APPLICATION OF STRASSEN'S ALGORITHM IN RHOTRIX ROW-COLUMN MULTIPLICATION

Ezugwu E. Absalom[1], Abdullahi M.[2], Sani B.[3] and Junaidu B. Sahalu[4]
Department of Mathematics, Ahmadu Bello University, Zaria-Nigeria

ABSTRACT

This paper presents a study of Strassen's algorithm and its application to rhotrix row-column multiplication. The special case of odd-sized rhotrices was considered basically by means of both padding and peeling approach. A sequential recursive implementation of the algorithm using java program was done for rhotrices of relatively medium sizes and also a parallel MPI framework implementation was likewise modelled and presented.

1.
2. **Keywords:** Strassen's algorithm, Rhotrix multiplication, Parallel computing

1.0 INTRODUCTION

Rhotrix is a new mathematical paradigm for matrix theory. The study of rhotrix is concerned with the representation of mathematical arrays of real numbers in rhomboidal form like structures, as an extension of ideas on matrix-tertions and matrix noitrets proposed by (Atanassov and Shannon, 1998). We represent the structure of n-dimensional rhotrix as follow.

$$R_n = \left\langle \begin{array}{ccccccc} & & & a_{11} & & & \\ & & a_{21} & & a_{12} & & \\ & a_{31} & & a_{22} & & a_{31} & \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & \\ a_{tt-2} & & a_{t-1t-1} & & a_{t-2t} & & \\ & a_{t-1t-2} & & a_{t-2t-1} & & & \\ & & a_{tt} & & & & \end{array} \right\rangle \qquad (1.1)$$

where: a_{11}, a_{12}, ..., a_{tt} denotes the main entries and c_{11}, c_{12}, ..., $c_{t\text{-}1t\text{-}1}$ in (1.1) denotes the heart entries of the rhotrix (Sani, 2004). A rhotrix would always have an odd dimension.

It is important to note that for any n-dimensional rhotrix denoted by R_n will have $|R_n| = \frac{1}{2}(n^2+1)$, entries (Ajibade, 2003) and $n \in 2Z+1$, thus we reiterate that all rhotrices are of odd dimension. Accordingly, the 5th and 7th dimension are given as:

$$R_5 = \left\langle \begin{matrix} & & a_{11} & & \\ & a_{12} & c_{11} & a_{12} & \\ a_{31} & c_{12} & a_{22} & c_{12} & a_{13} \\ & a_{31} & c_{22} & a_{23} & \\ & & a_{33} & & \end{matrix} \right\rangle \text{ and } R_7 = \left\langle \begin{matrix} & & & a_{11} & & & \\ & & a_{21} & c_{11} & a_{12} & & \\ & a_{31} & c_{21} & a_{22} & c_{12} & a_{13} & \\ a_{41} & c_{31} & a_{32} & c_{22} & a_{23} & c_{13} & a_{14} \\ & a_{42} & c_{32} & a_{33} & c_{23} & a_{24} & \\ & & a_{43} & c_{33} & a_{34} & & \\ & & & a_{44} & & & \end{matrix} \right\rangle$$

Consequently we can define the rows and columns of rhotrix main entries as:

$$\begin{matrix} a_{t,1} & a_{t-1,1} & . & . & . & a_{1,1} & & & & & & & & & & & & a_{1,1} & . & . & . & a_{1,t-1} & a_{1,t} \\ & a_{t,2} & a_{t-1,2} & . & . & . & a_{1,2} & & & & & & & & & & & a_{2,1} & . & . & . & a_{2,t-1} & a_{2,t} \\ & & a_{t,3} & a_{t-1,3} & . & . & . & a_{1,3} & & & & & & & & & a_{3,1} & . & . & . & a_{3,t-1} & a_{3,t} \\ & & & . & . & . & . & . & . & & & & & & & . & . & . & . & . & . & & \\ & & & & . & . & . & . & . & . & & & & & . & . & . & . & . & . & & & \\ & & & & & a_{t,t} & a_{t-1,t} & . & . & . & a_{1,t} & a_{t,1} & . & . & . & a_{t,t-1} & a_{t,t} & & & & & & \end{matrix}$$

and

while the rows of the heart entries are similarly defined (Sani, 2004).

Given any two rhotrix say *Rn* and *Qn* the set of entries a_{ij} for R_n and b_{ij} for Q_n multiply each other and entries c_{lk} for R_n and d_{lk} for Q_n also do likewise. Thus, the multiplication of any two rhotrices is thus defined as:

$$R_n \circ Q_n = (a_{i1j1,} c_{l1k1}) \circ (b_{i2j2,} d_{l2k2}) = \qquad (1.2)$$

The multiplication process of (1.2) is similar to that found in matrix, which is given by.

$$c_{ij} = \sum_{k=0}^{l-1} a_{ik} b_{kj} \qquad (1.3)$$

2.0 BACKGROUND FROM GROUP THEOREM TO COMPLEX NUMBER

The exponent of matrix multiplication has proved so elusive that it has been given a name ω, which in essence is described as the smallest number such that $o(n^{\omega+\varepsilon})$ multiplications suffice for all $\varepsilon > 0$. Similarly, since all n^2 must be part of any computation, ω is clearly at least 2.

The road to the current best upper bound on ω was built on clever ideas and increasingly complex combinatorial techniques. Many researchers long believed that the standard, $o(n^3)$ algorithm was the best possible. Then, in 1969, Volker Strassen stunned the research world with his $o(n^{2.81})$ algorithm for multiplying matrices. Still sometimes used in practice, Strassen's algorithm is reminiscent of a shortcut, first observed by Gauss, for multiplying complex numbers. Though finding the product $(a + bi)(c + di) = ac - bd + (bc + ad)i$ seems to require four multiplications, Gauss observed that it can actually be done with three multiplication *ac*, *bd*, and $(a + b)(c + d)$, because $bc + ad = (a + b)(c + d) - ac - bd$.

Let $A, B \in C = \Re[i]/(i^2 + 1)$,

where $A = (a + ib)$ and $B = (c + id)$,

then $AB = (ac - bd) + (ad + bc)i$.

This process uses 4 multiplications. How possible is it optimality? Certainly by definition:

$$m_1 = (a+b)(c+d) = (ac-bd) + bc - ad$$

$$m_2 = bc$$

$$m_3 = ad$$

$$AB = (m_1 - m_2 + m_3) + (m_2 + m_3)i$$

Thus, complex multiplication is accomplished with only 3 multiplications over $\Re$.

2.1 A Review of Strassen's Algorithm

In 1969, Strassen introduced an algorithm to compute matrix multiplications. This algorithm reduces the number of multiplications required for computing the product of two *2* x *2* matrices *A* and *B* from eight to seven, by expressing the entries of AB as linear combinations of products of linear combinations of the entries of A and B. This trick, applied to four blocks of 2^n x 2^n matrices, reduces the problem to seven multiplications of 2^{n-1} x 2^{n-1} matrices.

Let $A, B, C \in \Re^{2\times 2}$

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix},\ B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix},\ C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \tag{2.1}$$

$M_1 := (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})$
$M_2 := (A_{2,1} + A_{2,2})B_{1,1}$
$M_3 := A_{1,1}(B_{1,2} - B_{2,2})$
$M_4 := A_{2,2}(B_{2,1} - B_{1,1})$
$M_5 := (A_{1,1} + A_{1,2})B_{2,2}$
$M_6 := (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})$
$M_7 := (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})$
$C1,1 = M1 + M4 - M5 + M7$
$C1,2 = M3 + M5$
$C2,1 = M2 + M4$
$C2,2 = M1 - M2 + M3 + M6$

In essence, instead of using the normal 8 multiplications of trivial approach, Strassen's algorithm only uses 7. We can continue to apply Strassen's algorithm recursively to achieve the time complexity of $o(n^{\log 7}) = o(n^{2.807})$. Considering $A_{i,j}$, $B_{i,j}$, and $C_{i,j}$ as matrices instead of scalars allows matrix multiplication over $n = 2^N$ size matrices to be performed using only $7^N = 7^{\log_2 n} = n^{\log_2 7} = O(n^{2.81})$. Which means it will run faster than the conventional algorithm for sufficiently large matrices.

Unless the matrix has a dimension of 2k you cannot apply Strassen's algorithm directly. A method called *dynamic peeling* is able to solve this problem effectively (Spence and Kanodia). Dynamic peeling makes matrix dimensions even by stripping off an extra row or column as needed, and putting their contributions back to the final result later.

This interesting feature motivated us to incorporate this algorithm into the present research area. From our earlier definition of rhotrices, we related its row-column multiplication to be similar in every aspect with that of matrices multiplication and hence the importance of the present study. We believe that the Strassen's algorithm can also be applied to the row-column multiplication of rhotrices even though; rhotrix is always of odd dimension.

Huss-Lederman pioneered the design of an efficient and portable serial implementation of Strassen's algorithm called DGEFMM (Huss-Lederman and Jacobson, 1996). DGEFMM is designed to replace DGEMM and obtained better performance for all matrix sizes while minimizing the temporary storage. DGEMM, $C = op(A) \times op(B)$ (IBM Engineering and Scientific Subroutine Library).

There are several parallel implementtation of Strassen's algorithm on distributed memory architectures. The methods typically belong to three classes. The first class is to use the conventional algorithm at the top level (across processors) and Strassen's algorithm at the bottom level (within a processor).One basic application of this is that of the Fox's Broadcast-Multiply-Roll (BMR) methods (Fox *et al.*, 1987). Another similar implementation approach is found in (Ohtaki *et al.*, 2004) but is more focused on a distribution scheme particularly for heterogeneous clusters. The second class is to use Strassen's algorithm at both the top and bottom level. Chou (Chou et al.,

1995) decomposes the matrix A into 2 x 2 blocks of submatrices, then further decomposes each submatrix into four 2 x 2 blocks (i.e., 4 x 4 blocks). This way he is able to identify 49 multiplications and uses 7 or 49 processors to perform multiplication concurrently.

Table 2.1: Strassen's Algorithm

Phase 1	
T1 = A11 + A22	T6 = B11 + B22
T2 = A21 + A22	T7 = B12 − B22
T3 = A11 + A12	T8 = B21 − B11
T4 = A21 − A11	T9 = B11 + B12
T5 = A12 − A22	T10 = B21 + B22
Phase 2	
Q1 = T1 × T6	Q5 = T3 × B22
Q2 = T2 × B11	Q6 = T4 × T9
Q3 = A11 × T7	Q7 = T5 × T10
Q4 = A22 × T8	
Phase 3	
T1 = Q1 + Q4	T3 = Q3 + Q1
T2 = Q5 − Q7	T4 = Q2 − Q6
Phase 4	
C11 = T1 − T2	C12 = Q3 + Q5
C21 = Q2 + Q4	C22 = T3 − T4

We equally believe that if such task can be achieved by portioning matrices into sub-blocks of 2 x 2, we can equally extend this method to rhotrices, by a way of partitioning the heart entries into 2 x 2 sub-blocks and then applying the odd dimension techniques of solving the Strassen's algorithm as discussed in (Huss-Lederman et al., 1996). The last class is using Strassen's algorithm at the top level and the conventional one at the extra matrix additions imposed by Strassen's algorithm becomes less compared to the saved matrix multiplication cost. Therefore Strassen's algorithm is better used across processors on the top level.

Finally, for matrices with odd dimensions, some technique must be applied to make the dimensions even, apply Strassen's algorithm to the altered matrix, and then correct the results. Originally, Strassen suggested padding the input matrices with extra rows and columns of zeros, so that the dimensions of all the matrices encountered during the recursive calls are even. After the product has been computed, the extra rows and columns are removed to obtain the desired result. We call this approach static padding, since padding occurs before any recursive calls to Strassen's algorithm. Alternatively, each time Strassen's algorithm is called recursively, an extra row of zeros can be added to each input with an odd row-dimension and an extra column of zeros can be added for each input with an odd column-dimension. This approach to padding is called dynamic padding since padding occurs throughout the execution of Strassen's algorithm. A version of dynamic padding is used in (Huss-Lederman and Jacobson, 1996).

Let $C = AB$ be the desired matrix product, and let

$$\tilde{A} = \left(\begin{array}{ccc|c} & A & & \begin{matrix}0\\ \vdots \\ 0\end{matrix} \\ \hline * & \cdots & * & * \end{array}\right) \text{ and } \tilde{B} = \left(\begin{array}{ccc|c} & B & & \begin{matrix}*\\ \vdots \\ *\end{matrix} \\ \hline * & \cdots & * & * \end{array}\right)$$

be a matrix with an even dimension, where $*$ denotes an unspecified value. Then Strassen's algorithm can be used to compute

$$\tilde{A}\tilde{B} = \left(\begin{array}{ccc|c} & AB & & \begin{matrix}*\\ \vdots \\ *\end{matrix} \\ \hline * & \cdots & * & * \end{array}\right)$$

The product $C = AB$ appears in the upper-left block independent of the values substituted for the $*$'s in A and B. The same would be true if the zeros would have been placed in the bottom row of B instead of the last column of A. In the above diagram one extra row and column has been inserted; however, if desired, additional rows and columns can be inserted so long as the resulting matrix has even dimension. Furthermore, the extra rows and columns could have been inserted in arbitrary locations so long as the column index in the first matrix matches up with the row index in the second matrix. Typically the extra rows and columns contain zeros and are inserted so that A and B are in the top-left or bottom-right blocks (Douglas et al, 1994).

Another approach, called dynamic peeling, deals with odd dimensions by stripping off the extra rows and/or columns as needed, and adding their contributions to the final result in a later round of fixup work. The fixup is required to include the contribution of the rows and columns that were deleted. More specifically, let A be an m x k matrix and B be a k x n matrix. Assuming that m, k, and n are all odd, A and B are partitioned into the block matrices

$$A = \left(\begin{array}{c|c} A_{11} & a_{12} \\ \hline a_{21} & a_{22} \end{array}\right),$$

where A_{11} has even dimensions, a_{12} is a column vector, a_{21} is a row vector, and a_{22} is a single element. Similarly,

$$B = \left(\begin{array}{c|c} B_{11} & b_{12} \\ \hline b_{21} & b_{22} \end{array}\right),$$

where B_{11} has even dimensions, b_{12} is a column vector, b_{21} is a row vector, and b_{22} is a single element.

$$\left(\begin{array}{c|c} C_{11} & c_{12} \\ \hline c_{21} & c_{22} \end{array}\right) = \left(\begin{array}{c|c} A_{11}B_{11} + a_{12}b_{21} & A_{11}b_{12} + a_{12}b_{22} \\ \hline a_{21}B_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{array}\right),$$

Given this seven sub-matrix products, we can compute *W* as

W = $S_5 + S_6 + S_4 – S_2$
= (A + D)(E + H) + (B - D)(G + H) + D(G - E) – (A + B)H
= AE + DE + AH + DH + BG – DG + BH – DH + DG – DE – AH – BH
= AE + BG

Similarly we can compute X as

X = $S_1 + S_2$
= A(F - H) + (A + B)H
= AF – AH + AH + BH
= AF + BH

Again we can compute Y as

Y = $S_3 + S_4$

where $A_{11}B_{11}$ is computed using Strassen's algorithm, and the other computations constitute the fixup work. In Section 3, we discussed how we actually applied similar Strassen's technique to design our own rhotrix multiplication algorithm and coded its computations.

3.0 VERIFICATION OF STRASSEN'S ALGORITHM BY DIVIDE-AND-CONQUER TECHNIQUE

Interestingly, Strassen's algorithm organizes arithmetic involving the sub-arrays A through H so that we can compute I, J, K, and L using just seven recursive matrix multiplications. We can easily verify that they work correctly.

Let us assume that *n* a power of two and let us partition P, Q, and R each into four $\frac{n}{2} \times \frac{n}{2}$ matrices, so that we can write R = PQ as

$\begin{pmatrix} W & X \\ Y & Z \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix}\begin{pmatrix} E & F \\ G & H \end{pmatrix}$ then,

Let's defines seven sub-matrix products:

S_1 = A(F - H)
S_2 = (A + B)H
S_3 = (C + D)E
S_4 = D(G - E)
S_5 = (A + D)(E + H)
S_6 = (B - D)(G + H)
S_7 = (A - C)(E + F)
= (C + D)E + D(G - E)
= CE + DE + DG – DE
= CE + DG

Finally, we can compute Z as

Z = $S_1 – S_7 – S_3 + S_5$
= A(F - H) – (A - C)(E + F) – (C + D)E + (A + D)(E + H)
= AF – AH – AE + CE – AF + CF – CE – DE + AE + DE + AH + DH + CF + DH

Thus, we can compute $R = PQ$ using seven recursive multiplications of matrices of size $\frac{n}{2} \times \frac{n}{2}$. Similarly, if we let a function $t(n)$ denote the running time of the algorithm on an input of size n, and characterize $t(n)$ using an equation that

relates $t(n)$ to values of the function t for problem sizes smaller than n. We get recursion equation

$$t(n) = \begin{cases} b & if\ \ n < 2 \\ 7t(\frac{n}{2}) + kn & f\ \ n > 2, \end{cases} \qquad (3.1)$$

If we use the iterative substation method and we assume that the matrix size is fairly large enough, then by substituting (3.1) of the recurrence for each occurrence of the function t on the right-hand side, we can characterize the running time $t(n)$ as

$$t(n) = 7t(\tfrac{n}{2}) + kn^2,$$

for some constant $k > 0$. Hence by the master theorem, we have the following:

Theorem 1: we can multiply two $n \times n$ matrices in $O(n^{\log_2 7})$ time.

Thus, with a fair bit of additional computation, we can perform the multiplication for $n \times n$ matrices in time $O(n^{2.808})$, which is $o(n^3)$ time.

3.1 Strassen's Algorithm and Rhotrix Row-Column Multiplication

In accordance with the Winograd's variant of Strassen's algorithm, which uses seven matrix multiplications and 15 matrix additions we extend and implement similar algorithmic computation on rhotrix multiplication. It is well-known that this is the minimum number of multiplications and additions possible for any recursive matrix multiplication algorithm based on division into quadrants. The division of the matrices into quadrants follows equation (2.1).

Our major difficulty is on partitioning rhotrix elements into 2 x 2 blocks of quadrants knowing that rhotrix has an odd dimension. In actual sense this might seem complicated, but we can still combine both Strassen's algorithm with the standard multiplication technique of matrix to achieve our goal. In this section we do not discuss the theoretical proof behind this algorithm as they are covered elsewhere. The following steps illustrate the splitting process of rhotrices into main and heart entries.

Definition: Sifting is the process of separating main entries or heart entries form the entire given entries of the rhotrix. It is demoted by the symbol $\oplus\!\!\!\rightarrow$ with the main entries on the left and the heart entries on the right.

The above definition can be illustrated by the following example.

$$R_5 = \left\langle \begin{matrix} & a_{11} & \\ a_{21} & c_{11} & a_{12} \\ a_{31} & c_{21} & a_{22} & c_{12} & a_{13} \\ a_{32} & c_{22} & a_{23} \\ & a_{33} & \end{matrix} \right\rangle = \left\langle \begin{matrix} & a_{11} & \\ a_{21} & & a_{12} \\ a_{31} & a_{22} & a_{13} \\ a_{32} & & a_{23} \\ & a_{33} & \end{matrix} \right\rangle \oplus\!\!\!\rightarrow \left\langle \begin{matrix} & c_{11} & \\ c_{21} & & c_{12} \\ & c_{22} & \end{matrix} \right\rangle$$

Note that the following relations is true from the above definition

$$where \left\langle \begin{matrix} & a_{11} & \\ a_{21} & & a_{12} \\ a_{31} & a_{22} & a_{13} \\ a_{32} & & a_{23} \\ & a_{33} & \end{matrix} \right\rangle \subset \left\langle \begin{matrix} & & a_{11} & & \\ & a_{21} & c_{11} & a_{12} & \\ a_{31} & c_{21} & a_{22} & c_{12} & a_{13} \\ & a_{32} & c_{22} & a_{23} & \\ & & a_{33} & & \end{matrix} \right\rangle$$

$$and \left\langle \begin{matrix} & c_{11} & \\ c_{21} & & c_{12} \\ & c_{22} & \end{matrix} \right\rangle \subset \left\langle \begin{matrix} & & a_{11} & & \\ & a_{21} & c_{11} & a_{12} & \\ a_{31} & c_{21} & a_{22} & c_{12} & a_{13} \\ & a_{32} & c_{22} & a_{23} & \\ & & a_{33} & & \end{matrix} \right\rangle$$

Where $\subset$ denotes rhotrix embedment and $\oplus\!\!\!\rightarrow$ denotes rhotrix sifting.
Similarly,

$$Q_5 = \left\langle \begin{matrix} & & b_{11} & & \\ & b_{21} & d_{11} & b_{12} & \\ b_{31} & d_{21} & b_{22} & d_{12} & b_{13} \\ & b_{32} & d_{22} & b_{23} & \\ & & b_{33} & & \end{matrix} \right\rangle = \left\langle \begin{matrix} & & b_{11} & & \\ & b_{21} & & b_{12} & \\ b_{31} & & b_{22} & & b_{13} \\ & b_{32} & & b_{23} & \\ & & b_{33} & & \end{matrix} \right\rangle \oplus \left\langle \begin{matrix} & d_{11} & \\ d_{21} & & d_{12} \\ & d_{22} & \end{matrix} \right\rangle$$

$$\text{where } \left\langle \begin{matrix} & & b_{11} & & \\ & b_{21} & & b_{12} & \\ b_{31} & & b_{22} & & b_{13} \\ & b_{32} & & b_{23} & \\ & & b_{33} & & \end{matrix} \right\rangle \subset \left\langle \begin{matrix} & & b_{11} & & \\ & b_{21} & d_{11} & b_{12} & \\ b_{31} & d_{21} & b_{22} & d_{12} & b_{13} \\ & b_{32} & d_{22} & b_{23} & \\ & & b_{33} & & \end{matrix} \right\rangle$$

$$\text{and } \left\langle \begin{matrix} & d_{11} & \\ d_{21} & & d_{12} \\ & d_{22} & \end{matrix} \right\rangle \subset \left\langle \begin{matrix} & & b_{11} & & \\ & b_{21} & d_{11} & b_{12} & \\ b_{31} & d_{21} & b_{22} & d_{12} & b_{13} \\ & b_{32} & d_{22} & b_{23} & \\ & & b_{33} & & \end{matrix} \right\rangle$$

It is important to observe that the heart entries of R_n and Q_n from the above extraction are not themselves rhotrices, but represents computational step among group of steps required in performing a multiplication tasks.

Once we are able to split this, then the next step is to find a way of partitioning these entries into blocks of *2* x *2*, as required by the desired algorithm.

$$R_n = \left\langle \begin{matrix} & & a_{11} & & \\ & a_{21} & & a_{12} & \\ a_{31} & & a_{22} & & a_{13} \\ & a_{32} & & a_{23} & \\ & & a_{33} & & \end{matrix} \right\rangle \Rightarrow \left\langle \begin{matrix} & & a_{11} & & \\ & a_{21} & & a_{12} & \\ a_{31} & & a_{22} & & a_{13} \\ & a_{32} & & a_{23} & \\ & & a_{33} & & \end{matrix} \right\rangle \text{ and } \left\langle \begin{matrix} & a_{11} & \\ a_{21} & & a_{12} \\ & a_{22} & \end{matrix} \right\rangle \left\langle \quad \right\rangle$$

If the rhotrices R_n, Q_n are not of type 2^n x 2^n, we have two options, first option is to either fill the missing rows and columns with zeros (a method referred to as padding approach) and partition R_n and Q_n into equally sized block matrices or we delete the un-partitioned rows and columns (a method referred to as peeling approach) and then later incorporate them into the computation process. But first let's consider the padding approach as shown below.

$$R_n = \left\langle \begin{matrix} & & a_{11} & & \\ & a_{21} & & a_{12} & \\ a_{31} & & a_{22} & & a_{13} \\ 0 & & a_{32} & & a_{23} & & 0 \\ & 0 & & a_{33} & & 0 & \\ & & 0 & & 0 & & \\ & & & 0 & & & \end{matrix} \right\rangle \text{ and } Q_n = \left\langle \begin{matrix} & & b_{11} & & \\ & b_{21} & & b_{12} & \\ b_{31} & & b_{22} & & b_{13} \\ 0 & & b_{32} & & b_{23} & & 0 \\ & 0 & & b_{33} & & 0 & \\ & & 0 & & 0 & & \\ & & & 0 & & & \end{matrix} \right\rangle$$

$$A_{11} = \left\langle a_{21} \begin{matrix} a_{11} \\ \\ a_{22} \end{matrix} a_{12} \right\rangle,\ A_{12} = \left\langle a_{23} \begin{matrix} a_{13} \\ \\ 0 \end{matrix} 0 \right\rangle,\ A_{21} = \left\langle 0 \begin{matrix} a_{31} \\ \\ 0 \end{matrix} a_{32} \right\rangle \text{and } A_{22} = \left\langle 0 \begin{matrix} a_{33} \\ \\ 0 \end{matrix} 0 \right\rangle$$

$$B_{11} = \left\langle b_{21} \begin{matrix} b_{11} \\ \\ b_{22} \end{matrix} b_{12} \right\rangle,\ A_{12} = \left\langle b_{23} \begin{matrix} b_{13} \\ \\ 0 \end{matrix} 0 \right\rangle,\ A_{21} = \left\langle 0 \begin{matrix} b_{31} \\ \\ 0 \end{matrix} b_{32} \right\rangle \text{and } A_{22} = \left\langle 0 \begin{matrix} b_{33} \\ \\ 0 \end{matrix} 0 \right\rangle$$

and

$$C_{11} = \left\langle c_{21} \begin{matrix} c_{11} \\ \\ c_{22} \end{matrix} c_{12} \right\rangle,\ C_{12} = \left\langle c_{23} \begin{matrix} c_{13} \\ \\ 0 \end{matrix} 0 \right\rangle,\ C_{21} = \left\langle 0 \begin{matrix} c_{31} \\ \\ 0 \end{matrix} c_{32} \right\rangle \text{and } C_{22} = \left\langle 0 \begin{matrix} c_{33} \\ \\ 0 \end{matrix} 0 \right\rangle$$

with $A_{i,j}, B_{i,j}$ *and* $C_{i,j} \in \Re^{2^{n-1} \times 2^{n-1}}$ then

$$C_{1,1} = A_{1,1} B_{1,1} + A_{1,2} B_{2,1}$$
$$C_{1,2} = A_{1,1} B_{1,2} + A_{1,2} B_{2,2}$$
$$C_{2,1} = A_{2,1} B_{1,1} + A_{2,2} B_{2,1}$$
$$C_{2,2} = A_{2,1} B_{1,2} + A_{2,2} B_{2,2}$$

With this construction we have not reduced the number of multiplications. We still need 8 multiplications to calculate the $C_{i,j}$ matrices, the same number of multiplications we need when using standard matrix multiplication.

Now comes the important part. We define new matrices

$$M_1 := (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})$$
$$M_2 := (A_{2,1} + A_{2,2})B_{1,1}$$
$$M_3 := A_{1,1}(B_{1,2} - B_{2,2})$$
$$M_4 := A_{2,2}(B_{2,1} - B_{1,1})$$
$$M_5 := (A_{1,1} + A_{1,2})B_{2,2}$$
$$M_6 := (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})$$
$$M_7 := (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})$$

which are then used to express the $C_{i,j}$ in terms of M_k. Because of our definition of the M_k we can eliminate one matrix multiplication and reduce the number of multiplications to 7 (one multiplication for each M_k) and express the $C_{i,j}$ as

$$C_{1,1} = M1 + M4 - M5 + M7$$
$$C_{1,2} = M3 + M5$$
$$C_{2,1} = M2 + M4$$
$$C_{2,2} = M1 - M2 + M3 + M6$$

We iterate this division process n times until the sub matrices degenerate into numbers (elements of the ring C). At the end of the recursive computation we now do away with the zeros added to augments the missing rows and columns to get back our rhotrix.

Alternatively we consider one of the key approaches to dealing with matrices with odd dimension, the peeling method discussed in (Huss-Lederman et al., 1996). The peeling approached discussed can be used to apply Strassen's algorithm to smaller rhotrices obtained by deleting rows and columns. In this case, after applying the Strassen's algorithm, still we need to do some additional work, referred to as fixup. The fixup has to be taken into consideration in other to include the contribution of the initial rhotrix's rows and columns that were deleted earlier on.

The peeling approach corresponds to blocking methods found in Cannon's algorithm and is strictly applied to only the rhotrix main entries whose dimension is usually odd, and not to the heart entries whose dimension would always be even. The input rhotrices are partitioned into blocks, possibly of different sizes, such that the rhotrix products is compatible with the dimensions of the blocks, that is, if A_{ij} is the ij -block of the rhotrix $R(A)$ and B_{jK} is the jk -blocks of the rhotrix $Q(B)$, then the column dimension of A_{ij} must be equal to the row dimension of B_{jK} for all i, j, and k. If the dimensions of the blocks are compatible in this sense, then C_{ik}, the ik-

block of $C = AB$, is equal to $\sum_{k=1}^{n} a_{i,k} b_{k,j}$ where j runs over the blocks in the *i*-th columns of the block rhotrix obtained by partitioning A. After partitioning the input rhotrices into blocks, Strassen's algorithm can then be applied to compute the block products of rhotrices whose dimensions are even. The fixup work includes the block products that are not computed with Strassen's algorithm and the work required to combine the block products.

One partitioning scheme chooses the upper left-hand block to be the largest sub-rhotrix, with even dimensions, containing the element in the upper left-hand corner. In this scheme there can be at most four block, which occurs when both the row and column dimension are odd. In this case

$$R(A) = \left\langle \begin{array}{ccc} & a_{11} & \\ a_{21} & & a_{12} \\ a_{31} & a_{22} & a_{13} \\ a_{32} & & a_{23} \\ & a_{33} & \end{array} \right\rangle \left\langle \begin{array}{ccc} & A_{11} & \\ a_{31} & & a_{13} \\ a_{32} & & a_{23} \\ & a_{33} & \end{array} \right\rangle$$

if we assume $a_{33} \Leftrightarrow a_{22}$ then

$$\left\langle \begin{array}{ccc} & A_{11} & \\ a_{21} & & a_{12} \\ & a_{22} & \end{array} \right\rangle$$

where A_{11} is an (m - 1) x (k - 1) matrix, a_{12} is a (m - 1) x 1 matrix, a_{21} is a 1 x (k - 1) matrix, a_{22} is a 1 x 1 matrix

$$R(B) = \left\langle \begin{matrix} & b_{11} & \\ b_{21} & & b_{12} \\ b_{31} & b_{22} & b_{13} \\ b_{32} & & b_{23} \\ & b_{33} & \end{matrix} \right\rangle \left\langle \begin{matrix} & & \\ & B_{11} & \\ b_{31} & & b_{13} \\ b_{32} & & b_{23} \\ & b_{33} & \end{matrix} \right\rangle$$

if we assume $b_{33} \Rightarrow b_{22}$, $b_{23}, b_{13} \Rightarrow b_{12}$ and $b_{31}, b_{32} \Rightarrow b_{21}$ then,

$$\left\langle \begin{matrix} & B_{11} & \\ b_{21} & & b_{12} \\ & b_{22} & \end{matrix} \right\rangle$$

and B_{11} is an (k - 1) x (n - 1) matrix, b_{12} is a (k - 1) x 1 matrix, $b_{21}$is a 1 x (n- 1) matrix, b_{22} is a 1 x 1 matrix. The product $R = AB$ is computed as

$$\left\langle \begin{matrix} & R_{11} & \\ r_{21} & & r_{12} \\ & r_{22} & \end{matrix} \right\rangle = \left\langle \begin{matrix} A_{11}B_{11} + a_{12}b_{21} & & \\ & & A_{11}b_{12} + a_{12}b_{22} \\ a_{21}B_{11} + a_{22}b_{21} & & \\ & a_{21}b_{12} + a_{22}b_{22} & \end{matrix} \right\rangle$$

The above peeling approach for which we applied to solve for the rhotrix main entries as stated before do not apply to the heart elements with even dimension, hence the good news is that we can directly apply the blocks:

$$\left\langle \begin{matrix} & R_{11} & \\ R_{21} & & R_{12} \\ & R_{22} & \end{matrix} \right\rangle = \left\langle \begin{matrix} & C_{11} & \\ C_{21} & & C_{12} \\ & C_{22} & \end{matrix} \right\rangle \left\langle \begin{matrix} & D_{11} & \\ D_{21} & & D_{12} \\ & D_{22} & \end{matrix} \right\rangle$$

where C_{11}, C_{12}, C_{21}, and C_{22} are (m - 1) x (k-1) matrices and D_{11}, D_{12}, D_{21} and D_{22} are (k - 1) x (n - 1) matrices respectively.

3.2 Strassen's Algorithm Computation Approach

The computation process can be divided into two major parts; the first part handles the recursive computation of 2 x 2 blockable matrices with even dimensions while the latter parts handles the fixup (i.e., multiplication resulting from the recursive computation with the peeled off row and column) case resulting from the earlier peeling processes carried out. For the first part, the computation can proceeds as shown in Table 3.1 for both rhotrix main entries and heart entries.

Strassen's algorithm for the computation of the heart entries as shown below. The algorithm basically describes how to perform a single level recursion on 2 × 2

Figure 3.1a and 3.1b shows the computation graph for the Winograd variant of Strassen's algorithm. The sub-blocks rhotrices A and B, and C and D are the initial inputs, shown labelling the vertices at the top of the two graphs. All edges are directed from inputs, higher vertices to lower ones below.

Table 3.1: Strassen's Algorithm for Rhotrix Multiplication

Computation for rhotrix main entries		Computation for rhotrix heart entries	
Phase 1 and 2		**Phase 1 and 2**	
S1 = A21 + A22	T1 = B12 − B11	S1 = C21 + A22	T1 = D12 − D11
S2 = S1 + A11	T2 = B22 – T1	S2 = S1 + C11	T2 = D22 – T1
S3 = A11 – A21	T3 = B22 − B12	S3 = C11 – C21	T3 = D22 − D12
S4 = A12 – S2	T4 = B21 − T2	S4 = C12 – S2	T4 = D21 − T2
Phase 3		**Phase 3**	
P1 = A11 × B11	P5 = S3 × T3	P1 = C11 × D11	P5 = S3 × T3
P2 = A12 × B21	P6 = S4 × B22	P2 = C12 × D21	P6 = S4 × D22
P3 = S1 × T1	P7 = A22 × T4	P3 = S1 × T1	P7 = C22 × T4
P4 = S2 × T2		P4 = S2 × T2	
Phase 4		**Phase 4**	
R11 = U1 = P1 + P2		R11 = U1 = P1 + P2	
U2 = P1 + P4		U2 = P1 + P4	
U3 = U2 + P5		U3 = U2 + P5	
R21 = U4 = U3 + P7		R21 = U4 = U3 + P7	
R22 = U4 = U3 + P7		R22 = U4 = U3 + P7	
U6 = U2 + P3		U6 = U2 + P3	
R12 = U7 = U6 + P6		R12 = U7 = U6 + P6	

Thus, for example, in figure 3.1, A11 and A21 are initial inputs, which are combined to produce S3. S3 is subsequently combined with T3 to produce P5, and so on, until the final results, sub-blocks of R, are produced at the bottom level.

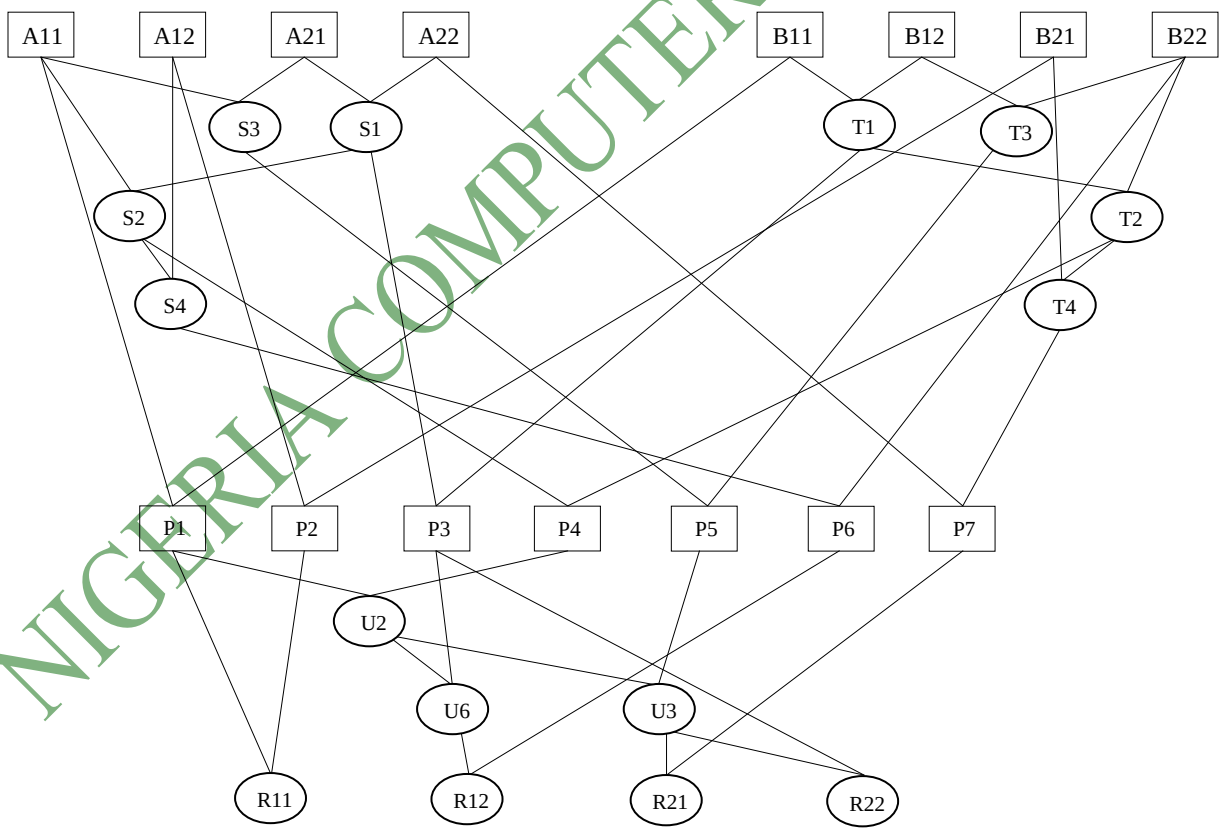

Figure 3.1: Dependency graph for Winograd's variant of Strassen's Algorithm [Huss-Lederman]

3.3 Strassen's Algorithm Fixup Case

The second part of our computation deals with the dynamic peeling approach explained earlier on, based on the previous analysis, it has been shown that the dynamic peeling could be a useful method, but it has not been previously tested through actual implementation for rhotrices. We wrote a java program to implement the essentially data-driven computation and presented a framework model for the parallel MPI implementation.

After combining operations, there are potentially three fixup steps:

Step1:

$$R_{11} = a_{12}b_{21} + R_{11},$$

Step2:

$$r_{12} = \left\langle A_{11} \quad a_{12} \right\rangle \circ \left\langle b_{12} \quad b_{22} \right\rangle + r_{12},$$

Step3:

$$\left\langle r_{21} \quad r_{22} \right\rangle = \left\langle a_{21} \quad a_{22} \right\rangle \circ \left\langle B_{11} \quad b_{12} \quad b_{22} \right\rangle + \left\langle r_{21} \quad r_{22} \right\rangle.$$

The three steps can be computed using the standard rhotrix multiplication method. It should be noted that, A_{11} and B_{11} are the result obtained from our initial recursive computation for even dimensioned blockable rhotrices and a_{12}, a_{21}, a_{22}, b_{12}, b_{21} and b_{22} are the peeled off row and column elements.

4.0 PROGRAM TASK GRAPH FOR STRASSEN'S ALGORITHM

It is straightforward to represent Strassen's algorithm in a task graph. A task graph is a directed graph whose edges represent dependence relationship and nodes represent tasks (either sequential or parallel programs). The node located at the arc tail should be executed before the node at the arc head. Figure 3.2 displays the corresponding task graph of one-level recursion of Strassen's algorithm. Winograd's variant of Strassen's algorithm uses the required 7 multiplications and minimal number of 15 additions/subtractions. This is simplified into the following 4 stages of computations.

Stages (1) and (2)

S1 = A21 + A22,
S2 = S1 – A11 = A21 + A22 – A11,
S3 = A11 – A21,
S4 = A12 – S2 = A12 – A21 - A22 + A11,
T1 = B12 – B11,
T2 = B22 – T1 = B22 – B12 + B11,
T3 = B22 – B12,
T4 = B21 – T2 = B21 – B22 + B12 – B11.

Stages (3) compute the seven products

P1 = A11B11,
P2 = A12B21,
P3 = S1T1,
P4 = S2T2,
P5 = S3T3,
P6 = S4B22,
P7 = A22T4,

And stage (4) computes

U1 = P1 + P2,
U2 = P1 + P4,
U3 = U2 + P5,
U4 = U3 + P7,
U5 = U3 + P3,
U6 = U2 + P3,
U7 = U6 + P6.

Hence we can easily verify that R11= U1, R12 = U7, R21 = U4, and R22 = U5. We can now apply the above task notation to produce a simplified task graph as shown below.

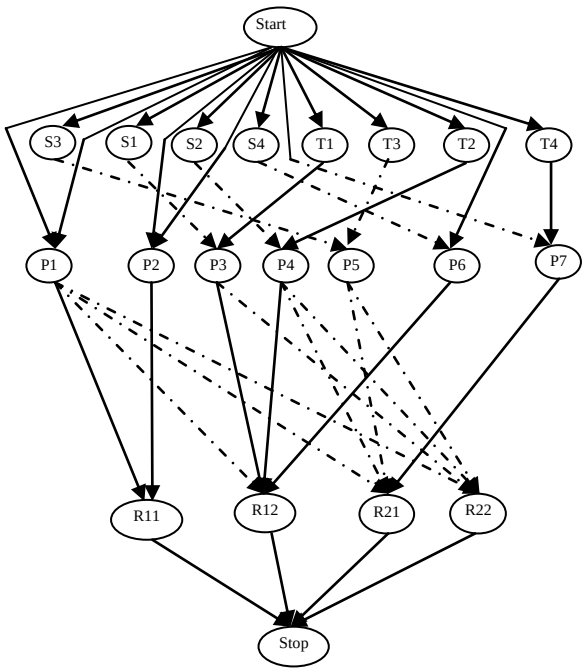

Figure 3.2: Task graph for Winograd's variant of Strassen's Algorithm

5.0 A FRAMEWORK OF TASK-PARALLEL APPROACH USING MPI

We can use a straightforward method to parallelize Strassen's algorithm. Based on the algorithm, there exist seven matrix multiplications (i.e., P1-P7). If we employ seven processes to compute the multiplications, the program design becomes very simple. The basic idea is that we divide the task graph in Figure 3.2 into seven parts and each part is handled by a process. Figure 3.3 depicts the task partitioning across the seven processes. Nodes with the same colour belong to a single process. It is important to note that the positions of nodes in Figure 3.3 are identical to those in Figure 3.2 were computations are carried out for step 1 and 2 of S and T, step 3 of U and step 4 of R. Since Figure 4 directly reflects our program design model, it is straightforward to write an MPI program based on the graph. For instance, process P3 performs computations of S1, T1, U4 and R21 which correspond to the brown nodes on the right.

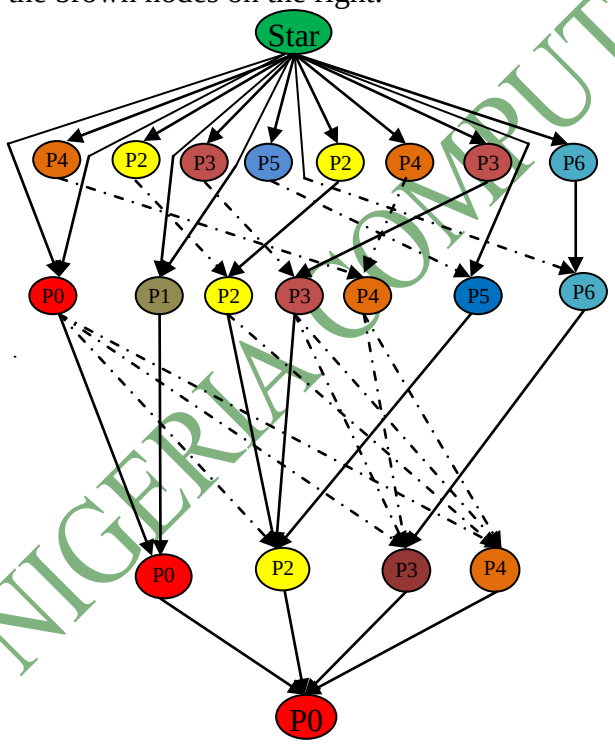

Figure 3.3: A model of MPI task processes graph to implement Strassen's algorithm.

6.0 CONCLUSION

We have described a theoretical computational model for rhotrix multiplication using the Strassen's algorithm, in which case we were able to implement a sequential version of the algorithm and at the same time presented a parallel framework for the MPI implementation on distributed memory systems. This method we believe is scalable and portable to other platforms because most of the desired called subroutines already exist in a standard library. Hence it is worthwhile to put more effort into this method so that the basic operation of rhotrix multiplication could be improves further.

7.0 REFERENCES

Ajibade A.O. (2003). The Concept of Rhotrix in Mathematical Enrichment, International Journal of Mathematical Education in Science and Technology, 175-179.

Atanassov K.T. and Shannon A.G. (1998). *International Journal of Mathematics Education in Science and Technology*, 29, 898-903.

Chou C., Deng Y., Li G. and Wang Y. (1995). Parallelizing Strassen's Method for Matrix Multiplication on Distributed-Memory MIMD Architectures, Computers for Mathematics with Applications, 30(2):49.

Desprez F. and Suter.Mixed F. (2001). Parallel Implementation of the Top Level Step of Strassen and Winograd Matrix Multiplication Algorithms, Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS'01), San Francisco.

Douglas C., M. Heroux, G. Slishman and R. M. Smith. (1994). A portable level 3 BLAS Winograd variant of Strassen's matrix-matrix multiply algorithm. GEMMW: Journal of Computational Physics, 110:1-10.

Fisher P.C. and Probert R.L. (1974). Efficient Procedures for using Matrix Algorithms in Automation Languages and Programming, Number 14 in

Lecture Notes in Computer Science, Page 413-427. Springer-Verlag.

Fox G.C., Hey A.I and Otto S. (1987). Matrix Algorithms on the Hypercube I: Matrix Multiplication, Parallel Computing 4:17-31.

Huss-Lederman S. and Jacobson E. (1996). Implementation of Strassen's Algorithm for Matrix Multiplication, Proceedings of the 1996 ACM/IEEE conference on Supercomputing, Pittsburgh.

IBM Engineering and Scientific Subroutine Library Version 3 Release 3 Guide and Reference. Document Number SA22-7272-04, IBM.

Jacobson S.E.M., Johnson J.R., Tsao A. and Turnbull T. (1996). Strassen's algorithm for matrix multiplication: Modeling, analysis, and implementation. Technical report, Center for Computing Sciences. Technical Report CCS-TR-96-147.

Ohtaki Y., Takahashi D., Boku T. and Sato M. (2004). Parallel Implementation of Strassen's Matrix Multiplication Algorithm for Heterogeneous Clusters, Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04), Santa Fe.

Sani B. (2004). The Row-Column Multiplication of High Dimensional Rhotrices, International Journal of Mathematical Education in Science and Technology, 777-781.

Spence M. and Kanodia V. Efficient Implementation of Strassen's Algorithm for Matrix Multiplication, Technical Report, Rice University.

Strassen V. (1969). Gaussian Elimination is not Optima Number. Mat13:354-356.