ALGORITHMIC DESIGN - HOMEWORK 1 - MATRIX MULTIPLICATION

KEERTHANA C J

September 2020

1 Generalization of Strassen

The strassen algorithm focuses on square matrix multiplication having dimensions in the powers of 2. As for the first step in generalization, the strassen algorithm was generalized for square matrices of any dimesion. For this purpose a dynamic peeling strategy is used. The dynamic peeling has been proved as more efficient than any padding technique.[1]. In the next step, the strassen algorithm is generalized for rectangular matrices. The following method is opted for the rectangular matrices:

- A rectangular matrix multiplication takes the form $A^{m\times k} \times B^{k\times n}$ resulting in matrix $C^{m\times n}$
- The matrices A and B are split according to the minimum among the 3 dimensions.
- Let us suppose that m is the minimum among the 3 dimensions. Split matrix A and B with m in each dimension. The following would be obtained:

$$\left[\begin{array}{c|c} A_{11} = A^{m\times m} & A_{12} = A^{m\times (k-m)} \end{array}\right] \tag{1}$$

$$\left[\begin{array}{c|c} B_{11} = B^{m\times m} & B_{12} = B^{k\times (n-m)} \\ \hline B_{21} = B^{(k-m)\times m} & B_{22} = B^{(k-m)\times (n-m)} \end{array}\right] \tag{2}$$

- This results to a block matrix multiplication of dimension $A^{1\times 2}$ and $B^{2\times 2}$. The block multiplication is done as per the naive algorithm resulting in the C matrix as follows:

$$\left[\begin{array}{c|c} C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} & C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22} \end{array}\right] \tag{3}$$

- The spiltting of the matrix varies as per the minimum dimension.If k is the minimum dimension, the matrix is split as:

$$\left[\begin{array}{c} A_{11} = A^{k\times k} \\ \hline A_{21} = A^{(m-k)\times k} \end{array}\right] \tag{4}$$

$$\left[\begin{array}{c|c} B_{11} = B^{k\times k} & B_{12} = B^{k\times(n-k)} \end{array}\right] \tag{5}$$

- If n is the minimum dimension, then we have as follows:

$$\left[\begin{array}{c|c} A_{11} = A^{n\times n} & A_{12} = A^{n\times(k-n)} \\ \hline A_{21} = A^{(m-n)\times n} & A_{22} = A^{(m-n)\times(k-n)} \end{array}\right] \tag{6}$$

$$\left[\begin{array}{c} B_{11} = B^{n\times n} \\ \hline B_{21} = B^{(k-n)\times n} \end{array}\right] \tag{7}$$

- Depending on the minimum dimension, the A and B matrix dimension gets reduced as 1×2 or 2×1 or 2×2.
- The $(1, 1)$ square blocks can always be multiplied using strassen and the rectangular blocks can be multiplied by splitting recursively using this nonsquare strassen method.

Using the above methodology, different matrix dimensions were tested and the following results were obtained.

m	k	n	Generalized Strassen	Naive
234	785	547	0.111541	0.139643
1734	2285	547	1.766239	11.859441
1734	785	3547	4.044999	8.460419
1734	3785	2047	10.695219	84.513829
3234	3785	3547	28.697538	254.150779

Table 1: Comparison of Generalized Strassen & Naive algorithms for various matrix dimensions

2 Reducing Memory allocations on Strassen

The major memory consumption of Strassen occurs when we allocate matrices recursive calls (P matrices)and the S matrices. In each recursive call, we allocate 10 S matrices and 7 P matrices of size equalling to half of the matrix dimension involved in each recursive call. This can be reduced by allocating lesser matrices for each call. The following are used for reducing the memory allocation

- Allocating less matrices means, using the C matrix for storing instead allocating new matrices.For example: Of the 7 P matrices used, 4 P matrices can be used from the memory used for C matrix.The remaining 3 P matrices have to allocated.
- The next option is to reduced the space used for S matrices. Each S matrix is not of use after its calculation purpose is over. Hence, only 2 S matrices are needed to be allocated simultaneously. The S matrix can be made free once its purpose is over.

- Hence, at each recursive call only 5 matrices of dimension equal to half of matrix dimension at each recursive call is active.

The following table shows the effects of reducing memory allocation in the Strassen algorithm

n	Strassen	Memory efficient Strassen
256	0.039241	0.034341
512	0.134955	0.128026
1024	0.977672	0.854484
2048	7.121078	6.646829
4096	53.669534	44.410660

Table 2: Memory efficient Strassen algorithm operates with comparatively lesser time than the original Strassen

References

[1] Steven Huss-Lederman et al. "Implementation of Strassen's Algorithm for Matrix Multiplication". In: *Proceedings of the 1996 ACM/IEEE Conference on Supercomputing.* Supercomputing '96. Pittsburgh, Pennsylvania, USA: IEEE Computer Society, 1996, 32–es. ISBN: 0897918541. DOI: `10.1145/369028.369096`. URL: `https://doi.org/10.1145/369028.369096`.