Adaptive Matrix Multiplication

Data Structure and Algorithm Analysis

Group 21
WEI Jinqi
LUO Peiyu
HAN Xudong
LI Xingze

SUSTech Southern University of Science and Technology

CATALOGUE

SUSTech Southern University of Science and Technology

1 Background

Background

Topic

In this project we will implement and analyze, empirically and theoretically, a method for multiplying square matrices. The method will adapt to use algorithmic techniques depending on the system architecture on which it is run.

Goals

Matrix Method

Basic Variables

Random Matrix

Basic Method

Multiply Algorithm

Pseudocode

Brute-force Algorithm

Strassen Algorithm

Comparing and Optimizing

Runtime Comparing

Algorithm Optimizing

Adaptive Algorithm

2 Algorithm

Algorithm

Square Matrix Multiply Algorithm (Runtime: $\Theta(n^3)$)

SQUARE-MATRIX-MULTIPLY(A, B)

$n = A.rows$

let C be a new $n \times n$ matrix

for i = 1 **to** n

    for j = 1 **to** n

        $c_{i,j} = 0$

        for k = 1 **to** n

            $c_{i,j} = c_{i,j} + a_{i,k} \cdot b_{k,j}$

return C

Algorithm

Strassen Algorithm (Runtime: $\Theta(n^{\lg 7})$)

```
STRASSEN(A, B)
n = A.rows
if n == 1
    return A[1, 1] * B[1, 1]
let C be a new n × n matrix
A[1, 1] = A[1..n / 2][1..n / 2]
A[1, 2] = A[1..n / 2][n / 2 + 1..n]
A[2, 1] = A[n / 2 + 1..n][1..n / 2]
A[2, 2] = A[n / 2 + 1..n][n / 2 + 1..n]
B[1, 1] = B[1..n / 2][1..n / 2]
B[1, 2] = B[1..n / 2][n / 2 + 1..n]
B[2, 1] = B[n / 2 + 1..n][1..n / 2]
B[2, 2] = B[n / 2 + 1..n][n / 2 + 1..n]
S[1] = B[1, 2] - B[2, 2]
S[2] = A[1, 1] + A[1, 2]
S[3] = A[2, 1] + A[2, 2]
S[4] = B[2, 1] - B[1, 1]
S[5] = A[1, 1] + A[2, 2]
S[6] = B[1, 1] + B[2, 2]
S[7] = A[1, 2] - A[2, 2]
S[8] = B[2, 1] + B[2, 2]
S[9] = A[1, 1] - A[2, 1]
S[10] = B[1, 1] + B[1, 2]
P[1] = STRASSEN(A[1, 1], S[1])
P[2] = STRASSEN(S[2], B[2, 2])
P[3] = STRASSEN(S[3], B[1, 1])
P[4] = STRASSEN(A[2, 2], S[4])
P[5] = STRASSEN(S[5], S[6])
P[6] = STRASSEN(S[7], S[8])
P[7] = STRASSEN(S[9], S[10])
C[1..n / 2][1..n / 2] = P[5] + P[4] - P[2] + P[6]
C[1..n / 2][n / 2 + 1..n] = P[1] + P[2]
C[n / 2 + 1..n][1..n / 2] = P[3] + P[4]
C[n / 2 + 1..n][n / 2 + 1..n] = P[5] + P[1] - P[3] - P[7]
return C
```

Strassen Algorithm For Any Matrix (Runtime: $\Theta(n^{\lg 7})$)

```
STRASSEN(A, B)
n = A.rows
if n == 1
    return A[1, 1] * B[1, 1]
if n % 2 == 1
    n = n - 1
let C be a new n × n matrix
A[1, 1] = A[1..n / 2][1..n / 2]
A[1, 2] = A[1..n / 2][n / 2 + 1..n]
A[2, 1] = A[n / 2 + 1..n][1..n / 2]
A[2, 2] = A[n / 2 + 1..n][n / 2 + 1..n]
B[1, 1] = B[1..n / 2][1..n / 2]
B[1, 2] = B[1..n / 2][n / 2 + 1..n]
B[2, 1] = B[n / 2 + 1..n][1..n / 2]
B[2, 2] = B[n / 2 + 1..n][n / 2 + 1..n]
S[1] = B[1, 2] - B[2, 2]
S[2] = A[1, 1] + A[1, 2]
S[3] = A[2, 1] + A[2, 2]
S[4] = B[2, 1] - B[1, 1]
S[5] = A[1, 1] + A[2, 2]
S[6] = B[1, 1] + B[2, 2]
S[7] = A[1, 2] - A[2, 2]
S[8] = B[2, 1] + B[2, 2]
S[9] = A[1, 1] - A[2, 1]
S[10] = B[1, 1] + B[1, 2]
P[1] = STRASSEN(A[1, 1], S[1])
P[2] = STRASSEN(S[2], B[2, 2])
P[3] = STRASSEN(S[3], B[1, 1])
P[4] = STRASSEN(A[2, 2], S[4])
P[5] = STRASSEN(S[5], S[6])
P[6] = STRASSEN(S[7], S[8])
P[7] = STRASSEN(S[9], S[10])
C[1..n / 2][1..n / 2] = P[5] + P[4] - P[2] + P[6]
C[1..n / 2][n / 2 + 1..n] = P[1] + P[2]
C[n / 2 + 1..n][1..n / 2] = P[3] + P[4]
C[n / 2 + 1..n][n / 2 + 1..n] = P[5] + P[1] - P[3] - P[7]
return C
```

Algorithm

Parallel Strassen Algorithm (Runtime: $\Theta(n^{\lg 7}/\lg^2 n)$)

```
P-STRASSEN(A, B)
n = A.rows
if n == 1
    return A[1, 1] * B[1, 1]
let C be a new n×n matrix
parallel
    A[1, 1] = A[1..n / 2][1..n / 2]
    A[1, 2] = A[1..n / 2][n / 2 + 1..n]
    A[2, 1] = A[n / 2 + 1..n][1..n / 2]
    A[2, 2] = A[n / 2 + 1..n][n / 2 + 1..n]
    B[1, 1] = B[1..n / 2][1..n / 2]
    B[1, 2] = B[1..n / 2][n / 2 + 1..n]
    B[2, 1] = B[n / 2 + 1..n][1..n / 2]
    B[2, 2] = B[n / 2 + 1..n][n / 2 + 1..n]
parallel
    S[1] = B[1, 2] - B[2, 2]
    S[2] = A[1, 1] + A[1, 2]
    S[3] = A[2, 1] + A[2, 2]
    S[4] = B[2, 1] - B[1, 1]
    S[5] = A[1, 1] + A[2, 2]
    S[6] = B[1, 1] + B[2, 2]
    S[7] = A[1, 2] - A[2, 2]
    S[8] = B[2, 1] + B[2, 2]
    S[9] = A[1, 1] - A[2, 1]
    S[10] = B[1, 1] + B[1, 2]
parallel
    P[1] = STRASSEN(A[1, 1], S[1])
    P[2] = STRASSEN(S[2], B[2, 2])
    P[3] = STRASSEN(S[3], B[1, 1])
    P[4] = STRASSEN(A[2, 2], S[4])
    P[5] = STRASSEN(S[5], S[6])
    P[6] = STRASSEN(S[7], S[8])
    P[7] = STRASSEN(S[9], S[10])
parallel
    C[1..n / 2][1..n / 2] = P[5] + P[4] - P[2] + P[6]
    C[1..n / 2][n / 2 + 1..n] = P[1] + P[2]
    C[n / 2 + 1..n][1..n / 2] = P[3] + P[4]
    C[n / 2 + 1..n][n / 2 + 1..n] = P[5] + P[1] - P[3] - P[7]
return C
```

3 Experiments and Results

Timeline

<table>
<tr><th></th><th>Monday</th><th>Tuesday</th><th>Wednesday</th><th>Thursday</th><th>Friday</th><th>Saturday</th><th>Sunday</th></tr>
<tr><td>Week 11</td><td>Team Formed</td><td colspan="3">Topic and Requirements Definition</td><td colspan="3">Paper Reading and Reference Collection</td></tr>
<tr><td>Week 12</td><td colspan="2">Basic Matrix Method</td><td>Brute-force Algorithm</td><td rowspan="3">Weekly Meeting for Progress Reporting</td><td colspan="3">Strassen Algorithm</td></tr>
<tr><td>Week 13</td><td colspan="2">Strassen Optimizing</td><td></td><td colspan="3">Parallel Algorithm</td></tr>
<tr><td>Week 14</td><td colspan="3">Algorithm Comparing</td><td colspan="3">Presentation Preparing</td></tr>
<tr><td>Week 15</td><td>Final Presentation</td><td colspan="3">Paper Preparing</td><td>All Uploading</td><td colspan="2"></td></tr>
</table>

Contribution

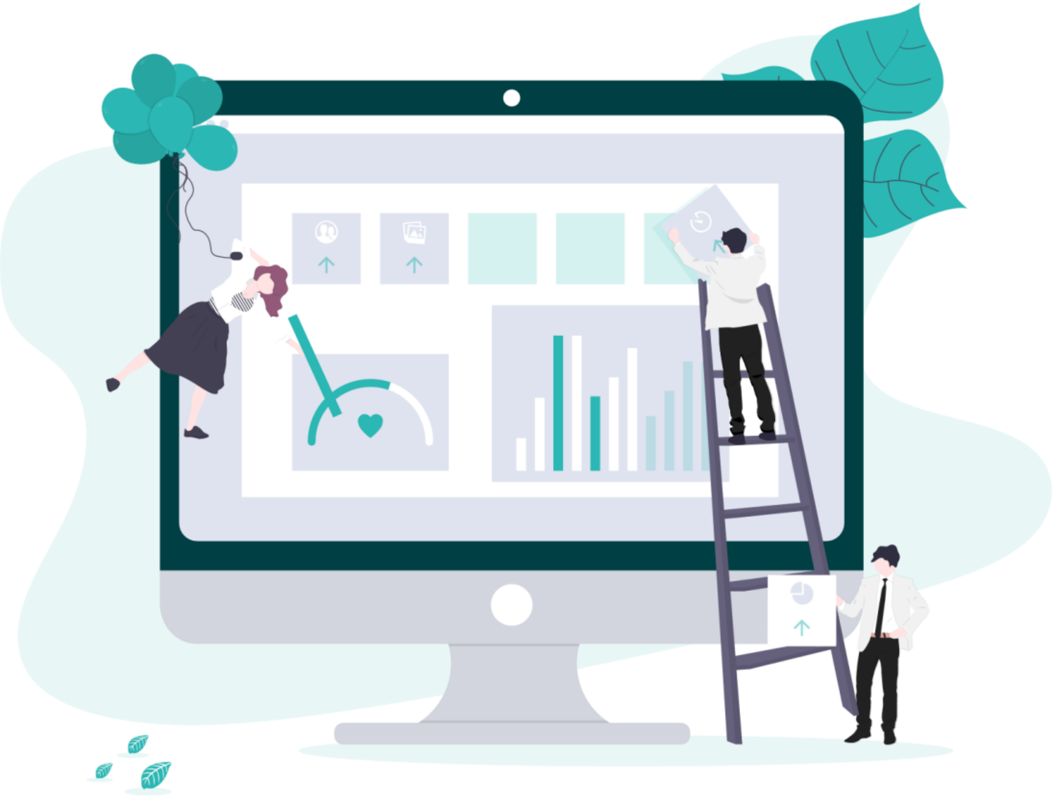
SUSTech
Southern University of Science and Technology

WEI — Brute-force and Strassen Algorithm — 25%

HAN — Pseudocode and Parallel Algorithm — 25%

LI — Parallel Optimizing and Debugging — 25%

LUO — Basic Matrix Method and Paper — 25%



Asymptotic Bounds

Square Matrix Multiply Algorithm

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

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

where

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$$
$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$
$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$$
$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$$

Then

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

Asymptotic Bounds

Strassen Algorithm

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

$$S_1 = B_{12} - B_{22}, \quad S_2 = A_{11} + A_{12}, \quad S_3 = A_{21} + A_{22}, \quad S_4 = B_{21} - B_{11}, \quad S_5 = A_{11} + A_{22},$$

$$S_6 = B_{11} + B_{22}, \quad S_7 = A_{12} - A_{22}, \quad S_8 = B_{21} + B_{22}, \quad S_9 = A_{11} - A_{21}, \quad S_{10} = B_{11} + B_{12}$$

$$P_1 = A_{11} \cdot S_1, \quad P_2 = S_2 + B_{22}, \quad P_3 = S_3 \cdot B_{11}, \quad P_4 = A_{22} \cdot S_4, \quad P_5 = S_5 \cdot S_6, \quad P_6 = S_7 \cdot S_8, \quad P_7 = S_9 \cdot S_{10}$$

$$C_{11} = P_5 + P_4 - P_2 + P_6, \qquad C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4, \qquad C_{22} = P_5 + P_1 - P_3 - P_7$$

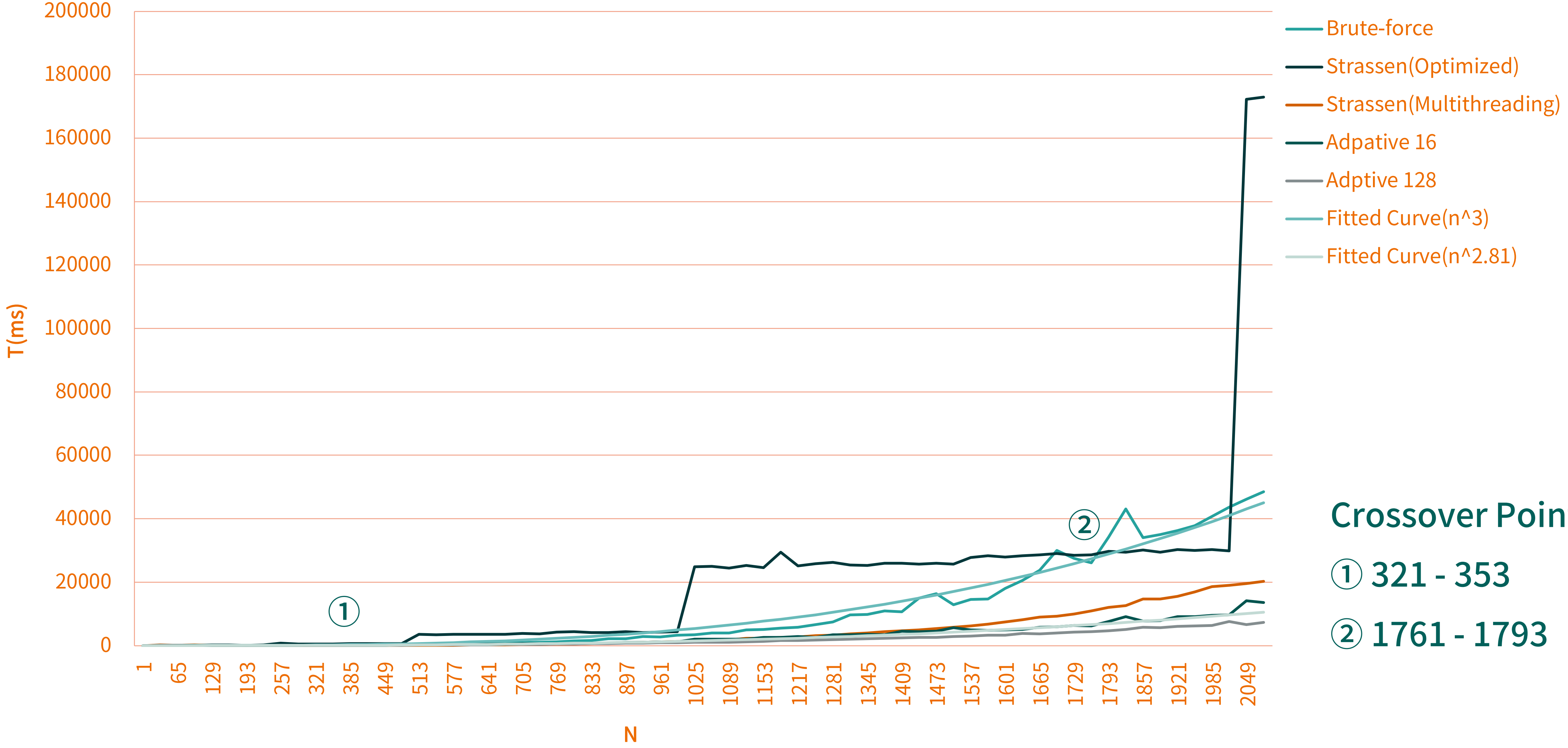
Then

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ 7T\left(\frac{n}{2}\right) + \Theta(n^2), & n > 1 \end{cases}$$

Compare Runtime

Compare Runtime

Southern University of Science and Technology



Crossover Point:

① 321 - 353

② 1761 - 1793

4 Conclusion

CONCLUSION

In this project, we first implemented the brute-force algorithm of matrix multiplication and Strassen algorithm. After that, the existing algorithms are optimized from two directions of multithreading and code structure, and more ideal results are obtained. After comparison, when the matrix size is small, the brute-force algorithm is faster. But when the matrix size is large, the Strassen algorithm has greater advantages. In addition, the parallel algorithm can improve the utilization of resources and speed up the computation significantly. Finally, we propose a method that can select different algorithms according to the matrix itself, so as to improve the operational efficiency of matrix multiplication.

SUSTech Southern University of Science and Technology

Thanks
FOR LISTENING

December 21, 2020