Information Processing Letters 36 (1990) 241-245
North-Holland

1 December 1990

EFFICIENT ALLOCATION OF CHAIN-LIKE TASK ON CHAIN-LIKE NETWORK COMPUTERS

Jang-Ping SHEU

Department of Electrical Engineering, National Central University, Chungli 32054, Taiwan

Zen-Fu CHIANG

Department of Information Engineering, Tatung Institute of Technology, Taipei, Taiwan

Communicated by K. Ikeda
Received 4 July 1990

In this paper, we propose an algorithm to improve Bokhari's method for allocation of chain-like task on chain-like network computers. The time complexity of our algorithm is reduced from the time complexity $O(m^3n)$ of Bokhari's to $O(\min(m, n)m^2)$, where m is the number of modules and n is the number of processors. In addition, our algorithm relaxes Bokhari's constraints of all processors to be utilized and $n < m$.

Keywords: Distributed computing systems, task allocation, chain-like task, task turnaround time

1. Introduction

A distributed computing system has been defined as an interconnected collection of autonomous processors and integrated by a system-wide operating system. The often cited advantages of distributed computing systems include modularity, flexibility, extensibility, availability and integrity which make these systems attractive to many types of applications. But there exist some problems that prevent the system performance from increasing linearly as the number of processors increases. One of the major problems is to allocate several tasks over the processors optimally. A number of studies have been reported in the literature. They are basically the graph-theoretical method [1,2,8], the mathematical programming approach [4,5] and the heuristic method [3,6,7]. The problem we investigate is allocation of a chain-like task on the chain-like network computers which is first presented by Bokhari [1]. In this paper, we propose an algorithm which is more efficient than Bokhari's for solving the task assignment problem.

In the following, we examine the problem of optimally distributing a chain-like task over a chain-like network computer. It is assumed that the cost for computation of each module of the task on the processor is known. For any two adjacent modules, the communication cost between them is also assumed to be known (if the adjacent modules are coresident, the cost of communication between them is assumed to be zero). The time required for a processor to finish the assigned work equals to the sum of the costs to compute all of the modules that reside on it plus the communication costs between its adjacent processors. The quantity e_i represents the computation cost for module i of the chain-like task. The $c_{j,j+1}$ represents the communication cost between modules j and $j+1$ of the task. We assume these costs are known quantities.

We also assume that there is little or no precedence relationship among the modules of the task. We work under the constraint that each processor has a contiguous subchain modules assigned to it. That is, partitions of chains have to be such that modules j and $j+1$ are assigned to the same or to adjacent processors. We call this the contiguity constraint [1].

The purpose of task allocation on distributed computing systems is to reduce the task turnaround time by increasing the system throughput [6]. Let $t_p(A)$ denote the execution cost of the processor p according to a certain task allocation A. We call $t_p(A)$ the processor turnaround time of processor p. Let

$$t(A) = \max_{1 \leqslant i \leqslant n} t_p(A)$$

be the task turnaround time of A. It is easy to see that $t(A)$ is the total time required to complete the whole task according to allocation A. The optimal allocation cost function is to find an allocation A_{opt} such that $t(A_{opt}) \leqslant t(A)$ for all possible allocations A. Then

$$t(A_{opt}) = \min_{A} \max_{1 \leqslant p \leqslant n} t_p(A).$$

2. Task allocation on chain-like network computers

In this section, we propose an algorithm for allocation of a chain-like task on the chain-like network computers. Bokhari has presented an algorithm to solve this problem with time complexity $O(m^3 n)$, where m is the number of modules and n is the number of processors. Our algorithm reduces the time complexity from $O(m^3 n)$ to $O(\min(m, n) m^2)$.

Given a task T with m modules connected in a chain-like fashion, and a chain-like network computer with n processors, find the allocation of task T to processors that minimizes the task turnaround time. This problem first presented in [1] assumes that n, the number of processors, is less than m, the number of modules, and all processors are to be utilized. But if all processors are utilized, the allocation is not necessarily optimal. The extra processing and waiting time due to interprocessor communication grows rapidly and system performance quickly begins to degrade. We solve this problem without Bokhari's processors constraint. The relation between the number of modules m and the number of processors n is arbitrary and permits some processors not to be utilized if such an allocation is more efficient.

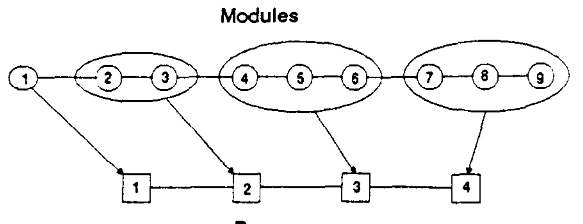

Fig. 1. Allocation of a nine-module chain-like task on a four-processor chain-like network computer.

The contiguity constraint ensures that two modules that communicate with each other lie on directly connected processors. For example, Fig. 1 shows an allocation of a nine-module chain-like task on a four-processor chain-like network computer. Thus, the turnaround time of processor 2 for task allocation as shown in Fig. 1 is equal to

$$c_{1,2} + c_{3,4} + \sum_{2 \leqslant i \leqslant 3} e_i.$$

Based on our objective function, the solution of this problem is to find an allocation which minimizes the maximum processor turnaround time.

In the following, we first present Bokhari's algorithm [1]. In Fig. 2, we show a task allocation graph by using Bokhari's method. Each layer corresponds to a processor and the label on each node corresponds to a subchain of modules. Each layer contains all pairs $\langle h, i\rangle$ such that $1 \leqslant h \leqslant i \leqslant m$. A node labeled $\langle h, i\rangle$ is connected to all nodes $\langle i+1, j\rangle$ in the layer below it. All nodes in the first (last) layer are connected to node s (d). Any path connecting node s to node d corresponds to an allocation of modules to processors. If a path contains the node $\langle h, i\rangle$ of layer k, this represents the assignment of modules h through i to processor k. For example, the thick edges of Fig. 2 correspond to the allocation of Fig. 1. To avoid a congested diagram, many nodes and edges have been omitted from Fig. 2.

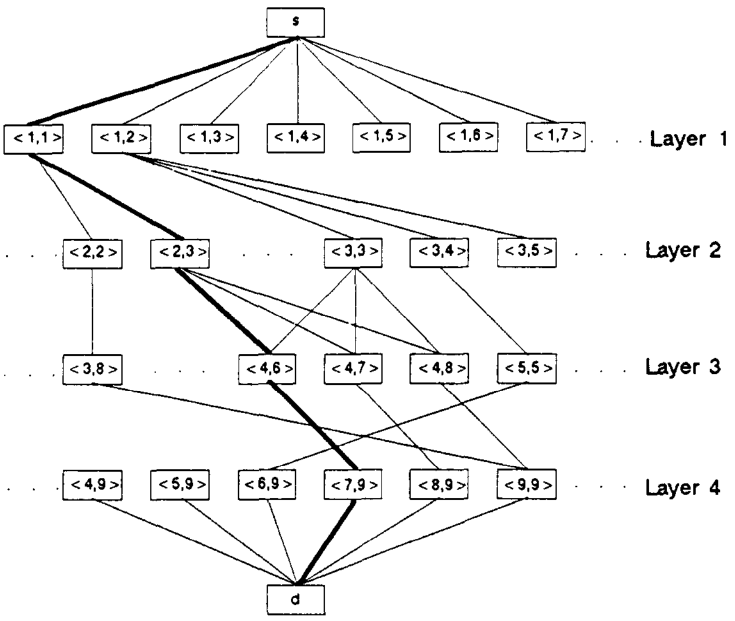

Fig. 2. Bokhari's task allocation graph for Fig. 1.

Our solution is also to draw up a task allocation graph which is less complex than Bokhari's method. We generate the task allocation graph for this problem as follows. The kth layer (except the last layer) contains $m-k+1$ nodes and a sequence numbers k, $k+1, \ldots, m$ is labeled at each node from left to right. The last layer consists of only one node labeled m. All nodes in the first layer are connected to node s. A node i is connected to all nodes, whose labels are greater than i, in the layer below it. The edge that connects source node s and node i of the first layer represents the assignment of modules 1 through i to processor 1. The edge that connects node a in layer $k-1$ and node b in layer k represents the assignment of modules $a+1$ through b to processor k. According to our method, the task allocation graph corresponding to Fig. 2 is shown in Fig. 3. Note that there are only $\min(m, n)$ layers in the task allocation graph.

After constructing such a graph, we can add the weights to the edges of this task allocation graph as follows. The weight of edge joining source node s to node i of the first layer is equal to the sum of all computation costs of modules 1 through i and communication cost between modules i and $i+1$. The weight of edge joining node a in layer $k-1$ and b in layer k equals the sum of all computation costs of modules $a+1$ through b and communication costs between modules a and $a+1$, and between modules b and $b+1$.

Finally, we apply Bokhari's labeling procedure to construct a table, which lists the minimum task turnaround time versus variety number of utilized

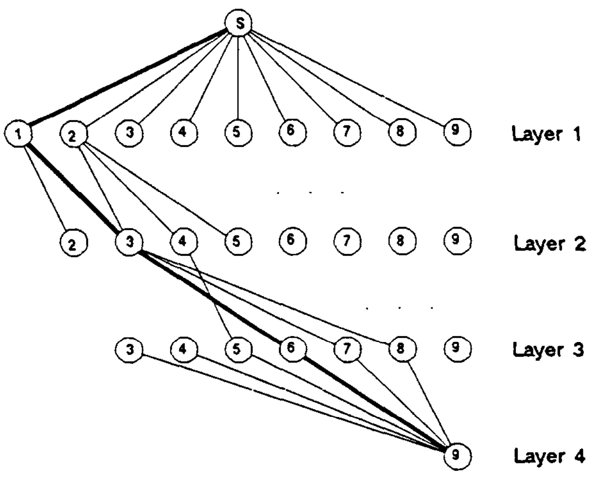

Fig. 3. The improved task allocation graph of Fig. 2.

processors, called a task turnaround time table. With this table, the optimal allocation can be found. Now, we present the formal algorithm as follows.

Algorithm. Find the optimal allocation of a chain-like task on a chain-like network computer.

Step 1. Apply the following rules to construct the task allocation graph.

(1) There is one distinguished node called the source node, denoted s.

(2) In addition to the source node, there are $\min(m, n)$ layers in the task allocation graph.

(3) The kth layer consists of $m-k+1$ nodes and a sequence numbers $k, k+1, \ldots, m$ is labeled at each node from left to right.

(4) A node labeled h in layer $k-1$ is connected to all nodes, in layer k, labeled number greater than h. All nodes in the first layer are connected to node s.

(5) Each edge joining source node s to node j, for $1 \leqslant j \leqslant m$, has weight

$$c_{j,j+1} + \sum_{1 \leqslant h \leqslant j} e_h.$$

/ * The first item represents the communication cost between modules j and $j+1$ and the second item represents the computation cost sum of modules 1 through j. * /

(6) Each edge joining node a in layer $k-1$ to node b in layer k has weight

$$c_{a,a+1} + c_{b,b+1} + \sum_{a+1 \leqslant h \leqslant b} e_h.$$

/ * The first item represents the communication cost between modules a and $a+1$, the second represents the communication cost between modules b and $b+1$ and the third represents the computation cost sum of modules $a+1$ through b. * /

Step 2. Initially all nodes are given infinite labels except the source node, where it is labeled zero.

Step 3. For $k = 1$ to $\min(m, n)$ do

/ * Bokhari's labeling procedure * /

Examine each edge e emanating downwards from layer $k-1$. Suppose it connects node a in layer $k-1$ to node b in layer k. Let the weights on this edge be $W(e)$. Then replace $L(b)$ by $\min\{L(b), \max(W(e), L(a))\}$.

Record the value of $L(m)$ in layer k to the task turnaround time table for k processors utilized.

Step 4. (1) Search the task turnaround time table to find the minimum task turnaround time.

(2) If k processors are utilized then we trace backwards from node m in layer k to source node s to find an optimal allocation.

Each feasible path from source node s to node m in each layer of the task allocation graph corresponds to a partition of the sequence numbers $1, 2, \ldots, m$. Each subsequence corresponds to an assignment of modules on a processor. For example, a feasible path 1–3–6–9 in Fig. 3 associates module 1 with processor 1, modules 2 and 3 with processor 2, modules 4, 5, and 6 with processor 3 and modules 7, 8, and 9 with processor 4. It is obvious that each feasible path of the task allocation graph corresponds to an allocation. Under the contiguity constraint, each allocation corresponds to a partition of the sequence $1, 2, \ldots, m$. In the task allocation graph exactly one feasible path corresponds to each partition. Thus, all allocations are in the task allocation graph. The labeling procedure examines each path emanating downwards from each layer, so all feasible paths are inspected. As a result, the proposed algorithm

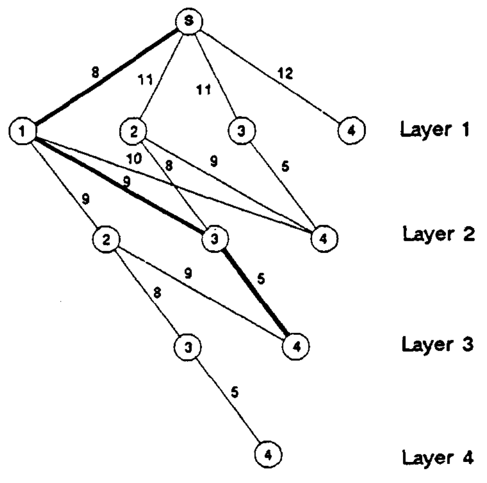

Fig. 4. The task allocation graph of Example 2.1.

Table 1
Task turnaround time table of T

Number of processors used	Turnaround time
1	12
2	10
3	9
4	9

can correctly find one optimal allocation of single chain-like task on a chain-like network computer.

The total number of nodes is $O(\min(m, n)m)$. Since each node has at most m edges connected to it, there are $O(\min(m, n)m^2)$ edges in all. The algorithm looks at each edge in the assignment graph exactly once. Thus, the complexity of this algorithm is $O(\min(m, n)m^2)$. We now give an example to show how this algorithm works.

Example 2.1. A chain-like task T, which consists of four modules, has computation costs $e_1 = 5$, $e_2 = 2$, $e_3 = 2$, and $e_4 = 3$ and communication costs $c_{1,2} = 3$, $c_{2,3} = 4$ and $c_{3,4} = 2$. The chain-like network computer consists of four processors.

Figure 4 shows the constructed task allocation graph. Our algorithm is used to construct the task turnaround time table as shown in Table 1. Searching the task turnaround time table of T, we find that three or four processors utilized is the optimal allocation. Let $[j, k] \to h$ represent modules j through k of task T assigned to processor h. In Fig. 4 there exist two feasible paths that have the same minimum task turnaround time 9. One path is $[1, 1] \to 1$, $[2, 3] \to 2$, and $[4, 4] \to 3$. The other path is $[1, 1] \to 1$, $[2, 2] \to 2$, $[3, 3] \to 3$, and $[4, 4] \to 4$. Here, we choose the first path as our solution. The thick edges corresponding to this optimal allocation is shown in Fig. 4.

3. Conclusions

In this paper, we consider the problem of finding efficient algorithm for allocation of a chain-like task on the chain-like network computers. Bokhari first presents a solution to this problem with time complexity $O(m^3n)$. Our algorithm can improve Bokhari's method. The time complexity of the proposed algorithm is $O(\min(m, n)m^2)$. In addition, the constraints of all processors to be utilized and $n < m$ are relaxed. In some cases, if all processors are utilized, the allocation is not necessarily optimal. The extra processing and waiting time due to interprocessor communications will increase the task turnaround time.

References

[1] S.H. Bokhari, Partitioning problems in parallel, pipelined, and distributed computing, *IEEE Trans. Comput.* **37** (1988) 48–57.

[2] W.T. Chen and J.P. Sheu, Task assignment in loosely-coupled multiprocessor systems, *J. Chinese Inst. Eng.* **10** (1987) 721–726.

[3] K. Efe, Heuristic models of task assignment scheduling in distributed systems, *IEEE Comput.* **15** (1982) 50–56.

[4] V.M. Lo, Heuristic algorithms for task assignment in distributed systems, *IEEE Trans. Comput.* **37** (1988) 1384–1397.

[5] P.Y. Ma, E.Y.S. Lee and M. Tsuchiya, A task allocation model for distributed computing system, *IEEE Trans. Comput.* **31** (1982) 41–47.

[6] C.C. Shen and W.H. Tsai, A graph matching approach to optimal task assignment in distributed computing system using a minimax criterion, *IEEE Trans. Comput.* **34** (1985) 197–203.

[7] J.B. Sinclair, Efficient computation of optimal assignments for distributed tasks, *J. Parallel and Distributed Comput.* **4** (1987) 342–362.

[8] H.S. Stone, Multiprocessor scheduling with the aid of network flow algorithms, *IEEE Trans. Software Engrg.* **3** (1977) 85–93.