Abstract—This paper presents a new technique for the optimum placement of processors to minimize the total effective communication load under multi-processor communication dominated environment. This is achieved by placing heavily loaded processors near each other and lightly loaded ones far away from one another in the physical grid locations. The results are mathematically proved for the Algorithms are described.

Keywords—Ascending Sort Index Vector, Effective Communication Load, Effective Distance Matrix, Optimal Placement, Sorting Order.

I. INTRODUCTION

OPTIMUM placement of VLSI blocks has been extensively studied and various solutions are provided [1], [2] for different requirements. In this paper we consider the efficiency of communication as the main criterion while placing the VLSI blocks (say Processors).

When several processors are to be placed in a grid layout, their location can be chosen to minimize the overall effective Communication Load among them. This maximizes the total traffic transportation in a given time.

II. SINGLE LINE PLACEMENT

The Processors are to be uniformly located on a single line grid as shown in fig.1 at L1, L2, L3, …LN. The physical distance between adjacent grids is taken as 1 unit.

Thus, the different distances are,

\[D_{12} = 1, \quad D_{13} = 2, \quad D_{1N} = N-1 \]

and so on. The distance between the jth and kth location is given by,

\[D_{jk} = |k-j|\]

with \(D_{kk} = 0\)

Let \(P_1, P_2, \ldots, P_N\) be the given processors. The communication among the processors is assumed to be of Broad-Cast type and the traffic from one processor to other processors is assumed to be known as follows:

\(T_{jk} \) is the broad cast traffic load to be transported from j to k. \(T_{jj}\) is obviously zero. The unit of traffic load can be bytes or packets or frames. The traffic load represents the amount of data to be transported say from the output buffer of a processor.

Important assumption: We assume that the traffic load from a specific source processor to all other processors is of the same magnitude. This value is denoted by \(T_j\) for Processor \(P_j\) for \(j = 1\) to \(N\) and is written as,

\[T_1 = T_{12} = T_{13} = \ldots = T_{1N}\]

\[T_2 = T_{21} = T_{23} = \ldots = T_{2N}\]

\[\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdOTS_{jk} = |k-j| (1)

with \(D_{kk} = 0\)

\[E_k = \sum_{j=1}^{N} T_{kj} D_{kj}\] (3)

Here, each product term represents the traffic load from the source to the destination multiplied by the corresponding distance. This is a metric that represents the transportation burden of that path. The sum of the products gives the total effective load for that Processor.

Thus \(E_k\) gives the Effective Communication Load for Processor \(P_k\). The Effective Communication Load (ECL) also represents the energy consumed in transporting the traffic loads through corresponding distances, because, the power required is proportional to the traffic load and the time required is proportional to the distance of travel. Thus the ‘traffic-load distance’ product represents Energy.

IV. OBJECTIVE OF THE PAPER

The sum of ECL’s of each processor is given by Eq.(3) and the overall total is given by,
Our objective is to find the optimum placement of given Processors at proper locations to minimize \( E \) (the total Effective communication Load for the entire system) as given by Eq.\( (4) \).

\[
E = E_1 + E_2 + ... + E_N = \sum_{j=1}^{N} \sum_{k=1}^{N} E_{j,k} \tag{4}
\]

V. BASIC PRINCIPLE

The basic principle is to place the heavily loaded Processors nearer to each other and lightly loaded ones far away from one another. In other words, higher the traffic load value, lower should be the distance covered by that load and vice-versa.

Now the Effective Communication Load of Processor \( P_1 \), using Eqs.\( (3) \) and \( (2) \) is,

\[
E_1 = T_1.D_{12} + T_1.D_{13} + ... + T_1.D_{1N} = T_1.(D_{12} + D_{13} + ... + D_{1N})
\]

Similarly, for \( P_j \),

\[
E_j = T_j.(D_{j1} + D_{j2} + ... + D_{jN})
\]

This is rewritten as,

\[
E_j = T_j.D_j \tag{5}
\]

where

\[
D_j = D_{j1} + D_{j2} + ... + D_{jN} \tag{6}
\]

Now the overall total Effective Communication Load can be expressed using Eqs.\( (4) \) and \( (5) \) as,

\[
E = T_1.D_1 + T_2.D_2 + ... + T_j.D_j + ... + T_N.D_N
\]

That is,

\[
E = \sum_{j=1}^{N} T_j.D_j \tag{7}
\]

Thus \( E \) is the scalar product of two vectors \( T \) and \( D \) given by,

\[
T = [T_1 \ T_2 \ . \ . \ . \ T_N] \tag{8}
\]

\[
D = [D_1 \ D_2 \ . \ . \ . \ D_N] \tag{9}
\]

Please note that \( T_j \) gives the traffic load of Processor \( P_j \) and \( D_j \) gives the distance to be covered from location \( L_j \).

Our objective is to minimize \( E \) by properly placing the Processors such that when \( T_1 \) is maximum, \( D_1 \) is minimum and vice versa. If \( T \) is in the descending order, \( D \) has to be in the ascending order. This means, the sort order of \( T \) should be opposite to that of \( D \).

VI. VALUES OF VECTOR \( D \)

Consider the case when \( N = 4 \) as shown in Fig. 2. Here \( D_{12} = 1 \), \( D_{13} = 2 \) and \( D_{14} = 3 \). Therefore,

\[D_1 = 1 + 2 + 3 = 6 \]

Similarly,

\[D_2 = D_{21} + D_{23} + D_{24} = 1 + 1 + 2 = 4 \]
\[D_3 = D_{31} + D_{32} + D_{34} = 2 + 1 + 1 = 4 \]
\[D_4 = D_{41} + D_{42} + D_{43} = 3 + 2 + 1 = 6 \]

Therefore when \( N = 4 \),

\[
D = [6 \ 4 \ 4 \ 6] \tag{10}
\]

Similarly when \( N = 5 \),

\[
D = [10 \ 7 \ 6 \ 7 \ 10] \tag{11}
\]

In general, for a given \( N \), the \( k \) th element of \( D \), designated as \( D_k \) is given by,

\[
D_k = \frac{1}{2}[(N - k)(N - k + 1) + k(k - 1)] \tag{12}
\]

For \( k = 1, 2, ..., N \).

The minimum of \( D \) occurs at position \((N+1)/2\) in vector \( D \), when \( N \) is odd. When \( N \) is even there are 2 equal minimums at positions \( N/2 \) and \((N/2)+1\).

VII. ASCENDING SORT INDEX OF VECTOR \( D \)

The Ascending Sort Index Vector (ASIV) of a given vector gives the positions of the smallest element, then the next smallest element and so on in the ascending order. A given vector and its ASIV have the same size. Let \( G = [G(1) \ G(2) \ . \ . \ . \ G(N)] \) be the ASIV of \( D \), then the elements of \( G \) are determined as follows.

\( G(1) \) = Position of the smallest element of \( D \) in \( D \).
\( G(2) \) = Position of the second smallest element of \( D \) in \( D \).
…………………………………………………………
\( G(N) \) = Position of the largest element of \( D \) in \( D \).

The minimum of \( D \) occurs at position \((N+1)/2\) in vector \( D \), when \( N \) is odd. When \( N \) is even there are 2 equal minimums at positions \( N/2 \) and \((N/2)+1\).

The diagram shows the distances among Processors.

![Fig. 2 Distances among Processors](image)

Please note that \( T_j \) gives the traffic load of Processor \( P_j \) and \( D_j \) gives the distance to be covered from location \( L_j \).

Our objective is to minimize \( E \) by properly placing the Processors such that when \( T_1 \) is maximum, \( D_1 \) is minimum and vice versa. If \( T \) is in the descending order, \( D \) has to be in the ascending order. This means, the sort order of \( T \) should be opposite to that of \( D \).

Please note that \( T_j \) gives the traffic load of Processor \( P_j \) and \( D_j \) gives the distance to be covered from location \( L_j \).

Our objective is to minimize \( E \) by properly placing the Processors such that when \( T_1 \) is maximum, \( D_1 \) is minimum and vice versa. If \( T \) is in the descending order, \( D \) has to be in the ascending order. This means, the sort order of \( T \) should be opposite to that of \( D \).
G is basically a Permutation Vector. When D is permuted according to G, D gets sorted in the ascending order. The permutation operation is carried as follows.

For \( j = 1, 2, \ldots, N \) Let
\[
F(j) = D(G(j))
\]
(13)

Then, from the definition of G, we know that
\[
F(1) = \text{smallest element of } D.
\]
\[
F(2) = \text{next smallest element of } D.
\]
\[\vdots\]
\[
F(N) = \text{largest element of } D.
\]

Therefore F is the ascending sorted version of D.

Thus ASIV of a given vector represents the ascending sort order of that vector. From Eq.(10), the ASIV of D for \( N=4 \), written as, ASVI(D) is given by,
\[
G = \text{ASIV}(D) = \text{ASIV}(\begin{bmatrix} 6 & 4 & 4 & 6 \end{bmatrix}) = \begin{bmatrix} 2 & 3 & 1 & 4 \end{bmatrix}
\]

For \( N=5 \),
\[
G = \text{ASIV}(D) = \text{ASIV}(\begin{bmatrix} 10 & 7 & 6 & 7 & 10 \end{bmatrix})
\]
\[
= \begin{bmatrix} 3 & 2 & 4 & 1 & 5 \end{bmatrix}
\]
(14)

Because of the symmetric nature of vector D, (see Eqs (10) and (11)) the elements of G which is the AVIS of D can be determined and it can be shown that \( G[k] = k \text{ th element of } G \) is given by,
\[
G[k] = \frac{(N+k)}{2} \text{ when } N \text{ and } k \text{ are both odd or both even , and}
G[k] = \frac{(N+1-k)}{2} \text{ when } N \text{ odd and } k \text{ even or vice-versa.}
\]

VIII. DESCENDING SORT INDEX OF VECTOR T

The Descending Sort Index Vector (DSIV) of a given vector gives the positions of the largest element, then the next largest element and so on in that order. A given vector and its DSIV have the same size. Let
\[
S = [S(1) \ S(2) \ldots S(N)]
\]
be the DSIV of the given vector T. Then,
\[
S(1) = \text{Position of the largest element of } T \text{ in } T.
\]
\[
S(2) = \text{Position of the second largest element of } T \text{ in } T.
\]
\[\vdots\]
\[
S(N) = \text{Position of the smallest element of } T \text{ in } T.
\]

Since position \( j \) in vector T represents the \( j \) th processor, \( P_s(k) \) is the processor having the \( k \) th largest traffic load. That is,
\[
P_s(1) = \text{Processor having the largest traffic load.}
\]
\[
P_s(2) = \text{Processor having the second largest traffic load.}
\]
\[\vdots\]
\[
P_s(N) = \text{Processor having the smallest traffic load.}
\]

Example 1:
Let \( N=5 \) and
\[
T = [50 \ 60 \ 70 \ 80 \ 90]
\]
Elements of T give the traffic loads of Processors
\( P_1, P_2, \ldots, P_5 \) in that order. Now by inspection,
\[
S = \text{DSIV}(T) = [5 \ 4 \ 3 \ 2 \ 1]
\]
This means \( P_1 \) has the largest traffic load, \( P_4 \) has the next highest traffic and so on. The ASIV of D when \( N = 5 \) as given by Eq.(14) is reproduced here,
\[
G = \text{ASIV}(D) = [3 \ 2 \ 4 \ 1 \ 5]
\]

From the basic principle of minimization of dot product, we know that DSIV(T) should match with ASIV(D). Writing one vector below the other we get,
\[
S = \text{DSIV}(T) = [5 \ 4 \ 3 \ 2 \ 1] \text{ is order of Processors.}
\]
\[
G = \text{ASIV}(D) = [3 \ 2 \ 4 \ 1 \ 5] \text{ is order of Locations.}
\]
From this, we see that
\( P_5 \) should be placed at location 3.
\( P_4 \) should be placed at location 2.
\[
\vdots
\]
\( P_1 \) should be placed at location 5.
Therefore the general rule is,
place Processor \( P_{s(k)} \) at Location \( L_{g(k)} \) for \( k=1,2,\ldots, N \)

IX. OPTIMUM PLACEMENT ALGORITHM

To place \( N \) processors for minimum ECL:
1. Get the T vector from the given traffic load data.
2. Calculate the D vector from Eq.(12).
3. Find the DSIV of T by a suitable sorting algorithm (say Bubble sort) and call it S. That is, get \( S = \text{DSIV}(T) \).
4. Similarly, get \( G = \text{ASIV}(D) \)
5. place Processor \( P_{s(k)} \) at Location \( L_{g(k)} \) for \( k=1,2,\ldots, N \)

X. OPTIMUM PLACEMENT ON A 2-DIMENSIONAL GRID

Consider a 2-D grid of \( M \) rows and \( N \) columns as shown in Fig. 3. The Locations of the grid points are marked as 11, 12,..., and so on. The total manhattan distance to be covered by a given node to reach all other nodes depends on the position of the node. Let the node under consideration be at location \( (r,c) \) where \( r = \text{row value and } c = \text{column value of the node position.} \)
Then the total distance from this node to all other nodes is given by,
\[
D(r,c) = \sum_{i=1}^{M} \sum_{j=1}^{N} |c-j| + |r-i| \]
(15)
for \( r = 1,2,\ldots, M \text{ and } c = 1,2,\ldots, N \)

After the summation, this equation can be expressed as
\[
D(r,c) = \frac{1}{2} [M(c-c-1) + (N-c)(N-c+1)]
\]
\[
+ \frac{1}{2} [N(r-1) + (M-r)(M-r+1)]
\]
(16)

For optimum placement, we need to know the position of smallest \( D(r,c) \), next smallest \( D(r,c) \) and so on. To determine this, we mark the smallest value by metric 1, next smallest by 2 and so on. Consider the Example of a 3x7 grid. The smallest distance occurs at the center of the
grid and the next smallest at the immediate neighborhood and so on as shown in Fig. 4. We call these distances as effective distances.

Fig. 4 Distribution of Distances when rows = odd and columns = odd

Thus the table of Fig. 4 gives the effective distance matrix ED. For this ED, the ASIV is given by,

$$G = ASIV(ED) = [(2, 4) (2, 3) (1, 4) (2, 5) (3, 4) \ldots (3, 7)]$$

The size of G is 15 (which is equal to MxN).

G[1] gives the position of the smallest element of D.

G[2] gives the position of the next smallest element of D.

……………………………………………………………………

G[15] gives the position of the largest element of D.

The nature of ED matrix

When the no of rows of the grid M is odd and the no of columns N is odd, the center of the grid area is at row (M+1)/2 and column (N+1)/2 as in Fig 4. In this case the ij th element of ED is given by,

$$ED(i, j) = \left| i - \frac{1}{2} (M + 1) \right| + \left| j - \frac{1}{2} (N + 1) \right| + \frac{1}{2}$$

When M = even and N = odd, the ED matrix appears as shown in Fig. 6.

Fig. 6 Distribution of Distances when Rows = odd and columns = even

In this case, the ij th element of ED matrix is given by,

$$ED(i, j) = \left| i - \frac{1}{2} (M + 1) \right| + \left| j - \frac{1}{2} (N + 1) \right|$$

When M = odd and N = even, the ED matrix appears as shown in Fig. 7.

Fig. 7 Distribution of Distances when rows = even and columns = odd

In this case, the ij th element of ED matrix is given by Eq.(19).

XI. OPTIMUM PLACEMENT ALGORITHM

To place MxN processors for minimum ECL in a 2-D grid

1. Get the T vector from the given traffic load data. Size of T vector is MxN.

2. Calculate the ED (Effective Distance) matrix from Eqs.(17), (18) or (19) which is applicable.

3. Find the DSIV of T by a suitable sorting algorithm (say Bubble sort) and call it S. That is, get S = DSIV(T). Then, P_{sk} is the processor having the k th largest traffic load.

4. Get G = ASIV(ED).

G[1] = gives the row-column position of the smallest element of ED.

G[2] = gives the row-column position of the 2 nd smallest element of ED.

……………………………………………………………………

G[MxN] = gives the row-column position of the largest element of ED.

In general, G[k] = gives the row-column position of the k th smallest element of ED.
element of ED, for \( k=1,2,\ldots, M\times N \).
5. Place Processor \( PS(k) \) at location given by \( G(k) \) for \( k=1,2,\ldots, M\times N \).
   This minimizes the total ECL.

XII. EXAMPLE

Let the grid size be 3x7. The ED matrix is given as shown in Fig. 4. Let the Traffic Load be given by (values assumed arbitrarily) the \( T \) vector of size 21 as,

\[
T = [20 \ 25 \ 2 \ 10 \ 15 \ 5 \ 7 \ 9 \ 12 \ 13 \ 24 \ 3 \ 8 \ 21 \\
11 \ 14 \ 6 \ 23 \ 16 \ 1 \ 19]
\]

Then DSIV of \( T \) is,

\[
S = \begin{bmatrix}
2 & 11 & 18 & 14 & 1 & 21 & 19 & 5 & 16 & 10 & 9 & 15 & 4 & 8 \\
13 & 7 & 17 & 6 & 12 & 3 & 20
\end{bmatrix}
\]

From the matrix of Fig. 4, \( G \) is given by

\[
G = \begin{bmatrix}
(2,4) & (1,4) & (2,3) & (2,5) & (3,4) & (1,3) & (1,5) \\
(2,2) & (2,6) & (3,3) & (3,5) & (1,2) & (1,6) & (2,1) \\
(2,7) & (3,2) & (3,6) & (1,1) & (1,7) & (3,1) & (3,7)
\end{bmatrix}
\]

Now, from vectors \( S \) and \( G \),

- For \( k = 1 \), Processor 2 is placed at location (2,4).
- For \( k = 2 \), Processor 11 is placed at location (1,4).
- For \( k = 3 \), Processor 18 is placed at location (2,3).

In general, Processor \( PS(k) \) is placed at location \( G(k) \).
- For \( k = 21 \), Processor 20 is placed at location (3,7).

XIII. PRIORITY CONSIDERATIONS

When different Processors have different Priority levels based on the functionality and system logic, the placement location of the Processors can be modified to take care of the Priorities. Let the Priority of Processor \( P_k \) be quantified by a number \( R_k \) for \( k = 1, 2, 3, \ldots, M\times N \), such that higher the numerical value of \( R_k \) greater is the priority represented by it. For higher overall efficiency, higher priority processors should be placed closer to one another. Therefore from the consideration of priority, a high priority processor should be placed to have a low Effective Distance. The Traffic Load consideration also demands a similar requirement. Hence, the Priority number \( R_k \) for \( k = 1, 2, 3, \ldots, M\times N \) such that higher the numerical value of \( R_k \) greater is the priority represented by it.

\[ ET_k = R_k \cdot T_k \quad (20) \]

for \( k = 1, 2, \ldots, M\times N \).

Thus \( ET_k \) of Processor \( k \) represents both its Priority and Traffic Load combined together. The ET vector for the system is given by,

\[
ET = \begin{bmatrix} R_1 \cdot T_1 & R_2 \cdot T_2 & \ldots & R_J \cdot T_J \end{bmatrix} \quad (21)
\]

Where \( J = M\times N \) is the total number of Processors.

Now we use the ET vector instead of \( T \) vector in the Optimum placement Algorithm of section 11. This takes care of both Traffic Load and Priority requirements.

XIV. CONCLUSION

In this paper, we have presented a new technique to determine the placement of Processing Blocks to achieve maximum communication efficiency in both single line and 2-D grids.

REFERENCES