University of Campinas

Instituto de Matemática, Estatística e Computação Científica.

EP3 - Algoritmos Subcúbicos para Multiplicação Matricial.

MT404

Vinícius Oliveira Martins

November, 2023

1 Multiplicação Matricial

Matrizes e vetores são os dois elementos fundamentais da álgebra linear. Dada duas matrizes reais, $A \in \mathbb{R}^{m\times p}$ e $B \in \mathbb{R}^{p\times n}$, o produto $AB = C$ é tal que

$$c_{i,j} = \sum_{k=1}^{p} a_{i,k}b_{k,j} \ , \quad i = 1, ..., m \ , \quad j = 1, ..., n \ . \tag{1}$$

Assim, o algoritmo usual para calcularmos $C = AB$ é construir cada elemento $c_{i,j}$ usando a equação (1). Como sabemos que $C \in \mathbb{R}^{m\times n}$ é fácil ver que realizaremos mn somas do tipo (1). Assim, basta determinarmos quantas operações (adições e multiplicações) são feitas nas somas do tipo (1). Como o somatório é para $k = 1$ até $k = p$ teremos $p - 1$ adições e como cada fator da soma envolve uma multiplicação teremos p multiplicações. Logo, para realizarmos o produto $C = AB$ faremos $mn(p-1)$ adições e mnp multiplicações (ou $mn(2p - 1)$ operações).

Uma maneira mais interessante de expressar a complexidade temporal da multiplicação matricial é introduzindo a razão entre o custo computacional de uma multiplicação e de uma adição

$$\omega = \frac{c_p}{c_a} \ , \tag{2}$$

onde c_p é o custo de uma multiplicação e c_a é o custo de uma adição. Assim, podemos definir a complexidade temporal da multiplicação usual de duas matrizes como

$$g_u(m, n, p) = wmnp + mn(p - 1) = (1 + w)mnp - mn \ . \tag{3}$$

No caso em que A e B são matrizes quadradas $n \times n$

$$g_u(n) = (1 + w)n^3 - n^2 \ . \tag{4}$$

O valor de ω depende de cada processador, porém para processadores modernos sabemos que $\omega \approx 1$. Assim, o valor que calcularemos para esse parâmetro vale apenas para o processador utilizado para o desenvolvimento deste texto. Mais especificamente o processador utilizado foi um Intel Core i7-1065G7 @ 1.30GHz.

2 Algoritmo de Winograd

Uma tentativa para melhorar a complexidade temporal da multiplicação matricial é o Algoritmo de Winograd. Esse algoritmo se baseia na ideia de que o somatório (1) é na verdade o produto interno entre a linha i de A e a coluna j de B. Com isso (e algumas contas) temos que

$$c_{i,j} = \sum_{k=1}^{n/2}(a_{i,2k-1} + b_{2k,j})(a_{i,2k} + b_{2k-1,j}) - \sum_{k=1}^{n/2} a_{i,2k-1}a_{i,2k} - \sum_{k=1}^{n/2} b_{2k-1,j}b_{2k,j} \ . \tag{5}$$

A vantagem desse algoritmo é que o segundo somatório é calculado apenas n vezes (uma vez para cada linha de A) e o terceiro somatório também é calculado n vezes (uma vez para cada coluna de B).

Assim, para matrizes quadradas $n \times n$ calculamos $\frac{3+\omega}{2}n^3 - n^2$ vezes o primeiro somatório e $\frac{1+\omega}{2}n^2 - n$ vezes o segundo e o terceiro somatório de (5). Como ainda temos mais duas subtrações (que são interpretadas como adições) para cada elemento de C, temos que a complexidade temporal do algoritmo de Winograd é

$$g_w(n) = \frac{3+\omega}{2}n^3 + (2+\omega)n^2 - 2n \ , \tag{6}$$

que tem ordem $O(n^3)$. Vale notar também que para esse algoritmo devemos usar mais memória do processador que o algoritmo usual já que devemos salvar os valores do segundo e terceiro somatório, ou seja a complexidade espacial do algoritmo de Winograd é maior que a do algoritmo usual e para algumas aplicações isso pode ser um limitante no uso do algoritmo de Winograd.

Podemos usar esses dois algoritmo de multiplicação matricial para determinar o valor de ω de um processador. Para isso, devemos considerar o seguinte limite

$$\tau = \lim_{n\to\infty} \frac{g_w}{g_u} \ , \tag{7}$$

que podemos calcular

$$\tau = \lim_{n\to\infty} \frac{\frac{3+\omega}{2}n^3 + (2+\omega)n^2 - 2n}{(1+w)n^3 - n^2} \tag{8}$$

$$= \lim_{n\to\infty} \frac{\frac{3+\omega}{2} + (2+\omega)\frac{1}{n} - 2\frac{1}{n^2}}{(1+w) - \frac{1}{n}} \tag{9}$$

$$= \frac{3+w}{2+2w} \ . \tag{10}$$

Assim, se $\tau > 1$ temos que o algoritmo de Winograd é mais eficiente que o usual. Logo

$$\frac{3+\omega}{2+2\omega} > 1 \Rightarrow 3+\omega < 2+2\omega \Rightarrow \omega > 1 \ , \tag{11}$$

ou seja o algoritmo de Winograd é mais eficiente que o usual em um processador tal que $w > 1$. Essa condição é satisfeita por basicamente todo processador já que por construção a multiplicação é uma operação mais custosa que a adição.

Além disso, se calcularmos a razão entre o tempo de execução da multiplicação entre duas matrizes quaisquer $n \times n$ (n par) do algoritmo de Winograd e o algoritmo usual podemos determinar o valor do parâmetro ω para o processador que utilizamos. Para isso, basta fazermos τ como sendo a razão mencionada e calcularmos o valor de ω, mais precisamente

$$\omega = \frac{2\tau - 3}{1 - 2\tau} \ . \tag{12}$$

Essa expressão vale apenas para o limite em que $n \to \infty$. Para determinar qual valor n é suficientemente grande iremos começar supondo $n = 100$ e depois aumentaremos esse valor até que o valor de ω tenda para algum limite. Como não temos a possibilidade de executar esse método em um processador dedicado apenas para essa tarefa (o computador utilizado executa tarefas internas enquanto executa o método) calcularemos o valor τ para $E = 100$ matrizes aleatórias e consideramos a média para o cálculo de ω. Assim, na verdade a equação (12) se torna

$$\omega = \frac{2\bar{\tau} - 3}{1 - 2\bar{\tau}} \ , \tag{13}$$

sendo

$$\bar{\tau} = \frac{1}{100} \sum_{l=1}^{100} \tau_l \ , \tag{14}$$

onde τ_l é a razão τ calculada para o ensaio l.

Além disso, calculamos o erro nessa razão. Consideramos o erro padrão como medida de erro para esse método já que estamos interessados em determinar o valor de ω. Assim, sendo T_w o tempo de execução do algoritmo de Winograd e T_u o tempo de execução do algoritmo usual (ambas variáveis aleatórias), temos

$$e_{std}(T_w) = \frac{\sigma(T_w)}{\sqrt{E}} \ , \quad e_{std}(T_u) = \frac{\sigma(T_u)}{\sqrt{E}} \ , \tag{15}$$

os erros associados as medições de T_w e T_u. Fazendo propagação de incertezas, temos o erro de $\bar{\tau}$

$$e_{std}(\bar{\tau}) = \sqrt{\left(\frac{e_{std}(T_w)}{\mu(T_w)}\right)^2 + \left(\frac{e_{std}(T_u)\mu(T_w)}{\mu(T_u)^2}\right)^2} \ , \tag{16}$$

onde $\mu(T_w)$ é a média de T_w e $\mu(T_u)$ é a média de T_u. Com isso, temos também o erro de ω

$$e_{std}(\omega) = \frac{4}{(2\bar{\tau} - 1)^2} e_{std}(\bar{\tau}) \ . \tag{17}$$

As figuras a seguir contêm os resultados obtidos para $n = 100$, $n = 200$, $n = 300$ e $n = 400$.

Obs.: Devido a limitação de caracteres para a exposição dos resultados no terminal de comando temos a seguinte mudança de notação: $avr\_std \leftrightarrow \mu(T_u)$; $error\_std \leftrightarrow e_{std}(T_u)$; $avr\_wino \leftrightarrow \mu(T_w)$; $error\_wino \leftrightarrow e_{std}(T_w)$; $tau \leftrightarrow \bar{\tau}$; $error\_tau \leftrightarrow e_{std}(\bar{\tau})$; $w \leftrightarrow \omega$; $error\_w \leftrightarrow e_{std}(\omega)$.

```
avr_std = 0.35075794414000017 error_std = 0.008708527723913635
avr_wino = 0.33155486949999985 error_wino = 0.0003585297964673453


tau = 0.9452526308788728 error_tau = 0.02349073760427456
w = 1.245915982632434 error_w = 0.11849053631671272
```

Figura 1: ω para o algoritmo de Winograd. Matrizes de dimensão $n = 100$ e quantidade de ensaios $E = 100$

```
avr_std = 2.7552591275200013 error_std = 0.008882525457970037
avr_wino = 2.612646250239998 error_wino = 0.004063744272225032


tau = 0.9482397587016185 error_tau = 0.003394179119551987
w = 1.2309489075592555 error_w = 0.01689328099934478
```

Figura 2: ω para o algoritmo de Winograd. Matrizes de dimensão $n = 200$ e número de ensaios $E = 100$

```
avr_std = 9.529117331619998 error_std = 0.008482668503944512
avr_wino = 9.048088370800004 error_wino = 0.006748123597586293


tau = 0.9495200925668299 error_tau = 0.0011026929784317748
w = 1.2245946655906834 error_w = 0.00545703063815537
```

Figura 3: ω para o algoritmo de Winograd. Matrizes de dimensão $n = 300$ e número de ensaios $E = 100$

```
avr_std = 22.50231678012999 error_std = 0.028924789107956633
avr_wino = 21.36794667193999 error_wino = 0.026921626948133864


tau = 0.9495887414938682 error_tau = 0.0017091686016420114
w = 1.2242549861841652 error_w = 0.008455787330883396
```

Figura 4: ω para o algoritmo de Winograd. Matrizes de dimensão $n = 400$ e número de ensaios $E = 100$

Podemos ver que a diferença entre os valores de ω calculados para $n = 300$ e $n = 400$ apresentam uma diferença na quarta casa decimal e os erros foram da ordem de 10^{-3}. Assim, podemos considerar o valor de ω como sendo $1,22$ e portanto o algoritmo de Winograd é mais eficiente que o usual.

Considero que o algoritmo de Winograd não é vantajoso em situações práticas. Mesmo tendo uma complexidade menor que o algoritmo usual podemos observar que essa vantagem não é substancial já que o valor de $\bar{\tau}$ se manteve próximo a $0,95$, ou seja o algoritmo de Winograd possui um tempo de execução (em média) de 95% do tempo de execução do algoritmo usual. Além disso, como já foi comentado a complexidade espacial do algoritmo de Winograd é maior que a do algoritmo usual.

3 Algoritmo de Strassen-Winograd

Um outro algoritmo de multiplicação matricial é o Algoritmo de Strassen-Winograd. Esse algoritmo apresenta uma diminuição considerável na complexidade temporal em relação ao algoritmo usual e o algoritmo de Winograd. Esse terceiro algoritmo tem sua

ideia baseada no fato que a multiplicação de duas matrizes 2×2 pode ser calculada usando apenas 7 produtos (um produto a menos que o algoritmo usual) e 18 adições, esse é o algoritmo de Strassen. O algoritmo de Strassen-Winograd é um refinamento do algoritmo de Strassen que em vez de 18 são realizadas 15 adições. Mais precisamente, dadas duas matrizes A e B de dimensões 2×2 podemos calcular o produto $C = AB$ da seguinte forma

$$\begin{aligned} c_{11} &= m_1 + m_2 \ , \quad c_{12} = r_3 + m_6 \ , \\ c_{21} &= r_2 + m_7 \ , \quad c_{22} = r_2 + m_3 \ , \end{aligned} \tag{18}$$

onde

$$\begin{aligned} p_1 &= a_{21} + a_{22} \ , \quad q_1 = b_{12} - b_{11} \ , \\ p_2 &= p_1 - a_{11} \ , \quad q_2 = b_{22} - q_1 \ , \\ p_3 &= a_{11} - a_{21} \ , \quad q_3 = b_{22} - b_{12} \\ p_4 &= a_{12} - p_2 \ , \quad q_4 = b_{21} - q_2 \ , \end{aligned}$$

$$\begin{aligned} m_1 &= a_{11}b_{11} \ , \quad m_5 = p_3q_3 \ , \\ m_2 &= a_{12}b_{21} \ , \quad m_6 = p_4b_{22} \ , \\ m_3 &= p_1q_1 \ , \quad m_7 = a_{22}q_4 \ , \\ m_4 &= p_2q_2 \ , \end{aligned}$$

$$\begin{aligned} r_1 &= m_1 + m_4 \ , \quad r_3 = r_1 + m_3 \ , \\ r_2 &= r_1 + m_5 \ . \end{aligned}$$

Dentre as contas que constituem a equação (18) não usamos a propriedade de comutação entre os números reais. Isso da a ideia de que cada elemento de A e B pode ser na verdade um bloco 2×2 e podemos usar o algoritmo de Strassen-Winograd nestes blocos. Assim, esse algoritmo calcula de maneira recursiva o produto $C = AB$ mas com A e B de dimensões $2^k \times 2^k$ onde $k \in \mathbb{N}$. Esse algoritmo recursivo exige uma complexidade espacial maior que o algoritmo usual (essa é um propriedade de algoritmos recursivos, sua complexidade espacial é maior que suas versões iterativas) porém a complexidade temporal do algoritmo de Strassen-Winograd é notavelmente menor que dos algoritmos anteriores.

A função da complexidade temporal do algoritmo de Strassen-Winograd [1] é

$$g_s(n) = (5 + w)n^{\log_2 7} - 5n^2 \ , \tag{19}$$

de ordem $O(n^{\log_2 7})$. Assim, o algoritmo de Strassen-Winograd é um algoritmo de multiplicação matricial subcúbico.

Para facilitar a comparação da complexidade temporal entre esse algoritmo e o algoritmo usual podemos fazer a simples substituição na dimensão do problema, $n = 2^k$. Assim, teremos a complexidade do algoritmo de Strassen-Winograd

$$g_s(k) = (5 + w)7^k - 5 \cdot 2^{2k} \ . \tag{20}$$

E a complexidade do algoritmo usual

$$g_u(k) = (1 + w)2^{3k} - 2^{2k} \ . \tag{21}$$

Nas duas figuras a seguir estão as curvas da razão entre a complexidade temporal do algoritmo de Strassen-Winograd e do algoritmo usual em dois casos. O primeiro caso (figura (12)) é supondo que ω é igual a 1. Podemos observar que para valores de k menores ou iguais a 8 o algoritmo usual apresenta complexidade temporal menor. Porém, para valores de k maiores ou iguais a 9 o algoritmo de Strassen-Winograd é superior. O segundo caso (figura (6) é usando o valor (aproximado) para ω que calculamos anteriormente, ≈ 1.2. Neste caso, vemos que a curva possui o mesmo formato (o que obviamente era esperado), mas a intersecção com reta $y = 1$ acontece para um valor de k menor. Com isso, vemos que neste caso para valores de k maiores ou iguais a 8 a complexidade temporal do algoritmo de Strassen-Winograd é menor que a complexidade temporal do algoritmo usual.

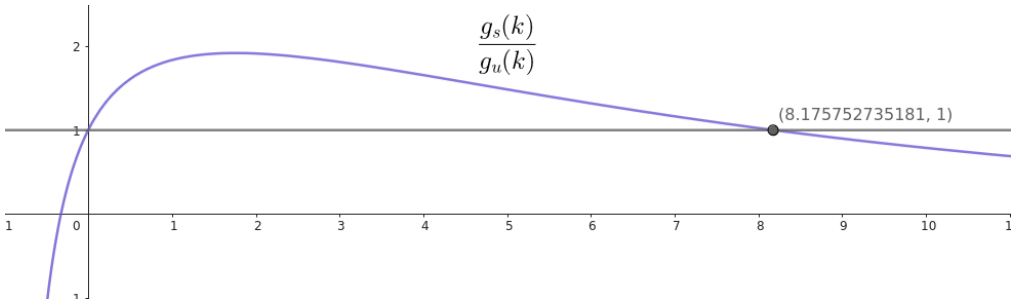

Figura 5: Curva da razão entre a complexidade temporal do algoritmo de Strassen-Winograd e o algoritmo usual de multiplicação matricial. Supondo $\omega = 1$.

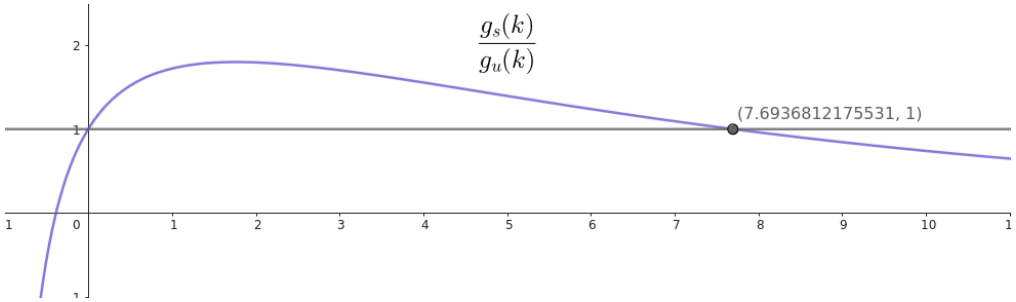

Figura 6: Curva da razão entre a complexidade temporal do algoritmo de Strassen-Winograd e o algoritmo usual de multiplicação matricial. Usando o valor aproximado $\omega = 1.2$.

Com isso, vemos que processadores com valores de ω maiores tiram maior vantagem do algoritmo de Strassen-Winograd. Fazendo o mesmo gráfico das figuras anteriores mas agora usando $\omega = 1,75$ vemos que o algoritmo de Strassen-Winograd é mais vantajoso que o usual para valores de k maiores ou igual a 7. Podemos observar também que o ponto de máximo presente nas figuras anteriores tende a diminuir quando aumentamos o valor de ω. Assim, podemos supor que existe um valor máximo de ω em que o algoritmo de Strassen-Winograd é superior ao usual para valores maiores ou igual a algum $\bar{k}$. Aumentando o valor de ω encontramos que para $\omega = 11$ o algoritmo de Strassen-Winograd é sempre superior ao algoritmo usual (figura (7)).

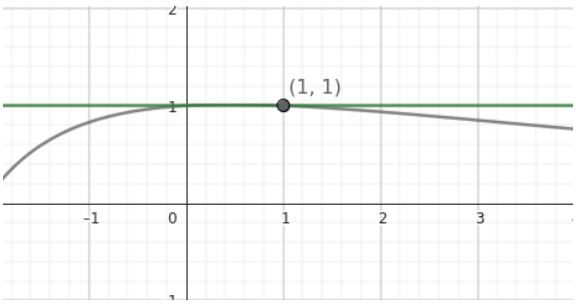

Figura 7: Curva da razão entre a complexidade temporal do algoritmo de Strassen-Winograd e o algoritmo usual de multiplicação matricial. Supondo $\omega = 11$.

Para comparar o resultado teórico acima com a implementação computacional [2] calculamos o tempo de execução do algoritmo usual (recursivo) e o algoritmo de Strassen-Winograd (figura (8)). Podemos observar claramente que o algoritmo de Strassen-Winograd tem tempo de execução menor para todos os casos calculados. Isso acontece pois o número de vezes que as funções recursivas são acionadas é menor no algoritmo de Strassen-Winograd que o algoritmo usual recursivo.

```
avr_standard = {32: 0.07687279279999998, 64: 0.6286987551699997, 128: 5.099702066770005, 256: 41.66193154519}
error_standard = {32: 0.007687279279999998, 64: 0.06286987551699998, 128: 0.5099702066770005, 256: 4.166193154519}
```

```
avr_strassen = {32: 0.07297986602000002, 64: 0.5270566627300005, 128: 3.71647881268, 256: 25.483390262640015}
error_strassen = {32: 3.1588989860478194e-05, 64: 0.0003225336775665037 3, 128: 0.0017576951155443537, 256: 0.01641761859034401}
```

Figura 8: Comparação de tempo de execução (em segundos) entre a multiplicação de matrizes usual e o algoritmo de Strassen-Winograd para matrizes de dimensões $n = 32$ ($k = 5$), $n = 64$ ($k = 6$), $n = 128$ ($k = 7$) e $n = 256$ ($k = 8$). Número de ensaios $E = 100$ para cada valor de k.

Referências

[1] A. Saa, *EP3 - Algoritmos subcúbicos para multiplicação matricial*, 2023. [Online]. Available: http://vigo.ime.unicamp.br/mt404/EP3.pdf

[2] V. O. Martins, "subcubic-matrix-multiplication," 2023. [Online]. Available: https://gitlab.com/mt404/subcubic-matrix-multiplication