Actividad 3: Aplicaciones de arreglos

EDyA 361075 Sánchez Torres, Carlos Eduardo

16 de febrero de 2021

1. Descripción de la actividad

Los arreglos son una estructura interna, estática de datos. Son indispensables para la solución de una gran diversidad de aplicaciones. Considerando que posees los conocimientos básicos del concepto de arreglos, elige un problema simple o clásico e implementa su solución en un programa computacional.

2. Descripción del problema elegido

El objetivo del presente trabajo es el desarrollo del algoritmo Strassen $\mathcal{O}(n^{2,8074})$ [1] y compararlo con el algoritmo estándar $\mathcal{O}(n^3)$. Aunque existen otros algoritmos como Coppersmith–Winograd [2] o Le Gall [3], con eficiencia $\mathcal{O}(n^{2,375477})$ y $\mathcal{O}(n^{2,3728639})$ respectivamente. No se implementaran por falta de conocimientos en las matemáticas expuestas en dichos trabajos.

2.1. Algoritmo estándar

El algoritmo estándar (figura 1) para multiplicar matrices: Si $A_{m\times n} = (a_{ij})$ y $B_{n\times p} = (b_{ij})$, entonces el producto $C = AB$, podemos definirlo como la entrada c_{ij} para cada $i, j = 1, 2, ..., n$ por

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \tag{1}$$

Donde debemos computar n^2 entradas y cada una es la suma n elementos.

2.2. Un algoritmo divide y vencerás

Asumamos que $C = A_{n\times n}B_{n\times n}$, donde $n \geq 2$ y es una potencia de 2, garantizando $n/2$ pertenezca a los enteros. Entonces, partimos cada matriz $n \times n$ en submatrices $n/2 \times n/2$, reescribiendo:

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \tag{2}$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{12}B_{12} + A_{22}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix} \tag{3}$$

Cada bloque se obtiene recursivamente sobre (2) (ver figura 2) hasta llegar al producto escalar:

$$C_{11} = A_{11}B_{11} + A_{12}B_{21} \tag{4}$$

$$C_{12} = A_{12}B_{12} + A_{22}B_{22} \tag{5}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21} \tag{6}$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22} \tag{7}$$

Detalles en la implementación para partir las matrices: copiar las matrices en submatrices o calcular los índices sobre la matriz original. Se opto por la primera debido a las facilidades con numpy.

2.3. Algoritmo de Strassen

En lugar de sumar recursivamente 8, Strassen creo un algoritmo de 7 productos recursivos y preferencia sobre las sumas (Figura 3).

Si

$$P_1 = A_{11}B_{12} - A_{11}B_{22} \tag{8}$$

$$P_2 = A_{11}B_{22} + A_{12}B_{22} \tag{9}$$

$$P_3 = A_{21}B_{11} + A_{22}B_{11} \tag{10}$$

$$P_4 = A_{22}B_{21} - A_{22}B_{11} \tag{11}$$

$$P_5 = A_{11}B_{11} + A_{11}B_{22} + A_{22}B_{11} + A_{22}B_{22} \tag{12}$$

$$P_6 = A_{12}B_{21} + A_{12}B_{22} - A_{22}B_{21} - A_{22}B_{22} \tag{13}$$

$$P_7 = A_{11}B_{11} + A_{11}B_{12} - A_{21}B_{11} - A_{21}B_{12} \tag{14}$$

Entonces,

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{12}B_{12} + A_{22}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix} = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_5 + P_1 - P_3 - P_7 \end{bmatrix} \tag{15}$$

La demostración es simple: cumplir las sentencias según las operaciones definidas para las matrices. Strassen no explica como llego a la solución, se limita a demostrarlo y en el fondo, no importa.

3. Resultados

El algoritmo estándar es ineficiente, sobre todo cuando su entrada son grandes volúmenes de datos (véase «Big Data»), sus tres for anidados (los primeros dos por cantidad de entradas n^2 y el último por el producto punto n) provocan $\mathcal{O}(n^3)$.

El algoritmo recursivo es igual de ineficiente que algoritmo estándar, porque el método maestro $T(n) = 8T(n/2) + \theta(n^2)$ (8 sumas recursivas y las adiciones son n^2) resulta en $\mathcal{O}(n^{log_2(8)}) = \mathcal{O}(n^3)$.

El algoritmo Strassen es más eficiente que algoritmo estándar, porque el método maestro $T(n) = 7T(n/2) + \theta(n^2)$ (7 sumas recursivas y las adiciones son n^2) resulta en $\mathcal{O}(n^{log_2(7)}) = \mathcal{O}(n^{2,80735492206})$. Este algoritmo fue el más rápido hasta el 2010 y creado por Volker Strassen [4].

Al comparar el algoritmo estándar y el de Strassen, en valores pequeños ($n \approx 10000$) no se observan diferencias. La ventaja de estos algoritmos es entorno a los millones.

4. Anexo

```cpp
Matrix<T> operator*(const Matrix &rhs)const{
    if(this->COL != rhs.ROW){
        throw "MATRIX MULTIPLICATION CANNOT HAPPEN WITH THE GIVEN MATRICES";
    }
    Matrix<T> result(this->ROW,rhs.COL, 0);
    for(int i = 0; i < this->ROW; i++){
        for(int j = 0; j < rhs.COL; j++){
            for(int k = 0; k < this->COL; k++){
                result.matrix[i][j] += (this->matrix[i][k]*rhs.matrix[k][j]);
            }
        }
    }
    return result;
}
```

Figura 1: Algoritmo estándar

```python
def square_matrix_multiply_recursive(A, B):
    if len(A) == 1:
        return A * B
    a11, a12, a21, a22 = split(A)
    b11, b12, b21, b22 = split(B)
    c11 = square_matrix_multiply_recursive(a11, b11) + square_matrix_multiply_recursive(a12, b21)
    c12 = square_matrix_multiply_recursive(a11, b12) + square_matrix_multiply_recursive(a12, b22)
    c21 = square_matrix_multiply_recursive(a21, b11) + square_matrix_multiply_recursive(a22, b12)
    c22 = square_matrix_multiply_recursive(a21, b12) + square_matrix_multiply_recursive(a22, b22)
    return np.vstack((np.hstack((c11, c12)), np.hstack((c21, c22))))
```

Figura 2: Algoritmo recursivo

```python
def strassen(A, B):
    if len(A) == 1:
        return A * B
    a11, a12, a21, a22 = split(A)
    b11, b12, b21, b22 = split(B)
    p1 = strassen(a11, b12 - b22)
    p2 = strassen(a11 + a12, b22)
    p3 = strassen(a21 + a22, b11)
    p4 = strassen(a22, b21 - b11)
    p5 = strassen(a11 + a22, b11 + b22)
    p6 = strassen(a12 - a22, b21 + b22)
    p7 = strassen(a11 - a21, b11 + b12)
    c11 = p5 + p4 - p2 + p6
    c12 = p1 + p2
    c21 = p3 + p4
    c22 = p5 + p1 - p3 - p7
    return compress(c11, c12, c21, c22)
```

Figura 3: Algoritmo Strassen

Referencias

[1] Cormen, T. H., &; Leiserson, C. E. (2009). Introduction to Algorithms, 3rd edition. In Introduction to algorithms, 3rd edition (pp. 75-82).

[2] Coppersmith, Don; Winograd, Shmuel (1990), ”Matrix multiplication via arithmetic progressions”(PDF), Journal of Symbolic Computation, 9 (3): 251, doi:10.1016/S0747-7171(08)80013-2

[3] Le Gall, François (2014), ”Powers of tensors and fast matrix multiplication”, Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC 2014), arXiv:1401.7714, Bibcode:2014arXiv1401.7714L

[4] Strassen, Volker, Gaussian Elimination is not Optimal, Numer. Math. 13, p. 354-356, 1969