Curso: POO II

Profesor: Ruben Rivas Medina

Informe de la PC4

- Benjamin Diaz
- Gabriel Spranger
- Rodrigo Céspedes

2019-1

INTRODUCCIÓN

Para esta oportunidad, se nos pidió realizar un programa que haga multiplicaciones de matrices.

Las multiplicaciones se hicieron de dos maneras dentro de nuestro programa y son las siguientes:

-Síncrona
Se refiere a la realización de forma secuencial de todas las instrucciones y operaciones que requiera hacer nuestro programa. Es la manera mediante la cual se haría si no se tuviese algún tipo de optimización. Es decir, es la más lenta (para ciertos casos).

-Asíncrona
Se refiere a la realización de forma paralela de las instrucciones y operaciones que requiera hacer nuestro programa. Para ello, es necesario lanzar hilos, los cuales ejecutarán una determinada función en paralelo, en teoría debería demorar menos que la manera síncrona (para ciertos casos).

OBJETIVO

- Aprender cómo se compara el performance de la multiplicación de matrices de forma síncrona y asíncrona. Además, saber en qué casos cuál de los dos algoritmos es mejor.

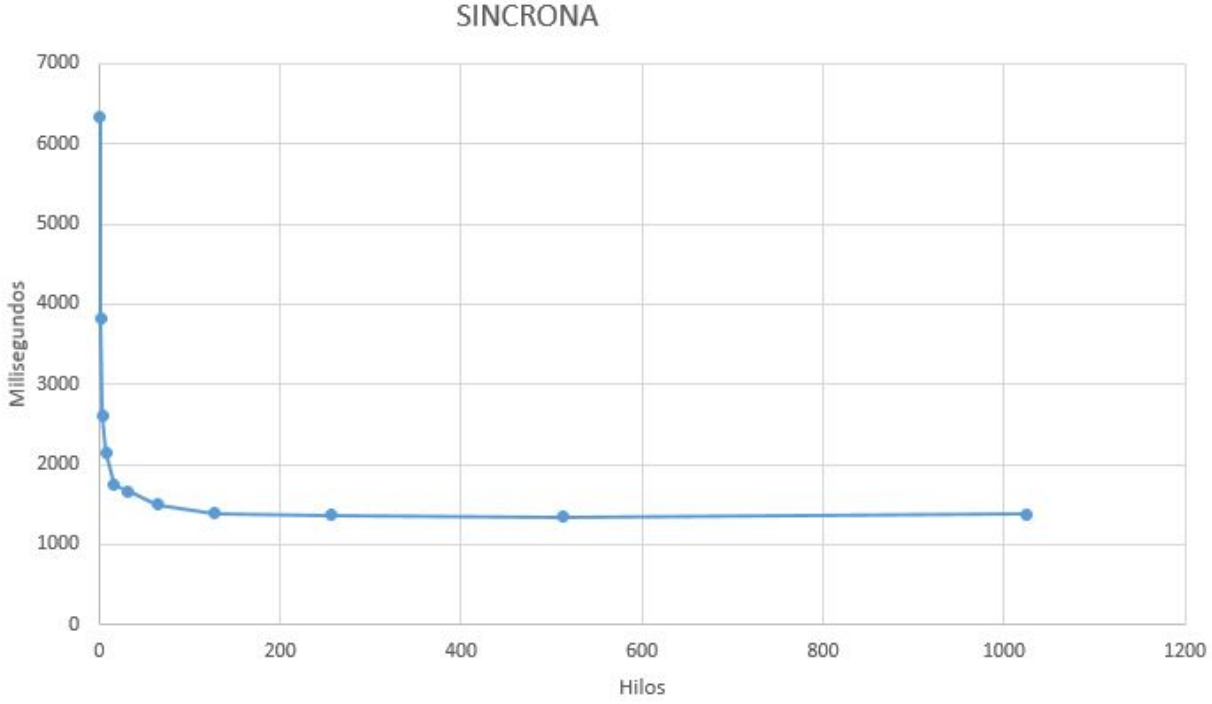
PROCEDIMIENTO

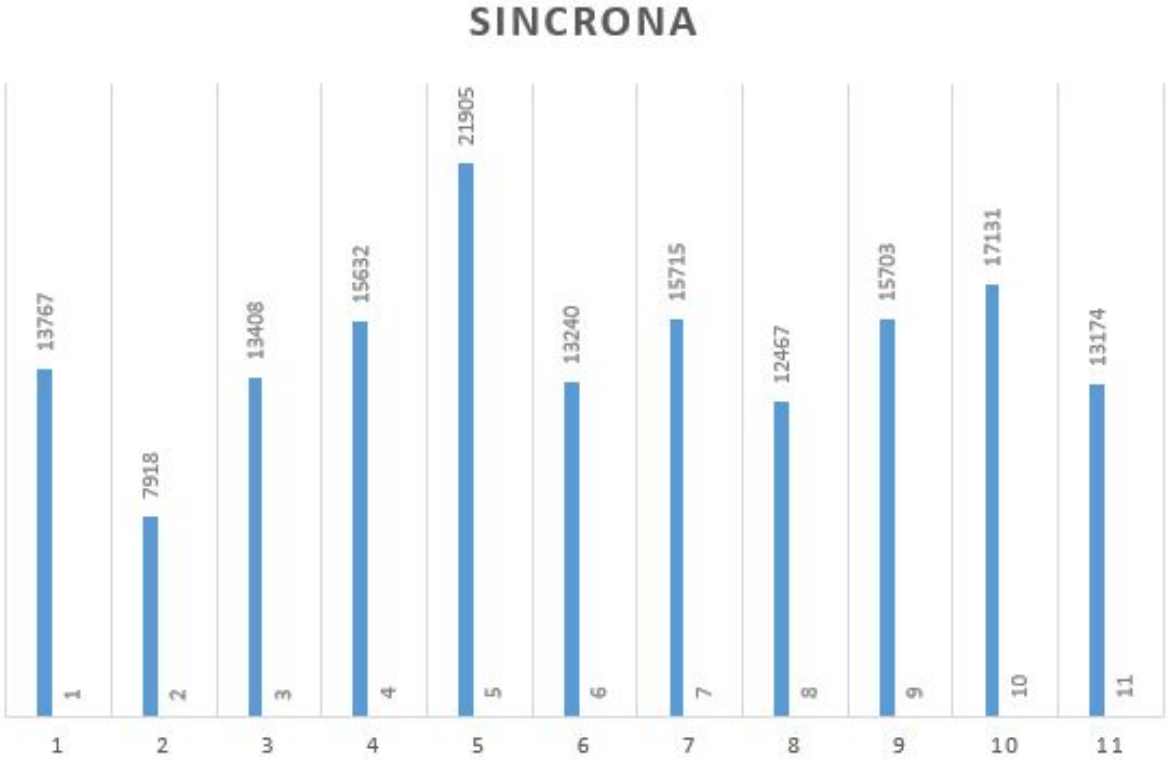
1. Se realizó un programa para multiplicar matrices en C++ de dos formas: una concurrente y la otra secuencial.
2. Se probó multiplicar dos matrices de números enteros de dimensiones 1024x1024 de forma síncrona y se tomaron tiempos de ejecución.
3. Se probó multiplicar dos matrices de números enteros de dimensiones 1024x1024 de forma asíncrona y se tomaron tiempos de ejecución.
4. Se analizaron los resultados y se sacaron conclusiones.

RESULTADOS Y GRÁFICOS

Hilos	Asincrona
1	6347 milisegundos
2	3827 milisegundos
4	2613 milisegundos
8	2149 milisegundos
16	1745 milisegundos
32	1662 milisegundos
64	1499 milisegundos
128	1396 milisegundos
256	1366 milisegundos
512	1352 milisegundos
1024	1380 milisegundos

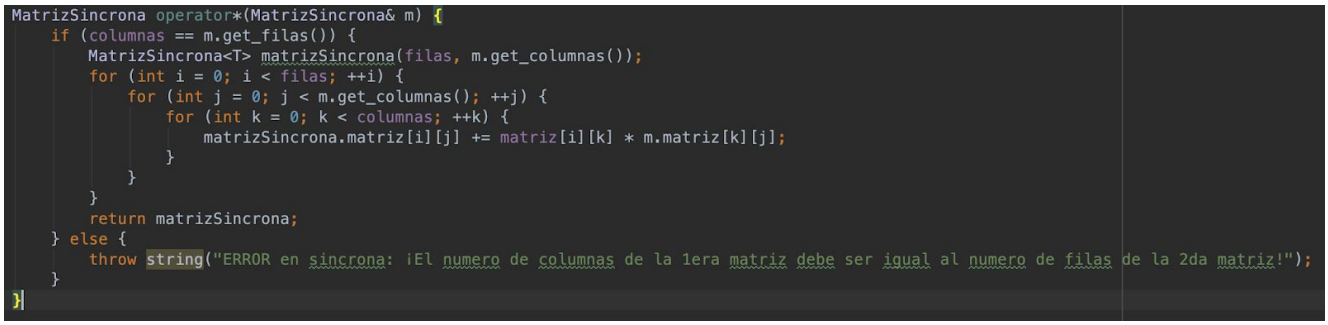
Hilos	Sincrona
1	13767 milisegundos
1	7918 milisegundos
1	13408 milisegundos
1	15632 milisegundos
1	21905 milisegundos
1	13240 milisegundos
1	15715 milisegundos
1	12467 milisegundos
1	15703 milisegundos
1	17131 milisegundos
1	13174 milisegundos


SINCRONA
Milisegundos
Hilos
7000
6000
5000
4000
3000
2000
1000
0
0
200
400
600
800
1000
1200


SINCRONA
13767
7918
13408
15632
21905
13240
15715
12467
15703
17131
13174
1
2
3
4
5
6
7
8
9
10
11

ALGORITMO

Síncrono: Este consta de sumar la multiplicación de cada elemento de cada fila de la 1era matriz, por cada elemento de cada columna de la 2da matriz, hasta que ya no queden filas ni columnas por multiplicar. En otras palabras, el algoritmo multiplicación matricial común y corriente. En cuanto a la implementación en la clase *MatrizSincrona*, se comprueba primero que el número de columnas de la 1era matriz sean igual al número de filas de la 2da matriz. Caso contrario, se lanzaba una excepción.



```
MatrizSincrona operator*(MatrizSincrona& m) {
    if (columnas == m.get_filas()) {
        MatrizSincrona<T> matrizSincrona(filas, m.get_columnas());
        for (int i = 0; i < filas; ++i) {
            for (int j = 0; j < m.get_columnas(); ++j) {
                for (int k = 0; k < columnas; ++k) {
                    matrizSincrona.matriz[i][j] += matriz[i][k] * m.matriz[k][j];
                }
            }
        }
        return matrizSincrona;
    } else {
        throw string("ERROR en sincrona: ¡El numero de columnas de la 1era matriz debe ser igual al numero de filas de la 2da matriz!");
    }
}
```

Asíncrono: Este es similar al anterior, solo se le añade una cosa: dependiendo de la cantidad de hilos, se le asigna a cada uno un pedazo de la multiplicación matricial. Es decir, cada uno hace una parte de la multiplicación. Obviamente, la multiplicación matricial solo depende de la fila y columna que se están operando, no en filas ni columnas anteriores. Por ello, se puede paralelizar sin que genere ningún conflicto; no hay necesidad de usar mutex.

```
void multiplicar_matrices(unsigned int lim_inf, unsigned int lim_sup, MatrizAsincrona<T>& matrizAsincrona1, MatrizAsincrona<T>& matrizAsincrona2, MatrizAsincrona<T>& matrizAsincrona3) {
    T r;
    for (int i = lim_inf; i < lim_sup; ++i) {
        for (int j = 0; j < matrizAsincrona1.get_columnas(); ++j) {
            r = matrizAsincrona1.matriz[i][j];
            for (int k = 0; k < matrizAsincrona2.get_columnas(); ++k) {
                matrizAsincrona3.matriz[i][k] += r*matrizAsincrona2.matriz[j][k];
            }
        }
    }
}

MatrizAsincrona operator*(MatrizAsincrona& m) {
    if (columnas == m.get_filas()) {
        MatrizAsincrona matrizAsincrona(filas, m.get_columnas());
        vector<thread> hilos;
        unsigned int lim_inf, lim_sup;
        for (int i = 0; i < NUM_HILOS; ++i) {
            lim_inf = i*(filas/NUM_HILOS);
            lim_sup = (i+1)*(filas/NUM_HILOS);                                  // ref(matriz)
            hilos.emplace_back(&MatrizAsincrona::multiplicar_matrices, this, lim_inf, lim_sup, ref(*this), ref(m), ref(matrizAsincrona));
        }
        for (int i = 0; i < NUM_HILOS; ++i) {
            hilos[i].join();
        }
        return matrizAsincrona;
    } else {
        throw string("ERROR en asincrona: ¡El numero de columnas de la 1era matriz debe ser igual al numero de filas de la 2da matriz!");
    }
}
```

INTERPRETACIÓN DE RESULTADOS

El algoritmo síncrono se demora menos que el asíncrono hasta que la matriz llega a, aproximadamente, una dimensión de 130x130. De ahí en adelante, el algoritmo asíncrono le gana de lejos al síncrono.

Tomando como matriz de análisis una de 1024x1024, el algoritmo síncrono, nos daba resultados distintos cada vez, pero constantes en el sentido que siempre se demoraba más que el asíncrono.

En cuanto a la cantidad de hilos, el performance sube drásticamente al usar una cantidad e hilos en la vecindad de 16. Luego, cuando se usa una gran cantidad de hilos, así como 512 o 1024, se puede notar que se llega a un umbral, es decir, que el performance ya no puede mejorar más: se mantiene constante. La función hilos vs tiempo se asemeja a $f(x) = -\log(x)$.

CONCLUSIONES

-Si se quiere multiplicar dos matrices de máximo 130x130, entonces usar el algoritmo síncrono.

-Si se quiere multiplicar dos matrices de más de 130x130, entonces usar definitivamente el algoritmo asíncrono. Sin embargo, cabe resaltar que el número de hilos que se usan para este algoritmo ya no afecta el performance desde cierto umbral, el cual es aproximadamente 256 hilos.

-El ***sweet spot*** para los hilos en relación al tiempo de ejecución está alrededor de 256 hilos, usar más de esa cantidad, en este caso, podría resultar en cambios no tan drásticos en el performance (aumentar la cantidad de hilos es como acercarse más al umbral ya mencionado).

-NOTA: El destructor en nuestro programa, el cual libera la memoria alocada dinámicamente, está comentado. Esto se debe a que cuando lo descomentamos, nos da un error que dice que un puntero está siendo liberado antes de ser creado, lo cual nos parece raro. Hemos tratado de solucionarlo, pero sin éxito.