LABORATORIO: Comparación de desempeño - Bucles anidados

Uberto García

I. Implementación

Recorrer dos bucles de dos formas distintas puede tener un impacto muy grande a la hora de compilar, un ejemplo de esto es la implementación del siguiente código:

```
#include <iostream>
#include <ctime>
using namespace std;
int main()
{
    int MAX = 500;
    double A[MAX][MAX];
    double x[MAX], y[MAX]= {0};
    /* Inicializando "A" y "x", ↩
       asignando y = 0 */
    for (int j = 0; j < MAX; j++){
        for (int i = 0; i < MAX; i++){
            A[j][i] = 0;
        }
        x[j] = 1;
    }
    unsigned i1, f1, i2, f2;
    /*Primer Bucle Anidado*/
    i1 = clock();
    for (int i = 0; i < MAX; i++)
        for (int j = 0; j < MAX; j++)
            y[i] += A[i][j]*x[j];
    f1 = clock();
    /* Asignamiento de y = 0 */
    for(int m = 0; m < MAX; m++)
        y[m] = 0;
    /*Segundo Bucle Anidado*/
    i2 = clock();
    for (int j = 0; j < MAX; j++)
        for (int i = 0; i < MAX; i++)
            y[i] += A[i][j]*x[j];
    f2 = clock();
    double time1 = (double(f1-i1)/↩
       CLOCKS_PER_SEC);
    printf("Bucle 1: %f segundos\n", ↩
       time1);
    double time2 = (double(f2-i2)/↩
       CLOCKS_PER_SEC);
    printf("Bucle 2: %f segundos\n", ↩
       time2);
    return 0;
}
```

Como se puede observar se recorre 2 veces “A”, se tomará como indicador de fila el primer parámetro y como indicador de columna el segundo parámetro. La primera vez se hace según las filas y luego las columnas, la segunda vez se hace el proceso viceversa, comenzando con columnas y luego filas. Al final se imprime el tiempo que tomo cada uno. Por otro lado, la variable “MAX” determina la cantidad de filas y columnas.

II. Comparación

Al inicio, con valores menores asignándose a “MAX”, la diferencia no existe o es insignificante, pero mientras más alto sea, se empiezan a notar las diferencias en su tiempo de ejecución.

Pronto se ve que el tiempo del segundo bucle puede llegar a ser 2 veces o incluso 3 veces más lento que el primer bucle, y esto se debe a como se ingresa y como está construido la matriz.

Cada fila es un Array por lo que “A[i]” es la fila “i”. Cuando se llama a “A[i][j]”, la memoria caché guarda en su interior al array “A[i]” y a partir de este accede al elemento en su posición “j”.

Esto ocurre en el primer bucle, ya que el primer “for” itera hacia otra fila solo si ya se ha terminado de recorrer todos los elementos de esa fila, lo que permite una eficiente forma de acceso.

Sin embargo, en el segundo bucle se accede de forma contraria, ya que el primer “for” itera hacia otra columna solo hasta que todos los elementos con ese número de columnas en todas las filas se hayan recorrido, lo que hace que se tenga que llamar a los arrays de las filas a la memoria caché una y otra vez.

III. Conclusion

La conclusión es que el segundo bucle demora más que el primero debido a su forma de acceder a la Matriz “A”, teniendo que cargar en el caché un array de fila en cada momento que llame a un elemento “A[i][j]”, lo que en arrays y matrices de pequeños tamaños no tiene mucho impacto, pero cuando se hacen más grandes se ve claramente debido a que la memoria caché se ha llenado y requiere reemplazar algunos datos, lo que la obliga a volver a llamar a ese array cuando otra vez se requiera un elemento de esa fila.

Click aquí para ir al repositorio de Github