附录 A 应用举例

A.2 应用: 矩阵乘积的快速算法

A.2.1 矩阵乘积的普通方法

设 $A=[a_{ij}], B=[b_{ij}] \in \mathbb{R}^{n\times n}$, 则 $C=[c_{ij}]=AB$, 其中

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

易知, 总运算量为: n^3（乘法）+ n^3-n^2（加法）$=2n^3$.

算法 A.1. 矩阵乘积: IJK 顺序

```
for i = 1 to n do
    for j = 1 to n do
        for k = 1 to n do
            c_ij = c_ij + a_ik b_kj
        end for
    end for
end for
```

以上是计算矩阵乘积的常规实现方式, 但并不是速度最快的实现方式. 基于矩阵乘积在实际应用中的重要性, 人们在不断寻求计算矩阵乘积的快速方法. 对于矩阵乘法的加速, 有如下两种策略:

- 基于硬件的优化.
- 基于算法的优化, 如著名的 Strassen 算法.

A.2.2 基于硬件的加速方法

对于硬件优化, 有循环展开、基于内存布局等的优化技巧, 这里只演示一个最简单技巧: 调换循环顺序, 比如将原来的 IJK 顺序改成 IKJ 顺序:

算法 A.2. 矩阵乘积: IKJ 顺序

```
for i = 1 to n do
    for k = 1 to n do
        for j = 1 to n do
            c_ij = c_ij + a_ik b_kj
```

```
    end for
  end for
end for
```

例 A.1 在个人电脑 (CPU i9-10900K, 内存 128G) 上用 C 语言测试. (`test_dgemm.c`)

- `n=1024`, `gcc` 编译加 `-O2` 选项, IKJ 顺序大概比 IJK 顺序快 2.5 倍. 如果加 `-O3` 选项, 则快 3.5 倍.
- `n=2048`, `gcc` 编译加 `-O2` 选项, IKJ 顺序大概比 IJK 顺序快 6 倍. 如果加 `-O3` 选项, 则快 8 倍.

实测速度

测试环境: 双 CPU Xeon 4215R (8 核 16 线程), 32G 内存. 带 `-O2` 编译选项.

N	IKJ	KIJ	IJK	JIK	KJI	JKI
1024	0.550	0.570	3.430	3.610	7.920	7.770
2048	4.360	4.880	25.050	27.560	97.100	99.610

✍ **注记:** 影响矩阵乘积的硬件因素有许多, 比如:

- 向量化指令运算, 如 X86 架构上的 AVX 指令集等.
- 循环展开 (loop unrolling), 编译器中加优化选项后, 一般会自动进行循环展开.
- 缓存分块 (cache blocking, 也称 tiling), 即对 A, B, C 进行分块, 使得每次计算时, A, B, C 的子块都在缓存中, 从而大幅提高缓存命中率.

```
for ii = 1 to n with step s do   % 其中 s 表示块的大小, 根据缓存大小来确定
    for jj = 1 to n with step s do
        for kk = 1 to n with step s do
            for i = ii to ii + s do
                for j = kk to jj + s do
                    for k = kk to kk + s do
                        c_ij = c_ij + a_ik b_kj
                    end for
                end for
            end for
        end for
    end for
end for
```

- 多线程, CPU 并行 (OpenMP, MPI), GPU 并行等.
- 融合乘加 (FMA, Fused Multiply-Add), 即计算 $a \times b + c$ 时将乘法和加法组合成单一指令且仅

执行一次舍入操作, 不仅能提高运算速度, 还能增强数值精度.

- ... ...

关于内存优化可参考

[1] R. E. Bryant and D. R. O'Hallaron, *Computer Systems: A Programmer's Perspective*, 3rd edn, Pearson, 2016. 《深入理解计算机系统》, 简称 CSAPP, 龚奕利, 贺莲译.

A.2.3 Strassen 方法

德国数学家 Strassen 在 1969 年提出了计算矩阵乘积的快速算法, 将运算量降为约 $O(n^{2.81})$. 下面以二阶矩阵为例, 描述 Strassen 方法的实现过程. 设

$$A=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \quad B=\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}.$$

则 $C=AB$ 的每个分量为

$$\begin{aligned} c_{11}&=a_{11}b_{11}+a_{12}b_{21}, \quad c_{12}=a_{11}b_{12}+a_{12}b_{22},\\ c_{21}&=a_{21}b_{11}+a_{22}b_{21}, \quad c_{22}=a_{21}b_{12}+a_{22}b_{22}. \end{aligned}$$

在 Strassen 算法中, 我们并不直接通过上面的公式来计算 C, 而是先计算下面 7 个量:

$$\begin{aligned} x_1&=(a_{11}+a_{22})(b_{11}+b_{22}),\\ x_2&=(a_{21}+a_{22})b_{11},\\ x_3&=a_{11}(b_{12}-b_{22}),\\ x_4&=a_{22}(b_{21}-b_{11}),\\ x_5&=(a_{11}+a_{12})b_{22},\\ x_6&=(a_{21}-a_{11})(b_{11}+b_{12}),\\ x_7&=(a_{12}-a_{22})(b_{21}+b_{22}). \end{aligned}$$

于是, C 的各元素可以表示为:

$$\begin{aligned} c_{11}&=x_1+x_4-x_5+x_7,\\ c_{12}&=x_3+x_5,\\ c_{21}&=x_2+x_4,\\ c_{22}&=x_1+x_3-x_2+x_6. \end{aligned}$$

易知, 总共需要做 7 次乘法和 18 次加法.

下面考虑一般情形. 我们采用分而治之的思想, 先将矩阵 A, B 进行 2×2 分块, 即

$$A=\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B=\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix},$$

则 $C = AB$ 也可以写成 2×2 分块形式, 即

$$\begin{aligned} C_{11} &= X_1 + X_4 - X_5 + X_7, \\ C_{12} &= X_3 + X_5, \\ C_{21} &= X_2 + X_4, \\ C_{22} &= X_1 + X_3 - X_2 + X_6, \end{aligned}$$

其中

$$\begin{aligned} X_1 &= (A_{11} + A_{22})(B_{11} + B_{22}), \\ X_2 &= (A_{21} + A_{22})B_{11}, \\ X_3 &= A_{11}(B_{12} - B_{22}), \\ X_4 &= A_{22}(B_{21} - B_{11}), \\ X_5 &= (A_{11} + A_{12})B_{22}, \\ X_6 &= (A_{21} - A_{11})(B_{11} + B_{12}), \\ X_7 &= (A_{12} - A_{22})(B_{21} + B_{22}). \end{aligned}$$

需要 7 次子矩阵的乘积和 18 次子矩阵加法. 假定采用普通方法计算子矩阵的乘积, 即需要 $(n/2)^3$ 乘法和 $(n/2)^3$ 次加法, 则采用 Strassen 方法计算 A 和 B 乘积的运算量为

$$7 \times \left((n/2)^3 + (n/2)^3 - (n/2)^2\right) + 18 \times (n/2)^2 = \frac{7}{4}n^3 + \frac{11}{4}n^2.$$

大约是普通矩阵乘积运算量的 $\dfrac{7}{8}$.

在计算子矩阵的乘积时, 我们仍然可以采用 Strassen 算法.

Strassen 算法的计算量

用 $t(n)$ 表示计算两个 n 阶矩阵乘积的计算量, 则由递归计算过程可知

$$t(n) = 7t(n/2) + 18(n/2)^2.$$

假定 n 是 2 的整数次幂 (即 $n = 2^p$), 则可得乘法次数为

$$7^{\log_2 n} = n^{\log_2 7}.$$

加减运算次数为

$$18\left(\frac{n^2}{4} + \frac{7n^2}{4^2} + \frac{7^2n^2}{4^3} + \cdots + \frac{7^{p-1}n^2}{4^p}\right) = \frac{18n^2}{4}\left(1 + \frac{7}{4} + \frac{7^2}{4^2} + \cdots + \frac{7^{p-1}}{4^{p-1}}\right).$$

根据等比数列求和公式, 上式等于

$$6n^2\left(\left(\frac{7}{4}\right)^p - 1\right) = 6n^2\left(\frac{7}{4}\right)^{\log_2 n} - 6n^2 = 67^{\log_2 n} - 6n^2 = 6n^{\log_2 7} - 6n^2.$$

因此, 总计算量为

$$t(n) = n^{\log_2 7} + 6n^{\log_2 7} - 6n^2 \approx \mathcal{O}\left(n^{2.807\ldots}\right).$$

表 A.1. Strassen 算法的加速比

n	2^{10}	2^{15}	2^{20}	2^{25}	2^{30}	2^{40}
加速比 (考虑总计算量)	1.09	2.12	4.13	8.05	15.69	59.65
加速比 (只考虑乘法运算)	3.80	7.41	14.45	28.17	54.92	208.78

课外阅读

记两个 n 阶矩阵乘积的运算量为 $O(n^\alpha)$. 在 Strassen 算法出现之前, 大家一直认为 $\alpha = 3$. 但随着 Strassen 算法的出现, 大家意识到这个量级是可以小于 3 的. 但 α 能不能进一步减少? 如果能的话, 它的极限又是多少? 这些问题引起了众多学者的极大兴趣. 1978 年, Pan 证明了 $\alpha < 2.796$. 一年后, Bini 等又将这个上界改进到 2.78. 在 Pan 和 Bini 等的工作的基础上, Schönhage 于 1981 年证明了 $\alpha < 2.522$. 而上界首次突破 2.5 是由 Coppersmith 和 Winograd 提出来的, 他们证明了 $\alpha < 2.496$. 1990 年, 他们又在 Strassen 算法的基础上提出了一种新的矩阵乘积计算方法, 将运算量级降至著名的 2.376. 这个记录一直持续了近二十年, 直到 2010 年前后, Stothers, Vassilevska-Williams 和 Le Gall 分别将这个上界降到 2.37293, 2.3728642 和 2.3728639. 2021 年, Alman 和 Williams 将上界进一步降到 2.3728596. 虽然有许多学者相信, α 的极限应该是 2, 但至今无法证实.

相关参考文献:

[1] V. Strassen, Gaussian elimination is not optimal, *Numer. Math.*, 13 (1969), 354–356.

[2] V. Y. Pan, Strassen's algorithm is not optimal, In *Proc. FOCS*, 19 (1978), 166–176.

[3] D. Bini, M. Capovani, F. Romani and G. Lotti, $O(n^{2.7799})$ complexity for $n \times n$ approximate matrix multiplication, *Inf. Process. Lett.*, 8 (1979), 234–235.

[4] A. Schönhage, Partial and total matrix multiplication, *SIAM J. Comput.*, 10 (1981), 434–455.

[5] D. Coppersmith and S. Winograd, On the asymptotic complexity of matrix multiplication, In *Proc. SFCS*, 82–90, 1981.

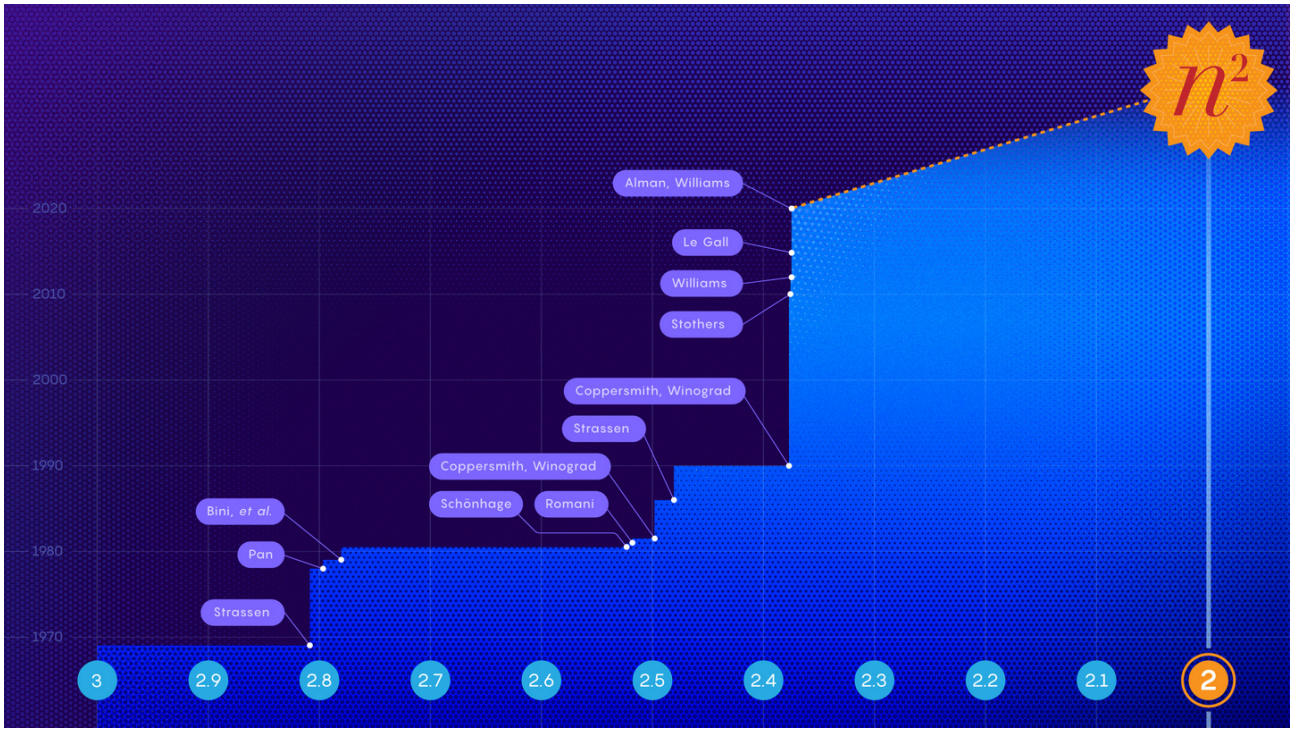
[6] D. Coppersmith and S. Winograd, Matrix multiplication via arithmetic progressions, *J. Symbolic Computation*, 9 (1990), 251–280.

[7] A. J. Stothers, *On the complexity of matrix multiplication*, PhD thesis, 2010.

[8] V. Vassilevska-Williams, Breaking the Coppersmith – Winograd barrier, In *44th ACM Symposium on Theory of Computing (STOC 2012)*, 2012.

[9] F. Le Gall, Powers of tensors and fast matrix multiplication, *In 39th International Symposium on Symbolic and Algebraic Computation (ISSAC 2014)*, 2014, 296–303. arXiv:1401.7714.

[10] J. Alman and V. V. Williams, A refined laser method and faster matrix multiplication, Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA), 2021.



(https://www.quantamagazine.org/mathematicians-inch-closer-to-matrix-multiplication-goal-20210323/)