Programa de Formação

Itaú Analytics

Rômulo Madureira Rodrigues

TAREFA 01 - Multiplicação de Matrizes

Prof. Marcelo Xavier Guterres

SÃO PAULO

2018

0.1 Introdução

A multiplicação de matrizes é uma das operações básicas da Álgebra Linear e sua implementação computacional é utilizada em várias aplicações importantes da ciência da computação, como inteligência artificial e processamento gráfico. Conforme a ordem das matrizes a serem multiplicadas cresce, mais recurso computacional é necessário para que a operação seja efetuada em um tempo aceitável, porém, com a limitação de recursos computacionais e as exigências de tempo de execução das aplicações, otimizações deste algoritmo são necessárias em alternativa ao método tradicional, chamado de *naive*, que tem complexidade $\mathcal{O}(n^3)$.

Na literatura existem algumas outras versões que tornam a complexidade inferior à do algoritmo *naive*, porém, que ainda exigem bastante dos recursos computacionais, dada essa limitação, soluções computacionais como o paralelismo das operações tem sido adotadas para aumentar o desempenho das aplicações.

O objetivo desse trabalho é implementar o algoritmo tradicional na linguagem Python e detalhar os algoritmos mais importantes conehcidos de multiplicação de matrizes.

0.2 Algoritmo Tradicional em Python

Na Álgebra Linear, a multiplicação de matrizes pode ser escrita conforme a Equação 1. O algoritmo tradicional, Algoritmo 1, implementa computacionalmente de forma direta o cálculo dessa equação.

$$C_{i,j} = \sum_k A_{i,k} B_{k,j} \tag{1}$$

Algoritmo 1: MULTIPLICADOR DE MATRIZES(A, B)

se n^o *colunas*$[A] \neq n^o$ *linhas*$[B]$ **então**
  devolva erro "Dimensões incompatíveis
senão
  para $i \leftarrow 1$ ***até*** n^o *linhas*$[A]$ **faça**
    para $j \leftarrow 1$ ***até*** n^o *linhas*$[B]$ **faça**
      $C[i,j] \leftarrow 0$ **para** $j \leftarrow 1$ ***até*** n^o *linhas*$[B]$ **faça**
        $C[i,j] \leftarrow C[i,j] + A[i,j] * B[i,j]$
      fim
    fim
  fim
  devolva C
fim

Adaptando o algoritmo tradicional na linguagem python foi escrito o seguinte trecho de código.

```
def zeros(m_rows,n_columns):
        '''
```

```
        Create m by n list matrix of zeros.
        '''
        return [[0 for j in range(n_columns)] for i in range(m_rows)]

def naivemul(A,B):
        '''
        Multiply to list matrix using naive method.
        '''

    # Get matrix dimensions
        m_rowsA = len(A)
        m_rowsB = len(B)
        n_columnsA = len(A[0])
        n_columnsB = len(B[0])

    # Check multiplication requirement
        if n_columnsA != m_rowsB:
                raise ValueError('Incompatible dimensions.')
        else:

        # Create output matrix filled with zeros
                C = zeros(m_rowsA, n_columnsB)

        # Fill output matrix with multiplication results
                for i in range(m_rowsA):
                        for j in range(n_columnsB):
                                for k in range(n_columnsA):
                                        C[i][j] += A[i][k]*B[k][j]
        return C
```

Neste trecho de código foram utilizadas apenas as bibliotecas nativas da linguagem, também foi criada uma função adicional para a construção da matriz de zeros, necessária para a criação da matriz resultado da equação.

0.3 Principais algoritmos de multiplicação de matrizes

O algoritmo tradicional apresenta uma solução direta do problema de multiplicação de matrizes, porém, a presença de 3 iterações encapsuladas torna a complexidade do algoritmo $\mathcal{O}(n^3)$ fazendo o cálculo bastante oneroso para matrizes de alta ordem. Existe uma limitação natural na redução da complexidade, uma vez que matrizes possuem duas dimensões, nenhuma solução direta pode ter complexidade inferior a $\mathcal{O}(n^2)$, ao longo dos anos algumas soluções tem sido propostas que reduzem a complexidade em relação ao método natural, Com destaque para os métodos de Strassen e Coppersmith-Winograd.

0.3.1 Método de Strassen

Em 1969, o matemático Volker Strassen propôs uma variante do algoritmo *dividir para conquistar*, que consiste em subdividir as matrizes em quatro matrizes quatro outras matrizes cada e fazer a multiplicação recursiva, reduzindo o número de multiplicações necessárias utilizando de cálculos já realizados, o procedimento que pode ser explicitado da seguinte forma [1]:

Sendo $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C} \in \mathcal{R}^{2^n \times 2^n}$, dado que:

$$C = AB \tag{2}$$

Particiona-se $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ da seguinte forma:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}, \tag{3}$$

Strassen propõe a seguinte cadeia de operações para a solução da Equação 2. Definem-se as matrizes.

$$\begin{aligned}
\mathbf{M}_1 &= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\
\mathbf{M}_2 &= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\
\mathbf{M}_3 &= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\
\mathbf{M}_4 &= \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\
\mathbf{M}_5 &= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\
\mathbf{M}_6 &= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\
\mathbf{M}_7 &= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})
\end{aligned} \tag{4}$$

O cálculo de $\mathbf{C}$ se reduz a:

$$\begin{aligned}
\mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 + \mathbf{M}_5 + \mathbf{M}_7 \\
\mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\
\mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\
\mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6
\end{aligned} \tag{5}$$

Este algoritmo possui complexidade assintótica de $\mathcal{O}(n^{2,807})$, porém, para matrizes de baixa ordem ele tem um desempenho inferior ao algoritmo tradicional devido ao seu uso de memória inicial ser maior nesses casos.

0.3.2 Método de Coppersmith-Winograd

Em 1987, David Coppersmith e Terry Winograd criaram uma variante do algoritmo de Strassen que realiza eliminações de operações via progressões aritméticas, trazendo uma substancial melhoria em tempo de execução. O arranjo matemático é descrito em [2], diminuindo a complexidade para o patamar de $\mathcal{O}(n^{2,376})$. Esse patamar não foi superado por cerca de 20 anos, atualmente alguns refinamentos do método de Coppersmith-Winograd obtém soluções com complexidade ligeiramente menor, como por exemplo o obtido por [3], alcançando $\mathcal{O}(n^{2,373})$.

0.3.3 Paralelismo

Desde a solução de Coppersmith-Winograd, a evolução na redução complexidade na multiplicação de matrizes tem sido bastante incipiente, com o advento de técnicas de processamento paralelo, tem-se buscado a redução do tempo de execução desses algoritmos. Por exemplo, [4] realiza as operações de multiplicação das submatrizes geradas pelo método de Strassen em *multithreads*, porém o desempelnho é comprometido devido a limitação de memória RAM da máquina que deixa a comunicação dos resultados lenta.

Os mais modernos desenvolvimentos tem conseguido bons resultados em tempo de execução utilizado o advento da computação distribuída e técnicas de *MapReduce* [5], dada a fácil disponibilidade de recursos computacionais distribuídos atualmente.

0.3.4 Conclusão

Essa tarefa proporcionou a compreensão do esforço computacional necessário para realizar multiplicações de matrizes, começando pelo algoritmo tradicional até a solução de Coppersmith-Winograd, que foi a ultima grande evolução na resolução deste problema. Enquanto não há evoluções significativas na teoria, na prática, o processamento paralelo tem obtido ganhos em tempo de execução, melhorando a eficiência das aplicações.

Referências Bibliográficas

[1] M. Anderson and S. Barman, "The coppersmith-winograd matrix multiplication algorithm." ICT, 2009.

[2] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM, 1987.

[3] V. V. Williams, "Multiplying matrices in $\mathcal{O}(n^{2,373})$ time." Stanford University, 2014.

[4] K. H. Randall, "Cilk: Ecient multithreaded computing." Massachussets Institute of Technology, 1998.

[5] R. B. Zadeh and G. Carlsson, "Dimension independent matrix square using mapreduce (dimsum)." Stanford University, 2014.