# 实验报告

## 一、Strassen 矩阵乘法的实现

### 1.原理

假设矩阵 $A$ 和矩阵 $B$ 都是 $N \times N(N=2^n)$ 的方矩阵，求 $C=AB$，如下所示：

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

其中 $\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$

如上分解完矩阵 ABC 之后，创建 10 个 N/2xN/2 的矩阵 S1-S10

$$\begin{aligned} S_1 &= B_{12} - B_{22} & S_6 &= B_{11} + B_{22} \\ S_2 &= A_{11} + A_{12} & S_7 &= A_{12} - A_{22} \\ S_3 &= A_{21} + A_{22} & S_8 &= B_{21} + B_{22} \\ S_4 &= B_{21} - B_{11} & S_9 &= A_{11} - A_{21} \\ S_5 &= A_{11} + A_{22} & S_{10} &= B_{11} + B_{12} \end{aligned}$$

然后由此计算 P1-P7

$$\begin{gathered} P_1 = A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\ P_2 = S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\ P_3 = S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\ P_4 = A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11} \\ P_5 = S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\ P_6 = S_7 \cdot S_8 = A_{12} \cdot B_{21} + A12 \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22} \\ P_7 = S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12} \end{gathered}$$

最后根据 Pi 计算 C11-C22，拼成矩阵 C 得到结果

$$\begin{gathered} C_{11} = P_5 + P_4 - P_2 + P_6 \\ C_{12} = P_1 + P_2 \\ C_{21} = P_3 + P_4 \\ C_{22} = P_5 + P_1 - P_3 - P_7 \end{gathered}$$

## 2.思路

由于 Strassen 矩阵乘法只适用于 N x N（$N=2^n$）的方矩阵，所以对于一般的矩阵乘法需要将两个矩阵补零至该情况再计算，最后只输出需要部分即可。

## 3.代码

见附件“786.cpp”

## 4.正确性检验

提交 YOJ#786，证明结果正确

| | | | | | | |
|---|---|---|---|---|---|---|
| #786 矩阵乘法 | ✔ Accepted | 100 | 1731 ms | 1088 KB | cpp / 7149 B | 李馨雨(2020202279) |

# 二、Strassen 矩阵乘法与朴素算法对比

## 1.结果展示

用所给数据进行试验，得到如下结果：（代码见 compare.cpp）

100 100 100

```
朴素矩阵算法开始时钟: 1791
朴素矩阵算法结束时钟: 1796
Strassen矩阵算法开始时钟: 1796
Strassen矩阵算法结束时钟: 2571
朴素矩阵算法: 5 Clocks..0 Sec
Strassen矩阵算法: 775 Clocks..0 Sec
```

200 200 200

```
朴素矩阵算法开始时钟: 4181
朴素矩阵算法结束时钟: 4226
Strassen矩阵算法开始时钟: 4226
Strassen矩阵算法结束时钟: 9855
朴素矩阵算法: 45 Clocks..0 Sec
Strassen矩阵算法: 5629 Clocks..5 Sec
```

300 300 300

```
朴素矩阵算法开始时钟: 9938
朴素矩阵算法结束时钟: 10079
Strassen矩阵算法开始时钟: 10080
Strassen矩阵算法结束时钟: 48011
朴素矩阵算法: 141 Clocks..0 Sec
Strassen矩阵算法: 37931 Clocks..37 Sec
```

400 400 400

```
朴素矩阵算法开始时钟: 14923
朴素矩阵算法结束时钟: 15277
Strassen矩阵算法开始时钟: 15278
Strassen矩阵算法结束时钟: 53587
朴素矩阵算法: 354 Clocks..0 Sec
Strassen矩阵算法: 38309 Clocks..38 Sec
```

500 500 500

```
朴素矩阵算法开始时钟: 21809
朴素矩阵算法结束时钟: 22523
Strassen矩阵算法开始时钟: 22524
Strassen矩阵算法结束时钟: 60946
朴素矩阵算法: 714 Clocks..0 Sec
Strassen矩阵算法: 38422 Clocks..38 Sec
```

得到如下表:

| 矩阵大小 | 朴素算法（时钟数） | Strassen 算法（时钟数） |
| ---: | ---: | ---: |
| 100 | 5 | 775 |
| 200 | 45 | 5629 |
| 300 | 141 | 37931 |
| 400 | 354 | 38309 |
| 500 | 714 | 38422 |

## 2.对比分析

朴素算法的时间复杂度为 $O(n^3)$，而 Strassen 算法的时间复杂度为 $O(n^{2.375})$,本应具有性能优势。但是根据 1 中的结果,可以发现在所给数据下,相比朴素算法,Strassen 算法的耗时不但没有减少，反而剧烈增多，效果更差。

## 3.原因与改进

通过实验和查阅相关资料，猜测导致 Strassen 算法性能较差的原因可能是：采用 Strassen 算法作递归运算，需要创建大量的动态二维数组，其中分配堆内存空间将占用大量计算时间，从而掩盖了 Strassen 算法的优势。

针对这一点，对 Strassen 算法进行改进。为其设置一个界限，当矩阵的维度小于这个界限时，采用朴素算法计算矩阵而不再分治递归，以此节省多次分配空间所花费的时间。通过实验，选择 N=128 作为界限值，改动如下，改进后计算时间大幅下降。

```
void Strassen(int N, int** A, int** B, int** C){
    int newSize = N / 2;
    if(N <= 128){   //边界情况
        mul(A, B, C, N, N, N);
    }else{
        int** A11;    //矩阵分块
```

## 三、结语

本次实验中，成功实现了矩阵相乘的 Strassen 算法并通过 YOJ 验证了其正确性。还通过对比朴素算法与 Strassen 算法的运行情况，发现了 Strassen 算法的缺陷并进行了改进，完成了实验要求。