MINIMIZATION OF TASK TURNAROUND TIME FOR DISTRIBUTED SYSTEMS

Chiun-Chieh Hsu, Sheng-De Wang and Te-Son Kuo

Department of Electrical Engineering

National Taiwan University, Taipei, Taiwan

ABSTRACT

This paper deals with the problem of assigning a partitioned task to a distributed computing system. Considering communication overhead and idle time, we successfully develop a mathematical model to describe the cost function, which is defined to evaluate the task turnaround time, under a genaral model of distributed computing systems. Task assignment is formulated as a DU-mapping, which maps a directed acyclic task graph onto an undirected system graph. The search of optimal DU-mapping is NP-complete and is transformed into a state space search problem. An approach called critical sink underestimate is developed to attain an optimal DU-mapping and the most nodes in the state space tree are pruned. Experimental results reveal that this method performs very well due to its close evaluation to the real cost.

I. INTRODUCTION

The rapid progress of VLSI and computer networking technologies has made distributed computing systems economically attractive for many computer applications. Although distributed computing systems are capable of enhancing system throughput and resource utilization, they raise some problems that prevent the widespread use of distributed computing system. One of the major problems is the throughput degradation casued by imbalance of processor's load and the large amount of interprocessor communication overhead due to inadequate task assignment.

Therefore, in this paper, the aim of task assignment intends to attain the minimum response (task turnaround) time to maximize system throughput. A task is assumed to be partitioned into a set of modules with precedence relationships. There are two conflicting policies to diminish task turnaround time, namely, assigning all modules to a single processor to save communication overhead and distributing them evenly to all processors to balance load. We require to make a compromise to achieve the minimum task turnaround time.

Many approaches to the task assignment in distributed computing system have been identified. They can be roughly classified into four catagories: graph-theoretical [1-3], mathematical programming [4,5], heuristics [6-9], and simulated annealing [10]. In general, these approaches are not mutually exclusive. Many of them ignore precedence constraints, which is an important characteristic in a real programming environment. Even though they include the precedence constraints, idle time due to queuing delay and control overhead is also neglected. In this article, we present an approach which takes both of them into consideration. The problem dealt with in this paper assumes a limited number of heterogeneous processors, non-identical interprocessor links, not necessarily full-connected system structure, partitioned tasks, and precedence constraints among modules. We see that the task assignment problem is based on a general model of distributed computing systems.

This paper is organized as follows: Section II treats the issue of problem formulation. The concepts of trigger time, activation time, start time, and the computation of cost function are defined in section III. A well-informed approach, critical sink underestimate, are proposed in section IV. In section V, an illustrative example is given and experimental results are discussed. Finally, conclusions are made in section VI.

II. PROBLEM FORMULATION

A graph G is known to be composed of a vertex set V and an edge set E, where $E \subseteq V * V$. Based on the assumptions aforementioned, a distributed computing system is modeled as an undirected graph $G_P = (V_P, E_P)$ as depicted in Fig.1.1, where V_P is the set of processors and E_P is the set of interprocessor communication links. There exists self-loop $(\lambda,\lambda) \in E_P$ for each processor λ, which implies that different modules can be assigned to the same processor [7].

A task is modeled as a directed graph $G_T = (V_T, E_T)$ as depicted in Fig.1.2, where V_T is the set of modules and E_T is the set of intermodule arcs. For an arc $(\alpha,\beta) \in E_T$, we say that α is the parent of β or β is the child of α. Directed cycles need to be merged into a larger module in advance. Therefore, a task graph dealt with in this paper is always affirmed to be a directed acyclic graph (DAG). In a DAG task graph, there is at least one module without parent and at least one module without child, which are called source and sink modules respectively.

Since the task and distributed computing system are respectively modeled as a DAG graph and an undirected graph, the problem of task assignment is then formulated as a special kind of graph mapping problem. We call it Totally Directed-to-Undirected graph mapping or DU-mapping for short, which is defined as follows:

Definition 1: Let there is a DAG task graph $G_T = (V_T, E_T)$ and an undirected system graph $G_S = (V_S, E_S)$. We define DU-mapping of G_T to G_S is a graph matching which not only maps all modules and intermodule arcs of G_T onto the processors and interprocessor links of G_S respectively (possibly many to one), but also maps all parents in G_T before their children.

From the previous discussions, we see that **the problem of task assignment in distributed computing systems is turned out to be the problem of DU-mapping.** From the viewpoint of sink modules, the task turnaround time is the maximun finish-execution time among all sink modules, where finish-execution time denotes the moment when a module completes its execution. This kind of sink module is called critical sink module; and the critical sink module with the minimum finish-execution time among all possible DU-mappings is called the minimum critical sink module. Henceforth, **the search of an optimal task assignment in distributed computing systems is transformed into finding the DU-mapping with the minimum critical sink module.**

III. COMPUTATION OF COST FUNCTION

For module α and its child β, the **trigger time** of α to β is the moment when α finishes transmitting a message to β. A module α is activated if all of its parents have triggered α. Thus, the **activation time** of α is the moment when the last parent of α finishes transmitting a message to α. If α is activated and its assigned processor is released by the last module assigned to this processor, it is runnable. In other words, this processor now starts executing α. This moment is called the **start time** of α. All terms described above have the phenomenon of time accumulation as a consequence of precedence constraints among modules.

Let us introduce some terms, which will be used in the remainder of this article. Note that all these terms are associated with a DU-mapping M.

1) $M(\alpha)$ denotes the processor to which module α is assigned.
2) Release time $RL_\alpha(M)$ denotes the moment when the processor $M(\alpha)$ is released by the last module assigned to $M(\alpha)$, and is ready to execute α.
3) Finish-execution time $FE_\alpha(M)$ denotes the moment when α completes its execution at its assigned processor.
4) Finish-transmission time $FT_\alpha(M)$ denotes the moment when module α finishes transmitting messages to its children which have already been assigned.
5) $E_\alpha(\lambda)$ denotes the elapsed time required for module α to be fully executed at processor λ.
6) $C_{\alpha,\beta}[\tau,\lambda]$ denotes the amount of communication time between processors τ and λ at which modules α and β reside respectively. Communication time is assumed to be zero if τ and λ are the same processor. It is an infinite value if there is no interprocessor link between τ and λ, or no intermodule arc from α to β.
7) $TIME_\lambda(M)$ denotes the current elapsed time recorded in processor λ at any point during the computation of the cost function (task turnaround time). $TIME_\lambda(M)$ includes all execution time, communication time and idle time.

Let us discuss how to compute the turnaround time for a totally assigned task. Initially, the trigger times, activation times and start times of all source modules are set to be zero; and also the elasped times of all processors. Based on this initialization, the start time of some assigned module and then the trigger times of this module to its children are computed. This procedure is repeated from the first assigned module to the last one.

Suppose a parent module α forks to children h_i for $1 \le i \le n$, and transmits messages to children in the order of $h_1, h_2, \ldots, h_n$. The trigger time of α to the ith transmitted child h_i associated with a DU-mapping M is

$$TRG_{\alpha,hi}(M) = ST_\alpha(M) + E_\alpha(M(\alpha)) + \sum_{j=1}^{i} C_{\alpha,hj}[M(\alpha),M(h_j)] \quad (1)$$

where $ST_\alpha(M)$ denotes the start time of parent α. In the mean time, the finish-execution time $FE_\alpha(M)$ and finish-transmission time $FT_\alpha(M)$ are also updated, where $FE_\alpha(M) = ST_\alpha(M) + E_\alpha(M(\alpha))$ and $FT_\alpha(M) = FE_\alpha(M) + \sum_{j=1}^{n} C_{\alpha,hj}[M(\alpha),M(h_j)]$. Note that n denotes the number of assigned children for partial assignments. Both terms will be used in the evaluation of turnaround time for a partially assigned task in the next section.

For module β and the set of its parents F_β. The activation time and start time of β are given by

$$ACT_\beta(M) = \max_{\alpha \in F_\beta} TRG_{\alpha,\beta}(M) \quad (2)$$

$$ST_\beta(M) = \max\{ RL_\beta(M), ACT_\beta(M) \} \quad (3)$$

The processor elapsed time requires to be updated while computing trigger time, start time, and finish-execution time. Processor's delay time can be obtained by subtracting the three terms to the last elapsed time.

An activated module β can not be executed (or run) until its assigned processor $M(\beta)$ is released from previously assigned module. As a consequence, β will be idle between the activation time and start time. This period of time is called the idle time of β, which is defined as

$$IDLE_\beta(M) = ST_\beta(M) - ACT_\beta(M) \quad (4)$$

Let us use a simple example to expound how to compute these terms. Suppose there are four modules in a partitioned task, where module A forks to modules B and C, B and C join to D. Besides, there is a three-processor system where the three processors are connected to each other. A DU-mapping M = {(A,1), (B,2), (C,3), (D,3)} indicates that Module A is assigned to processor 1, B to processor 2 and others to processor 3; and A transmits messages to B first and then to C. All the execution times and communication times are 10 except that the execution time of C is 20. The steps required to compute these terms are as follows:

1. $RUN_A = ACT_A = 0$, $TIME_1 = FE_A = 10$,
   $TIME_1 = FT_A = TRG_{A,B} = 20$, $TIME_1 = FT_A = TRG_{A,C} = 30$;
2. $ACT_B = 20$,
   $TIME_2 = RUN_B = \max\{RL_B, ACT_B\} = \max\{TIME_2, ACT_B\} = 20$,
   $TIME_2 = FE_B = 30$, $TIME_2 = FT_B = TRG_{B,D} = 40$;
3. $ACT_C = 30$,
   $TIME_3 = RUN_C = \max\{RL_C, ACT_C\} = \max\{TIME_3, ACT_C\} = 30$,
   $TIME_3 = FE_C = 50$,

$TIME_3 = FT_C = TRG_{C,D} = 50;$

4. $ACT_D = \max\{TRG_{B,D}, TRG_{C,D}\} = 50,$
   $TIME_3 = RUN_D = \max\{RL_D, ACT_D\} = \max\{TIME_3, ACT_D\} = 50,$
   $TIME_3 = FE_D = 60.$

In the above steps, the term M is droped for the sake of simplicity.

After all terms of the modules for a totally assigned task have been computed, the finish-execution time of the ith sink module $FE_{si}(M)$ associated with a DU-mapping M is given by

$$FE_{si}(M) = ST_{si}(M) + E_{si}(M(si)) \tag{5}$$

Hence, the task turnaround time TA(M) is

$$TA(M) = \max_{1 \le i \le r} FE_{si}(M) \tag{6}$$

where r denotes the number of sink modules. The task turnaround time of optimal assignment M_{opt} among all possible DU-mappings M is defined as

$$TA(M_{opt}) = \min_{M \in M} TA(M) = \min_{M \in M} \max_{1 \le i \le r} FE_{si}(M) \tag{7}$$

IV. CRITICAL SINK UNDERESTIMATE

The problem of searching an optimal DU-mapping is further formulated as a state space search problem. This kind of problem finds a solution in a dynamic tree. Guided by some cost function, the dynamic tree is expanded to the optimal answer node while many nodes are pruned. The approach of how to evaluate cost to obtain an optimal solution has been developed sophisticatedly in the well-known A^* algorithm from artificial intelligence [11]. For the evaluation cost, the closer to the real cost, the more nodes A^* algorithm prunes. In A^* algorithm, f(x), the evaluation cost of node x in state space tree, is defined as the sum of the real cost g(x) from the root node to current node x and the estimated cost h(x) from current node x to goal node; i.e., f(x) = g(x) + h(x). g(x) can be computed exactly but h(x) is only a heuristic estimate. To guarantee the optimality of solution, the estimated cost h(x) must be not larger than the real cost from node x to goal node and the state space tree is expanded in a least-cost-first manner.

The computation of estimated task turnaround time, selection rules and branching rules applied to A^* algorithm will be presented in the following. This approach is called critical sink underestimate (CSU). Note that the definitions of trigger time, activation time, start time and idle time of the assigned modules for a partially assigned task are the same as in section III.

A. Computation of Estimated Task Turnaround Time

A partially assigned task can be divided into two groups: the assigned modules and the unassigned ones. For an unassigned module α, we can also separate its parents into the sets of assigned parents F_α and unassigned ones $\bar{F}_\alpha$. Since we cannot ensure that its unassigned parent will be assigned to a different or the same processor as it is, the communication time between α and its unassigned parent cannot be added to the evaluation cost to avoid overestimate. Nevertheless, the communication time between α and its assigned parent can be included in the evaluation cost.

The partial DU-mapping M_x corresponds to a path traversing from the root node to node x in a state space tree. In this path, the module with smaller depth is implied to be triggered earlier than the one with larger depth. $P\alpha$, the set of available processors to be assigned to an unassigned module α, contains the same or adjacent processors to its parents'. If $P\alpha$ is empty, then there is no feasible solution for node x. In other words, the cost of node x is infinite.

For a partially assigned task, we apply the procedure described in section III to compute the real cost for all assigned modules. Then, we compute the estimated cost for all unassigned modules in terms of estimated activation time and finish-execution time. The estimated activation time of α at processor $\lambda \in P\alpha$ associated with M_x is given by

$$EACT_{\alpha,\lambda}(M_x) = \max \left\{ \max_{\beta \in F\alpha} \{FT_\beta(M_x) + C_{\beta,\alpha}[M_x(\beta),\lambda]\}, \max_{\beta \in \bar{F}\alpha} FE_\beta(M_x) \right\} \tag{8}$$

Hence, the estimated finish-execution time of unassigned α is

$$FE_\alpha(M_x) = \min_{\lambda \in P\alpha} \left\{ \max[\, TIME_\lambda(M_x),\ EACT_{\alpha,\lambda}(M_x)] + E_\alpha(\lambda) \right\} \tag{9}$$

Note that, in the cost-estimate procedure, elapsed time $TIME_\lambda(M_x)$ will not be updated as in section III. After the estimated finish-execution times of all unassigned modules have been computed, the estimated task turnaround time for node x is

$$ETA(x) = \max_{1 \le i \le r} FE_{si}(M_x) \tag{10}$$

As mentioned before, s_i denotes the ith sink module and r the number of sink modules.

The ignorance of communication time among unassigned modules in equation (8) and the inclusion of the minimum term in equation (9) make estimated activation time and estimated finish-execution time be underestimates. Although ETA is an underestimate for the partial DU-mapping M_x, it is not enough to guarantee to achieve an optimal DU-mapping. However, if it cooperates with the following two rules, an optimal DU-mapping will be obtained. This characteristic is called admissibility in A^* algorithm.

B. LCDF Selection Rule, Greedy Branching Rules and Well-Informed Algorithm

In the A^* algorithm, we use a CLOST list to hold all of the expanded nodes and an OPEN list to store all of the generated but unexpanded nodes in a state space tree. Initially, the start node is with zero depth, zero cost and no assignment. It is the first expansion node (E-node). Besides, OPEN list consists of only the start node and CLOST list is empty.

After initialization, we select the node with the least cost and largest depth in OPEN list to be E-node. Moreover, the former has higher priority than the latter. This kind of selection is called least-cost-and-depth-first (LCDF).

After E-node has been selected, tree expansion is done

and nodes are generated by E-node according to the following greedy branching rules:

1) Each of the activated modules is to be assigned to the same or adjacent processors to its parents' according to the partial DU-mapping M_x and system topology.
2) If at least one assigned brother of the activated module has been assigned to the same processor as its parents', then the other unassigned brothers and itself are restricted to being assigned to the same processor as their parents'.

Rule 1 is very clear, so it does not require further explanation. Let us make use of a simple example to expound rule 2. Suppose there are two modules B and C which are forked from A and two interconnected processors 1 and 2. For a partial assignment {(A,1), (B,1)} corresponding to a path in a state space tree, two nodes (C,1) and (C,2) are generated in tree expansion. Both of them indicate that A transmits messages to B and then to C. For assignment {(A,1), (B,1), (C,2)}, C can be executed immediately but B cannot because B should wait for processor 1 being released by its parent A. For another partial assignment {(A,1), (C,2)}, two nodes (B,1) and (B,2) are generated in tree expansion. Both of them imply that A transmits messages to C and then to B. The two assignments {(A,1), (B,1), (C,2)} and {(A,1), (C,2), (B,1)} generate the same start times for B and C, and then the same ETA. Hence, it enables us to neglect the node (C,2) in the tree expansion for {(A,1), (B,1)} without affecting the search of an optimal assignment. So, the number of nodes in the state space tree can be diminished. Thus, for the partial assignment {(A,1), (B,1)}, C is restricted to being assigned to processor 1 since its brother B has been assigned to the same processor 1 as its parent A.

The selection and branching procedures alternate repeatedly until a goal state (optimal assignment) is reached. That is, the depth of E-node is equal to the number of modules.

By gathering together all the ideas presented in this section, the proposed A^* algorithm is as follows:

Algorithm 1:

1) E-node ⟵ start node; OPEN list ⟵ start node;
   CLOST list ⟵ null;
2) WHILE(E-node is not goal node)
3) BEGIN
4) Move the first node in OPEN list to be E-node;
5) Traverse backward from E-node to root node to find the partial assignment M_x;
6) According to greedy branching rules, select available processors for each activated module to do tree expansion;
7) Compute estimated task turnaround time ETA for each currently generated node;
8) Insert the generated nodes to OPEN list and sort them in the non-decreasing order of ETA value and the non-increasing order of depth (LCDF);
9) Store E-node to CLOST list;
10) ENDWHILE;
11) END of Algorithm 1.

The presented algorithm is admissible due to the following three reasons:

(1) Estimated task turnaround time is an underestimate.
(2) LCDF always chooses the node with the least cost to do tree expansion.
(3) Branching rule 1 generate all possible assignments for each of activated modules.

Therefore, the proposed algorithm will attain an underestimate for the optimal DU-mapping to achieve adimissibility [11]. This is the reason why this approach is called critical sink **underestimate** (CSU). Note that branching rule 2 has no influence on the admissibility of CSU, but will reduce the number of nodes in state space tree.

V. ILLUSTRATIVE EXAMPLE and EXPERIMENTAL RESULTS

Before using an example to illustrate the proposed approach, we define pruning rate as the percentage of nodes pruned by CSU

$$PR = \frac{N_E - N}{N_E} * 100\%$$

where N_E is the number of nodes generated by exhaustive search and N the number of nodes generated when a solution is found. Pruning rate serves as a performance metric for evaluating the proposed algorithm in this paper due to its strong indication of saving of time and space complexities.

The illustrative distributed computing system is composed of three processors and task is partitioned into five modules with precedence relationships. They are modeled as an undirected and DAG graphs as portrayed in Figs.2.1 and 2.2 respectively. We want to assign the five modules to the three processors in order to attain the minimum task turnaround time, which corresponds to finding the optimal DU-mapping of Fig.2.2 to Fig.2.1. The execution times of each module to all processors and the intermodule communication time of each adjacent modules are shown in Tab.1.1 and Tab.1.2 respectively. Without loss of generality, in Tab.1.2, we assume that the intermodule communication times for all interprocessor links are the same.

For this example, the state space tree generated by CSU is plotted in Fig.3. In Fig.3, the number in each node (circle) represents its estimated cost and the number in each square represents the ordering of tree expansion. Besides, the data in each parenthesis stands for some module assigned to a specific processor. In this case, we obtain the minimum task turnaround time 140 and the optimal DU-mapping is

$$M_{opt} = \{ (A,2), (B,1), (C,2), (D,1), (E,3) \}.$$

In Fig.3, only 25 nodes and 6 tree expansions are generated by CSU. Compared the above results with exhaustive search having branching rule 2, which generates 481 nodes and 208 tree expansions, the proposed algorithm saves 94.8 percent of node generations. Compared the results with exhaustive search but without branching rule 2, which generates 697 nodes and 262 tree expansions, the proposed algorithm saves 96.4 percent of node generations.

Several computation steps will be given to explain how to compute ETA value. From the second tree expansion of Fig.3, we have partial DU-mapping $M_x = \{(A,1)\}$ whose evaluation cost is computed as follows:

(1) Initially, $TIME_1 = TIME_2 = TIME_3 = 0$; $ST_A = 0$.
For the assigned module A, we have
(2) $TIME_1 = FE_A = 40$.
For the activated but unassigned modules B and C, both of them can be assigned to processors 1 and 2. Hence, using equations (8) and (9), we obtain
(3) $FE_B = \min\{ \max\{40,40\}+50, \max\{0,40+10\}+100 \} = 90$;
(4) $FE_C = \min\{ \max\{40,40\}+10, \max\{0,40+60\}+30 \} = 50$.
(5) $FE_D = 140$, $FE_E = 60$.
Using equation (10), the evalution cost is
(6) $ETA = \max\{ FE_D, FE_E \} = 140$.

Note that the term M_x is dropped for the sake of simplicity.

Based on algorithm 1, a task assignment simulator has been developed in VAX 8200 by using C language and hundreds of experiments have been made. In these experiments, sets of execution and communication times are generated by a random number generator, which is parameterized with some mean EC (Execution time: Communication time) ratio. From Tab.2, we observe that the range of average pruning rates for three different EC ratios versus different number of processors are as high as 95.0% to 97.5%. Moreover, the prining rates increase with the increasing number of modules for all EC ratios.

VI. CONCLUSION

The assignment problem dealt with in this paper is based on a general model of distributed computing system. It takes precedence constraint among modules into account. Besides, the cost function to measure its performance includes not only execution time and communication time, but also idle time. Though it is so hard to describe this kind of cost function under such a general model, we successfully develop a mathematical model to do so.

It is known that the problem of task assignment with precedence constraint is NP-complete [12]. A well-informed A^* algorithm is proposed to efficiently solve the task assignment in distributed computing systems. It is formulated as searching an optimal DU-mapping in a state space tree. An approach, critical sink underestimate (CSU), is developed to obtain an optimal task assignment and many nodes are bounded. The example and experiments show that the proposed approach is an effective method to solve the task assignment optimization problem in distributed computing systems.

REFERENCES

[1] S.H.Bokhari, "Dual processor scheduling with dynamic reassignmnet," IEEE Trans. Software Eng., vol. SE-5, pp.341-349, July 1979.

[2] H.S.Stone, "Multiprocessor scheduling with the aid of network flow algorithms," IEEE Trans. Software Eng., vol. SE-3, pp.85-93, Jan. 1979.

[3] H.S.Stone and S.H.Bokhari. "Control of distributed processes," Computer, vol. 11, pp.97-106, July, 1978.

[4] W.W.Chu, L.J.Holloway, M.T.Lan and K.Efe, "Task allocation in distributed data processing," Computer, vol. 13, pp.57-69, Nov. 1980.

[5] V.M.Lo, "Task assignment to minimize completion time," Proc. of the 5th Int'l Conf. on Distributed Computing Systems, Denver, Colorado, pp.329-336, May 1985.

[6] K.Efe, "Heuristic models of task assignment scheduling in distributed systems," Computer, vol.15, pp.50-56, June 1982.

[7] C.C.Shen and W.H.Tsai, "A graph matching approach to task assignment in distributed computing systems using a minimax criterion," IEEE Trans. Computer, vol. c-34, no.3, pp.197-203, March 1985.

[8] A.K.Ezzat, R.D.Bergeron and J.L.Pokoski, "Task allocation heuristics for distributed computing systems," Proc. of the 6th Int'l Conf. on Distributed Computing Systems, Cambridge, Mass., pp. 337-346, 1986.

[9] W.T.Chen and J.P.Sheu, "Task assignment in loosely-coupled multiprocessor systems," Journal of the Chinese Institute of Engineers, vol. 10, no. 6, pp. 721-726, 1987.

[10] J.Sheild, "Partitioning concurrent VLSI simulation programs onto a multiprocessor by simulated annealing," IEE Proceedings, vol. 134, Pc.E, no.1, pp.24-30, Jan.1987.

[11] N.J.Nilson, Principles of Artificial Intelligence, Tioga Publishing Co., 1980, chapter 2.

[12] M.R.Garey and D.S.Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

Tab.1.1. Execution Time

module \ processor	1	2	3
A	40	30	60
B	50	100	70
C	10	30	60
D	50	90	120
E	30	70	10

Tab.1.2. Intermodule Communication Time

module \ module	A	B	C	D	E
A	∞	10	40	∞	∞
B	∞	∞	∞	70	∞
C	∞	∞	∞	10	20
D	∞	∞	∞	∞	∞
E	∞	∞	∞	∞	∞

Tab.2. Average pruning rate for different number of processors and EC ratio

module \ EC ratio	4	1	0.25
3	96.3	96.7	97.5
4	95.2	95.2	96.8
5	95.0	95.1	96.6

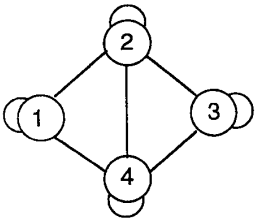

Fig.1.1 System Graph

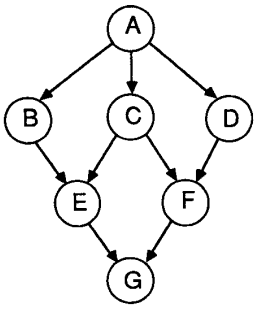

Fig.1.2 Task Graph

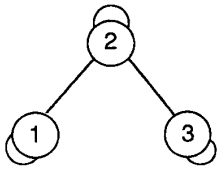

Fig.2.1 System Grpah

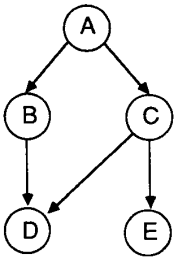

Fig.2.2 Task Graph

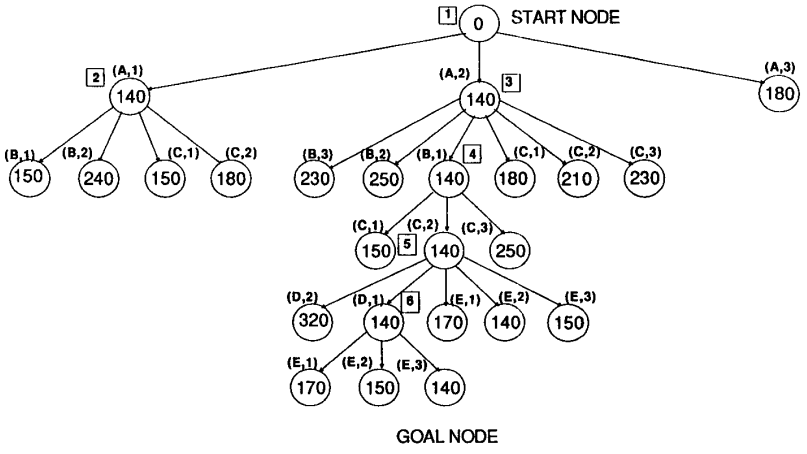

Fig.3 State space tree for CSU