunta código secuencial en el entregable) **sec.c** se realizó una muestra de 30 ejecuciones del código secuencial y se obtuvo que en promedio el programa secuencial demora **5.6[s]**. Y el mejor tiempo paralelo obtenido es de **1.52[s]**.

Chunk adaptativo

Chunk	Tasks	Private Matrix	Shared Matrix
160000	20	1.562	1.856
213000	15	1.628	1.778
320000	10	1.486	1.642
350000	9	1.63	1.598
380000	8	1.624	1.622
420000	7	1.78	1.758
500000	6	1.792	1.714
600000	5	2.102	1.82
700000	4	2.168	1.998
900000	3	2.458	2.46
1500000	2	3.95	4.136
3000000	1	10.326	10.242

Tabla 1: Chunk variable

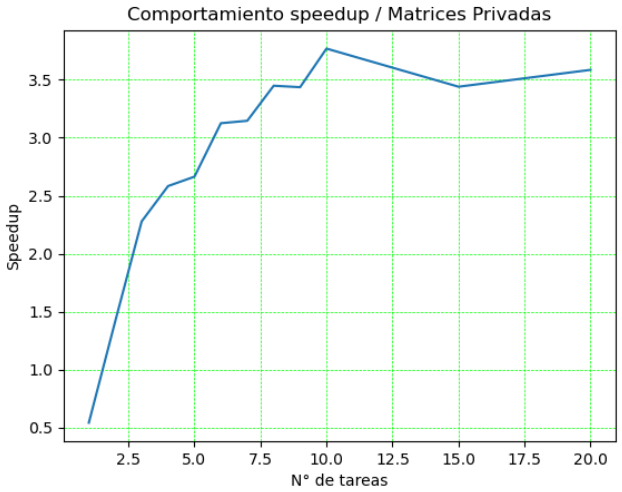

Figura 1: Speedup con chunk adaptado

Chunk fijo (100.000)

Chunk	Tasks	Private Matrix	Shared Matrix
100000	100	1.522	2.074
100000	50	1.774	2.184
100000	30	1.656	1.944
100000	20	1.56	1.964
100000	15	1.584	1.936
100000	10	1.472	1.914
100000	9	1.698	1.956
100000	8	1.61	1.952
100000	7	1.628	1.89
100000	6	1.628	1.906
100000	5	1.638	2.032
100000	4	1.726	2.326
100000	3	2.13	2.62
100000	2	2.766	3.364
100000	1	6.642	6.696

Tabla 2: Chunk fijo de 100.000

Se obtienen los siguientes gráficos se **Speedup** con el método de las matrices privadas tanto para *Tabla 1* y *Tabla 2*.

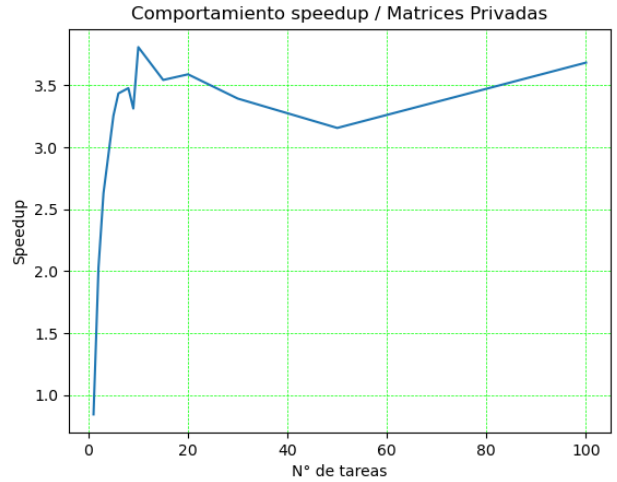

Figura 2: Speedup con chunk fijo 100.000

Se puede observar un speedup positivo desde la tarea número 2 se observa una gran mejora comparado a la solución secuencial. Además el Speedup máximo alcanzado es de **3.7** en ambas ejecuciones. Sin embargo se obtuvieron mejores tiempos en la ejecución con un chunk fijo de 100.000 independiente del número de tareas.

Imágenes resultantes

Mediante el siguiente script creado por mi, se realiza la visibilización de los resultados

```
% Leer los datos de los archivos
s1_s = fread(fopen("datosgrideados_sharedr.raw", "r"), "float");
s2_s = fread(fopen("datosgrideados_sharedi.raw", "r"), "float");

s1_p = fread(fopen("datosgrideados_privater.raw", "r"), "float");
s2_p = fread(fopen("datosgrideados_privatei.raw", "r"), "float");

% Reshape de los datos
re_s = reshape(s1_s, 2048, 2048);
re_p = reshape(s1_p, 2048, 2048);
im_s = reshape(s2_s, 2048, 2048);
im_p = reshape(s2_p, 2048, 2048);

% Crear n meros complejos
v_s = complex(re_s, im_s);
v_p = complex(re_p, im_p);

% Calcular transformada de Fourier e inversa
I_s = fftshift(ifft2(v_s));
I_p = fftshift(ifft2(v_p));

% Crear una figura con dos im genes en una fila
figure;

% Primera imagen
subplot(1,2,1);
imagesc(abs(I_s));
colormap('hot');
title('Shared matrix result image');

% Segunda imagen
subplot(1,2,2);
imagesc(abs(I_p));
colormap('hot');
title('Private matrix result image');
```

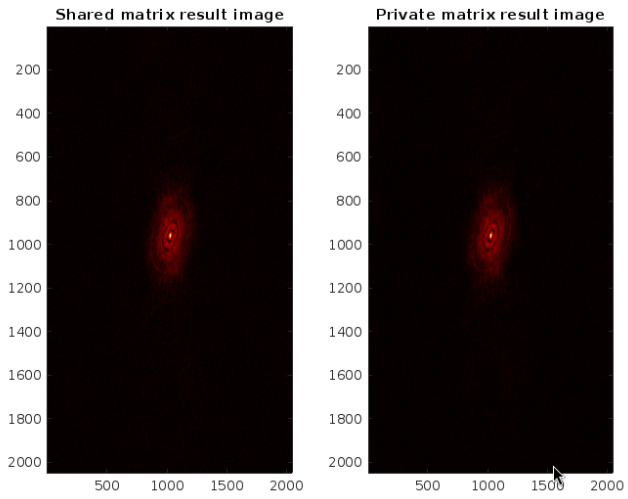

Figura 3: Imágenes obtenidas

Conclusión

Se logró paralelizar la solución correctamente llegando a tener una mejora de hasta **3.5x** veces mejor que la solución secuencial. Entre el método de las matrices privadas y compartidas, se obtuvo mejores tiempos en la solución con matrices privadas, dado que la acumulación de resultados no se realiza con exclusión mutua y además la acumulación final también es paralelizable.

Referencias

OpenMP Tasking Explained Ruud van der Pas. (2013). Retrieved from https://openmp.org/wp-content/uploads/sc13.tasking.ruud.pdf