POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN
Cómputo de Alto Rendimiento con Lenguajes de Alto Nivel 2022-II
Miniproyecto 1

Alumno: Edwin Enrique Pérez Rodríguez

18 de marzo de 2022

Resumen

En este miniproyecto se realiza el análisis de rendimiento al algoritmo de multiplicación de matrices tradicional, que tiene complejidad $O(n^3)$. Se analiza el **speed-up**, **eficiencia** y **escalabilidad**.

1. Introducción

El siguiente miniproyecto se analizan las ventajas de utilizar hilos para volver más eficientes algoritmos secuenciales. Se utiliza como ejemplo el algoritmo de multiplicación de matrices tradicional. En la sección 2 se menciona como se llevó acabo el experimento numérico usando código implementado en lenguaje de *Python*. La sección 3 muestra los resultados numéricos obtenidos y el análisis que se hace de ellos. Finalmente en la sección 4 encontramos las conclusiones que se pueden extraer del experimento.

2. Experimento e implementación numérica

El experimento numérico se realiza con código escrito en el lenguaje de programación *Python* versión 3.10.3. Se consideran matrices de tamaño $2^n \times 2^n$, por tal motivo, las matrices que se usan el experimento son de tamaño: $2^4 \times 2^4$, $2^5 \times 2^5$, $2^6 \times 2^6$, $2^7 \times 2^7$, $2^8 \times 2^8$, $2^9 \times 2^9$ y $2^{10} \times 2^{10}$.

El experimento se realiza considerando el algoritmo de multiplicación de matrices tradicional (Algoritmo 1) que es de complejidad $O(n^3)$. Este algoritmo será nuestro programa secuencial.

Para el programa concurrente se usaron $2, 4, 8$ y 16 hilos y se utilizan de la manera siguiente: dos hilos: un hilo calcula la primera mitad de la multiplicación de matrices y otro

Algorithm 1 Multiplicación de Matrices Tradicional

```
for i = 0 to 2^n do
    for j = 0 to 2^n do
        for k = 0 to 2^{n-1} do
            C[i][j] += A[i][k] + B[k][j]
        end for
    end for
end for
```

hilo calcula el resto, es decir, como se presenta en el Algoritmo 2; cuatro hilos: cada hilo calcula una cuarta parte de la multiplicación, es decir, como se muestra en el Algoritmo 3; para 8 hilos y 16 hilos se sigue una idea similar. Los experimentos se realizaron con matrices aleatorias utilizando la librería ***numpy***.

Algorithm 2 Programa concurrente con 2 hilos

```
Hilo 1
for i = 0 to 2^n do
    for j = 0 to 2^n do
        for k = 0 to 2^{n-1} do
            C[i][j] += A[i][k] + B[k][j]
        end for
    end for
end for

Hilo 2
for i = 0 to 2^n do
    for j = 0 to 2^n do
        for k = 2^{n-1} to 2^n do
            C[i][j] += A[i][k] + B[k][j]
        end for
    end for
end for
```

3. Resultados Numéricos y Análisis.

Los resultados del experimento numérico, con el programa secuencial, se pueden observar en la Tabla 1. Como se puede notar, no es eficiente utilizar el algoritmo clásico de multiplicación de matrices para matrices de tamaño mayores a 1000×1000.

Los resultados del experimento numérico, con el programa concurrente, se pueden ver en la Tabla 2. Se puede observar que con el uso de hilos se pueden calcular de manera más eficiente matrices de tamaños muy grandes. Se reduce en gran medida el tiempo de cálculo, en particular, es más rápido usar dos hilos que utilizar más de dos. El análisis

Algorithm 3 Programa concurrente con 4 hilos

```
Hilo 1
for i = 0 to 2^n do
    for j = 0 to 2^n do
        for k = 0 to 2^(n-2) do
            C[i][j]+ = A[i][k] + B[k][j]
        end for
    end for
end for

Hilo 2
for i = 0 to 2^n do
    for j = 0 to 2^n do
        for k = 2^(n-2) to 2 * 2^(n-2) do
            C[i][j]+ = A[i][k] + B[k][j]
        end for
    end for
end for

Hilo 3
for i = 0 to 2^n do
    for j = 0 to 2^n do
        for k = 2 * 2^(n-2) to 3 * 2^(n-2) do
            C[i][j]+ = A[i][k] + B[k][j]
        end for
    end for
end for

Hilo 4
for i = 0 to 2^n do
    for j = 0 to 2^n do
        for k = 3 * 2^(n-2) to 2^n do
            C[i][j]+ = A[i][k] + B[k][j]
        end for
    end for
end for
```

Tamaño Matriz	16×16	32×32	64×64	128×128	256×256	512×512	1024×1024
Tiempo Secuencial	0.005008	0.034721	0.284407	2.36366	59.3442	655.390	5652.30

Tabla 1: Resultados numéricos obtenidos de los tiempos de cálculo del algoritmo secuencial medido en segundos.

Tamaño de la Matriz	16×16	32×32	64×64	128×128	256×256	512×512	1024×1024
Tiempo Concurrente							
2 Hilos	0.004507	0.028737	0.044694	0.059782	0.058131	0.053685	0.048717
4 Hilos	0.005359	0.033220	0.283329	0.635212	1.209032	0.681731	1.25557
8 Hilos	0.006455	0.033745	0.269729	2.04569	7.02014	2.97566	4.85076
16 Hilos	0.008240	0.036338	0.241066	3.85247	13.5921	16.6047	11.1897

Tabla 2: Resultados numéricos obtenidos de los tiempos de cálculo del algoritmo concurrente medido en segundos.

Hilos	2	4	8	16
Speed-Up	116024.2	4501.78	1165.24	505.132
Eficiencia	58012.1	1125.444	145.655	31.5707

Tabla 3: Cálculo del **Speed-Up** y **Eficiencia** usando los valores obtenidos de las Tablas 1 y 2 usando la matriz de tamaño 1024×1024. Las mediciones se realizan en segundos.

de speed-up y eficiencia se realiza con la matriz de tamaño 1024×1024. Se obtienen los resultados que se pueden observar en la Tabla 3. Note que el *Speed-Up* disminuye cuando se aumenta el número de hilos, de igual manera sucede con la *Eficiencia*. El mayor *Speed-Up* y *Eficiencia* se alcanza cuando se usan dos hilos. Esto se puede observar mejor al visualizar la Figura 1 y la Figura 2. Hay un descenso muy pronunciado del *Speed-Up* y de la *Eficiencia* al aumentar a más de dos el número de hilos.

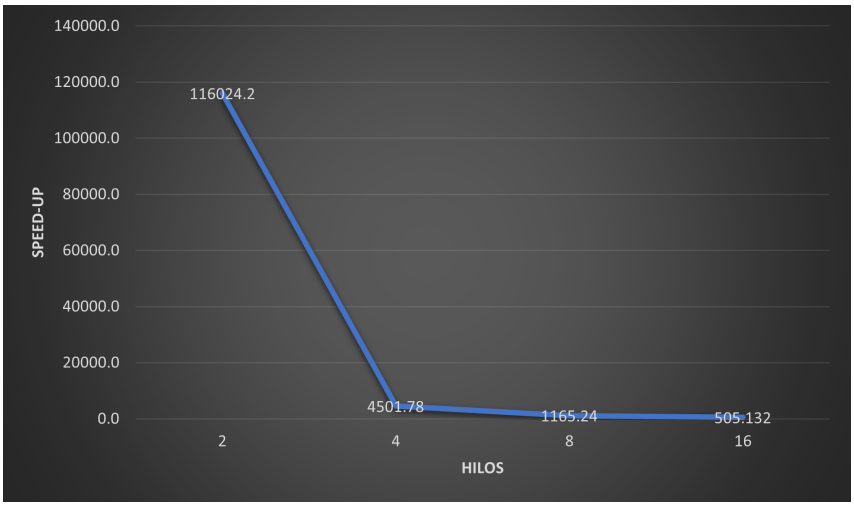

Figura 1: Gráfica del Speed-Up.

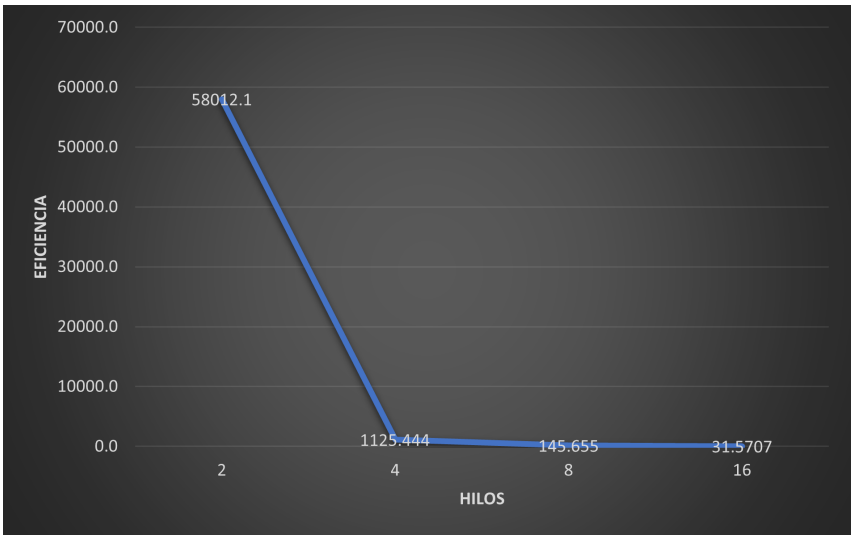

Figura 2: Gráfica de Eficiencia.

4. Conclusiones

Como se puede ver en los resultados numéricos obtenidos, se obtiene mejor tiempo de cálculo con dos hilos. Con cuatro hilos o más no se ve mejoría en el tiempo de cálculo, aunque sigue siendo mucho menor al tiempo obtenido con el algoritmo secuencial. Al considerar una matriz de tamaño 1024×1024, se puede notar también que al usar un número de hilos mayor a dos el **speed-up** y la **eficiencia** disminuye muy drásticamente.

Este experimento permite darse cuenta que un mayor número de hilos no necesariamente implica que un algoritmo mejore en tiempo, speed-up y eficiencia. Se debe hacer algún tipo de análisis antes que permita discernir cuantos hilos usar en un programa concurrente.

Como trabajo personal se podría hacer un análisis de tiempo, speed-up y eficiencia al Algoritmo de Strassen, que mejora considerablemente la complejidad del algoritmo de multiplicación de matrices, de $O(n^3)$ a $O(n^{\log_2 7})$.

Referencias

[1] Esquivel Flores, A. (2022). *Computación Concurrente: Mecanismos de Sincronización.* IIMAS-UNAM.

[2] NumPy Community. (2021). *NumPy.* Numpy. https://numpy.org/

[3] Python Software Foundation. (2021). *Python 3.10.0 documentation.* Python. https://docs.python.org/3/