UNIWERSYTET WROCŁAWSKI

Algorytm Strassena

Pracownia 1.19

Antoni Tomaszewski
Artur Derechowski

10 listopada 2018

1 OPIS PROBLEMU

Zadanie polega na napisaniu algorytmu Strassena mnożenia macierzy n*n i porównania go z klasycznym mnożeniem macierzy. Algorytm Strassena ma złożoność asymptotyczną

$$O(N^{\log 7})$$

co jest lepsze od standardowego mnożenia w asymptotycznym czasie

$$O(N^3)$$

Dla dużych macierzy algorytm Strassena powinien więc być szybszy. Algorytm ten zawiera jednak dużą stałą rzędu $O(N^2)$, dlatego dla małych macierzy spodziewamy się, że będzie on wolniejszy od klasycznego mnożenia.

2 ALGORYTM STRASSENA

Mając dwie macierze X i Y aby wyznaczyć ich iloczyn można najpierw podzielić je na bloki

$$Z = \begin{bmatrix} R & S \\ T & U \end{bmatrix} \qquad X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \qquad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix} \tag{2.1}$$

gdzie Z jest iloczynem macierzy X i Y. Następnie trzeba zauważyć, że zamiast robić 8 mnożeń bloków macierzy wystarczy zrobić 7.

$$R = AE + BH, \qquad S = AG + BH, \qquad T = CE + DF, \qquad U = CG + DH \tag{2.2}$$

Tę samą macierz można przedstawić w ten sposób:

$$R = P5 + P4 - P2 + P6, \qquad S = P1 + P2, \qquad T = P3 + P4, \qquad U = P5 + P1 - P3 - P7 \tag{2.3}$$

gdzie:

$$\begin{aligned} P1 &= A(G-H), & P2 &= (A+B)H, & P3 &= (C+D)E, & P4 &= D(F-E), \\ P5 &= (A+D)(E+H), & P6 &= (B-D)(F+H), & P7 &= (A-C)(E+G) \end{aligned} \tag{2.4}$$

Algorytm Strassena dzieli wtedy macierze na pół i wywołuje na nich kolejne mnożenie Strassena, z analogicznym podziałem dla coraz to mniejszych macierzy.

Widać, że algorytm Strassena działa najefektywniej dla macierzy rozmiaru 2^k. Wtedy zawsze można je dzielić na 2 i wywoływać rekurencyjnie Strassena dla mniejszych macierzy. Gdy macierz jest nieparzystego rozumiaru, to nie można jej pomnożyć, bo nie można jej podzielić na bloki. Problem pojawia się, gdy mamy więc macierz rozmiaru $2^k + 1$. Wtedy efektywnie tworzymy macierz 4 razy większą (resztę wypełniamy zerami), aby ją pomnożyć. W części doświadczalnej pracowni sprawdzamy więc tylko macierze o boku 2^k, aby zilustrować jak najbardziej efektywny algorytm Strassena. Stała, którą zaniedbujemy nie jest jednak duża, bo w najgorszym przypadku wynosi 4.

3 Analiza błędów

po implementacji algorytmu Strassena kolejną częścią pracowni jest analiza błędów algorytmu. Należy przeprowadzić obliczenia dla macierzy o rozmiarach od 4 do 500. Dla danej macierzy nieosobliwej trzeba też policzyć wartości błędów $\Delta(XX^{-1} - I)$ oraz $\Delta(X^{-1}X - I)$, gdzie

$$\Delta(X) := \sum_{i=1}^{n} \sum_{j=1}^{n} x_{i,j}^2$$

Jak te błędy mają się do klasycznego sposobu mnożenia macierzy?

Dla danych macierzy X Y V, obliczyć także $\Delta((XY)V - X(YV))$. Porównać to z klasycznym mnożeniem macierzy. To sprawdzi, czy mnożenie Strassena dobrze zachowuje łączność mnożenia macierzy.

Czy generowane macierze powinny być losowe?
Chyba musimy też stworzyć algorytm liczenia macierzy odwrotnej.

4 Pomiary

Porównujemy czasy mnożenia dwóch macierzy o danym boku n (tabela 5.1). Ponieważ algorytm Strassena wyrównuje macierze nie-kwadratowe do macierzy kwadratowych, może

Tabela 4.1: Czas mnożenie macierzy dwoma sposobami

n	Zwykły (wbudowany)	Zwykły	Strassen (R)	
32	2.8482e-5	0.000105749	0.012895832	
64	6.0576e-5	0.000894503	0.089886236	
96	0.000170009	0.003644241	0.550218023	
128	0.000166872	0.010069118	0.548687255	
160	0.000330841	0.015669477	3.860402354	
192	0.000502141	0.022380795	3.860977080	
224	0.000735925	0.032780297	3.851896002	
256	0.001038842	0.064917801	3.846245318	

tracić na tym dużo czasu. Dla przykładu pomnożenie wektorów k-elementowych zajmie mu tyle samo czasu, co pomnożenie macierzy kwadratowych o boku k. Dlatego w pomiarach bierzemy pod uwagę tylko macierze kwadratowe.

Można zauważyć, że algorytm Strassena obecnie wykonuje się o wiele wolniej od wbudowanego operatora mnożenia w języku Julia. Ten operator może być jednak zaimplementowany o wiele wydajniej, ponieważ jest częścią języka. Warto w takim razie sprawdzić, jak algorytm Strassena ma się do ręcznie napisanej funkcji, która działa tak samo jak wbudowany operator, ale nie korzysta z możliwych usprawnień systemowych.

Widać, że algorytm Strassena w wersji rekurencyjnej jest również o wiele wolniejszy od napisanej przez nas funkcji mnożącej macierze. To może być jednak spowodowane tym, że algorytm tworzy bardzo wiele wywołań rekurencyjnych (aż do macierzy stopnia 1), więc następnym krokiem będzie napisanie algorytmu w wersji iteracyjnej.

Czasy wykonania algorytmu Strassena są natomiast identyczne co do rzędu wielkości dla macierzy o rozmiarach $\left[2^k, 2^{k+1}\right)$. Dzieje się tak, ponieważ w algorytmie Strassena macierze są zawsze wyrównywane do parzystych wielkości poprzez wypełnienie zerami. Można zauważyć, że wszystkich wyrównań nigdy nie będzie więcej niż czterokrotność całej wielkości macierzy (dla macierzy o boku $2^k + 1$). Najefektywniej jest natomiast mnożyć macierze o boku 2^k.

Kolejnym problemem w implementacji algorytmu Strassena jest rekurencja. Implementacja opierająca się na wzorze naturalnie z niej korzysta, jednak dla komputera jest to proces wolniejszy od iteracyjnego rozwiązania. Rekurencyjna wersja algorytmu Strassena jest również o wiele mniej wydajna pod względem pamięciowym, gdzie już dla macierzy niewielkich rozmiarów obserwujemy znaczny wzrost w zaalokowanej pamięci (2GB RAM dla macierzy o boku 300).

5 Mnożenie kwaternionów

Algorytm Strassena daje oszczędność jednego mnożenia na osiem. Istnieją jednak inne przekształcenia, które mogą bardziej zmniejszyć liczbę kosztownych mnożeń na rzecz dodawa-

nia. Przykładem tego jest mnożenie kwaternionów.

5.1 KWATERNIONY

Kwaternion jest strukturą algebraiczną rozszerzającą liczby zespolone:

$$q = a + bi + cj + dk$$

gdzie a, b, c, d są liczbami rzeczywistymi,

$$i^2 = j^2 = k^2 = ijk = -1$$

Mnożąc klasycznie dwa kwaterniony zgodnie z zasadą rozdzielności wykonamy 16 mnożeń:

$$\begin{gathered}(x_1 + x_2 i + x_3 j + x_4 k) * (y_1 + y_2 i + y_3 j + y_4 k) = f1 + f2 + f3 + f4 \\ f_1 = x_1 y_1 - x_2 y_2 - x_3 y_3 - x_4 y_4 \\ f_2 = (x_1 y_2 + x_2 y_1 + x_3 y_4 - x_4 y_3) i \\ f_3 = (x_1 y_3 - x_2 y_4 + x_3 y_1 + x_4 y_2) j \\ f_4 = (x_1 y_4 + x_2 y_3 - x_3 y_2 + x_4 y_1) k\end{gathered} \tag{5.1}$$

5.2 PRZEKSZTAŁCENIE

Zapiszemy powyższą sumę w inny sposób, wykonując tylko 8 mnożeń zamiast 16. Niech:

$$\begin{gathered}[I] = x1y1 \\ [II] = x4y3 \\ [III] = x2y4 \\ [IV] = x3y2 \\ [V] = (x1 + x2 + x3 + x4)(y1 + y2 + y3 + y4) \\ [VI] = (x1 + x2 - x3 - x4)(y1 + y2 - y3 - y4) \\ [VII] = (x1 - x2 + x3 - x4)(y1 - y2 + y3 - y4) \\ [VIII] = (x1 - x2 - x3 + x4)(y1 - y2 - y3 + y4)\end{gathered} \tag{5.2}$$

Wtedy sumę $f_1 + f_2 + f_3 + f_4$ można zapisać jako:

$$\begin{gathered}f_1 = 2[I] - ([V] + [VI] + [VII] + [VIII])/4 \\ f_2 = -2[II] + ([V] + [VI] - [VII] - [VIII])/4 \\ f_3 = -2[III] + ([V] - [VI] + [VII] - [VIII])/4 \\ f_4 = -2[IV] + ([V] - [VI] - [VII] + [VIII])/4\end{gathered} \tag{5.3}$$

Widać więc, że iloczyn dwóch kwaternionów można wykonać, robiąc tylko 8 mnożeń zamiast 16[1]. Zapisanie tego przekształcenia w innej postaci pozwala więc na poprawę liczby mnożeń o 50%., podczas gdy algorytm Strassena mnożenia macierzy pozwalał jedynie na 12,5% oszczędności.

Dla algorytmu mnożącego kwaterniony można udowodnić, że nie da się go wykonać z liczbą mnożeń mniejszą niż 8.[1]

6 Plany

6.1 Algorytm łączony

Nie jest to opisane w poleceniu pracowni, ale ciekawie byłoby pokazać dla jak dużych macierzy w naszej implementacji opłaca się bardziej stosowac algorytm Strassena od klasycznego. Można też się zastanowić wtedy nad algorytmem, który stosuje dzielenie Strassena do pewnego momentu, a gdy macierze są dostatecznie małe, to mnoży je klasycznie. Taki algorytm powinien teoretycznie być szybszy od obu powyższych, bo łączy najlepszą część algorytmu Strassena (mnożenie dużych macierzy) z klasycznym mnożeniem, które jest szybsze dla małych macierzy.

6.2 Do zrobienia

- wzory na duże macierze odwrotne
- analiza błędów

Literatura

[1] Thomas D. Howell, Jean-Cleaude Lafon, *The Complexity of the Quaternion Product*, Department of Computer Science, Cornell University, Ithaca, N.Y. 1975.