Generalizing of a High Performance Parallel Strassen Implementation on Distributed Memory MIMD Architectures

Duc Kien Nguyen[1], Ivan Lavallee[2], Marc Bui[2]

[1]CHArt - Ecole Pratique des Hautes Etudes & Université Paris 8, France
Kien.Duc-Nguyen@univ-paris8.fr

[2]LaISC - Ecole Pratique des Hautes Etudes, France
Ivan.Lavallee@ephe.sorbonne.fr
Marc.Bui@ephe.sorbonne.fr

Abstract: Strassen's algorithm to multiply two $n \times n$ matrices reduces the asymptotic operation count from $O(n^3)$ of the traditional algorithm to $O(n^{2.81})$, thus designing efficient parallelizing for this algorithm becomes essential. In this paper, we present our generalizing of a parallel Strassen implementation which obtained a very nice performance on an Intel Paragon: faster 20% for $n \approx 1000$ and more than 100% for $n \approx 5000$ in comparison to the parallel traditional algorithms (as Fox, Cannon). Our method can be applied to all the matrix multiplication algorithms on distributed memory computers that use Strassen's algorithm at the system level, hence it gives us compatibility to find better parallel implementations of Strassen's algorithm.

1 Introduction

Matrix multiplication (MM) is one of the most fundamental operations in linear algebra and serves as the main building block in many different algorithms, including the solution of systems of linear equations, matrix inversion, evaluation of the matrix determinant and the transitive closure of a graph. In several cases the asymptotic complexities of these algorithms depend directly on the complexity of matrix multiplication - which motivates the study of possibilities to speed up matrix multiplication. Also, the inclusion of matrix multiplication in many benchmarks points at its role as a determining factor for the performance of high speed computations.

Strassen was the first to introduce a better algorithm [Str69] for MM with $O(N^{\log_2 7})$than the traditional one which needs $O(N^3)$ operations. Then Winograd variant [Win71] of Strassen's algorithm has the same exponent but a slightly lower constant as the number of additions/subtractions is reduced from 18 down to 15. The record of complexity owed to Coppersmith and Winograd is $O(N^{2.376})$, resulted from arithmetic aggregation [CW90]. However, only Winograd's algorithm and Strassen's algorithm offer better performance than traditional algorithm for matrices of practical sizes, say, less than 10^{20}[LPS92]. The

full potential of these algorithms can be realized only on large matrices, which require large machines such as parallel computers. Thus, designing efficient parallel implementations for theses algorithms becomes essential.

This research was started when a paper by Chung-Chiang Chou, Yuefan Deng, Gang Li, and Yuan Wang [CDLW95] on the Strassen parallelizing came to our attention. Their implementation already obtained a nice performance: in comparison to the parallel traditional algorithms (as Fox, Cannon) on an Intel Paragon, it's faster 20% for $n \approx 1000$ and more than 100% for $n \approx 5000$. The principle of this implementation is to parallelize the Strassen's algorithm at the system level - i.e. to stop on the recursion level r of execution tree - and the calculation of the products of sub matrices is locally performed by the processors. The most significant point here is to determine the sub matrices after having recursively executed r time the Strassen's formula (these sub matrices are corresponding to the nodes of level r in the execution tree of Strassen's algorithm) and then to find the result matrix from these sub matrices (corresponding to the process of backtracking the execution tree). It is simple to solve this problem for a sequential machine, but it's much harder for a parallel machine. With a definite value of r, we can manually do it like [CDLW95], [LD95], and [GSv96] made (r = 1, 2) but the solution for the general case has not been found.

In this paper, we present our method to determine all the nodes at the unspecified level r in the execution tree of Strassen's algorithm, and to show the expression representing the relation between the result matrix and the sub matrices at the level recursion r; this expression allows us to calculate directly the result matrix from the sub matrices calculated by parallel matrix multiplication algorithms at the bottom level. By combining this result with the matrix multiplication algorithms at the bottom level, we have a generalizing of the high performance parallel Strassen implementation in [CDLW95]. It can be applied to all the matrix multiplication algorithms on distributed memory computers that use Strassen's algorithm at the system level, besides the running time for these algorithms decreases when the recursion level increases hence this general solution gives us compatibility to find better implementations (which correspond with a definite value of the recursive level and a definite matrix multiplication algorithm at the bottom level).

2 Background

2.1 Strassen's Algorithm

We start by considering the formation of the matrix product $Q = XY$, where $Q \in \Re^{m\times n}$, $X \in \Re^{m\times k}$, and $Y \in \Re^{k\times n}$. We will assume that m, n, and k are all even integers. By partitioning

$$X = \begin{pmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{pmatrix}, Y = \begin{pmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{pmatrix}, Q = \begin{pmatrix} Q_{00} & Q_{01} \\ Q_{10} & Q_{11} \end{pmatrix}$$

where $Q_{ij} \in \Re^{\frac{m}{2} \times \frac{n}{2}}$, $X_{ij} \in \Re^{\frac{m}{2} \times \frac{k}{2}}$, and $Y_{ij} \in \Re^{\frac{k}{2} \times \frac{n}{2}}$, it can be shown [Win71, GL89] that the following computations compute $Q = XY$:

$$\begin{aligned}
M_0 &= (X_{00} + M_{11})(Y_{00} + Y11) \\
M_1 &= (X_{10} + X_{11})Y_{00} \\
M_2 &= X_{00}(Y_{01} - Y_{11}) \\
M_3 &= X_{11}(-Y_{00} + Y_{10}) \\
M_4 &= (X_{00} + X_{01})Y_{11} \\
M_5 &= (X_{10} - X_{00})(Y_{00} + Y_{01}) \\
M_6 &= (X_{01} - X_{11})(Y_{10} + Y_{11}) \\
Q_{00} &= M_0 + M_3 - M_4 + M_6 \\
Q_{01} &= M_1 + M_3 \\
Q_{10} &= M_2 + M_4 \\
Q_{11} &= M_0 + M_2 - M_1 + M_5
\end{aligned} \tag{1}$$

The Strassen's algorithm does the above computation recursively until one of the dimensions of the matrices is 1.

2.2 A High Performance Parallel Strassen Implementation

In this section, we will see the principle of the hight performance parallel Strassen implementation presented in [CDLW95], which is foundation for our generalizing.

First, decompose the matrix X into 2×2 blocks of sub matrices X_{ij} where $i, j = 0, 1$. Seconde, decompose further these four sub matrices into four 2×2 (i.e. 4×4) blocks of sub matrices x_{ij} where $i, j = 0, 1, 2, 3$.

$$X = \begin{pmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{pmatrix} = \begin{pmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{pmatrix}$$

Similarly, perform the same decomposition on matrix Y and get:

$$Y = \begin{pmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{pmatrix} = \begin{pmatrix} y_{00} & y_{01} & y_{02} & y_{03} \\ y_{10} & y_{11} & y_{12} & y_{13} \\ y_{20} & y_{21} & y_{22} & y_{23} \\ y_{30} & y_{31} & y_{32} & y_{33} \end{pmatrix}$$

Then, use the Strassen's formula to multiply the matrices X and Y, and get the following

seven matrix multiplication expressions:

$$\begin{cases} M_0 = (X_{00} + X_{11})(Y_{00} + Y_{11}) \\ M_1 = (X_{10} + X_{11})Y_{00} \\ M_2 = X_{00}(Y_{01} - Y_{11}) \\ M_3 = X_{11}(-Y_{00} + Y_{10}) \\ M_4 = (X_{00} + X_{01})Y_{11} \\ M_5 = (-X_{00} + X_{10})(Y_{00} + Y_{01}) \\ M_6 = (X_{01} - X_{11})(Y_{10} + Y_{11}) \end{cases}$$

Next, apply the Strassen's formula to these seven expressions to obtain 49 matrix multiplication expressions on sub matrices x and y. Taking M_0 as an example:

$$\begin{aligned} M_{00} &= (x_{00} + x_{22} + x_{11} + x_{33})\,(y_{00} + y_{22} + y_{11} + y_{33}) \\ M_{01} &= (x_{10} + x_{32} + x_{11} + x_{33})\,(y_{00} + y_{22}) \\ M_{02} &= (x_{00} + x_{22})\,(y_{01} + y_{23} - y_{11} - y_{33}) \\ M_{03} &= (x_{11} + x_{33})\,(y_{10} + y_{32} - y_{00} - y_{22}) \\ M_{04} &= (x_{00} + x_{22} + x_{01} + x_{23})\,(y_{11} + y_{33}) \\ M_{05} &= (x_{10} + x_{32} - x_{00} - x_{22})\,(y_{00} + y_{22} + y_{01} + y_{23}) \\ M_{06} &= (x_{01} + x_{23} - x_{11} - x_{33})\,(y_{10} + y_{32} + y_{11} + y_{33}) \end{aligned}$$

Similarly, each of the remaining six matrix multiplication expressions M_i for $i = 1, 2, \ldots, 6$ can also be expanded into six groups of matrix multiplications in terms of x and y.

$$\begin{aligned} M_{10} &= (x_{20} + x_{22} + x_{31} + x_{33})\,(y_{00} + y_{11}) \\ M_{11} &= (x_{30} + x_{32} + x_{31} + x_{33})\,y_{00} \\ M_{12} &= (x_{20} + x_{22})\,(y_{01} - y_{11}) \\ M_{13} &= (x_{31} + x_{33})\,(y_{10} - y_{00}) \\ M_{14} &= (x_{20} + x_{22} + x_{21} + x_{23})\,y_{11} \\ M_{15} &= (x_{30} + x_{32} - x_{20} - x_{22})\,(y_{00} + y_{01}) \\ M_{16} &= (x_{21} + x_{23} - x_{31} - x_{33})\,(y_{10} + y_{11}) \end{aligned}$$

$$\begin{aligned} M_{20} &= (x_{00} + x_{11})\,(y_{02} - y_{22} + y_{13} - y_{33}) \\ M_{21} &= (x_{10} + x_{11})\,(y_{02} - y_{22}) \\ M_{22} &= x_{00}\,(y_{03} - y_{23} - y_{13} + y_{33}) \\ M_{23} &= x_{11}\,(y_{12} - y_{32} - y_{02} + y_{22}) \\ M_{24} &= (x_{00} + x_{01})\,(y_{13} - y_{33}) \\ M_{25} &= (x_{10} - x_{00})\,(y_{02} - y_{22} + y_{03} - y_{23}) \\ M_{26} &= (x_{01} - x_{11})\,(y_{12} - y_{32} + y_{13} - y_{33}) \end{aligned}$$

$$\begin{aligned} M_{30} &= (x_{22} + x_{33})\,(y_{20} - y_{00} + y_{31} - y_{11}) \\ M_{31} &= (x_{32} + x_{33})\,(y_{20} - y_{00}) \\ M_{32} &= x_{22}\,(y_{21} - y_{01} - y_{31} + y_{11}) \\ M_{33} &= x_{33}\,(y_{30} - y_{10} - y_{20} + y_{00}) \\ M_{34} &= (x_{22} + x_{23})\,(y_{31} - y_{11}) \\ M_{35} &= (x_{32} - x_{22})\,(y_{20} - y_{00} + y_{21} - y_{01}) \\ M_{36} &= (x_{22} + x_{33})\,(y_{30} - y_{10} + y_{31} - y_{11}) \end{aligned}$$

$$
\begin{aligned}
M_{40} &= (x_{00} + x_{02} + x_{11} + x_{13})(y_{22} + y_{33}) \\
M_{41} &= (x_{10} + x_{12} + x_{11} + x_{13})\, y_{22} \\
M_{42} &= (x_{00} + x_{02})(y_{23} - y_{33}) \\
M_{43} &= (x_{11} + x_{13})(y_{32} - y_{22}) \\
M_{44} &= (x_{00} + x_{02} + x_{01} + x_{03})\, y_{33} \\
M_{45} &= (x_{10} + x_{13} - x_{00} - x_{02})(y_{22} + y_{23}) \\
M_{46} &= (x_{01} + x_{03} - x_{11} - x_{13})(y_{32} + y_{33})
\end{aligned}
$$

$$
\begin{aligned}
M_{50} &= (x_{20} - x_{00} + x_{31} - x_{11})(y_{00} + y_{02} + y_{11} + y_{13}) \\
M_{51} &= (x_{30} - x_{10} + x_{31} - x_{11})(y_{00} + y_{02}) \\
M_{52} &= (x_{20} - x_{00})(y_{01} + y_{03} - y_{11} - y_{13}) \\
M_{53} &= (x_{31} - x_{11})(y_{10} + y_{12} - y_{00} - y_{02}) \\
M_{54} &= (x_{20} - x_{00} + x_{21} - x_{01})(y_{11} + y_{13}) \\
M_{55} &= (x_{30} - x_{10} - x_{20} + x_{00})(y_{00} + y_{02} + y_{01} + y_{03}) \\
M_{56} &= (x_{21} - x_{01} - x_{31} + x_{11})(y_{10} + y_{12} + y_{11} + y_{13})
\end{aligned}
$$

$$
\begin{aligned}
M_{60} &= (x_{02} - x_{22} + x_{13} - x_{22})(y_{20} + y_{22} + y_{31} + y_{33}) \\
M_{61} &= (x_{12} - x_{32} + x_{13} - x_{33})(y_{20} + y_{22}) \\
M_{62} &= (x_{02} - x_{22})(y_{21} + y_{23} - y_{31} - y_{33}) \\
M_{63} &= (x_{13} - x_{33})(y_{30} + y_{32} - y_{20} - y_{22}) \\
M_{64} &= (x_{02} - x_{22} + x_{03} - x_{23})(y_{31} + y_{33}) \\
M_{65} &= (x_{12} - x_{32} - x_{02} + x_{22})(y_{20} + y_{22} + y_{21} + y_{23}) \\
M_{66} &= (x_{03} - x_{23} - x_{13} + x_{33})(y_{30} + y_{32} + y_{31} + y_{33})
\end{aligned}
$$

After finishing these 49 matrix multiplications, we need to combine the resulting M_{ij} where $i, j = 0, 1, \ldots, 6$ to form the final product matrix.

$$
Q = \begin{pmatrix} Q_{00} & Q_{01} \\ Q_{10} & Q_{11} \end{pmatrix} = \begin{pmatrix} q_{00} & q_{01} & q_{02} & q_{03} \\ q_{10} & q_{11} & q_{12} & q_{13} \\ q_{20} & q_{21} & q_{22} & q_{23} \\ q_{30} & q_{31} & q_{32} & q_{33} \end{pmatrix}
$$

First, define some variables $\delta_i = \begin{cases} -1, \text{ if } i = 4 \\ 1 \text{ otherwise} \end{cases}$ and $\gamma_i = \begin{cases} -1, \text{ if } i = 1 \\ 1 \text{ otherwise} \end{cases}$, the 4 x 4 blocks of sub matrices forming the product matrix Q can be written as:

$$
\begin{aligned}
q_{00} &= \textstyle\sum_{i \in S_1} \delta_i \left(M_{i0} + M_{i3} - M_{i4} + M_{i6}\right) \\
q_{01} &= \textstyle\sum_{i \in S_1} \delta_i \left(M_{i2} + M_{i4}\right) \\
q_{02} &= \textstyle\sum_{i \in S_3} M_{i0} + M_{i3} - M_{i4} + M_{i6} \\
q_{03} &= \textstyle\sum_{i \in S_3} M_{i2} + M_{i4}
\end{aligned}
$$

$$
\begin{aligned}
q_{10} &= \textstyle\sum_{i \in S_1} \delta_i \left(M_{i1} + M_{i3}\right) \\
q_{11} &= \textstyle\sum_{i \in S_1} \delta_i \left(M_{i0} + M_{i2} - M_{i1} + M_{i5}\right) \\
q_{12} &= \textstyle\sum_{i \in S_3} M_{i1} + M_{i3} \\
q_{13} &= \textstyle\sum_{i \in S_3} M_{i0} + M_{i2} - M_{i1} + M_{i5}
\end{aligned}
$$

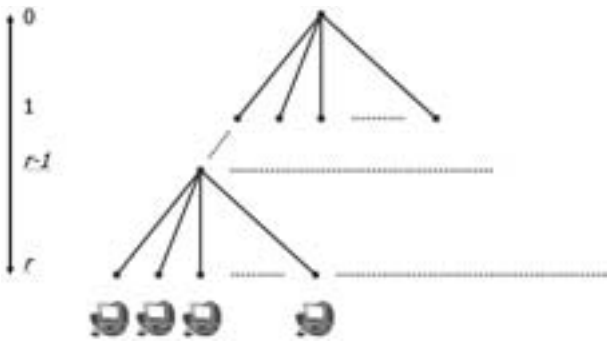

Figure 1: Principle of the Strassen parallelizing in [CDLW95].

$$
\begin{aligned}
q_{20} &= \textstyle\sum_{i \in S_2} M_{i0} + M_{i3} - M_{i4} + M_{i6} \\
q_{21} &= \textstyle\sum_{i \in S_2} M_{i2} + M_{i4} \\
q_{22} &= \textstyle\sum_{i \in S_4} \gamma_i \left(M_{i0} + M_{i3} - M_{i4} + M_{i6} \right) \\
q_{23} &= \textstyle\sum_{i \in S_4} \gamma_i \left(M_{i2} + M_{i4} \right)
\end{aligned}
$$

$$
\begin{aligned}
q_{30} &= \textstyle\sum_{i \in S_2} M_{i1} + M_{i3} \\
q_{31} &= \textstyle\sum_{i \in S_2} M_{i0} + M_{i2} - M_{i1} + M_{i5} \\
q_{32} &= \textstyle\sum_{i \in S_4} \gamma_i \left(M_{i1} + M_{i3} \right) \\
q_{33} &= \textstyle\sum_{i \in S_4} \gamma_i \left(M_{i0} + M_{i2} - M_{i1} + M_{i5} \right)
\end{aligned}
$$

As you saw above, it is not very simple although they have only 49 matrix multiplications. It become great complicated if we want to go further - when we have 343, 2401 or more matrix multiplications.

3 Generalizing of the Parallel Strassen Implementation

The principle of the method that has been presented is to parallelize the Strassen's algorithm at the system level - i.e. to stop on the recursion level r of execution tree - and the calculation of the products of sub matrices is locally performed by the processors. The most important point here is to determine the sub matrices after having applied r time the Strassen's formula, and to find the result matrix from the products of these sub matrices. In the preceding works, the solutions are given with fixed values of r (= 1, 2). But the solution for the general case has not been found.

Such are the problems with which we are confronted and the solution will be presented in this section.

3.1 Recursion Removal in Fast Matrix Multiplication

We represent the Strassen's formula:

$$\begin{aligned} &m_l = \sum_{i,j=0,1} x_{ij} SX(l,i,j) \times \sum_{i,j=0,1} y_{ij} SY(l,i,j) \\ &l = 0 \cdots 6 \\ &\text{and } q_{ij} = \sum_{l=0}^{6} m_l SQ(l,i,j) \end{aligned} \tag{2}$$

with

SX=

Pij	00	01	10	11
0	-1	0	1	1
1	1	0	0	0
2	0	1	0	0
3	1	0	-1	0
4	0	0	1	1
5	1	1	-1	-1
6	0	0	0	1

SY=

Pij	00	01	10	11
0	1	-1	0	1
1	1	0	0	0
2	0	0	1	0
3	0	-1	0	1
4	0	1	0	-1
5	0	0	0	1
6	1	-1	-1	1

SQ=

Pij	00	01	10	11
0	0	1	1	1
1	1	1	1	1
2	1	0	0	0
3	0	0	1	1
4	0	1	0	1
5	0	1	0	0
6	0	0	-1	0

Each of 7^k product can be represented as in the following:

$$\begin{aligned} &m_l = \sum_{i,j=0,n-1} x_{ij} SX_k(l,i,j) \times \sum_{i,j=0,n-1} y_{ij} SY_k(l,i,j) \\ &l = 0...7^k - 1 \\ &\text{and } q_{ij} = \sum_{l=0}^{7^k-1} m_l SQ_k(l,i,j) \end{aligned} \tag{3}$$

In fact, $SX = SX_1, SY = SY_1, SQ = SQ_1$. Now we have to determine values of matrices SX_k, SY_k, and SQ_k from SX_1, SY_1, and SQ_1. In order to obtain this, we extend the definition of tensor product in [KHJS90] for arrays of arbitrary dimensions as followed:

Definition. Let A and B are arrays of same dimension l and of size $m_1 \times m_2 \times \ldots \times m_l, n_1 \times n_2 \times \ldots \times n_l$ respectively. Then the tensor product (TP) is an array of same dimension and of size $m_1 n_1 \times m_2 n_2 \times \ldots \times m_l n_l$ defined by replacing each element of A with the product of the element and B.
$P = A \otimes B$ where $P[i_1, i_2, ..., i_l] = A[k_1, k_2, ..., k_l] B[h_1, h_2, ..., h_l], i_j = k_j n_j + h_j$ with $\forall 1 \leq j \leq l$;

Let $P = \bigotimes_{i=1}^{n} A_i = (...(A_1 \otimes A_2) \otimes A_3)... \otimes A_n)$ with A_i is array of dimension l and of size $m_{i1} \times m_{i2} \times \ldots \times m_{il}$. The following theorem allows computing directly elements of P

Theorem.

$$\begin{aligned} P[j_1, j_2, ..., j_l] &= \prod_{i=1}^{n} A_i[h_{i1}, h_{i2}, ..., h_{il}] \\ \text{where } j_k &= \sum_{s=1}^{n} \left(h_{sk} \prod_{r=s+1}^{n} m_{rk} \right). \end{aligned} \tag{4}$$

Proof. We prove the theorem by induction. With $n = 1$, the proof is trivial. With $n = 2$, it is true by the definition. Suppose it is true with $n - 1$. We show that it is true with n.

We have $P_{n-1}[t_1, t_2, ..., t_l] = \prod_{i=1}^{n-1} A_i[h_{i1}, h_{i2}, ..., h_{il}]$ where $t_k = \sum_{s=1}^{n-1} \left(h_{sk} \prod_{r=s+1}^{n-1} m_{rk} \right)$ with $\forall 1 \leq k \leq l$; and then $P_n = P_{n-1} \otimes A_n$.

By definition

$P_n[j_1, j_2, ..., j_l] = P_{n-1}[p_1, p_2, ..., p_l] A_n[h_{n1}, h_{n2}, ..., h_{nl}] = \prod_{i=1}^{n} A_i[h_{i1}, h_{i2}, ..., h_{il}]$

where $j_k = p_k m_{nk} + h_{nk} = m_{nk} \times \sum_{s=1}^{n-1} \left(h_{sk} \prod_{r=s+1}^{n-1} m_{rk} \right) + h_{nk}$

$= \sum_{s=1}^{n-1} \left(h_{sk} \prod_{r=s+1}^{n} m_{rk} \right) + h_{nk} = \sum_{s=1}^{n} \left(h_{sk} \prod_{r=s+1}^{n} m_{rk} \right)$

The theorem is proved. □

In particular, if all A_i have the same size $m_1 \times m_2 \times \ldots \times m_l$, we have $P[j_1, j_2, ..., j_l] = \prod_{i=1}^{n} A_i[h_{i1}, h_{i2}, ..., h_{il}]$ where $j_k = \sum_{s=1}^{n} \left(h_{sk} m_k^{n-s} \right)$.

Remark. $j_k = \sum_{s=1}^{n} \left(h_{sk} m_k^{n-s} \right)$ is a j_k's factorization in base m_k. We note $a = \overline{a_1 a_2 ... a_l}_{(b)}$ the a's factorization in base b hence $P[j_1, j_2, ..., j_l] = \prod_{i=1}^{n} A_i[h_{i1}, h_{i2}, ..., h_{il}]$ then $j_k = \overline{h_{i1} h_{i2} ... h_{in}}_{(m_k)}$.

Now we return to our algorithm. We have following theorem:

Theorem.

$$\begin{aligned} SX_k &= \bigotimes_{i=1}^{k} SX \\ SY_k &= \bigotimes_{i=1}^{k} SY \\ SQ_k &= \bigotimes_{i=1}^{k} SQ \end{aligned} \tag{5}$$

Proof. We prove the theorem by induction. Clearly it is true with $k = 1$. Suppose it is true with $k - 1$. The algorithm's execution tree is balanced with depth k and degree 7. Thanks to (3), we have at the level $k - 1$ of the tree:

$$\begin{aligned} &M_l = \\ &\left(\sum_{0 \leq i,j \leq 2^{k-1}-1} X_{k-1,ij} SX_{k-1}(l, i, j) \right) \times \left(\sum_{0 \leq i,j \leq 2^{k-1}-1} Y_{k-1,ij} SY_{k-1}(l, i, j) \right) \\ &0 \leq l \leq 7^{k-1} - 1 \end{aligned}$$

Then thanks to (2) at the level k we have

$$
\begin{array}{l}
M_l[l'] = \\
\sum\limits_{0\le i',j'\le 1}\left(\left(\sum\limits_{0\le i,j\le 2^{k-1}-1} X_{k-1,ij}[i',j']SX_{k-1}\left(l,i,j\right)\right)SX(l',i',j')\right)\times \\
\sum\limits_{0\le i',j'\le 1}\left(\left(\sum\limits_{0\le i,j\le 2^{k-1}-1} Y_{k-1,ij}[i',j']SY_{k-1}\left(l,i,j\right)\right)SY(l',i',j')\right) \\
0\le l\le 7^{k-1}-1 \\
0\le l'\le 6 \\
= \\
\sum\limits_{0\le i',j'\le 1}\left(\sum\limits_{0\le i,j\le 2^{k-1}-1} X_{k-1,ij}[i',j']SX_{k-1}\left(l,i,j\right)SX(l',i',j')\right)\times \\
\sum\limits_{0\le i',j'\le 1}\left(\sum\limits_{0\le i,j\le 2^{k-1}-1} Y_{k-1,ij}[i',j']SY_{k-1}\left(l,i,j\right)SY(l',i',j')\right) \\
0\le l\le 7^{k-1}-1 \\
0\le l'\le 6
\end{array}
\tag{6}
$$

where $X_{k-1,ij}[i',j'], Y_{k-1,ij}[i',j']$ are $2^k\times 2^k$ matrices obtained by division $X_{k-1,ij}, Y_{k-1,ij}$ in 4 sub matrices (i',j' indicate the sub matrix's quarter).

We present l,l' in the base 7, and i,j,i',j' in the base 2 and remark that $X_{k-1,ij}\left[i',j'\right] = X_k\left[\overline{ii'_2},\overline{jj'_2}\right]$. Then (6) becomes

$$
\begin{array}{l}
M[\overline{ll'}_{(7)}] = \\
\left(\sum\limits_{0\le \overline{ii'}_{(2)},\overline{jj'}_{(2)}\le 2^{k-1}-1} X_k[\overline{ii'}_{(2)},\overline{jj'}_{(2)}]SX_{k-1}\left(l,i,j\right)SX(l',i',j')\right)\times \\
\left(\sum\limits_{0\le \overline{ii'}_{(2)},\overline{jj'}_{(2)}\le 2^{k-1}-1} Y_k[\overline{ii'}_{(2)},\overline{jj'}_{(2)}]SY_{k-1}\left(l,i,j\right)SY(l',i',j')\right) \\
0\le \overline{ll'}_{(7)}\le 7^{k-1}-1
\end{array}
\tag{7}
$$

In addition, we have directly from (3):

$$
\begin{array}{l}
M[\overline{ll'}_{(7)}] = \\
\left(\sum\limits_{0\le \overline{ii'}_{(2)},\overline{jj'}_{(2)}\le 2^{k-1}-1} X_k[\overline{ii'}_{(2)},\overline{jj'}_{(2)}]SX_k\left(\overline{ll'}_{(7)},\overline{ii'}_{(2)},\overline{jj'}_{(2)}\right)\right)\times \\
\left(\sum\limits_{0\le \overline{ii'}_{(2)},\overline{jj'}_{(2)}\le 2^{k-1}-1} Y_k[\overline{ii'}_{(2)},\overline{jj'}_{(2)}]SY_k\left(\overline{ll'}_{(7)},\overline{ii'}_{(2)},\overline{jj'}_{(2)}\right)\right) \\
0\le \overline{ll'}_{(7)}\le 7^{k-1}-1
\end{array}
\tag{8}
$$

Compare (7) and (8) we have

$$
\begin{array}{l}
SX_k\left(\overline{ll'_7},\overline{ii'_2},\overline{jj'_2}\right) = SX_{k-1}\left(l,i,j\right)SX\left(l',i',j'\right) \\
SY_k\left(\overline{ll'_7},\overline{ii'_2},\overline{jj'_2}\right) = SY_{k-1}\left(l,i,j\right)SY\left(l',i',j'\right)
\end{array}
$$

By definition, we have

$$SX_k = SX_{k-1} \otimes SX = \overset{k}{\underset{i=1}{\otimes}} SX$$
$$SY_k = SY_{k-1} \otimes SY = \overset{k}{\underset{i=1}{\otimes}} SY$$

Similarly

$$SQ_k = SQ_{k-1} \otimes SQ = \overset{k}{\underset{i=1}{\otimes}} SQ$$

The theorem is proved. □

Thanks to Theorem 3.1 and Remark 3.1 we have

$$\begin{aligned} SX_k(l,i,j) &= \prod_{r=1}^{k} SX(l_r, i_r, j_r) \\ SY_k(l,i,j) &= \prod_{r=1}^{k} SY(l_r, i_r, j_r) \\ SQ_k(l,i,j) &= \prod_{r=1}^{k} SQ(l_r, i_r, j_r) \end{aligned} \tag{9}$$

Apply (9) in (3) we have nodes leafs m_l and all the elements of result matrix.

3.2 Generalizing

Now we known how to parallelize Strassen's algorithm in general case: firstwe stop at therecursion level r, thanks to the expressions (9) and (3),we have the entire corresponding sub matrices:

$$\begin{aligned} M_l = & \sum_{i,j=0,2^r-1} X_{ij} \left(\prod_{t=1}^{r} SX(l_t, i_t, j_t) \right) \\ & \times \\ & \sum_{\substack{i,j=0,2^r-1 \\ l=0...7^r-1}} Y_{ij} \left(\prod_{t=1}^{r} SY(l_t, i_t, j_t) \right) \end{aligned} \tag{10}$$

with

$$X_{ij} = \begin{pmatrix} x_{i*2^{k-r},j*2^{k-r}} & \dots & x_{i*2^{k-r},j*2^{k-r}+2^{k-r}-1} \\ \dots & \dots & \dots \\ x_{i*2^{k-r}+2^{k-r}-1,j*2^{k-r}} & \dots & x_{i*2^{k-r}+2^{k-r}-1,j*2^{k-r}+2^{k-r}-1} \end{pmatrix}$$
$$Y_{ij} = \begin{pmatrix} y_{i*2^{k-r},j*2^{k-r}} & \dots & y_{i*2^{k-r},j*2^{k-r}+2^{k-r}-1} \\ \dots & \dots & \dots \\ y_{i*2^{k-r}+2^{k-r}-1,j*2^{k-r}} & \dots & y_{i*2^{k-r}+2^{k-r}-1,j*2^{k-r}+2^{k-r}-1} \end{pmatrix}$$
$$i = 0, 2^r - 1, j = 0, 2^r - 1$$

The product M_l of the sub matrices $\left(\sum_{\substack{i=0,2^r-1 \\ j=0,2^r-1}} X_{ij} \left(\prod_{t=1}^{r} SX\left(l_t, i_t, j_t\right) \right) \right)$ and $\left(\sum_{\substack{i=0,2^r-1 \\ j=0,2^r-1}} Y_{ij} \left(\prod_{t=1}^{r} SY\left(l_t, i_t, j_t\right) \right) \right)$ is locally calculated on each processor by the sequential matrix multiplication algorithms.

Finally, thanks to (9) & (3) we have directly sub matrix elements of result matrix by applying matrix additions instead of backtracking manually the recursive tree to calculate the root in [LD95], [CDLW95], and [GSv96]:

$$\begin{aligned} Q_{ij} &= \sum_{l=0}^{7^r-1} M_l SQ_r(l,i,j) \\ &= \sum_{l=0}^{7^r-1} M_l \left(\prod_{t=1}^{r} SQ\left(l_t, i_t, j_t\right) \right) \end{aligned} \tag{11}$$

4 Conclusion

We have presented a general scalable parallelization for all the matrix multiplication algorithms on distributed memory computers that use Strassen's algorithm at inter-processor level. The running time for these algorithms decreases when the recursion level increases hence this general solution gives us compatibility to find better algorithms (which correspond with a definite value of the recursive level and a definite matrix multiplication algorithm at the bottom level). And from a different view, we have generalized the Strassen's formula for the case where the matrices are divided into 2^k parts (the case $k = 2$ gives usoriginal formulas) thus we have a whole new direction to parallelize the Strassen's algorithm. In addition, we are applying these ideas to all the fast matrix multiplication algorithms.

References

[CDLW95] Chung-Chiang Chou, Yuefan Deng, Gang Li, and Yuan Wang. Parallelizing Strassen's Method for Matrix Multiplication on Distributed Memory MIMD architectures. *Computers and Math. with Applications*, 30(2):4–9, 1995.

[CW90] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[GL89] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 2nd edition, 1989.

[GSv96] Brian Grayson, Ajay Shah, and Robert van de Geijn. A High Performance Parallel Strassen Implementation. *Parallel Processing Letters*, 6(1):3–12, 1996.

[KHJS90] B. Kumar, Chua-Huang Huang, Rodney W. Johnson, and P. Sadayappan. A tensor product formulation of Strassen's matrix multiplication algorithm. *Applied Mathematics Letters*, 3(3):67–71, 1990.

[LD95] Qingshan Luo and John B. Drake. A scalable parallel Strassen's matrix multiplication algorithm for distributed memory computers. In *Proceedings of the 1995 ACM symposium on Applied computing*, pages 221–226, Nashville, Tennessee, United States, 1995. ACM Press.

[LPS92] J. Laderman, V. Y. Pan, and H. X. Sha. On Practical Algorithms for Accelerated Matrix Multiplication. *Linear Adgebra and Its Applications*, 162:557–588, 1992.

[Str69] Volker Strassen. Gaussian Elimination is not Optimal. *Numer. Math.*, 13:354–356, 1969.

[Win71] Shmuel Winograd. On multiplication of 2 x 2 matrices. *Linear Algebra and its Applications*, 4:381–388, 1971.