International Journal of Theoretical & Applied Sciences, 10(1): 263-268(2018)

ISSN No. (Print): 0975-1718
ISSN No. (Online): 2249-3247

A Heuristic Tasks Allocation Algorithm

Kamini Raikwar[1], Virendra Upadhyay[1] and Manoj Kumar Shukla[3]
[1]Research Scholar Department of Mathematics M.G.C.G.V. Chitrakoot (Madhya Pradesh), India
[2]Department of Mathematics M.G.C.G.V. Chitrakoot (Madhya Pradesh), India
[3]Department of Mathematics,
Institute for Excellence in Higher Education, Bhopal (Madhya Pradesh), India

(Corresponding author: Kamini Raikwar)
(Received 09 February 2018, Accepted 20 April, 2018)
(Published by Research Trend, Website: www.researchtrend.net)

ABSTRACT: The technique reported in this paper provides either similar or batter optimum results optimal solution for assigning a set of m tasks of a program to a set of n processors where m>n in all the cases. The model has been coded in C++ and implemented on HP-workstation dual core processor machine and found satisfactory results.

I. INTRODUCTION

Allocation of modules or tasks of an application program to processors is an important design issue in the Distributed Computing Environment (DCE). If this allocation is not carried out properly, the available computational power cannot be exploited efficiently and consequently the throughput of the system decreases. This degradation in throughput Lint [11] and Govi [8] is due to excessive ITCC that arises from the interdependent tasks comprising an application residing on different processors. Thus to make an efficient use of resources, the ITCC need to be minimized and tasks should be assigned to the processor on which they run fast. Hence task allocation is to be carried out with the goal of minimizing the total sum of EC and ITCC incurred by the assignment.

In order to minimize the Total Computation Cost (TCC) of assignment it is required to exploit the specific capabilities of the processors and avoid excessive ITCC. The collaboration of EC and ITCC of any policy has been taken in to in consideration in different ways in the literature Aror81, Bokh [5], Bokh [6], Chu[7], Hous[9], Inde [10], Pric [18], Lo[12], Lo [14], Ston [21] and [22], Tows [23] and Lusa [15]. These techniques can be broadly classified into three categories: graph theoretic approach, mathematical programming approach and heuristic approach. Among them, the graph theoretic approach, alone or in combination with heuristics, have been widely applied to the task allocation problem. Ston [21-22], using network flow graphs derived an efficient task allocation algorithm for a two-processor system to obtain the optimal assignment. Extending Stone's results Lo [12], [13], [14], Arora & Rana [2] derived a solution for multiple processors using heuristics in combination with network flow model. Bokhari [5] and Arora & Rana [1] presented optimal solutions for program graphs constrained to tree like structure. For series-parallel program graphs, Towsley [23] presented a shortest path algorithm for optimal task allocation. Price and Pooch [18] have developed an efficient heuristic policy for a general form of program graph. The problem of assigning tasks of a random program graph to any number of processors with objective of minimizing total time using either graph theoretic approach or mathematical programming approach is complex. The graph theoretical approach is intractable for random program graphs and computing system having more than two processors. Therefore simpler, but sub-optimal solutions can be obtained by employing heuristics. One such heuristic search technique is presented in this chapter. The model considered the Impact of Collaborating EC and ITCC before deciding the fusion strategy. After the initial assignment process is over the impact of the sum of EC & ITCC of a candidate task on each of the processor and finally fuses the task, where the sum of its EC and ITCC is minimum.

The technique reported in this paper provides either similar or batter optimum results optimal solution for assigning a set of m tasks of a program to a set of n processors where $m > n$ in all the cases. The model has been coded in C++ and implemented on HP-workstation dual core processor machine and found satisfactory results.

II. PROBLEM STATEMENT

Consider the problem of finding optimal assignment of set of m tasks $T = \{t_1, t_2, t_3 \ldots t_m\}$ of m tasks to be allocated set of n processors $P = \{p_1, p_2, p_3, . p_n\}$. These processors are interconnected by communication link. These links provide means for transferring massages among the processors. Problem involves developing a generalized theory by using liner programming method. This technique can generate an optimal solution which (i) large number of computation task to be allocated (ii) balance utilization of processors in the DCS, and (iii) Minimize the overall computation cost.

III. THE PROPOSED METHOD

Initially the average load on the processors P_j is obtained by using equation (3.1) and (3.2) respectively and the total load may be calculated by using the equation (3.3)

(3.1) $L_{avg}(P_j) = \frac{L_j}{n}$

(3.2) $L_j = \sum_1^n ec_{ij}$ where $j = 1,2, \ldots \ldots n,$

The average load on a processor $L_{avg}(P_j)$ depends upon the different tasks on each processor in $ECM\ (,)$. The total processor load is given by

(3.3) $T_{lod} = \sum_1^n L_{avg}(P_j)$

The load on each processor is equal to the average load within a reasonable tolerance. In the present study a tolerance factor of 20% of average load has been considered.

To determine the allocation, initially determine "n" Minimally Linked Task (MLT) by using equation (3.4) and store the result in two dimensional array $MLT(,)$ the first column represents the task number and second column represents the sum of ITCC of task ti with all task tk-i. Rearrange the $MLT\ (,)$ in ascending order assuming the second column as sorted key.

(3.4) $MLT(i,k) = \sum_{i,k=1}^m cc_{ik}$

Also, arrange the $ECM\ (,)$ accordingly. To determine initial allocation apply the Yadav et al [24] algorithm] and store in an array $T_{ass}(j)$ (where $j = 1,2, \ldots, n$). The processor position are also store in a another linear array Aalloc (j). The value of $T_{TASK}(j)$ is also computed by adding the values of Aalloc (j) if a task t_i is assigned to processor p_j otherwise continue. The remaining $(m-n)$ task are then store in a liner array $T_{non\_ass}()$. Tasks assigned to processors p_j and stored in $T_{non\_ass}()$. which are obviously

$T = T_{ass}() \cup T_{non\_ass}()$

All the tasks stored in $T_{non\_ass}()$ fused with those assigned tasks stored in $T_{ass}()$ on the bases of minimum average of EC and ITCC.

The Fused Execution Cost (FEC) of a task $t_a \in T_{non\_ass}()$ with some other task $t_i \in T_{ass}()$ on processor p_j is obtained as:

(3.5) $FEC(j)_{ai} = [ec_{aj} + ec_{ij}]\ \ 1 \le a \le m, 1 \le i \le m, 1 \le j \le n$ and $a \ne i$

Let cc_{ai} be the ITCC between $t_a \in T_{non\_ass}()$ and $t_i \in T_{ass}()$. Fused Inter Task Communication Cost $(FITCC)$ for t_a with t_i is computed as:

(3.6) $FITCC(j)_{ai} = \sum_{i/_{t_i \in T_{ass}()}} [cc_{ai}]$

Here, $c_{ai} = 0$ if fused with t_i or $a = i$ and remaining c_{ai} value are added Minimum Average Fused Cost $(MAFC)$ is calculated as follows:

(3.7) $MAFC(j)_{ai} = min\left\{\begin{matrix}(FEC_{a1} + FITCC_{a1}) + (FEC_{a2} + FITCC_{a2}) + (FEC_{a3} + FITCC_{a3}), \\ \ldots \ldots \ldots, (FEC_{am} + FITCC_{am})\end{matrix}\right\}$

This process will be continued until all the tasks stored in $T_{non\_ass}()$ are fused. After complete allocation is achieved $PEC\ (j)\ and\ ITCC\ (j)\ as$:

(3.8) $PEC(j) = \sum_{j=1}^n e_{ij} x_{ij} = 1,2, \ldots \ldots m$

Where $x_{ij} = \left\{\begin{matrix}1, if\ task\ t\ is\ assign\ to\ processor\ p \\ 0, otherwise\end{matrix}\right\}$

(3.9) $ITCC(j) = \sum_{j=1}^n e_{ij} x_{ij}\ i = 1,2, \ldots \ldots m$

Where $x_{ij} = \left\{\begin{matrix}1, if\ task\ t\ is\ assign\ to\ processor\ p \\ 0, otherwise\end{matrix}\right\}$

Finally, Calculate the Total Optimal Cost (TOC) by summing up the values of $PEC\ (j)$ and $ITCC\ (j)$, and store the result in a linear array Over $TOC\ (j)$ where $j = 1,2, \dots n$. The maximum value of $TOC\ (j)$ will be the optimal cost of the system:

(3.10) $TOC\ (j) = Max\{PEC\ (j) + ITCC\ (j)\}$

The Mean Service Rate [MSR] of the processors in terms of T_{ass}(j). to be computed as and store the results in $MSR(j)$ (where $j = 1,2, \dots, n$).

(3.11) $MSR(j) = \frac{1}{TOC(j)}\ j = 1,2, \dots \dots n$

The overall throughout of the processors are calculated as and store the results of throughout in the linear arrays $TRP\ (j)$, where $j = 1, 2 \dots \dots, n$

(3.12) $T_{RP}(j) = \frac{T_{TASK}(j)}{TOC(j)}$

Algorithm

To give an algorithmic representation to the technique described in the section 3. Let us

Consider the DCS in which a set of m tasks $T = \{t_1. t_2, t_3 \dots t_m\}$ of m tasks to be allocated set of n processors $P = \{p_1, p_2, p_3, \dots p_n\}$.

Step-1. Input $m, n, ECM\ (,)$ and $ITCCM\ (,)$

Step-2. Initially the average load on the processors p_j is obtained

Step-3. Determine the "n" Minimally Linked Task

Step-4. Augmented $ECM\ (,)$ by introducing the $MLM\ ()$ and sort $ECM\ (,)$ in increasing order considering MLT as sorting key.

Step-5. Determine the initial allocations, on applying Yadav *et al* [24] algorithm. The initial allocation than store in an linear array Ta s s () and the processor position are store in $A_{alloc}(j)$ the valve of $T_{TASK}(j)$ is also computed by adding the value of $A_{alloc}(j)$. The remaining $m - n$ task are stored in $T_{non\_ass}()$.

Step-6. Select a task $t_a \in T_{non\_ass}()$ for fusion with some other task $t_l \in T_{ass}()$ on processor p_j and determine the FEC by using the equation (3.5), (3.6) and (3.7). This step will be continued until all the tasks stored in $T_{non\_ass}()$ are fused and apply Yadav *et al* [24] algorithm for assignment.

Step-7. After complete allocation compute $PEC\ (j), ITCC\ (j)$ and $TOC\ (j)$ by using equation (3.8), (3.9) and (3.10) respectively. Obtained the maximum value of $TOC\ (j)$ that will be the optimal cost of the system.

Step-8. Evaluate the $MSR\ (j)$ and $TRP\ (j)$ using equation (3.11) & (3.12)

Step-9. Stop

IV. IMPLEMENTATION OF THE ALGORITHM

The application of the above method is illustrated here using an example.

Example 4.1: To justify the application and usefulness of the present algorithm an example of a DCS is considered which is consisting of a set of "$n = 4$" processors $P = \{p_1, p_2, p_3, p_4\}$ connected by an arbitrary network and a set of "$m = 5$" executable tasks $T = \{t_1\ t_2, t_3, t_4, t_5\}$ which may be portion of an executable code or a data file.

Input of the Algorithm: Data required by the Algorithm is given below:

Step-1. Number of processors available in the system $(n) = 4$

Number of tasks to be executed $(m) = 5$

$$EMC(,) = \begin{bmatrix} & p1 & p2 & p3 & p4 \\ t1 & 8 & 4 & 6 & 5 \\ t2 & 6 & 5 & 4 & 2 \\ t3 & 8 & 4 & 5 & 7 \\ t4 & 4 & 7 & 6 & 5 \\ t5 & 8 & 8 & 2 & 6 \end{bmatrix}, \qquad ITCCM = \begin{bmatrix} & t1 & t2 & t3 & t4 & t5 \\ t1 & 0 & 100 & 3 & 5 & 6 \\ t2 & 100 & 0 & 6 & 4 & 3 \\ t3 & 3 & 6 & 0 & 5 & 2 \\ t4 & 5 & 4 & 5 & 0 & 6 \\ t5 & 6 & 3 & 2 & 6 & 0 \end{bmatrix}$$

Step-2. Actual average load to be assign to the processor after introducing the 20% Tolerance Factor (TF) the average load may be assign to the processors is calculated and same may be sore in linear array $LAVG()$

$$LAVG() = \begin{matrix} p1\ 10 \\ p2\ 08 \\ p3\ 07 \\ p4\ 07 \end{matrix}$$

Step-3. Determine the Minimally Link Task (MLT) and store the result in $MLT\ (,)$ as follows:

$$MLM(,) = \begin{matrix} t1\ 114 \\ t2\ 113 \\ t3\ 016 \\ t4\ 020 \\ t5\ 017 \end{matrix}$$

Step-4. Augmented $ECM\ (,)$ by introducing the $MLM\ ()$ and sort $ECM\ (,)$ in increasing order considering MLT as sorting key.

$$ECM(,) = \begin{matrix} & p1 & p2 & p3 & p4 & MLT \\ t1 & 8 & 4 & 6 & 5 & 114 \\ t2 & 6 & 5 & 4 & 2 & 113 \\ t3 & 8 & 4 & 5 & 7 & 16 \\ t4 & 4 & 7 & 6 & 7 & 20 \\ t5 & 8 & 8 & 2 & 6 & 17 \end{matrix}$$

$$ECM(,) = \begin{matrix} & p1 & p2 & p3 & p4 & MLT \\ t3 & 8 & 4 & 5 & 7 & 16 \\ t5 & 8 & 8 & 2 & 6 & 17 \\ t4 & 4 & 7 & 6 & 7 & 20 \\ t2 & 6 & 5 & 4 & 2 & 113 \\ t1 & 8 & 4 & 6 & 5 & 114 \end{matrix}$$

Step-5. Apply Yadav *et al* [24] algorithm to determine the initial allocation. The initial allocation than store in an linear array $T_{ass}()$ and the processor position are store in $Aalloc(\ )$. The remaining m-n tasks are stored in $T_{non\_ass}()$. The results are as follows:

$T_{ass}() = \{t2\ t4, t5, t3\ \}$

$Aa\ l\ l\ o\ c(j) = \{p4, p1, p3, p2\}$

$T_{non\_ass}() = \{t_1\}$

Step-6. After getting the initial allocation a task stored in $T_{non\_ass}()$ has been selected for assignment i.e. $t_2\ FEC(j)$, for t_2 with all the stored in $T_{ass}()$ is calculated as:

$FEC(j)$ in table 1

Table 1

Task	**Processors**	***FEC(j)***
t1+t2	*P4*	*7*
t1+t3	*P1*	*16*
t1+t4	*P3*	*12*
t1+t5	*P2*	*12*

Evaluate the $FITCC\ (j)$ for t_2 with the other assigned tasks stored in $T_{ass}()$ is calculated as:

Table 2.

Task	**Processors**	***FITCC(j)***
t1+t2	P4	27
t1+t3	P1	124
t1+t4	P3	124
t1+t5	P2	119

Calculated the $MAFC\ (j)$ by summing up the value of $FEC\ (j)$ and $FITCC\ (j)$ as:

Table 3.

Task	Processors	*FEC(j)*	*FITCC(j)*	*MAFC(j)*
t1+t2	P4	7	27	34
t1+t3	P1	16	124	140
t1+t4	P3	12	124	136
t1+t5	P2	12	119	131

The $MAFC\ (1)12$ is minimum i.e. 34. Therefore, task t1 is fused with Task t_2 executing on processor $p4$.
Step-7 & 8. Compute $PEC\ (j), ITCC\ (j), TOC\ (j), MSR\ (j)\ and\ TRP\ (j)$ given in table 4

Table 4.

Processors	PEC(j)	ITCC(j)	TOC(j)	*MSR (j)*	*TRP(j)*
P1	4	20	24	0.42	0.42
P2	4	16	20	0.50	0.50
P3	2	17	19	0.053	0.53
P4	7	27	34	0.029	0.58

The maximum of $TOC\ (j)$ is 34 i.e. the total busy cost of the system is 34 which is corresponds to processor $p4$ and depicted in Fig. 1.

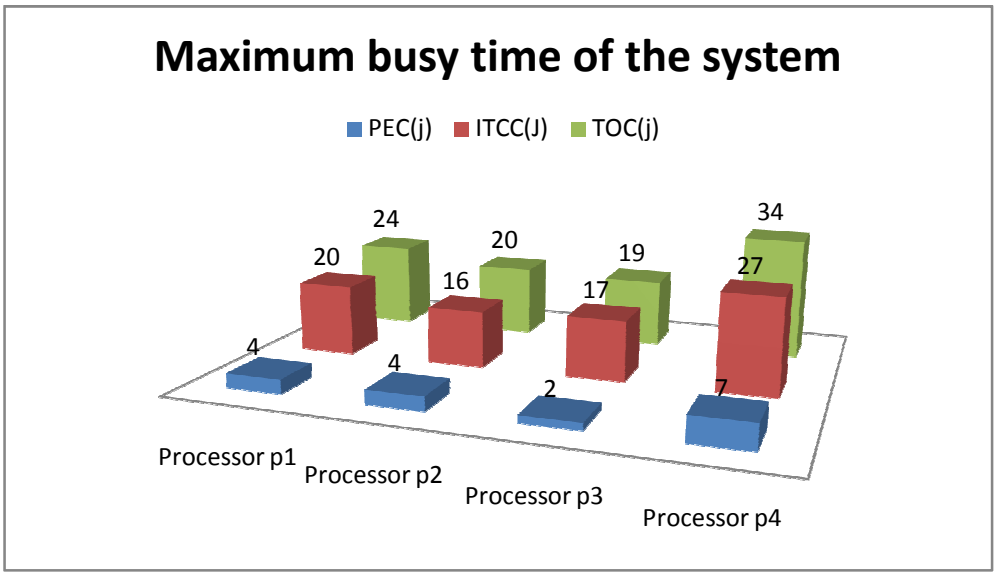

Fig. 1.

V. CONCLUSIONS

The present Chapter deals with the problem of optimal task allocation and load balancing in DCS. A heuristic tasks allocation algorithm is suggested to obtain appropriate solution of the problem. The load balancing mechanism is introduced in the algorithm by fusing the unallocated tasks of the basis of minimum of the average impact of EC and ITCC. It is observed that the time complexity of the present method is better than the other method which are based on the graph approach Ston [21], Bokh79, Bokh [4,5], Shen [20], Rao [19], Huan [9], Nico [17], Integer programming and branch & bound technique Ma [16].

REFERENCE

[1]. Arora, R.K., and Rana, S.P., (1979). "On module assignment in two processors distributed Systems", *Information Processing Letters,* Vol. **9**, No. 3, pp. 113-117.

[2]. Arora, R.K., and Rana, S.P., (1980). "Heuristic algorithms for process assignment in distributed computing systems", *Information Processing Letters*, Vol. **11**, No. 45, pp. 199-203.
[3]. Arora, R.K., and Rana, S.P., (1981). "On the design of process assigner for distributed computing systems", *The Australian Computer Journal,* Vol. **13**, No. 3, pp. 77-82.
[4]. Bokhari, S.H., (1979). "Dual processors scheduling with dynamic re-assignment", *IEEE Transactions on Software Engineering,* Vol. **SE-5**, pp. 341-349.
[5]. Bokhari, S.H., (1981). "A shortest tree algorithm for optimal assignment across space and time in distributed processor system", *IEEE Transactions on Software Engineering,* Vol. **SE-7**, No. 6, pp. 583-589.
[6]. Bokhari, S.H., (1988). "Partitioning problems in parallel, pipeline and distributed computing", *IEEE Transactions on Computers,* Vol. **C-37**, No. 1, pp. 48-57.
[7]. Chu, W.W., Holloway, L.J., M.T.L. Lan, and K. Efe, (1980). "Task allocation in distributed data processing", *Computers* **13**, No.11, pp.57-69.
[8]. Govil, Kapil, and Kumar, Avanish (2011). "A Modified and Efficient Algorithm for Static Task Assignment in Distributed Processing Environment" *International Journal of Computer Applications, 2011, "IJCA Journal*" No. 1, Art.1
[9]. Huang, X., and Cai, X.Y., (1987). "An efficient and flexible heuristic task assignment method for distributed computing systems" *2nd International Conference on Computers and Applications, China,* 682-688.
[10]. Inderkhya, B., Stone, H.S., and Cheng, L.X., (1986). "Optimal partitioning of randomly generated distributed programs", *IEEE Transactions on Computers and Software Engineering,* Vol. **SE-12**, No. 3, pp. 483-495.
[11]. Lint, B., and Agarwal, T., (1981). "Communication issues in the design and analysis of parallel algorithm", *IEEE Transactions on Software Engineering,* Vol. **SE-7**, No. 2, pp. 174-188.
[12]. Lo, V.M., (1983). "Task assignment in distributed systems", Ph.D. Thesis, University of Illinois at Urbana-Champaign.
[13]. Lo, V.M., (1984). "Heuristic algorithms for task assignment in distributed systems", *Proceedings of the 4th International Conference on Distributed Computing Systems,* 3Q-39.
[14]. Lo, V.M., (1988). "Heuristic algorithms for task assignment in distributed systems", *IEEE Transactions on Computers,* Vol. **37**, No. 11, pp. 1384-1397.
[15]. Lusa, A., and Potts., C. N., (2008). "A variable neighborhood search algorithm for the constrained task allocation problem" *Journal of the Operational Research Society.* **59**, 812-822.
[16]. Ma, P.Y.R., Lu, E.Y.S., and Tsuchiya, J., (1982). "A task allocation model for distributed computing systems", *IEEE Transactions on Computers",* Vol. **31**, No. 1, pp. 41-47.
[17]. Nicol, D.M. (1989)."Optimal Partitioning of Random Programs across two processor" *IEEE Trans. On Softwear Engineering*, Vol. **5**, No. 2.
[18]. Price C.C., and Pooch, U.W, (1982). "Search techniques for non-linear multiprocessor scheduling problem", *Naval Research Logistics Quarterly,* Vol. **29**, No. 2, pp. 13-233.
[19]. Rao, G.S., Stone, H.S, Hu, T.C., (1979). "Assignment of Task in Distributed processor System with limited Memory" *IEEE Trans. on Computer,* Vol. **C- 28**, No. 4, PP-291-299.
[20]. Shen., C.C., and Tsai, W.H., (1985). "A graph matching approach to optimal task assignment in distributing computing system using a minimax criterion", *IEEE Transactions on Computers,* Vol. **34**, No. 3, pp. 197-203.
[21]. Stone, H. S., (1977). "Multiprocessor scheduling with the aid of network flow algorithms", *IEEE Transactions on Software Engineering.* **SE-3**, l, pp. 85-93.
[22]. Stone, H.S., and Bokhari, S.H., (1978). "Control of Distributed Processor", computers, Vol. **11**, pp. 97-106.
[23]. Towsley, D.F., (1986). "Allocating Programs Containing Branches and Loops within a Multiple Processor System", *IEEE Transactions on Software Engineering*, **SE-12** (10), pp.1018-1024.
[24]. Yadav, P. K., Kumar, Avanish and Singh, M.P, (2004). "An Algorithm for Solving the Unbalanced Assignment Problems, *International Journal of Mathematica Sciences*, Vol. **12**(2), pp. 447-461.