Análisis de Algoritmos de Multiplicación de Matrices

Jean Paul Ari Maldonado

August 28, 2024

Abstract

Este informe presenta la implementación, análisis y comparación del rendimiento de diferentes algoritmos para la multiplicación de matrices: la multiplicación clásica con tres bucles anidados, la multiplicación por filas y columnas, y la multiplicación por bloques con seis bucles anidados. El código se implementó en C++ y se realizaron pruebas para medir el tiempo de ejecución de cada algoritmo con diferentes tamaños de matrices. Enlace del repositorio

1 Introducción

La multiplicación de matrices es una operación fundamental en muchos campos, como álgebra lineal, gráficos computacionales, inteligencia artificial, y más. Este informe examina varios métodos de multiplicación de matrices, evaluando su rendimiento en términos de tiempo de ejecución.

2 Implementación

2.1 Multiplicación de Matrices Clásica

El método clásico de multiplicación de matrices utiliza tres bucles anidados para calcular el producto de dos matrices. El siguiente código muestra la implementación en C++:

Listing 1: Multiplicación Clásica en C++

```cpp
#include <iostream>
#include <chrono>
#include <vector>
#include <algorithm>

using namespace std;

const int N = 200;

int main() {
    vector<vector<int>> A(N, vector<int>(N,
        1));
    vector<vector<int>> B(N, vector<int>(N,
        1));
    vector<vector<int>> res(N, vector<int>(N
        , 0));

    auto multMatrizClasica = [&]() {
        auto start = chrono::
            high_resolution_clock::now();
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                for (int k = 0; k < N; k++)
                    {
                    res[i][j] += A[i][k] * B
                        [k][j];
                }
            }
        }
        auto end = chrono::
            high_resolution_clock::now();
        return chrono::duration<double>(end
            - start).count();
    };

    double duration1 = multMatrizClasica();
    cout << "Tiempo de multiplicaci n
        cl sica: " << duration1 << "
        segundos" << endl;

    return 0;
}
```

2.2 Multiplicación de Matrices por Bloques

Este método optimiza el acceso a la memoria utilizando bloques pequeños de las matrices. Esto mejora la utilización de la memoria caché y puede reducir significativamente el tiempo de ejecución para matrices grandes.

Listing 2: Multiplicación por Bloques en C++

```
#include <iostream>
#include <chrono>
#include <vector>
#include <algorithm>

using namespace std;

const int N = 200;
const int S = 20;

int main() {
    vector<vector<int>> A(N, vector<int>(N,
        1));
    vector<vector<int>> B(N, vector<int>(N,
        1));
    vector<vector<int>> res(N, vector<int>(N
        , 0));

    auto multMatrizBloque = [&]() {
        auto start = chrono::
            high_resolution_clock::now();
        for (int ii = 0; ii < N; ii += S) {
            for (int jj = 0; jj < N; jj += S
                ) {
                for (int kk = 0; kk < N; kk
                    += S) {
                    for (int i = ii; i < min
                        (ii + S, N); ++i) {
                        for (int j = jj; j <
                             min(jj + S, N);
                            ++j) {
                            for (int k = kk;
                                 k < min(kk +
                                 S, N); ++k)
                                {
                                res[i][j] +=
                                     A[i][k]
                                    * B[k][j
                                    ];
                            }
                        }
                    }
                }
            }
        }
        auto end = chrono::
            high_resolution_clock::now();
        return chrono::duration<double>(end
            - start).count();
    };

    double duration2 = multMatrizBloque();
    cout << "Tiempo de multiplicaci n por
        bloques: " << duration2 << " segundos
        " << endl;

    return 0;
}
```

2.3 Multiplicación con Dimensiones Variables

Se realizaron pruebas de rendimiento adicionales con tamaños de matriz variables para ambos métodos de multiplicación (clásica y por bloques) para comparar su eficiencia en diferentes escalas.

Listing 3: Multiplicación con Dimensiones Variables en C++

```
#include <iostream>
#include <chrono>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
    int Ns[] = {100, 200, 250, 500, 1000};

    cout << "Tama o matriz\tMultClasica\
        tMultBloque" << endl;

    for (int N : Ns) {
        vector<vector<int>> A(N, vector<int
            >(N, 1));
        vector<vector<int>> B(N, vector<int
            >(N, 1));
        vector<vector<int>> res(N, vector<
            int>(N, 0));

        auto multMatrizClasica = [&]() {
            auto start = chrono::
                high_resolution_clock::now();
            for (int i = 0; i < N; i++) {
                for (int j = 0; j < N; j++)
                    {
                    for (int k = 0; k < N; k
                        ++) {
```

```
                    res[i][j] += A[i][k]
                         * B[k][j];
                }
            }
        }
        auto end = chrono::
            high_resolution_clock::now();
        return chrono::duration<double>(
            end - start).count();
    };

    auto multMatrizBloque = [&]() {
        int S = 20;
        auto start = chrono::
            high_resolution_clock::now();
        for (int ii = 0; ii < N; ii += S
            ) {
            for (int jj = 0; jj < N; jj
                += S) {
                for (int kk = 0; kk < N;
                     kk += S) {
                    for (int i = ii; i <
                         min(ii + S, N);
                        ++i) {
                        for (int j = jj;
                             j < min(jj +
                             S, N); ++j)
                            {
                            for (int k =
                                 kk; k <
                                min(kk +
                                S, N); ++
                                k) {
                                res[i][j
                                    ] +=
                                    A[i][
                                    k] *
                                    B[k][
                                    j];
                            }
                        }
                    }
                }
            }
        }
        auto end = chrono::
            high_resolution_clock::now();
        return chrono::duration<double>(
            end - start).count();
    };

    double duration1 = multMatrizClasica
        ();
    for (auto& row : res) {
        fill(row.begin(), row.end(), 0);
    }
    double duration2 = multMatrizBloque
        ();

    cout << N << "\t\t" << duration1 <<
        "\t\t" << duration2 << endl;
}

    return 0;
}
```

3 Análisis y Resultados

3.1 Análisis de Rendimiento

Los resultados mostraron que el método de multiplicación por bloques es más eficiente en términos de tiempo de ejecución, especialmente para matrices grandes. Esto se debe a una mejor utilización de la memoria caché, ya que los bloques más pequeños permiten un acceso más eficiente a la memoria.

3.2 Resultados de la Ejecución

A continuación se presentan los resultados del tiempo de ejecución de ambos métodos para diferentes tamaños de matriz. Como se observa, el método por bloques ofrece un rendimiento significativamente mejor para matrices más grandes.

Tamaño Matriz	Mult. Clásica (s)	Mult. Filas/Columna
100	0.005	0.004
200	0.040	0.030
250	0.075	0.055
500	0.600	0.450
1000	5.000	4.200

Table 1: Comparación de Tiempos de Ejecución para Diferentes Tamaños de Matrices

4 Conclusiones

El método de multiplicación por bloques es superior en términos de rendimiento para tamaños de matrices más grandes debido a la optimización del uso de memoria caché. El análisis sugiere que para aplicaciones que involucran matrices grandes, este método

debería ser preferido sobre los métodos clásicos de tres bucles anidados.