PROBLEMS

3.1 MOTIVATION

Communication occurs between entities (such as, file transfer packages, browsers, electronic mail software, etc.) in different systems. An entity is anything capable of sending or receiving information. A system is a physical object that contains one or more entities, such as computers, terminals, processors etc. Communication systems have homogenous and/or heterogeneous processors that are connected through a communication links. It provides the capability for the utilization of remote computing resources and allow for increased level of flexibility, reliability, and modularity. The distributed processing environments in which services provided for the network reside at multiple sites. Instead of single large machine being responsible for all aspects of process, each separate processor handles subset. In the distributed environments the programs or tasks are also often developed with the subsets of independent units under various environments. It has drawn tremendous attention in developing cost-effective and reliable applications to meet the desired requirements. Task allocation is the process of partitioning a set of programming modules into a number of processing groups, known as tasks, where each group executes on a separate processor. The general allocation problem is NP-complete. In these problems, suppose in any communication system there are ‘m’ tasks that are to be processed and it has at least ‘n’ processors which can process any of the m tasks but possibly in various time periods. Tasks are to be assigned to such processors so as to minimize the overall time to complete the execution of all ‘m’ tasks. Hungarian
approach [GILL 2002] is suggested for solving the assignment problem, but this approach is to be applicable only for m-tasks to the m-processors i.e. for balanced assignment problem. For the unbalanced assignment problem where m > n or n > m, the Hungarian method suggest to add the dummy tasks or processors to make the effectiveness matrix square.

The best-known research problem for such systems is the assignment problem, in which the total system time is to be minimized. The problem finding an assignment of tasks to nodes that minimizes total execution and communication cost (or time) was elegantly analyzed using a network flow model and network flow algorithms by [STON 1977, STON 1978] and a number of other researchers [YADA 2002, YADA 2003, KUMA 2001, SING 1999, SRIN 1999, PENG 1997, KUMA 1995a, KUMA 1995b, KUMA 1995c, KUMA 1995d, SAGA 1991, ZAHE 1991, BACA 1989, CHU 1969, KUMA 1999, KUMA 1996, KUMA 2002, KUMA 2006, CASA 1988, BOKH 1979] through various different algorithms. The load-balancing algorithms or load-leveling algorithms are based on the intuition that, for better resource utilization, it is desirable for the load in a distributed communication system to be balanced evenly. Thus a load-balancing algorithm tries to balance the total system load by transparently transferring the work load from heavily loaded nodes to lightly loaded nodes in an attempt to ensure good overall performance relative to some specific metric of system performance. When considering performance from the user point of view, the metric involved is often the response time of the processors. However, when performance is considered from the resource point of view, the metric involved is the total system throughput [SINH 2004]. Kumar et al [KUMA 1995a] suggested a modified and efficient task allocation approach in which authors have optimized the overall system
cost that include the execution and communication cost. Kumar et al [KUMA 1996]
and Kumar [KUMA 1999] have suggested task allocation technique, which are
dynamic in nature for optimizing the cost and reliability respectively. S. Srinivasan
[SRIN 1999] stated that software or a program to be run on the distributed computing
system is called a task, which is composed of intercommunication tasks, which
allocated to the different processor in the system. Several other methods have been
reported in the literature, such as, Integer Programming [DESS 1980], Branch and
Bound technique [RICH 1982], Matrix Reduction technique [SAGA 1991],
The Series Parallel Redundancy-Allocation problem has been studied with different
approaches, such as, Non-Linear techniques [TILL 1977], and Heuristic techniques

Therefore it is recorded that the research problem for such systems is of the
type tasks allocation problems, in which either system reliability is to be maximized
or total system time is to be minimized or overall effectiveness of the computing
systems is to be optimized under the pre specified constrains. Sager et al [SAGA
1991] discussed the tasks allocation problem in 1991; they considered a set of tasks
and a set of processors, which are less in comparison to the number of tasks. They
have suggested a heuristics approach by using matrix reduction technique for the
allocation of tasks to the various processors, and obtained optimal execution cost.
Kumar et al [KUMA 1995c, KUMA 1996] has studied the task allocation problem for
a set of heterogeneous processors on which the set of tasks to be allocated in such a
way that all the tasks got executed and load on each processor also maintained. These
models gave the total assignment cost and results were much better as compared to
Sagar et al [SAGA 1991]. The complexity of the algorithm is also less as they have mentioned the complexity comparisons to other approaches in their papers. Kumar [KUMA 2001] and Singh et al [SING 1999] also extended the work on tasks allocation problem for analyzing the various performance parameters. Recently, Yuan-Shun and Levitin [YUAN 2007] has used optimal resource allocation for maximizing performance and reliability in tree structured grid series. The genetic algorithm is used for reliability-oriented task assignment by Chin-Ching Chiu et al [CHIN 2006]. The work related to redundancy allocation problem with a mix of components is studied by Onishi et al [ONIS 2007].

PROBLEM-I

OPTIMAL TIME EVALUATION OF CS: AN EXHAUSTIVE SEARCH APPROACH

3.1.1 Objective

Let the given Communication System (CS) consists of a set \( P = \{ p_1, p_2, p_3, p_4, \ldots, p_n \} \) of \( n \) processors, interconnected by communication links and a set \( T = \{ t_1, t_2, t_3, t_4, \ldots, t_m \} \) of \( m \) tasks. The processing Execution Time [ET] of individual tasks corresponding to each processor are given in the form of matrix ETM (\( \cdot \)) of order \( m \times n \) and the Communication Time [CT] is taken in the square symmetric matrices CTM (\( \cdot \)) of order \( n \) respectively. The functions to measure ET and CT are then formulated. A procedure to assign all the tasks to the processors of the communicating systems based on execution time is to be designed in such a way that the overall time is to be optimizing under the pre specified constraints. The load on each processor has been also taken care off in order to balance the load of each processor.
3.1.2 Technique

Let the given system consists of a set \( P = \{ p_1, p_2, p_3, p_4, \ldots, p_n \} \) of \( n \) processors, interconnected by communication links and a set \( T = \{ t_1, t_2, t_3, \ldots, t_m \} \) of \( m \) tasks. The processing execution time of individual tasks corresponding to each processor are given in the matrix \( ETM(\cdot) \) of order \( m \times n \). The communication time is taken in the square symmetric matrix \( CTM(\cdot) \) of order \( n \). Initially, we obtain the task combinations in order to make the set of task(s) equals to number of processor. These combinations shall be \( n \times mC_{m-n} (= nl, \text{say}) \) and to be store in \( TCOMB(\cdot) \). Then we obtained an index, which is based on the processing time of the tasks to the processors for the execution of tasks to various processors, and also time of communication amongst the task to each of the combination, the maximum value of the index shall give us the optimal result. The assignment of tasks to processors may be done in different ways. To allocate the task to one of the processors, the minimum value of each row and column of \( ETM(\cdot) \) is obtained. Let \( \min \{ r_i \} \) represent the minimum row time value corresponding to the tasks \( t_i \) and \( \min \{ c_j \} \) represent the minimum column time value for processor \( p_j \). These values are then replaced to 0 in \( ETM(\cdot) \). For allocation purpose a modified version of row and column assignment method, namely Kumar et al [KUMA 1995c] is employed which allocates a task to a processor where it has minimum execution time. The overall execution time \([\text{Etime}]\) is expressed as the sum of execution and communication time of all the tasks as follows:

\[
\text{Etime} = \left[ \sum_{i=1}^{n} \left( \sum_{j=1}^{n} ET_{ij}x_{ij} \right) \right] + \left[ \sum_{j=1}^{n} \left( \sum_{i=1}^{n} CT_{ij}y_{ij} \right) \right]
\]

where,

\[
\text{ET} = \left[ \sum_{i=1}^{n} \left( \sum_{j=1}^{n} ET_{ij}x_{ij} \right) \right]
\]
where, \( x_{ij} = \begin{cases} 1, & \text{if } i^{th} \text{ task is assigned to } j^{th} \text{ processor} \\ 0, & \text{otherwise} \end{cases} \), and

\[
CT = \sum_{i=1}^{n} \left\{ \sum_{j=1}^{n} CT_{ij} y_{ij} \right\}
\]

\( y_{ij} = \begin{cases} 1, & \text{if the task assigned to processor } i \text{ communicate with the task assigned to processor } j \\ 0, & \text{otherwise} \end{cases} \)

Index = (Etime)\(^{l}\)

### 3.1.3 Computational Algorithm

To given an algorithmic representation to the technique mentioned in the previous section, let us consider a system in which a set of \( m \) tasks \( T = \{t_1, t_2, t_3, \ldots, t_m\} \) is to be executed on a set of \( n \) available processors \( P = \{p_1, p_2, p_3, \ldots, p_n\} \).

**Step-1:**

Input: \( m, n, \text{ETM}(\_), \text{CTM}(\_), \) ; \( nar := 0; \text{T ass} := \{ \_ \} \);

**Step-2:**

Select the task to form the combinations with other task(s), (say \( t_1, t_k \) & \( t_k, t_l \)). Store it in TCOMB(\( l \)), where \( l=1(1) nl \)

**Step-3:**

for \( l := 1 \) to \( nl \) do

begin

TCOMB(\( l \))

**Step-3.1:**

for \( k := 1 \) to \( n \) do

for \( i := 1 \) to \( n \) do

begin

Add \( k^{th} \) row of ETM(\_) to its \( i^{th} \) row. If all the values are become infinite, get next value of TCOMB(\( l \)) then repeat else Step-3.2.

end.

**Step-3.2:**

Modify CTM(\_) and, ETM(\_) as follow;

**Step-3.2.1:**

for \( k := 1 \) to \( n \) do

for \( j := 1 \) to \( n \) do

begin

Replace the corresponding values of \( k^{th} \) row and \( k^{th} \) column by zero in CTM(\_) and then add \( k^{th} \) row to \( i^{th} \) row and \( k^{th} \) column to \( i^{th} \) column, after that delete the \( k^{th} \) row and \( k^{th} \) column from CTM(\_).

end.
Step-3.2.2: Modify the ETM (.) by adding k\textsuperscript{th} row to i\textsuperscript{th} row and thereby deleting k\textsuperscript{th} row.

Step-3.2.3: Store modified, ETM (.) and, CTM (.) to NETM (.) and NCTM(.) respectively.

end.

Step-4:
for k := 1 to n do
for j := 1 to n do
begin
Find the minimum of k\textsuperscript{th} row (Say mn\textsubscript{kj}) of NETM (.) falling in j\textsuperscript{th} column and replace it by zero.
end.

Step-5:
for k := 1 to n do
for j := 1 to n do
begin
Find the minimum of j\textsuperscript{th} column (say mc\textsubscript{kj}) of NETM (.), which lies in k\textsuperscript{th} row, and replace it by zero.
end.

Step-6:
for j := 1 to n do
begin
for k := 1 to n do
begin
Search for a row in NETM (.), which has only one zero say, at the position (k,j) and assign task(s) corresponding to this position.
nar := nar + 1;
far(k) := j;
\text{T}_{\text{ass}} := \text{T}_{\text{ass}} \cup \{t_k\};
end;
end.

Step-7:
for j := 1 to n do
begin
for k := 1 to n do
begin
Search for a column which has only one zero entry say, at the position (k,j) and assign task(s) corresponding to this position.
nar := nar + 1;
far(k) := j;
\text{T}_{\text{ass}} := \text{T}_{\text{ass}} \cup \{t_k\};
end;
end.
Step-8:

if nar ≠ n
Pick-up an arbitrary zero entry say, at the position (k,j) and assign task(s) corresponding to this position.
nar : = nar + 1; far(k) : = j;
Tass : = Tass ∪{tk};
else
Check column(s) positions of zero(s) in unassigned row(s). Check the row(s) for any previous assignment in the corresponding column(s). Find the minimum of the entire elements for the remaining rows and replace it zero and then go to step-6.

Step-9:

Evaluate Execution Time as;
ET : = 0.0;
for i : = 1 to n do
ET : = ET + ET(i, far(i))
ETT(l):= ET

Step-10:

Evaluate Communication Time as;
CT : = 0.0;
for i = 1 to n do
CT : = CT + CT(i, far(i))
CTT(l):= CT

Step-11:

Execution Time [Etime] and Index are thus calculated as:
Step-11.1:
Etime(l): = ETT(l)+ CTT(l)
Step-11.2:
Index(l): = 1 /Etime(l)

Step-12:
If l < nl then go to step-3.

Step-13:
List: Index (l); Etime (l);
end.
Step-14:
Stop.

3.1.4 Implementation

Example-I

Consider a system consisting of a set T = {t1, t2, t3, t4} of 4 tasks and a set P =
{p1, p2, p3} of 3 processors,
Step-1: Input: 4, 3

\[
\begin{bmatrix}
   P_1 & P_2 & P_3 \\
   t_1 & 8 & 12 & 7 \\
   t_2 & 9 & 8 & 11 \\
   t_3 & 12 & 9 & 6 \\
   t_4 & 10 & 11 & 12
\end{bmatrix}
\]

\[
ETM(\cdot) = \begin{bmatrix}
   t_1 & t_2 & t_3 & t_4 \\
   t_1 & 0 & 3 & 6 & 9 \\
   t_2 & 3 & 0 & 4 & 5 \\
   t_3 & 6 & 4 & 0 & 7 \\
   t_4 & 9 & 5 & 7 & 0
\end{bmatrix}
\]

\[
CTM(\cdot) = \begin{bmatrix}
   3 & 4 & 2 & 4 & 2 & 3 & 1 & 4 & 1 & 3 & 1 & 2 \\
   4 & 3 & 4 & 2 & 3 & 2 & 4 & 1 & 3 & 1 & 2 & 1
\end{bmatrix}
\]

Step-2:

\[
TCOMB(\cdot) = \begin{bmatrix}
   3 & 4 & 2 & 4 & 2 & 3 & 1 & 4 & 1 & 3 & 1 & 2 \\
   4 & 3 & 4 & 2 & 3 & 2 & 4 & 1 & 3 & 1 & 2 & 1
\end{bmatrix}
\]

Step-3:

\[
TCOMB(1) = \begin{bmatrix}
   3 \\
   4
\end{bmatrix}
\]

Step-3.1-3.2:

\[
ETM(\cdot) = \begin{bmatrix}
   p_1 & p_2 & p_3 \\
   t_5 & t_4 & 22 & 20 & 18
\end{bmatrix}
\]

\[
NCTM(\cdot) = \begin{bmatrix}
   p_1 & p_2 & p_3 \\
   r_1 & 0 & 3 & 6 \\
   r_2 & 3 & 0 & 4 \\
   r_3 & r_4 & 6 & 4 & 0
\end{bmatrix}
\]
\[ NETM (, ) = \begin{array}{c} \ \{ i \} \ \{ j \} \\
1 & r_1 & 8 & 12 & 7 \\
2 & r_2 & 9 & 8 & 11 \\
3 & r_3 & 22 & 20 & 18 \\
\end{array} \]

Step-4 & 5:

On applying modified Hungarian method devised by Kumar et al [KUMA 1995c] to assign the task, \( \min \{ r_{ij} \} \) from \( NETM (, ) \) for every \( i, r_{13} = 7, r_{22} = 8, \)
\( r_{33} = 18 \). Making \( r_{13} = r_{22} = r_{33} = 0 \). Again \( \min \{ c_{ij} \} \) from \( NETM (, ) \) for every
\( j, \) are \( c_{11} = 8, c_{22} = 0, c_{33} = 0 \). Making \( c_{11} = 0 \), so that, we get,

\[ NETM (, ) = \begin{array}{c} \ \{ i \} \ \{ j \} \\
1 & r_1 & 0 & 12 & 0 \\
2 & r_2 & 9 & 0 & 11 \\
3 & r_3 & 22 & 20 & 0 \\
\end{array} \]

Step-6, 7& 8:

After implementing assignment process, the first set of the allocation is thus
obtained.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Processors</th>
<th>ET</th>
<th>CT</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>( p_1 )</td>
<td>08</td>
<td>03</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>( p_2 )</td>
<td>08</td>
<td>06</td>
</tr>
<tr>
<td>( t_3 * t_4 )</td>
<td>( p_3 )</td>
<td>18</td>
<td>04</td>
</tr>
</tbody>
</table>

Step-9:

\[ ETT (1) := 34 \]

Step-10:

\[ CTT (1) := 13 \]

Step-11:

\[ Etime (1) := 47 \]

Step-12:

\[ Index (1) := 0.02127650 \]

59
Step-13:

On repeating the above process, the assignments and their corresponding related values of ET, CT, Etime, and Indexes are thus obtained, which are shown in the following table-1:

<table>
<thead>
<tr>
<th>S. No.</th>
<th>ET</th>
<th>CT</th>
<th>Etime</th>
<th>INDEX</th>
<th>ASSIGNMENT-I</th>
<th>ASSIGNMENT-II</th>
<th>ASSIGNMENT-III</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>34</td>
<td>13</td>
<td>47</td>
<td>0.02127650</td>
<td>t₁  p₁</td>
<td>t₂  p₂</td>
<td>t₃ * t₄</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
<td>17</td>
<td>51</td>
<td>0.01960780</td>
<td>t₁  p₁</td>
<td>t₂  p₂</td>
<td>t₄ * t₃</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
<td>13</td>
<td>46</td>
<td>0.02173910</td>
<td>t₁  p₁</td>
<td>t₃  p₃</td>
<td>t₅ * t₄</td>
</tr>
<tr>
<td>4</td>
<td>33</td>
<td>22</td>
<td>55</td>
<td>0.01818181</td>
<td>t₁  p₁</td>
<td>t₃  p₃</td>
<td>t₄ * t₂</td>
</tr>
<tr>
<td>5</td>
<td>34</td>
<td>17</td>
<td>51</td>
<td>0.01960780</td>
<td>t₁  p₃</td>
<td>t₄  p₁</td>
<td>t₅ * t₃</td>
</tr>
<tr>
<td>6</td>
<td>34</td>
<td>22</td>
<td>56</td>
<td>0.01785710</td>
<td>t₁  p₃</td>
<td>t₄  p₁</td>
<td>t₅ * t₂</td>
</tr>
<tr>
<td>7</td>
<td>32</td>
<td>13</td>
<td>45</td>
<td>0.02222222</td>
<td>t₂  p₁</td>
<td>t₃  p₃</td>
<td>t₁ * t₄</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>16</td>
<td>48</td>
<td>0.02083333</td>
<td>t₂  p₂</td>
<td>t₃  p₃</td>
<td>t₄ * t₁</td>
</tr>
<tr>
<td>9</td>
<td>31</td>
<td>17</td>
<td>48</td>
<td>0.02083333</td>
<td>t₂  p₂</td>
<td>t₄  p₁</td>
<td>t₁ * t₃</td>
</tr>
<tr>
<td>10</td>
<td>31</td>
<td>16</td>
<td>47</td>
<td>0.02127650</td>
<td>t₂  p₂</td>
<td>t₄  p₁</td>
<td>t₃ * t₁</td>
</tr>
<tr>
<td>11</td>
<td>34</td>
<td>16</td>
<td>50</td>
<td>0.02000000</td>
<td>t₃  p₃</td>
<td>t₄  p₂</td>
<td>t₂ * t₁</td>
</tr>
<tr>
<td>12</td>
<td>34</td>
<td>22</td>
<td>56</td>
<td>0.01785710</td>
<td>t₃  p₃</td>
<td>t₄  p₂</td>
<td>t₁ * t₂</td>
</tr>
</tbody>
</table>

Step-14:

Stop.
Example-II

The results of a system which consist a set $T = \{t_1, t_2, t_3, t_4, t_5\}$ of 5 tasks and a set $P = \{p_1, p_2, p_3\}$ of 3 processors where,

$$
\begin{array}{ccc}
  & P_1 & P_2 & P_3 \\
t_1 & 8 & 12 & 7 \\
t_2 & 9 & 8 & 11 \\
t_3 & 12 & 9 & 6 \\
t_4 & 10 & 11 & 12 \\
t_5 & 7 & 6 & 2 \\
\end{array}
$$

$$
ETM(,) = \begin{array}{c}
  t_1 \\
t_2 \\
t_3 \\
t_4 \\
t_5 \\
\end{array} \begin{array}{c}
  0 \\
  3 \\
  6 \\
  9 \\
  8 \\
\end{array}
$$

$$
CTM(,) = \begin{array}{c}
  t_1 \\
t_2 \\
t_3 \\
t_4 \\
t_5 \\
\end{array} \begin{array}{c}
  3 \\
  0 \\
  4 \\
  5 \\
  6 \\
\end{array}
$$

The following tables shows the results as obtain after implementing the present algorithm

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Processors</th>
<th>ET</th>
<th>CT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1 * t_4$</td>
<td>$p_1$</td>
<td>18</td>
<td>3</td>
</tr>
<tr>
<td>$t_2$</td>
<td>$p_2$</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>$t_3 * t_5$</td>
<td>$p_3$</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Processors</th>
<th>Etime</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1 * t_4$</td>
<td>$p_1$</td>
<td>47</td>
<td>0.021276595</td>
</tr>
<tr>
<td>$t_2$</td>
<td>$p_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_3 * t_5$</td>
<td>$p_3$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.1.5 Conclusion

The first problem of the thesis discusses an assignment model through optimization technique for the performance enhancement of communication system. In this problem we have chosen such a communication system in which numbers of tasks are more than the number of processors. The present method deals the case when the index is based on the processing time of the tasks to the processors for the execution of tasks to various processors and also communication time amongst the tasks. The method is presented in computational algorithmic form and implemented on the several sets of input data to test the performance and effectiveness of the algorithm. The optimal result of the Example-I are shown in the following table:

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Processors</th>
<th>Etime</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_3$</td>
<td>$p_3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_2$</td>
<td>$p_2$</td>
<td>45</td>
<td>0.02222222</td>
</tr>
<tr>
<td>$t_1 t_4$</td>
<td>$p_1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The graphical representation of the results mentioned in Table-1 of Step-12 of the implementation part of this paper is shown in the following graphs:

Graph-1: Index Graph
Graph-2: Execution Time Graph

The following table shows the results of Example-II given in its implementation part of problem-I of this chapter, as obtained after implementing the present algorithm:

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Processors</th>
<th>Etime</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1 \times t_4$</td>
<td>$p_1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_2$</td>
<td>$p_2$</td>
<td>47</td>
<td>0.0212765</td>
</tr>
<tr>
<td>$t_3 \times t_5$</td>
<td>$p_3$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.1.5.1 Comparison

The run time complexity of the algorithm is measured $O(n \times m \cdot C_{m,n})$ time. Our time complexity is better than $O(m^2 \cdot n)$ of [SAGA 1991] and $O(n^m)$ to [PENG 1997, RICH 1982]. The performance of the algorithm is compared with that of [SAGA 1991] and [PENG 1997, RICH 1982] and results for complexity comparison are shown below:

<table>
<thead>
<tr>
<th>Tasks</th>
<th>$O(n \times m \cdot C_{m,n})$ Present−Alg</th>
<th>$O(m^2 \cdot n)$ [SAGA 1991]</th>
<th>$O(n^m)$ [PENG1997, RICH1982]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(m,n)$</td>
<td>(5,3)</td>
<td>30</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>(6,3)</td>
<td>60</td>
<td>108</td>
</tr>
<tr>
<td></td>
<td>(7,3)</td>
<td>105</td>
<td>147</td>
</tr>
<tr>
<td></td>
<td>(8,3)</td>
<td>168</td>
<td>192</td>
</tr>
<tr>
<td></td>
<td>(10,4)</td>
<td>630</td>
<td>400</td>
</tr>
<tr>
<td></td>
<td>(10,5)</td>
<td>756</td>
<td>500</td>
</tr>
</tbody>
</table>
OPTIMIZING EXECUTION TIME OF CS: MATRIX PARTITIONING APPROACH

3.2.1 Objective

Consider a communication system which consist a set \( P = \{p_1, p_2, \ldots, p_n\} \) of "n" processors, interconnected by an arbitrary network. The processors have local memory only and do not share any global memory. The processor graph is a convenient abstraction of the processors together with interconnection network. It has processors as nodes and there is a weighted edge and distance between two nodes if the corresponding processors can communicate with each other. The weight \( w_{ij} \) and distance \( d_{ij} \) on the edge between processors \( p_i \) and \( p_j \) represent the delay time involved in sending or receiving the message of \( d_{ij} \) length from one processor to another. In order to have an approximate estimate of this time, irrespective of the two processors, we use the average of the weights on all the edges in the processor graph.

A set \( T = \{t_1, t_2, \ldots, t_m\} \) of "m" tasks is considered at hand to be executed on "n" processors. The Execution Time (ET) of these tasks on all the processors is given in the form of Execution Time Matrix \( [ETM(.)] \) of order \( m \times n \). The Communication Time (CT) is taken in the form of a symmetric matrix named as Communication Time Matrix \( [CTM(.)] \), which is of order \( m \). In order to make the best use of the resources of the system for that we would like to distribute the load on each processor in such a way that allocated load on the processors should be evenly balanced. The proposed model discusses the following issues:

- Developing the method to form the sub problems,
- Formulating the method to assign all tasks,
3.2.2 Technique

Since the numbers of task are more than the number of processors, so that we divide the problem of unbalanced tasks assignment in to sub problems, which becomes the balanced tasks assignment problems. Obtain the sum of each row and each column except infinity time (infinity time should be kept aside) form the ETM (, ) and store the results in to Sum_Row() and Sum_Column(), each of them one dimensional arrays. Select the first set of tasks, (this set of tasks shall contain only as many tasks as the number of processor in the communication system) on the basis of minimum time against the tasks in the Sum_Row() array. Store the result in to ETM (, , ) a two dimensional array. Repeat the process until remaining tasks are either less than or equal to the number of processors. If the tasks are equal to the processors of the communication system then it will becomes the last sub problem, else to form the last problem we have to delete the column (processor) form ETM(,) on the basis of Sum_Column() array. Assignment of these tasks for each sub problem, we apply the Kumar et al algorithm [KUMA 1995c]. The communication time of those tasks, which are allocated on the same processor, becomes zero. For each sub problem we calculate the exaction time and communication time of each processor and store the result in a linear array PET(j) and PCT(j) respectively where j= 1,2,...n. Finally, sum up the value of PET (j) and PCT (j), (j=1,...,n) to obtain Etime. It is the total optimal time for the complete assignments.

\[
PET(j) = \sum_{i=1}^{n} \sum_{j=1}^{n} ET_{ij} \times x_{ij};
\]
\[ P \text{CT}(j) = \sum_{i=1}^{n} \sum_{j=1}^{n} CT_{ij} y_{ij} ; \]

\[ E\text{time} = \left[ \sum_{i=1}^{n} \left\{ \sum_{j=1}^{n} ET_{ij} x_{ij} \right\} + \sum_{j=1}^{n} \left\{ \sum_{i=1}^{n} CT_{ij} y_{ij} \right\} \right] \]

where, \( x_{ij} = \begin{cases} 1, & \text{if } i^{th} \text{ task is assigned to } j^{th} \text{ processor} \\ 0, & \text{otherwise} \end{cases} \)

and

\( y_{ij} = \begin{cases} 1, & \text{if the task assigned to processor } i \text{ communicates with the task assigned to processor } j \\ 0, & \text{otherwise} \end{cases} \)

### 3.2.3 Computational Algorithm

To given an algorithmic representation to the proposed method as discussed, we have to concentrate on such a system which consist a set \( P = \{p_1, p_2, \ldots, p_n\} \) of \( n \) processor and a set \( T = \{t_1, t_2, \ldots, t_m\} \) of \( m \) executable tasks which are to be processed by any one of the processor of the system.

**Step-1:**

Input: \( m, n, \text{ETM}(\cdot), \text{CTM}(\cdot) \)

**Step-2:**

Obtain the sum of each row of the ETM(\cdot) in such a way that, if any time values (s) is (are) \( \infty \) then keeping it aside along with sum of that row (just to avoid the condition \( \text{ET} + \infty = \infty \)). Store the results in one-dimensional array \( \text{Sum \_Row}(\cdot) \) of order \( m \).

**Step-3:**

Obtain the sum of each column of the ETM(\cdot), in such a way that if any time value (s) is (are) \( \infty \) then keeping it aside along with product of that column (just to avoid the condition \( \text{ET} + \infty = \infty \)). Store the results in one-dimensional array \( \text{Sum \_Column}(\cdot) \) of order \( n \).

**Step-4:**

Partitioned the execution time matrix ETM(\cdot) of order \( m \times n \) in to sub matrices such that the order of these matrices become square i.e. number of row should be equal to number of column. Partitioning to be made as mentioned in the following steps.

**Step-4.1:**

Select the \( n \) task on basis of \( \text{Sum \_Row}(\cdot) \) array i.e. select the 'n' task corresponding to most minimum sum to next minimum sum, if there is a tie select arbitrarily. (For the cases in which product is \( \text{ET} + \infty = \infty \), minimum value depends only ET and the impact of \( \infty \) is to be neglected.
Step-4.2:
Store the result in the two dimensional array $ETM(i, j)$ to form the sub matrices of the sub problems.

Step-4.3:
If all the tasks are selected then go to step 4.7 else steps 4.4

Step-4.4:
Repeat the step 4.1 to 4.3 until the number of task become less than $n$.

Step-4.5:
Select the remaining task say $r$, $r < n$, select the $r$ processors on the basis of $\text{Sum \_ Column}(i)$ array i.e. the processors corresponding to the most minimum sum to next minimum, if there is a tie select arbitrarily (for the cases in which product is $ET + \infty = \infty$, minimum value depend only $ET$ and the impact of $\infty$ is to be neglected.

Step-4.6:
Store the result in the two dimensional array $ETM(i, j)$, which is a last sub problems.

Step-4.7:
List of all the sub problems formed through Step 4.1 to 4.6 and repeat step 5 to step 12 to solve each of these sub problems.

Step-5:
Find the minimum of each row of $ETM(i, j)$ and replace it by 0.

Step-6:
Find the minimum of each column of $ETM(i, j)$ and replace it by 0.

Step-7:
Search for a row in $ETM(i, j)$ which has only one zero and assign the task(s) corresponding to this position. Add one to the counter that is $nar = nar + 1$ and also store this position.

Step-8:
Search for a column in $ETM(i, j)$ which has only one zero and assign the task(s) corresponding to this position. Add one to the counter that is $nar = nar + 1$ and also store this position.

Step-9:
Check whether $nar = n$ if not than pickup an arbitrary 0 and assigned task(s) corresponding to this position. Add one to the counter that is $nar = nar + 1$ and also store this position, else, Check column(s) position of 0’s in unassigned row(s). Check the row(s) for any previous assignment in the corresponding column(s). Find the minimum of the entire elements for the remaining rows and replace it zero, Go to Step-7, Else Step-10.

Step-10:
Evaluate Execution Time [PET ()].

Step-11:
Evaluate Communication Time [PCTC ()].

Step-12:
Execution Time (Etime) are thus calculated as:
$Etime = PET() + PCT()$

Step-13:
Stop.
3.2.4 Example

Consider a communication system which is consisting of a set \( P = \{p_1, p_2, p_3\} \) of "\( n = 3 \)" processors connected by an arbitrary network. The processors only have local memory and do not share any global memory. The processor connections graph is depicted in figure-1 and tasks execution graph also pictorially depicted in figure-2. A set \( T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\} \) of "\( m = 8 \)" executable tasks which may be portion of an executable code or a data file. The communication graph is depicted in figure-3.

\[\text{Figure- 1: Processors Graphs}\]

\[\text{Figure - 2: Tasks Execution Graph}\]
Step-1

On converting its matrix representations, we have, $m = 8$, $n = 3$,

\[
\begin{array}{c|ccc}
  & p_1 & p_2 & p_3 \\
\hline
  t_1 & 6 & 3 & 5 \\
  t_2 & 4 & 2 & 3 \\
  t_3 & 3 & 1 & 2 \\
  t_4 & 5 & 2 & \infty \\
  t_5 & 3 & 4 & 2 \\
  t_6 & 6 & \infty & 6 \\
  t_7 & 5 & 6 & 7 \\
  t_8 & \infty & 2 & 5 \\
\end{array}
\]

ETM() =
\[
\begin{array}{cccccccc}
& t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 \\
t_1 & 0 & 3 & 4 & 2 & 6 & 8 & 1 & 0 \\
t_2 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \\
t_3 & 4 & 0 & 0 & 4 & 3 & 2 & 0 & 0 \\
CTM(\cdot)= & t_4 & 2 & 0 & 4 & 0 & 5 & 3 & 2 & 5 \\
t_5 & 6 & 0 & 3 & 5 & 0 & 0 & 0 & 0 \\
t_6 & 8 & 0 & 2 & 3 & 0 & 0 & 6 & 8 \\
t_7 & 1 & 0 & 0 & 2 & 0 & 6 & 0 & 5 \\
t_8 & 0 & 5 & 0 & 5 & 0 & 8 & 5 & 0 \\
\end{array}
\]

Step-2

Obtain the sum of each row and column of ETM(\cdot), as:

\[
\text{Sum \_Row} = \begin{array}{cccccccc}
t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 \\
14 & 9 & 6 & 7+\infty & 9 & 12+\infty & 18 & 7+\infty \\
\end{array}
\]

\[
\text{Sum \_Column} = \begin{array}{ccc}
p_1 & p_2 & p_3 \\
32+\infty & 20+\infty & 30+\infty \\
\end{array}
\]
We partitioned the matrix ETM (,) to define the first sub problem ETM (1,) by selecting rows corresponding \( t_3, t_4, t_8 \) and second sub problem ETM (2,) by selecting rows corresponding \( t_2, t_5, t_6 \) and on the basis of the Sum_Column, by deleting columns corresponding \( p_1 \), and after the selecting the remaining two tasks \( t_1, t_7 \) to form the last sub problem ETM (3,), as there were only two tasks, for which we required two processors. So that the modified matrices are as;

**Sub Problem-I:**

<table>
<thead>
<tr>
<th></th>
<th>( p_1 )</th>
<th>( p_2 )</th>
<th>( p_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_3 )</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( t_4 )</td>
<td>5</td>
<td>2</td>
<td>( \infty )</td>
</tr>
<tr>
<td>( t_8 )</td>
<td>( \infty )</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

**Problem-II:**

<table>
<thead>
<tr>
<th></th>
<th>( p_1 )</th>
<th>( p_2 )</th>
<th>( p_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_2 )</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>( t_5 )</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>( t_6 )</td>
<td>6</td>
<td>( \infty )</td>
<td>6</td>
</tr>
</tbody>
</table>

and, **Sub Problem-III:**

<table>
<thead>
<tr>
<th></th>
<th>( p_2 )</th>
<th>( p_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>( t_7 )</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

**Step-4 & 5:**

On applying modified Hungarian method devised by Kumar et al [KUMA 1995c] to assign the tasks, on the basis of min \( \{ r_i \} \) and min \( \{ c_j \} \) from execution time matrices for every \( i \) and \( j \). We put \( r_{ij} = 0 \) and \( c_{ij} = 0 \), for every \( i \) and \( j \). So that, the modified matrices for each sub problem are mentioned below:
Sub Problem-I:

<table>
<thead>
<tr>
<th></th>
<th>( p_1 )</th>
<th>( p_2 )</th>
<th>( p_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_4 )</td>
<td>5</td>
<td>0</td>
<td>( \infty )</td>
</tr>
<tr>
<td>( t_5 )</td>
<td>( \infty )</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

NETM (1,) =

Sub Problem-II:

<table>
<thead>
<tr>
<th></th>
<th>( p_1 )</th>
<th>( p_2 )</th>
<th>( p_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_2 )</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>( t_4 )</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>( t_6 )</td>
<td>0</td>
<td>( \infty )</td>
<td>6</td>
</tr>
</tbody>
</table>

NETM (2,) =

and, Sub Problem-III:

<table>
<thead>
<tr>
<th></th>
<th>( p_2 )</th>
<th>( p_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_7 )</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>

NETM (3,) =

Step-6, 7& 8:

After implementing assignment process, the solution set, of the each of sub problem, the allocation is thus obtained.

Solution for the Sub Problem-I:

<table>
<thead>
<tr>
<th>Tasks ( \rightarrow ) Processors</th>
<th>ET</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_3 ) ( \rightarrow ) ( p_3 )</td>
<td>02</td>
</tr>
<tr>
<td>( t_4 ) ( \rightarrow ) ( p_1 )</td>
<td>05</td>
</tr>
<tr>
<td>( t_5 ) ( \rightarrow ) ( p_2 )</td>
<td>02</td>
</tr>
</tbody>
</table>

Solution for the Sub Problem-II:

<table>
<thead>
<tr>
<th>Tasks ( \rightarrow ) Processors</th>
<th>ET</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_2 ) ( \rightarrow ) ( p_2 )</td>
<td>02</td>
</tr>
<tr>
<td>( t_5 ) ( \rightarrow ) ( p_3 )</td>
<td>02</td>
</tr>
<tr>
<td>( t_6 ) ( \rightarrow ) ( p_1 )</td>
<td>06</td>
</tr>
</tbody>
</table>
Solution for the Sub Problem-III:

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Processors</th>
<th>ET</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>p₃</td>
<td>05</td>
</tr>
<tr>
<td>t₇</td>
<td>p₂</td>
<td>06</td>
</tr>
</tbody>
</table>

Step-9:

After implementing the process, we obtain the following set of complete assignments along with execution and communication times of each processor.

<table>
<thead>
<tr>
<th>Processor→ Tasks</th>
<th>PET()</th>
<th>PCT ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>p₁ → t₄, t₆</td>
<td>11</td>
<td>43</td>
</tr>
<tr>
<td>p₂ → t₂, t₇, t₈</td>
<td>10</td>
<td>26</td>
</tr>
<tr>
<td>p₃ → t₁, t₃, t₅</td>
<td>9</td>
<td>25</td>
</tr>
</tbody>
</table>

Step-10:

PET ( ) := 30
PCT ( ) := 94
Etime := 124

Step-12:

Stop.

3.2.5 Conclusion

The model discussed in this problem provide an optimal solution for assigning a set of “m” tasks of a program to a set of “n” processors where m > n in a communication system that to minimize the overall time of the system and the load of all allocated tasks on all the processors evenly balanced. The communication time and execution time on different processors has been obtained.

<table>
<thead>
<tr>
<th>Processors of communication system</th>
<th>P₁</th>
<th>P₂</th>
<th>P₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time of each processor PET(ₜ)</td>
<td>11</td>
<td>10</td>
<td>09</td>
</tr>
<tr>
<td>Communication time of each processor PCT(ₜ)</td>
<td>43</td>
<td>26</td>
<td>25</td>
</tr>
<tr>
<td>Total time of each processor time(ₜ)</td>
<td>54</td>
<td>36</td>
<td>34</td>
</tr>
</tbody>
</table>
The model addressed in this problem is based on the consideration of execution and communication times of the tasks to the processors. Keeping in view we suggested a modified method to assign all the tasks as per the required availability of processors so that none of the tasks get remains unexecuted and the present approach does not require to adding dummy processors. The algorithm is capable to improve the performance of the given communication system by optimally assigning the tasks to various processors of the system. The final results of the problem considered in the implementation of the algorithm are mentioned as,

\[
\begin{array}{cccc}
\text{Tasks} & \rightarrow & \text{Processors} & ET & CT & Time & Etime \\
\tau_4 ; \tau_6 & \rightarrow & P_1 & 11 & 43 & 54 \\
\tau_2 ; \tau_7 ; \tau_8 & \rightarrow & P_2 & 10 & 26 & 36 & 124 \\
\tau_1 ; \tau_3 ; \tau_5 & \rightarrow & P_3 & 09 & 25 & 34 \\
\end{array}
\]

The model discussed here, would be useful to the network system designer working in the field of distributed communication systems. The developed model is programmed in C++ and implemented the several sets of input data to test the effectiveness and efficiency of the algorithm. It is recorded that the model is suitable for arbitrary number of processors with the random program structure.

3.2.5.1 Comparison

The run time complexity of the algorithm is measured \( O(6mn^2) \) time. Here it is concluded that it remains same as [KUMA 1998, YADA 2002]. Our time complexity is better than \( O(m^2n) \) of [SAGA 1991] and \( O(n^m) \) to [PENG 1997, RICH 1982]. The performance of the algorithm is compared with that of [SAGA 1991] and [PENG 1997, RICH 1982] and result in both cases, i.e. when tasks increases against the fixed processors and the case in which the tasks remain fix, while increase in processors are shown below:
### Case-1: For fixed processors n = 3

<table>
<thead>
<tr>
<th>Tasks</th>
<th>(O(6mn - n^2))</th>
<th>(O(m^2 n))</th>
<th>(O(n^m))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m)</td>
<td>(\text{Present - Alg} )</td>
<td>(\text{[SAGA1991]})</td>
<td>(\text{[PENG1997, RICH1982]})</td>
</tr>
<tr>
<td>5</td>
<td>81</td>
<td>75</td>
<td>243</td>
</tr>
<tr>
<td>6</td>
<td>99</td>
<td>108</td>
<td>729</td>
</tr>
<tr>
<td>7</td>
<td>117</td>
<td>147</td>
<td>2187</td>
</tr>
<tr>
<td>8</td>
<td>135</td>
<td>192</td>
<td>6561</td>
</tr>
<tr>
<td>9</td>
<td>153</td>
<td>243</td>
<td>19683</td>
</tr>
<tr>
<td>10</td>
<td>171</td>
<td>300</td>
<td>59049</td>
</tr>
</tbody>
</table>

### Case-2: For fixed tasks \(m = 10\)

<table>
<thead>
<tr>
<th>Processors</th>
<th>(O(6mn - n^2))</th>
<th>(O(m^2 n))</th>
<th>(O(n^m))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)</td>
<td>(\text{Present - Alg} )</td>
<td>(\text{[SAGA1991]})</td>
<td>(\text{[PENG1997, RICH1982]})</td>
</tr>
<tr>
<td>3</td>
<td>171</td>
<td>300</td>
<td>59049</td>
</tr>
<tr>
<td>4</td>
<td>224</td>
<td>400</td>
<td>1048576</td>
</tr>
<tr>
<td>5</td>
<td>275</td>
<td>500</td>
<td>9765625</td>
</tr>
<tr>
<td>6</td>
<td>324</td>
<td>600</td>
<td>60466176</td>
</tr>
<tr>
<td>7</td>
<td>371</td>
<td>700</td>
<td>282475249</td>
</tr>
<tr>
<td>8</td>
<td>416</td>
<td>800</td>
<td>1073741824</td>
</tr>
</tbody>
</table>

### PROBLEM-III

A NEW APPROACH FOR ASSIGNMENT IN CS: BASED ON TASKS WEIGHT

#### 3.3.1 Objective

Consider a communication system which consist a set \(P = \{p_1, p_2, ..., p_n\}\) of “n” processors, interconnected by an arbitrary network. A set \(T = \{t_1, t_2, ..., t_m\}\) of “m” tasks is considered at hand to be executed on “n” processors. The Task Execution Time [TET] of these tasks on all the processors is given in the form of Task Execution Time Matrix [TETM (\(\_\))], of order \(m \times n\). The Task Weight (WT) are taken in the form of a linear array named as Task Weight Matrix [TWM (\(\_\))], which is of order \(m\). In
order to make the best use of the resources of these systems for that we would like to 
distribute the load on each processor in such a way that allocated load on the 
processors should be evenly balanced.

3.3.2 Proposed Method

Since the number of task are more than the number of processors, so that we 
divide the problem of unbalanced tasks assignment in to sub problems, which are to 
be balanced tasks assignment problems. First of all arrange the weights of the tasks in 
descending order and store the result in a linear array namely, wt_seq(). Select the 
first set of tasks, (this set of tasks shall contain only as many tasks as the number of 
processor in the communication system) on the basis of the wt_seq () array. Store the 
result in to TETM ( , , ) a two dimensional array. Repeat the process until remaining 
tasks are either less than or equal to the number of processors. If the tasks are equal to 
the processors of the communication system then it will becomes the last sub 
problem, else to form the last problem we have to add dummy task (s) to make it 
square matrix. Assignment of these tasks for each sub problem, we apply the Kumar 
et al algorithm [KUMA 1995c]. For each sub problem we calculate the task exaction 
time and store the result in a linear array TET (j) where j= 1, 2,...,n. Finally, obtain 
the Etime, which is the product of TET (j) and WT (j), (j=1,...,n). It is the total 
optimal time for the complete assignments.

\[
TET(j) = \sum_{i=1}^{n} \sum_{j=1}^{n} TET_{i,j} x_{i,j} ;
\]

\[
Etime = \left[ \sum_{i=1}^{n} \left( \sum_{j=1}^{n} WT_{i,j} \cdot TET_{i,j} x_{i,j} \right) \right]
\]

where, 
\[
x_{i,j} = \begin{cases} 
1, & \text{if } i^{th} \text{ task is assigned to } j^{th} \text{ processor} \\
0, & \text{otherwise}
\end{cases}
\]

76
3.3.3 Algorithm

To given an algorithmic representation to the proposed method as discussed above of this chapter, we consider a system which consist a set \( P = \{p_1, p_2, \ldots, p_n\} \) of \( n \) processor and a set \( T = \{t_1, t_2, \ldots, t_m\} \) of \( m \) executable tasks which are to be processed by any one of the processor of the system.

Step-1:
\[ \text{Input: \( m, n \) TETM(\( . \)), TWM(\( . \))} \]

Step-2:
Arrange the weight of the task in descending order and store the result in \( \text{wt_seq}() \) array.

Step-3:
Partitioned the execution time matrix TETM(\( . \)) of order \( m \times n \) to sub matrices such that the order of these matrices become square i.e. number of row should be equal to number of column. Partitioning to be made as mentioned in the following steps.

Step-4.1:
Select the \( n \) task on basis of \( \text{wt_seq}(\) ) array i.e. select the ‘n’ task corresponding to most minimum weight to next minimum, if there is a tie select arbitrarily.

Step-4.2:
Store the result in the two dimensional array ETM(\( ., . \)) to form the matrices for the sub problems.

Step-4.3:
If all the tasks are selected then go to step 4.7 else steps 4.4

Step-4.4:
Repeat the step 4.1 to 4.3 until the number of task become less than \( n \).

Step-4.5:
Add dummy task in order the make last sub problem as balanced problem.

Step-4.6:
Store the result in the two dimensional array TETM(\( ., . \)), which is a last sub problem.

Step-4.7:
List of all the sub problems formed through Step 4.1 to 4.6 and repeat step 5 to step 11 to solve each of these sub problems.

Step-5:
Find the minimum of each row of TETM and replace it by 0.

Step-6:
Find the minimum of each column of TETM and replace it by 0.

Step-7:
Search for a row in TETM, which has only one zero and assign the task(s) corresponding to this position. Add one to the counter that is \( \text{nar} = \text{nar} + 1 \) and also store this position.
Step-8:
Search for a column in TETM, which has only one zero and assign the
task(s) corresponding to this position. Add one to the counter that is
nar = nar + 1 and also store this position.

Step-9:
Check whether nar = n if not than pickup an arbitrary 0 and assigned
task(s) corresponding to this position. Add one to the counter that is
nar = nar + 1 and also store this position, else, Check column (s)
position of 0’s in unassigned row(s). Check the row(s) for any previous
assignment in the corresponding column(s). Find the minimum of the
entire elements for the remaining rows and replace it zero, go to Step-
7; else Step-10.

Step-10:
Evaluate Task Execution Time [TET ( )].

Step-11:
Execution Total Time [Etme] is thus calculated as:
Etme = ET * WT

Step-12:
Stop.

3.3.4 Implementation Of The Algorithm

Consider a communication system which is consisting a set P = {p1, p2, p3} of
"n = 3" processors connected by an arbitrary network. A set T = {t1, t2, t3, t4, t5, t6,
t7, t8} of “m = 8" executable tasks which may be portion of an executable code or a
data file. The tasks are different in nature and size so as the weights of tasks are
defined. The weights of the tasks are given in Task Weight Matrix TWM ( ) of order 1 x 8 The execution time per unit weight of the tasks on various processor is mentioned
in Task Execution Time Matrix, namely, TETM ( ) of order 8 x 3.

\[
\begin{array}{c|ccc}
  & p_1 & p_2 & p_3 \\
  t_1 & 6 & 3 & 5 \\
  t_2 & 4 & 2 & 3 \\
  t_3 & 3 & 1 & 2 \\
  t_4 & 5 & 2 & \infty \\
  t_5 & 3 & 4 & 2 \\
  t_6 & 6 & \infty & 6 \\
  t_7 & 5 & 6 & 7 \\
  t_8 & \infty & 2 & 5
\end{array}
\]
Step-2

The weights of the tasks are arranged in descending order and store the result in wt_seq() linear array.

\[ wt\_seq = t_4 \ t_6 \ t_5 \ t_2 \ t_1 \ t_1 \ t_4 \ t_2 \]

\[ wt\_seq = 20 \ 20 \ 30 \ 40 \ 50 \ 60 \ 70 \ 90 \]

Step-3

We partitioned the matrix TETM(,) to define the first sub problem TETM (1,) by selecting rows corresponding t_4, t_6, t_5 and second sub problem TETM (2,) by selecting rows corresponding t_5, t_6, t_7 and by the selecting the remaining two tasks t_1, t_2, t_6, to form the last sub problem TETM (3,), Where as t_4 represent the dummy task that is required to be adding to the last sub problem TETM (3,), to make it balance problem. So that the modified matrices are as;

Sub Problem-I:

\[
\begin{array}{c|ccc}
  & p_1 & p_2 & p_3 \\
  t_4 & 5 & 2 & \infty \\
  t_6 & \infty & 2 & 5 \\
  t_5 & 3 & 1 & 2 \\
\end{array}
\]

TETM (1, ) =

Sub Problem-II:

\[
\begin{array}{c|ccc}
  & p_1 & p_2 & p_3 \\
  t_5 & 3 & 4 & 2 \\
  t_6 & 6 & \infty & 6 \\
  t_7 & 5 & 6 & 7 \\
\end{array}
\]

TETM (2, ) =
and, Sub Problem-III:

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>6</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>$t_2$</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$t_4$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

TETM(3, ) =

Step-4 & 5:
On applying modified Hungarian method devised by Kumar et al. [KUMA 1995c] to assign the tasks, on the basis of min $\{r_i\}$ and min $\{c_j\}$ from task execution time matrices for every $i$ and $j$. We put $r_{ij} = 0$ and $c_{ij} = 0$, for every $i$ and $j$. So that, the modified matrices for each sub problem are mentioned below:

Sub Problem-I:

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_4$</td>
<td>0</td>
<td>0</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$t_5$</td>
<td>$\infty$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

TETM(1, ) =

Sub Problem-II:

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_5$</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_6$</td>
<td>0</td>
<td>$\infty$</td>
<td>0</td>
</tr>
<tr>
<td>$t_7$</td>
<td>0</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

and,

Sub Problem-III:

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$t_2$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_4$</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

TETM(3, ) =

80
After implementing the assignment process, the solution set of the each of sub problem, the allocation is thus obtained.

<table>
<thead>
<tr>
<th>Tasks → Processors</th>
<th>ET</th>
<th>WT</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution for the Sub Problem-I:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t4 → p1</td>
<td>5</td>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>t5 → p2</td>
<td>2</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>t3 → p3</td>
<td>2</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>Average Time</td>
<td></td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>Solution for the Sub Problem-II:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t5 → p2</td>
<td>4</td>
<td>40</td>
<td>160</td>
</tr>
<tr>
<td>t6 → p3</td>
<td>6</td>
<td>50</td>
<td>300</td>
</tr>
<tr>
<td>t7 → p1</td>
<td>5</td>
<td>60</td>
<td>300</td>
</tr>
<tr>
<td>Average Time</td>
<td></td>
<td></td>
<td>300</td>
</tr>
<tr>
<td>Solution for the Sub Problem-III:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t1 → p2</td>
<td>3</td>
<td>70</td>
<td>210</td>
</tr>
<tr>
<td>t2 → p3</td>
<td>3</td>
<td>90</td>
<td>270</td>
</tr>
<tr>
<td>Average Time</td>
<td></td>
<td></td>
<td>270</td>
</tr>
</tbody>
</table>

Step-9:
After implementing the process, we obtain the following set of complete assignments along with execution and communication times of each processor.

<table>
<thead>
<tr>
<th>Processor→ Tasks</th>
<th>PET()</th>
<th>Optimal Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1 → t1, t7</td>
<td>400</td>
<td></td>
</tr>
<tr>
<td>p2 → t1, t5, t6</td>
<td>410</td>
<td>630</td>
</tr>
<tr>
<td>p3 → t2, t3, t6</td>
<td>630</td>
<td></td>
</tr>
</tbody>
</table>

Step-10 & 11:
Etime: = 630

Step-12:
Stop.

3.3.5 Conclusion

The model discussed here provide an optimal solution for assigning a set of “m” tasks of a program to a set of “n” processors where m > n in a communication system that to maximize the overall time of the system and the load of all allocated tasks on all the processors evenly balanced. The model addressed here is based on the
consideration of task execution time to the processors. The task weights are also defined that represent their nature, size, etc. Keeping in view we suggested modified method to improve the performance of communication system by assigning all the tasks as per the required availability of processors so that none of the tasks get remains unexecuted and the present approach does not require to adding dummy processors, however dummy task may be added if the last sub problem contains less task as compare to processor. The final results of the problem considered in the implementation of the algorithm are mentioned as,

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Processors</th>
<th>Time</th>
<th>Etine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_4; t_6$</td>
<td>$p_1$</td>
<td>400</td>
<td></td>
</tr>
<tr>
<td>$t_7; t_5; t_8$</td>
<td>$p_2$</td>
<td>410</td>
<td>630</td>
</tr>
<tr>
<td>$t_2; t_3; t_6$</td>
<td>$p_3$</td>
<td>630</td>
<td></td>
</tr>
</tbody>
</table>

The model discussed here, would be useful to the network system designer working in the field of communication systems and other related systems such as computing system, computer communication networks, distributed systems etc. The developed model is programmed in C++ and implemented the several sets of input data to test the effectiveness and efficiency of the algorithm. It is recorded that the model is suitable for arbitrary number of processors with the random program structure.

**PROBLEM-IV**

**RELIABILITY IMPROVEMENT OF CS: MATRIX PARTITIONING APPROACH**

**3.4.1 Objective**

The objective of this problem is to maximize the total execution reliability by allocating all the tasks optimally to the communication system. For that purpose we
considered execution unreliability and uses the method of matrix partitioning. The Execution Unreliability (EUR) is presented by an array in the form of Execution Unreliability Matrix EURM( , ) of order m x n. A procedure is then formulated to assign the tasks to the processors of the communication systems based on execution unreliability and is to be designed so that the overall reliability becomes optimized. This problem presented a modified method to assign all the tasks as per the required availability of processors so that none of the tasks get remains unexecuted as number of tasks is more than number of processors.

3.4.2 Technique

For developing the technique we have form the sub problems using matrix partitioning i.e. each of the sub problem is of the type of balanced assignment problem. Devising a method to assign all tasks to the processors and formulating the unreliability function to measure EUR. Since the number of task is more than the number of processors, so that we divide the problem of unbalanced tasks assignment in to sub problems using matrix partitioning, this gives us balanced tasks assignment problems. First of all obtain the product of each row and each column except, the position where, the unreliability is zero (zero unreliability should be kept aside with the product of row or column) from the EURM( , ) and store the results into Product_Row() and Prouct_Column(), each of them are one dimensional arrays. Select the first set of tasks, (this set of tasks shall contain only as many tasks as the number of processor in the communication system) on the basis of minimum unreliability against the tasks in the Product_Row() array. Store the result in to EURM ( , , ) a two dimensional array. Repeat the process until remaining tasks are either less than or equal to the number of processors. If the tasks are equal to the
processors of the systems then it will becomes the last sub problem, else to form the last problem we have to delete the column (processor) from EURM(, ) on the basis of Product_Column( ) array i.e. this set shall contain only as many processors as number of tasks left, so that we delete the processors that have maximum unreliability in the Product_Column() array. For allocation purpose a modified version of row and column assignment method devised by Kumar et al [KUMA 1995c] is employed which allocates a task to a processor where it has minimum execution unreliability.

For each sub problem calculate the execution unreliability of each processor and store the result in a linear array PEUR(j) where \( j = 1,2,\ldots, n \).

\[
PEUR(j) = \prod_{i=1}^{n} \left\{ \sum_{j=1}^{n} e_{r_{ij}} x_{ij} \right\};
\]

where, \( x_{ij} = \begin{cases} 1, & \text{if } i^{th} \text{ task is assigned to } j^{th} \text{ processor} \\ 0, & \text{otherwise} \end{cases} \), and

3.4.3 Algorithm

Consider a communication system, which consist a set \( P = \{p_1, p_2, \ldots, p_n\} \) of \( n \) processor and a set \( T = \{t_1, t_2, \ldots, t_m\} \) of \( m \) executable tasks which are to be processed by any one processor of the system.

**Step-1:**
Input: \( m, n \), EURM (,).

**Step-2:**
Obtain the product of each row of the EURM(, ) in such a way that, if any unreliability(ies) is (are) zero then keeping it aside along with product amount of that row (just to avoid the condition EUR * 0 = 0.)
Store the results in one-dimensional array Product_ROW (, ) of order \( m \).

**Step-3:**
Obtain the product of each column of the EURM (, ), in such a way that if any unreliability (ies) is (are) zero then keeping it aside along with
product of that column (just to avoid the condition EUR * 0 = 0). Store
the results in one-dimensional array Product_Column (, ) of order n.

Step-4:
Partitioned the execution unreliability matrix EURM (, ) of order m x n
to sub matrices such that the order of these matrices become square i.e.
number of row should be equal to number of column. Partitioning to be
made as mentioned in the following steps.

Step-4.1:
Select the n task on basis of Product_Row (, ) array i.e. select
the 'n' task corresponding to most minimum product to next
minimum product, if there is a tie select arbitrarily. (For the
cases in which product is EUR * 0, minimum value depends
only EUR and the impact of zero is to be neglected).

Step-4.2:
Store the result in the two dimensional array EURM (, , ) to
form the sub matrices of the sub problems.

Step-4.3:
If all the tasks are selected then go to step 4.7 else steps 4.4

Step-4.4:
Repeat the step 4.1 to 4.3 until the number of task become less
than n.

Step-4.5:
Select the remaining task say r, r < n, select the r processors on
the basis of Product_Column (, ) array i.e. the processors
corresponding to the most maximum product to next maximum,
if there is a tie select arbitrarily (for the cases in which product
is EUR * 0, maximum value depend only EUR and the impact
of zero is to be neglected).

Step-4.6:
Store the result in the two dimensional array EURM (, , ), which
is a last sub problems.

Step-4.7:
List of all the sub problems formed through Step 4.1 to 4.6 and
repeat step 5 to step 11 to solve each of these sub problems.

Step-5:
Find the minimum of each row of EURM (, , ) and replace it by 0.

Step-6:
Find the minimum of each column of EURM(, , ) and replace it by 0.

Step-7:
Search for a row in EURM(, , ), which has only one zero and assign the
task(s) corresponding to this position. Add one to the counter that is
nar = nar + 1 and also store this position.

Step-8:
Search for a column in EURM(, , ), which has only one zero and assign
the task(s) corresponding to this position. Add one to the counter that is
nar = nar + 1 and also store this position.

Step-9:
Check whether nar = n if not than pickup an arbitrary 0 and assigned
task(s) corresponding to this position. Add one to the counter that is
nar = nar + 1 and also store this position, else, Check column (s)
position of 0's in unassigned row(s). Check the row(s) for any previous assignment in the corresponding column(s). Find the minimum of the entire elements for the remaining rows and replace it zero, go to Step-7; Else Step-10.

Step-10:
Evaluate Execution Unreliability [EUR]

Step-11:
Execution Reliability (Ereliability) is thus calculated as:
Ereliability = 1 - EUR

Step-12:
Stop.

3.4.4 Implementation

Example

Consider a system which consists a set \( T = \{t_1, t_2, t_3, t_4, t_5\} \) of 5 tasks and a set \( P = \{p_1, p_2, p_3\} \) of 3 processors, where,

Step-1: Input: 5, 3

\[
\begin{array}{ccc}
     & P_1 & P_2 & P_3 \\
\hline
  t_1 & 0.03 & 0.04 & 0.06 \\
\hline
  t_2 & 0.07 & 0.02 & 0.08 \\
  t_3 & 0.02 & 0.09 & 0.06 \\
  t_4 & 0.03 & 0.07 & 0.02 \\
  t_5 & 0.08 & 0.05 & 0.04 \\
\end{array}
\]

\( EURM() = \)

Step-2:
Obtain the product of each row and column of EURM (), i.e. the products of each row and each column are as:

\[
\text{Product }_{\text{Row}} = \begin{pmatrix}
  t_1 & t_2 & t_3 & t_4 & t_5 \\
  0.000072 & 0.000112 & 0.000108 & 0.000042 & 0.000160
\end{pmatrix}
\]

\[
\text{Product }_{\text{Column}} = \begin{pmatrix}
  p_1 & p_2 & p_3 \\
  0.0000001008 & 0.0000002520 & 0.0000002304
\end{pmatrix}
\]
Step-3:

We partitioned the matrix EURM (, ) to define the first sub problem EURM (, ) by selecting rows corresponding to t₂, t₃, t₅ and second sub problem EURM (, ) by selecting rows corresponding to the tasks t₁, t₄ and by deleting columns corresponding to p₁. So that the modified matrices for each sub problems are as:

Sub Problem-I:

\[
\begin{array}{ccc}
  & p_1 & p_2 & p_3 \\
 t_2 & 0.07 & 0.02 & 0.08 \\
t_3 & 0.02 & 0.09 & 0.06 \\
t_5 & 0.08 & 0.05 & 0.04
\end{array}
\]

and, Sub Problem-II:

\[
\begin{array}{cc}
  & p_2 & p_3 \\
 t_1 & 0.04 & 0.06 \\
t_4 & 0.07 & 0.02
\end{array}
\]

Step-4 & 5:

On applying modified Hungarian method devised by Kumar et al [KUMA 1995c] to assign the tasks, on the basis of min \{r_i\} and min \{c_j\} from unreliability matrices for every i and j. We put r_ij = 0 and c_ij = 0, for every i and j. On applying this to all sub problems, the modified matrices for each sub problem are mentioned below:

\[
\begin{array}{ccc}
  & p_1 & p_2 & p_3 \\
 t_2 & 0.07 & 0.00 & 0.08 \\
t_3 & 0.00 & 0.09 & 0.06 \\
t_5 & 0.08 & 0.05 & 0.00
\end{array}
\]

\[
\begin{array}{cc}
  & p_2 & p_3 \\
 t_1 & 0.00 & 0.06 \\
t_4 & 0.07 & 0.00
\end{array}
\]

87
After implementing assignment process, the allocation is thus obtained.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Processors</th>
<th>EUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>p1</td>
<td>0.04</td>
</tr>
<tr>
<td>t2</td>
<td>p2</td>
<td>0.02</td>
</tr>
<tr>
<td>t3</td>
<td>p1</td>
<td>0.02</td>
</tr>
<tr>
<td>t4</td>
<td>p3</td>
<td>0.02</td>
</tr>
<tr>
<td>t5</td>
<td>p3</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Step-9:

EUR : = 0.0000000128

Ereliability : = 0.9999999872

Step-10:

Stop.

3.4.5 Conclusion

Maximizing the reliability of any system is one of the major parameter to enhance its performance. So that the present algorithm suggested here is capable for maximizing the overall reliability of communication system through task allocation. In these system tasks are allocated in such a way that their individual reliability of processing is optimized as well as it improve the overall reliability. In this approach not only that the loads of each processor get evenly balanced and none of the task gets unprocessed. This problem provide an optimal solution for assigning a set of “m” tasks of a program to a set of “n” processors where m > n for a communication system, that is to maximize the overall reliability of the system; the load of all allocated tasks on all the processors is evenly balanced. The execution unreliability and reliability of different processors has been obtained.
<table>
<thead>
<tr>
<th>Processors</th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution unreliability of each processor PEUR (,)</td>
<td>0.02</td>
<td>0.0008</td>
<td>0.0008</td>
</tr>
<tr>
<td>Execution Unreliability [EUR]</td>
<td></td>
<td></td>
<td>0.0000000128</td>
</tr>
<tr>
<td>Total Reliability [Ereliability()] = (1-EUR)</td>
<td></td>
<td></td>
<td>0.9999999872</td>
</tr>
</tbody>
</table>

The final result of Example is as:

\[
\begin{align*}
\text{Tasks} & \rightarrow \text{Processors} & \text{EUR} & \text{ER} \\
t_3 & \rightarrow p_1 & & \\
t_4 * t_2 & \rightarrow p_2 & 0.0000000128 & 0.9999999872 \\
t_4 * t_3 & \rightarrow p_3 & & 
\end{align*}
\]

This problem provides the optimal solution for improving the reliability and also deals the performance on the bases of maximizing system reliability.

**PROBLEM-V**

**OPTIMIZING THE EXECUTION TIME OF CS: THROUGH CLUSTERING**

**3.5.1 Objective**

The main objective of this problem is to minimize the total program execution period by allocating the tasks in such a way that the allocated load on each processor should be evenly balanced. The model utilized the mathematical programming technique for execution of the module considering that each module to be executed through all the processors. The Execution Time and Inter Tasks Communication Time are considered for developing the algorithm. The impact of Inter Processor Distance also considered and it is mentioned in the Inter Processor Distance Matrix [IPDM (,)] of the order n. The ET and ITCT are presented by arrays in the form Execution Time.
Matrix [ETM (i)] of order m x n and Inter Tasks Communication Time Matrix [ITCTM (i)] of order m.

3.5.2 Technique

Since the number of task are more than the number of processors, so that it is required to form the clusters of tasks. To forming the cluster of tasks arrange the upper diagonal value of ITCTM (i) in descending order and store the result in a linear array maxct() also store their respective positions in two dimensional array pos(i). The maximum number of tasks in a cluster shell be governed by the formula \( nt = [m/n] \), where \( nt \) is the number of tasks in a cluster. Select the first value of maxct() and its corresponding positions from pos(i) say, \((t_i, t_k)\) and store the cl(i,j) where \( i = 1,2,\ldots \text{no. of cluster and } j=1,2,\ldots nt\). if \( j < nt \) then pickup the next value from maxct() and its corresponding positions from pos(i) say, \((t_{i'}, t_{k'})\) select the task form this combination which not already exist in cl(i,j) say, \( t_i \) and store the same in cl(i,j). This process continues until the entire cluster to be formed. Some of the tasks, which are not involved in any cluster, are known as isolated tasks. Assign these clusters and isolated tasks by using the KSY Algorithm [KUMA 1995c]. Calculate the exaction time and inter tasks communication time with processor distance of each processor and store the result in a linear array pet(j) and pitctpd(j) respectively where \( j = 1,2,\ldots n \).

\[
pet(i) = \sum_{j=1}^{m} c_{x(i,j)} \cdot x_{ij}, \quad i = 1,2,\ldots n
\]

\[
\text{Where } x_{ij} = 0, \text{ if } t_i \text{ and } t_j \text{ are on the same processor.}
\]

\[
\text{1, otherwise}
\]

and

\[
pitctpd(j) = \sum_{i=1}^{m} c_{y(i,j)} \cdot d_{y(i,j)} \cdot x_{ij}, \quad j = 1,2,\ldots n
\]

\[
\text{Where } x_{ij} = 1, \text{ if } t_i \text{ is on the } j^{th} \text{ processor.}
\]

\[
0, \text{ otherwise}
\]
Finally, sum up the value of pet(j) and pitctpd(j), (j=1,.....,n) and store the result the ttp(j) and pickup the maximum value of ttp(j) i.e. total optimal system time for busy period.

3.5.3 Computational Algorithm

The method discussed here is to determine the tasks allocations in communication systems based on the following components.

- Determine the initial allocation
- Determine the cluster of m-n tasks
- Determine the final allocation
- Computation the total system time of busy period.

3.5.4 Algorithm

To given an algorithmic representation to the technique mentioned in the previous section, let us consider a system in which a set T= \{t_1, t_2, t_3... t_m\} of “m” tasks is to be executed on a set P= \{p_1, p_2, p_3... p_n\} of “n” available processors.

Step-1: 
Input: m, n; m is the number of modules of a task, n is the number of processors/
Input: etm();//matrix to hold the execution time of each task to each processor/
Input: itctm();//matrix to hold the Communication time amongst the tasks/

Step-2: 
maxct()=0;//linear array to store the inter task communication time/
ian()=0;//linear array to store the tasks for initial assignment/
ian()=0;//linear array to store the remaining tasks for initial assignment/
bmin=0;//Variable for selecting the best minimum/
Tasc()=0;//linear array to hold the tasks in order to assignment made/
Tnon-as()=0;//linear array to store the non assigned tasks/
numade = 0;//variable for counting the number of the assignment made/
alloc()←0; //linear array to hold processor’s position in order of assignment/
msr()←0; //linear array to hold processor mean service rate/
trp()←0; //linear array to hold the throughput of the processors/
mst()←0; //linear array to hold processor mean service time /
pos()←0; //to dimensional array to hold to the corresponding position of maxct()/
cpos()←0; // to dimensional array to hold to the position of common element //
netm()←0; //operational matrix for execution time/
nitctm()←0; // operational matrix for inter task communication time //
ttask()←0; //linear array to store the total number of task to the processors/

Step-3: for i ← 1 to n do
    for j ← 1 to n do
        store the etm(i) in netm(i) as
        netm(i)← etm(i)
    repeat
    repeat
    for i ← 1 to n do
        for j ← 1 to n do
            store the itctm(i) in nitctm(i) as
            nitctm(i)← itctm(i)
        repeat
        repeat

Step-4: set k ← m (m-1)/2

Step-5: for i ← 1 to m do
    for j ← 1 to k do
        arrange the upper diagonal values of nitctm(i) in non-ascending order and store the result in maxct(), until j = k
    repeat
    repeat

Step-5.1: for i ← 1 to k do
    for j ← 1 to 2 do
store the combinations of tasks of maxct() in pos(.)

repeat

repeat

Step-5.1.1: \( nt \leftarrow 0 \)

set count \( \leftarrow \frac{m}{n} \)

Step-5.1.2: \( nt \leftarrow \text{count} \)

for \( i \leftarrow 1 \) to \( m \) do

for \( j \leftarrow 1 \) to \( k \) do

pick-up the maximum value of maxct() and check the corresponding combination in \( \text{pos}(.) \), say \( (t_j, t_k) \)

if \( nt = \text{count} \)

then

store the cluster in a linear array \( c_l(.) \)

else

if \( \text{pos}(i, j) < \text{pos}(i, k) \)

\( nmax \leftarrow \text{pos}(i, j) \), until \( j = k \)

repeat for \( i \)

\( nmax < \text{pos}(i, k) \), until \( j = k \)

endif

check the corresponding combination of \( nmax \) in \( \text{pos}(.) \), say \( (t_j, t_k) \), store the cluster in a linear array \( c_l(.) \)

repeat

repeat

\( nt \leftarrow nt + \text{count} \)

Step-5.1.3: \( k \leftarrow \frac{(m - nt)(m - nt)}{2} \)

Step-5.1.4: modify the \( \text{pos}(.) \) by deleting the combination and reduce maxct() by eliminating the corresponding values.

modify the \( nctm(.) \) by adding the \( i^{th} \) and \( k^{th} \) rows together, also modify the \( nctm(.) \) by adding the \( i^{th} \) and \( k^{th} \) rows and then column, remove \( k^{th} \) rows from \( nctm(.) \) and \( k^{th} \) column from \( nctm(.) \)

goto Step-5.1.2.
Step-6: 

for \( k \leftarrow 1 \) to \( n \) do

for \( j \leftarrow 1 \) to \( n \) do

find out minimum of \( k^{th} \) rows, say \( m_{r_{kj}} \), of \( netm(i,j) \) lying in \( j^{th} \) column and subtract \( m_{r_{kj}} \) from all the values of \( k^{th} \) rows

repeat

repeat

Step-6.1: 

for \( j \leftarrow 1 \) to \( n \) do

for \( k \leftarrow 1 \) to \( n \) do

find out minimum of \( j^{th} \) column, say \( m_{c_{kj}} \), of \( netm(i,j) \) lying in \( k^{th} \) row and subtract \( m_{c_{kj}} \) from all the values of \( j^{th} \) column

repeat

repeat

Step-7: 

for \( k \leftarrow 1 \) to \( n \) do

for \( j \leftarrow 1 \) to \( n \) do

row in \( netm(, \) ) has only one zero at position \((1,2)\)

assign task \( t_i \) to \( p_k \);

\( alloc(k) \leftarrow j; \ allocate(2) \)

\( nomade \leftarrow nomade+1; \ nomade=1 \)

\( t_{ass} \leftarrow t_{ass} \cup \{t_k\}; \{t_i\} \)

repeat

repeat

Step-7.1: 

for \( j \leftarrow 1 \) to \( n \) do

for \( k \leftarrow 1 \) to \( n \) do

search for a column in \( netm(, \) ) which has only one zero, say at position \((k,j)\); assign task(s) corresponding to \( k^{th} \), say, \( t_k \), row to \( p^{th} \) processor, say \( p_p \),

\( alloc(k) \leftarrow j \)

\( nomade \leftarrow nomade+1 \)

\( t_{ass} \leftarrow t_{ass} \cup \{t_k\} \)

repeat

repeat
Step-8: if nomade# n
then
  pick-up an arbitrary zero,
  go to Step-7
else
  go to Step-7.1
endif

Step-8.1: check column(s) position of zero(s) in unassigned row(s) check the row (s)
any previous assignment in the corresponding column(s) store the positions
of the common elements in cpos(), say (i,j), find the minimum element of
all the elements of the remaining rows, say minij, subtract minij, from these
elements add minij, at the common positions and then go to Step-6.

Step-9: for k ← 1 to m do
  for j ← 1 to n do
    compute the etij by summing-up th value of etij for each processors
    and store the result in a linear array pet(j)
    compute the mean service rate of the pth processor, say p, stored in
    alloc(k) for the assigned tasks corresponding to kth, say tk, in tset()
    msr(j) ← 1/pet(j)
    repeat
    repeat

Step-9.1: for j ← 1 to n do
  Compute the maen services time of the processors and store the
  result in a linear array mst(j)
  mst(j) ← 1/msr(j)
  repeat

Step-9.2: for i ← 1 to n do
  pcount ← 0
  for j ← 1 to m do
    compute the processor’s throughput and store the result in
    a linear array trp(i) as,
    if i = alloc(j)
    then
      pcount ← pcount+1
    else
    95
next j
endif

task(i) ← pcount
repeat

trp(i) ← task/mst(i)
repeat

Step-10: tcc ← 0

for i ← 1 to m do
    for j ← 1 to n do
        compute the lct for each processor as,
        pcc(i) ← tcc + lctcm(i,j)
    repeat
repeat

Step-11: for j ← 1 to n do

    compute the total busy time for each processor
    tpb(t) ← pet(t)+pct(t)
repeat

and select the maximum value from tpb(t)

tost ← max{tpb(t)}/tost is the total system time

Step-12: Stop.

3.5.5 Implementation

Consider a communication system which is consisting of a set \( P = \{p_1, p_2, p_3\} \)
of “n = 3” processors connected by an arbitrary network. The processors only have local memory and do not share any global memory. A set \( T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\} \) of “m = 8” executable tasks which may be portion of an executable code or a data file.
\[
\begin{array}{ccc}
p_1 & p_2 & p_3 \\
p_1 & 0 & 4 & 2 \\
\text{ipdm}(.) = & p_2 & 4 & 0 & 3 \\
p_3 & 2 & 3 & 0 \\
\end{array}
\]

\[
etm(.) =
\begin{array}{ccc}
t_1 & p_1 & p_2 & p_3 \\
t_1 & 6 & 3 & 5 \\
t_2 & 4 & 2 & 3 \\
t_3 & 3 & 1 & 2 \\
t_4 & 5 & 2 & \infty \\
t_5 & 3 & 4 & 2 \\
t_6 & 6 & \infty & 6 \\
t_7 & 5 & 6 & 7 \\
t_8 & \infty & 2 & 5 \\
\end{array}
\]

\[
itctm(.) =
\begin{array}{cccccccccc}
t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 \\
t_1 & 0 & 3 & 4 & 2 & 6 & 8 & 1 & 0 \\
t_2 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \\
t_3 & 4 & 0 & 0 & 4 & 3 & 2 & 0 & 0 \\
t_4 & 2 & 0 & 4 & 0 & 5 & 3 & 2 & 5 \\
t_5 & 6 & 0 & 3 & 5 & 0 & 0 & 0 & 0 \\
t_6 & 8 & 0 & 2 & 3 & 0 & 0 & 6 & 8 \\
t_7 & 1 & 0 & 0 & 2 & 0 & 6 & 0 & 5 \\
t_8 & 0 & 5 & 0 & 5 & 0 & 8 & 5 & 0 \\
\end{array}
\]
\[ \text{Cluster} = cl(i, j) = \begin{bmatrix} 1 : t_1, t_6, t_8 \\ 2 : t_3, t_4, t_5 \end{bmatrix} \]

tasks \( t_2 \) and \( t_7 \) are not involved in any cluster known as isolated tasks. After applying the KSY [KUMA 1995c] algorithm the final allocations are

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Processor</th>
<th>Time taken by the tasks for execution on particular processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>( P_3 )</td>
<td>5</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>( P_1 )</td>
<td>4</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>( P_2 )</td>
<td>1</td>
</tr>
<tr>
<td>( t_4 )</td>
<td>( P_2 )</td>
<td>2</td>
</tr>
<tr>
<td>( t_5 )</td>
<td>( P_2 )</td>
<td>4</td>
</tr>
<tr>
<td>( t_6 )</td>
<td>( P_3 )</td>
<td>6</td>
</tr>
<tr>
<td>( t_7 )</td>
<td>( P_1 )</td>
<td>5</td>
</tr>
<tr>
<td>( t_8 )</td>
<td>( P_3 )</td>
<td>5</td>
</tr>
</tbody>
</table>

Execution time of each processor \( \text{pet}(i) = (9, 7, 16) \)

Inter tasks communication time with inter-processor distance of each processor \( \text{pitctpd}(i) = (48, 74, 106) \)

Total time of busy period of each processor \( \text{ttbpt}(i) = (57, 81, 122) \)

Maximum value of \( \text{ttbpt}(i) \)
\[ = 122 \]

Tostbp (i.e. total optimal system time for busy period) \[ = 122 \]

3.5.6 Conclusion

The model discussed here provides an optimal solution in a Communication System to maximize the overall throughput and the balanced load of all allocated tasks on each processor. This approach forms the clusters before making the assignments in the example mentioned in the implementation. Two clusters have been formed of containing three tasks. There is two tasks are not involved in any of the cluster treated as isolated tasks. The Inter Task Communication Time with Inter
Processor Distance and Execution Time on different processors has been obtained.

The total optimal system cost for busy period in the example mentioned in the body of the chapter is found 122 units. The final results are shown in the following table:

<table>
<thead>
<tr>
<th>Processor → Tasks</th>
<th>Pet()</th>
<th>Pitctpd()</th>
<th>ttbpt()</th>
<th>Tostbp</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁ → t₂, t₇</td>
<td>9</td>
<td>48</td>
<td>57</td>
<td>122</td>
</tr>
<tr>
<td>P₂ → t₃, t₄, t₅</td>
<td>7</td>
<td>74</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>P₃ → t₁, t₆, t₈</td>
<td>16</td>
<td>106</td>
<td>122</td>
<td></td>
</tr>
</tbody>
</table>

The graphical representation of the optimal assignment has been shown in the figure-4. Out of two clusters one goes to processor p₂ while the other to processor p₃. The isolated tasks are executed on processor p₁. The total optimal system time for busy period is 122 units which include the impact of the processor distance and inter tasks communication time. The developed model is programmed in C++ and implemented the several sets of input data are used to test the effectiveness and efficiency of the algorithm. It is found that the model is suitable for arbitrary number of processors with the random program structure.

![Optimal Assignment Graph](image)

**Figure 4:** Optimal Assignment Graph