

Partitioning multiple-chain-like task across a host-satellite system

Alain Billionnet

CEDRIC, Institut d'Informatique d'Entreprise, 18 allée Jean Rostand, 91025 Evry Cedex, France

Communicated by L. Kott

Received 11 April 1991

Revised 6 November 1992

Abstract

We propose an algorithm to improve Bokhari's method for partitioning multiple-chain-like tasks across a host-satellite system. The time complexity of our algorithm is reduced from the time complexity $O(m^2n \log \max(m, n))$ of Bokhari to $O(mn \log \max(m, n))$, where n is the number of satellites and m the number of modules in each of the n chains which must be partitioned between a satellite and the host.

Key words: Algorithms; Analysis of algorithms; Host-satellite system; Pipelining; Task allocation; Dynamic programming; Complexity

1. Introduction

A major problem arising in multiple processor systems is the optimal allocation of tasks. A number of studies have been reported in the literature (see references). The problem we investigate is the allocation of a multiple-chain-like task on a set of n satellite processors and a host which was first considered by Bokhari [4]. In this paper, we propose an algorithm which solves this task allocation problem more efficiently than the algorithm of Bokhari.

In the problem considered by Bokhari [4], a large host computer is connected to n satellite computers which receive data from a real-time environment. The data streams entering each satellite have to be processed in a pipeline fashion. Each satellite computer is able to partition its workload between itself and the larger, more powerful host to improve its individual processing

time. However, moving some modules to the host adversely impacts the performance of the other satellites. As noted by Bokhari it is the complex interaction between the loads of the satellites via the shared host which makes this optimal partitioning problem difficult.

Each satellite receives a chain-structured task, i.e. a task made up of m modules numbered $1, \dots, m$. Like Bokhari, we can assume without loss of generality that each chain has exactly m modules. This assumption is made in order to simplify the presentation. Each task has an intercommunication pattern such that module j is connected only to modules $j + 1$ (if $j < m$) and $j - 1$ (if $j > 1$). Each satellite has a continuous subchain of modules to process and the n remaining subchains are processed by the host (see Fig. 1).

For each module j ($j = 1, \dots, m$) of the chain assigned to satellite i ($i = 1, \dots, n$), the time re-

quired to run it on the satellite is $es_{i,j}$, and the time to run it on the host is $eh_{i,j}$. For any two adjacent modules j and $j + 1$, the communication cost between them is also assumed to be known. If the adjacent modules are co-resident (on a satellite or on the host) the cost of communication between them is assumed to be zero; otherwise it is equal to $c_{i,j}$ (if modules j and $j + 1$ belong to the chain associated to satellite i).

Number the modules from right to left and define the partition point of each chain by the highest numbered module in that chain that is assigned to the satellite. When these n chains are partitioned between the host and the n satellites, the time required by the entire system to complete the processing of one frame of data from each stream is determined by the greater of (1) the individual load of the most heavily loaded satellite and (2) the sum of the collective loads of the host.

In order to simplify the presentation, we add to each chain one fictitious module numbered 0 linked to module 1 and such that $es_{i,0} = c_{i,0} = 0$ and another fictitious module numbered $(m + 1)$ linked to module m and such that $c_{i,m} = eh_{i,m+1} = 0$, for $i = 1, \dots, n$. We can thus assume that modules 0 are assigned to satellites and modules $m + 1$ to the host.

For each satellite i ($i = 1, \dots, n$) let $k(i)$ denote the number of the module which is the partition point of the corresponding chain, i.e., modules 0 to $k(i)$ are assigned to satellite i and modules $k(i) + 1$ to $m + 1$ are assigned to the host. Remark that $0 \leq k(i) \leq m$.

The load of satellite i is equal to the sum of the times to execute all of the modules that

reside on it, $\sum_{(k=0,k(i))} es_{i,k}$, plus a communication overhead for the time taken to transmit the final result to the host, $c_{i,k(i)}$. Similarly, the load of the host is equal to the sum of the times to execute all of the modules that reside on it, $\sum_{(i=1,n)} \sum_{(k=k(i)+1,m+1)} eh_{i,k}$, plus the sum of the communication times required to transmit results from the n satellites to the host, $\sum_{(i=1,n)} c_{i,k(i)}$.

The partitioning problem we consider here is to determine the partition point of each chain in order to minimize

$$\max \left\{ \max_{i=1,n} \{ \text{load of satellite } i \}, \text{ load of the host} \right\}$$

It can be stated as follows: for all $i \in \{1, \dots, n\}$ determine $k(i) \in \{0, 1, 2, \dots, m\}$ such that the cost

$$\max \left\{ \max_{i=1,n} \left(\sum_{(k=0,k(i))} es_{i,k} + c_{i,k(i)} \right), \sum_{(i=1,n)} \left(\sum_{(k=k(i)+1,m+1)} eh_{i,k} + c_{i,k(i)} \right) \right\}$$

is minimal.

To solve this problem Bokhari associates with it an *assignment graph* which is a doubly weighted graph, i.e., a graph which has two weights associated with each edge e : a sum weight $\sigma(e)$ and a bottleneck weight $\beta(e)$. Bokhari then proves that the optimal sum-bottleneck path between two distinct nodes of this graph corresponds to the optimal assignment. As usual, a path P is a sequence of edges e_1, e_2, \dots, e_p . The sum weight of path P , denoted $S(P)$, is the sum of all $\sigma(e_i)$ and the bottleneck weight, denoted $B(P)$, is the largest of all $\beta(e_i)$. The sum-bottleneck weight of path P is defined as $\max\{S(P), B(P)\}$ and an

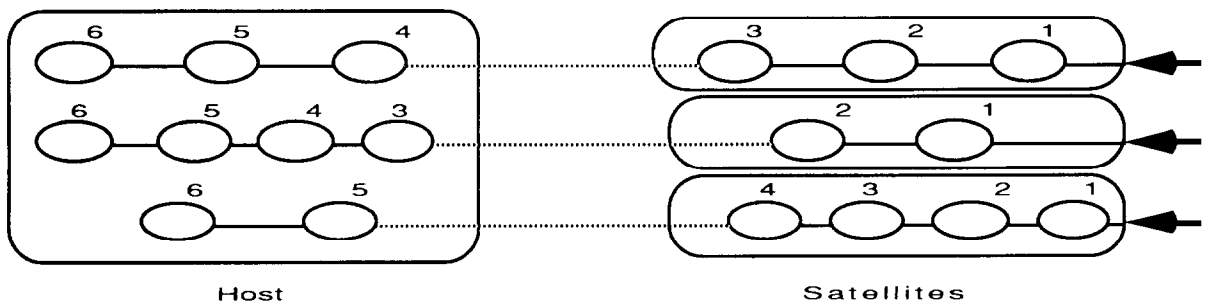


Fig. 1. Representation of a host-satellite system.

optimal sum–bottleneck path is a path for which the sum–bottleneck weight is minimal. Bokhari proposes an algorithm of complexity $O(m^2n \cdot \log \max\{m, n\})$ to find an optimal sum–bottleneck path in the assignment graph and therefore an optimal assignment. We propose in the following a quite different approach for this partitioning problem and an algorithm which takes $O(mn \log \max\{m, n\})$ time.

2. Notation

For all $(i, j) \in \{1, \dots, n\} \times \{0, \dots, m\}$, let $s(i, j)$ be the load of satellite i if modules 0 to j are assigned to it (module j is the partition point) and $h(i, j)$ the corresponding load of the host, i.e., $s(i, j) = \sum_{(k=0,j)} es_{i,k} + c_{i,j}$ and $h(i, j) = \sum_{(k=j+1,m+1)} eh_{i,k} + c_{i,j}$.

Let $L = (s_1, s_2, \dots, s_{n(m+1)})$ be the list of all elements $s(i, j)$ such that $s_1 \leq s_2 \leq \dots \leq s_{n(m+1)}$ and for all $t \in \{1, 2, \dots, n(m+1)\}$ let $(u(t), v(t)) \in \{1, \dots, n\} \times \{0, \dots, m\}$ be such that $s(u(t), v(t))$ corresponds to element s_t of L . The quantity s_t is thus the load of satellite $u(t)$ if modules 0 to $v(t)$ are assigned to it. For all $t \in \{1, 2, \dots, n(m+1)\}$, we denote $h(u(t), v(t))$ by h_t .

For all $(i, t) \in \{1, \dots, n\} \times \{n, n+1, \dots, n(m+1)\}$ let $f_i(s_t)$ be the minimal load of the host induced by satellite i when the load of this satellite is located before s_t in list L , i.e.,

$$f_i(s_t) = \min\{h_k : k \leq t \text{ and } u(k) = i\}.$$

For all $t \in \{n, n+1, \dots, n(m+1)\}$, we denote by $f(s_t)$ the minimal total load of the host when the load of each satellite is located before s_t in list L , i.e.,

$$f(s_t) = \sum_{(i=1,n)} f_i(s_t).$$

3. Main results

Theorem 1. *Let F^* be the cost of an optimal partitioning. Then*

$$F^* = \min_{t=n,n+1,\dots,n(m+1)} \max\{f(s_t), s_t\}.$$

Proof. (i) By definition, for all $t \in \{n, n+1, \dots, n(m+1)\}$, $f(s_t)$ is a possible load of the host, the load of each satellite being less than or equal to s_t . $\max\{f(s_t), s_t\}$ is therefore greater than or equal to the cost of a partitioning and consequently

$$\min_{t=n,n+1,\dots,n(m+1)} \max\{f(s_t), s_t\} \geq F^*.$$

(ii) Consider an optimal partitioning. Let S be the greatest load of the satellites and H the load of the host. The cost F^* of this partitioning is equal to $\max\{H, S\}$. Let r be the greatest index belonging to $\{n, n+1, \dots, n(m+1)\}$ and such that $s_r = S$. By definition $f(s_r) \leq H$ and since $s_r = S$ we have $\max\{f(s_r), s_r\} \leq \max\{H, S\} = F^*$ and finally

$$\min_{t=n,n+1,\dots,n(m+1)} \max\{f(s_t), s_t\} \leq F^*. \quad \square$$

Now consider two consecutive elements s_{t-1} and s_t of list L and examine how to compute $f(s_t)$ from $f(s_{t-1})$ and $f_{u(t)}(s_{t-1})$.

Property 2. *For all $t \in \{n+1, n+2, \dots, n(m+1)\}$,*

$$f(s_t) = f(s_{t-1}) - f_{u(t)}(s_{t-1}) + \min\{f_{u(t)}(s_{t-1}), h_t\}.$$

Proof. By definition $f_i(s_t) = \min\{h_k : k \leq t \text{ and } u(k) = i\}$.

If $i \neq u(t)$ then $f_i(s_t) = \min\{h_k : k \leq t-1 \text{ and } u(k) = i\} = f_i(s_{t-1})$.

If $i = u(t)$ then $f_i(s_t) = \min\{\min\{h_k : k \leq t-1 \text{ and } u(k) = i\}, h_t\} = \min\{f_i(s_{t-1}), h_t\}$.

Since we have, by definition, $f(s_t) - f(s_{t-1}) = \sum_{(i=1,n)} [f_i(s_t) - f_i(s_{t-1})]$, we deduce

$$\begin{aligned} f(s_t) - f(s_{t-1}) &= \sum_{(i=1,n; i \neq u(t))} [f_i(s_{t-1}) - f_i(s_{t-1})] \\ &\quad + f_{u(t)}(s_t) - f_{u(t)}(s_{t-1}) \\ &= \min\{f_{u(t)}(s_{t-1}), h_t\} - f_{u(t)}(s_{t-1}). \quad \square \end{aligned}$$

We now describe the details of the optimal partitioning algorithm which is derived directly from Theorem 1 and Property 2.

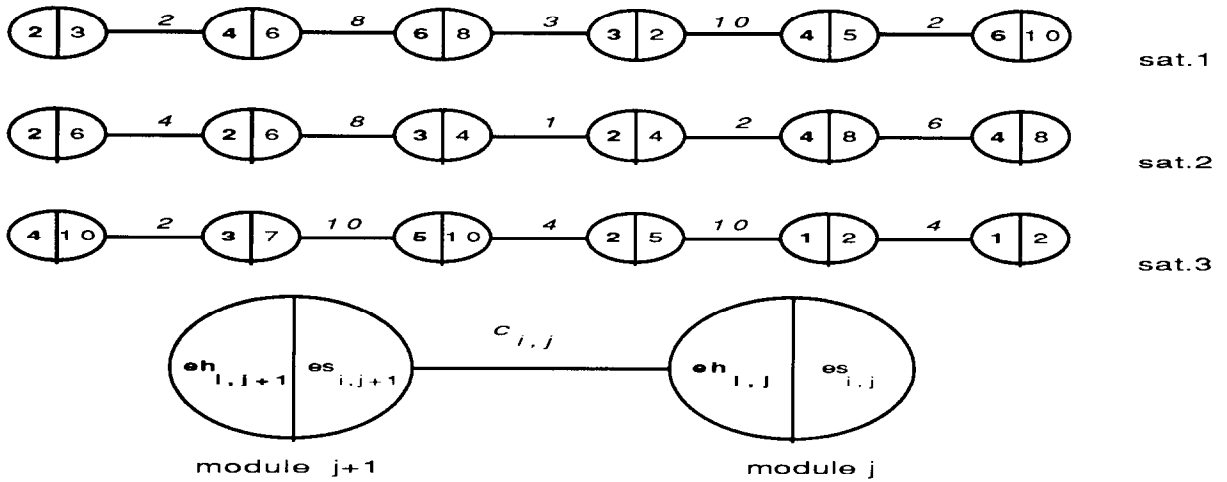


Fig. 2. An example with three satellites and six modules in each chain.

4. The optimal partitioning algorithm

input: $es_{i,j}$, $eh_{i,j}$ and $c_{i,j}$
output: the cost F^* of an optimal partitioning and, for all $i = 1, \dots, n$, the highest numbered module assigned to satellite i , $k(i)$

begin

comment: computing the $s(i, j)$'s

1. **for** $i = 1$ to n **do**
 $s(i, 0) \leftarrow 0$
2. **for** $j = 1$ to m **do**
 $s(i, j) \leftarrow s(i, j - 1) + es_{i,j} - c_{i,j-1} + c_{i,j}$

endo

comment: computing the $h(i, j)$'s

3. **for** $i = 1$ to n **do**
 $h(i, m) \leftarrow 0$
4. **for** $j = m - 1, m - 2, \dots, 0$ **do**
 $h(i, j) \leftarrow h(i, j + 1) + eh_{i,j+1} - c_{i,j+1} + c_{i,j}$

endo

endo

5. **for** $t = 1, 2, \dots, n(m + 1)$ **do** find
 $u(t), v(t), s_t, h_t$ **enddo**

comment: computing

$$\min_{t=n, n+1, \dots, n(m+1)} \max\{f(s_t), s_t\}$$

comment: initialization

$$f \leftarrow 0$$

6. **for** $t = 1$ to n **do** $f_{u(t)} \leftarrow h_t; f \leftarrow h_t; k(u(t)) \leftarrow 0$

enddo

$$F^* \leftarrow f$$

comment: finding the optimal partitioning

Table 1

t	$u(t)$	$v(t)$	s_t	h_t
1	1	0	0	25
2	2	0	0	17
3	3	0	0	16
4	3	1	6	19
5	1	1	12	21
6	3	3	13	16
7	2	1	14	19
8	3	2	14	24
9	2	2	18	11
10	1	3	20	15
11	2	3	21	8
12	1	2	25	25
13	3	5	28	6
14	3	4	29	17
15	2	4	32	12
16	1	4	33	14
17	1	5	33	4
18	1	6	34	0
19	2	5	34	6
20	2	6	36	0
21	3	6	36	0

Table 2

t	f_1	f_2	f_3	f	F^*	$k(1)$	$k(2)$	$k(3)$
4	25	17	16	58	58	0	0	0
5	21	17	16	54	54	1	0	0
6	21	17	16	54	54	1	0	0
7	21	17	16	54	54	1	0	0
8	21	17	16	54	54	1	0	0
9	21	11	16	48	48	1	2	0
10	15	11	16	42	42	3	2	0
11	15	8	16	39	39	3	3	0
12	15	8	16	39	39	3	3	0
13	15	8	6	29	29	3	3	5
14	15	8	6	29	29	3	3	5
15	15	8	6	29	29	3	3	5
16	14	8	6	28	29	3	3	5
17	4	8	6	18	29	3	3	5
18	0	8	6	14	29	3	3	5
19	0	6	6	12	29	3	3	5
20	0	0	6	6	29	3	3	5
21	0	0	0	0	29	3	3	5

```

7. for  $t = n + 1$  to  $n(m + 1)$  do
    if  $h_t < f_{u(t)}$  then  $f \leftarrow f - f_{u(t)} + h_t$ ;  $f_{u(t)} \leftarrow h_t$ 
    endif
    if  $\max\{f, s_t\} < F^*$  then  $F^* \leftarrow \max\{f, s_t\}$ ;
     $k(u(t)) \leftarrow v(t)$  endif
enddo
end

```

Theorem 3. *The previous algorithm finds an optimal partitioning in most $O(mn \log \max(m, n))$ time.*

Proof. Loops 1 and 3 require $O(nm)$ time. The time required to run loop 5 is $O(mn \log(mn))$ since N elements can be sorted in time $O(N \log N)$ (for example by heapsort [11]). Loop 6 requires $O(n)$ time and loop 7, $O(nm)$ time. Thus the overall complexity of the algorithm is $O(mn \log \max(m, n))$. \square

5. Example

Consider the example described by Fig. 2. Table 1 gives $u(t)$, $v(t)$, s_t and h_t for $t = 1, 2, \dots, 21$. At the initialization of the algorithm: $f_1 = 25$, $f_2 = 17$, $f_3 = 16$, $f = 58$, $F^* = 58$, $k(1) = 0$, $k(2) = 0$ and $k(3) = 0$. The values of f_1 , f_2 , f_3 , f , F^* , $k(1)$, $k(2)$ and $k(3)$ during the

execution of the algorithm are shown in Table 2. We see that in the optimal partitioning, modules 0 to 3 must be assigned to satellite 1, modules 0 to 3 must be assigned to satellite 2 and modules 0 to 5 must be assigned to satellite 3. The corresponding load of the host is 29 and the corresponding loads of satellites 1, 2, 3 are respectively 20, 21, 28. The cost of this partitioning is 29.

References

- [1] A. Billionnet, M.-C. Costa and A. Sutter, Les problèmes de placement dans les systèmes distribués, *Technique Sci. Inform.* **8** (3) (1989) 307–337.
- [2] A. Billionnet, M.C. Costa and A. Sutter, An efficient algorithm for a task allocation problem, *J. ACM* **39** (3) (1992) 502–518.
- [3] S.H. Bokhari, *Assignment Problems in Parallel and Distributed Computing* (Kluwer Academic Publishers, Norwell, MA, 1987).
- [4] S.H. Bokhari, Partitioning problems in parallel, pipelined, and distributed computing, *IEEE Trans. Comput.* **37** (1) (1988) 48–57.
- [5] W.W. Chu, L.J. Holloway, M. Lan and K. Efe, Task allocation in distributed data processing, *Computer* (November 1980) 57–69.
- [6] K. Efe, Heuristic models of task assignment scheduling in distributed systems, *Computer* **15** (1982) 50–56.
- [7] D. Fernandez-Baca, Allocating modules to processors in a distributed system, *IEEE Trans. Software Engrg.* **15** (11) (1989) 1427–1436.

- [8] A. Gabrielian and D.B. Tyler, Optimal object allocation in distributed computing system, in: *Proc. Internat. Conf. on Distributed Computing Systems*, San Francisco (1984) 88–95.
- [9] V.M. Lo, Heuristic algorithms for task assignment in distributed systems, *IEEE Trans. Comput.* **37** (11) (1988) 1384–1397.
- [10] P.R. Ma, E.Y.S. Lee and M. Tsuchiga, A task allocation model for distributed computing systems, *IEEE Trans. Comput.* **31** (1) (1982) 41–47.
- [11] K. Mehlhorn, *Data Structures and Algorithms 1: Sorting and Searching* (Springer, Berlin, 1984).
- [12] C.C. Price and S. Krishnaprasad, Software allocation models for distributed computing systems, in: *Proc. Internat. Conf. on Distributed Computing Systems*, San Francisco (1984) 40–48.
- [13] J.P. Sheu and Z.F. Chiang, Efficient allocation of chain-like task on chain-like network computers, *Inform. Process. Lett.* **36** (1990) 241–245.
- [14] J.B. Sinclair, Efficient computation of optimal assignments for distributed tasks, *J. Parallel Distributed Comput.* **4** (4) (1987) 342–362.
- [15] H. Stone, Multiprocessor scheduling with the aid of network flow algorithms, *IEEE Trans. Software Engrg.* **3** (1977) 85–94.
- [16] H. Stone, Critical load factors in two processors distributed systems, *IEEE Trans. Software Engrg.* **4** (1978) 254–258.
- [17] D. Towsley, Allocating programs containing branches and loops within a multiple processor system, *IEEE Trans. Software Engrg.* **12** (10) (1986) 1018–1024.