STATE UNIVERSITY OF NEW YORK AT

STONY BROOK

CEAS TECHNICAL REPORT 665

A Multi-Job Load Sharing Strategy for Divisible Jobs on Bus Networks

J. Sohn & T.G. Robertazzi

April 16, 1993

A Multi-Job Load Sharing Strategy for Divisible Jobs on Bus Networks

Jeeho Sohn and Thomas G. Robertazzi, *Senior Member, IEEE*

Dept. of Electrical Engineering,

SUNY at Stony Brook,

Stony Brook, N.Y. 11794

Abstract

In this paper, a load sharing problem involving the optimal load allocation of a linear data file over N processors interconnected in a bus-oriented network is investigated. Three distinct types of bus networks are examined in the case when there is more than one outstanding divisible job in the network. These are a network with a control processor for load distribution, a network without control processor but whose processors are equipped with front-end processors for communications off-loading, and finally a network without control processor and with processors without front-end processors. It is found that a *multi-job* scheme outperforms a single-job scheme in terms of the total solution finish time. Closed form solutions and simple recursive algorithms for the determination of the optimal load allocation are also presented.

I. Introduction

In recent years, there has been of great deal of interest in distributed sensor networks [1]. In distributed sensor networks, measurements are made by spatially distinct sensors. The data is, then, broadcast to a site where the spatially disparate readings are fused so that meaningful decisions can be made regarding these measurements. One major issue for distributed sensor networks is the trade-off between communication and computation [2]. That is, the decision of how much time should be spent to communicate and how much time should be spent to process (compute) the measurements becomes an important problem.

Related to the distributed sensor network problem are a number of papers which deal with scheduling and load sharing in multiprocessors [3, 4]. However most work assumes that a job can be assigned to a single processor. Only recently has there been interest in multiprocessor scheduling with jobs that need to be assigned to more than one processor [5, 6, 7].

Recently there has been work on a load sharing problem involving a *divisible* job. A divisible job is a job that can be arbitrarily partitioned in a linear fashion among a number of processors. Applications include the processing of very long linear data files as in signal and image processing and Kalman filtering.

In [8], recursive expressions for calculating the optimal load allocation for linear daisy chains of processors were presented. This is based on the simplifying premise that for an optimal allocation of load, all processors must stop processing at the same time. Intuitively, this is because otherwise some processors would be idle while others were still busy. Analogous solutions have been developed for tree networks [9] and bus networks [10, 11]. Asymptotic solutions for systems with large or even an infinite number of processors and limitations in performance when adding processors appear in [12, 19]. Closed form solutions were presented in [13] for bus and tree architectures where processor and link speeds are homogeneous. In [14], the concept of an equivalent processor that behaves identically to

a collection of processors in the context of a linear daisy chain of processors and a proof that, for a linear daisy chain of processors load sharing a divisible job, the optimal solution involves all processors stopping at the same time are introduced. An analytic proof for bus networks that for a minimal time solution all processors must stop computing at the same time is shown in [15, 16]. Previous proofs were heuristic. The equivalence of first distributing load either to the left or to the right from a point in the interior of a linear daisy chain is demonstrated in [18]. Optimal sequences of load distribution in tree networks are described in [17, 20]. A new load distribution strategy for tree networks [21] and linear daisy chains [22] is also discussed.

All of the previous work concentrated on investigating more efficient ways of distributing load to minimize the total solution time under the premise that there is only one job present in the network. Most practical computer systems operate in an environment where multiple jobs may be submitted. An intriguing question is how to efficiently handle the submission of multiple jobs in a distributed network. Under a naive, *single-job scheme*, a distributed network sequentially processes one job at a time. In this paper we propose a more sophisticated *multi-job scheme* for bus networks that exploits the special structure of a divisible job on a bus network to yield a smaller finish (solution) time than the single-job scheme.

This paper is organized as follow: Closed form solutions and simple recursive algorithms for the determination of the optimal load allocation for three types of bus networks – a network with a control processor, a network without control processor but with processors that are equipped with front-end processors, and a network without control processor and with processors without front-end processors – are presented in section II, III and IV, respectively. In section V, a performance evaluation is presented which compares the single-job scheme with the multi-job scheme. Finally the conclusion appears in section VI.

II. Architecture 1:

Bus Network with a Control Processor

Consider first the case where the network model consists of a queueing system for incoming jobs, a control processor for distributing the processing load, and N processors attached to a linear bus as in Fig. 1. New arriving jobs enter the queueing system and wait there for service. In this paper, the queueing system is assumed to be of an infinite size so that there are no *turned away*, *lost* or *blocked* jobs. There is also no restriction of service discipline, i.e., it can be *first-in first-out* (FIFO), *last-in first-out* (LIFO), etc. Under the supervision of the control processor, one of the waiting jobs in the queue can be delivered to the processor. The control processor distributes the processing load among the N processors interconnected through a bus type communication medium in order to obtain the benefits of parallel processing. The control processor does no processing itself. It does not matter whether the processors are equipped with front-end processors or not because the load distribution is performed by the control processor. Each processor may have a different computing speed.

The following notations will be used throughout this paper:

$\hat{\alpha}_i, \hat{\beta}_i$: The fraction of the entire processing load of the first job and the second job, respectively, that is assigned to the ith processor in *the single-job scheme*.

α_i, β_i : The fraction of the entire processing load of the first job and the second job, respectively, that is assigned to the ith processor in *the multi-job scheme*.

Z : A constant that is inversely proportional to the channel speed of bus.

w_i : A constant that is inversely proportional to the computing speed of the ith processor.

T_{cm_m} : The time that it takes to transmit the entire set of measurement data of the mth job over the channel when $Z = 1$.

T_{cp_m} : The time that it takes for the ith processor to process (compute) the entire load of the mth job when $w_i = 1$.

T_{f_m} : The finish time of the entire processing load of the mth job.

The timing diagram for the bus network with a control processor in the single-job scheme and the multi-job scheme are depicted in Fig. 2 and Fig. 3, respectively. In the single-job scheme, the load distribution (transmission or communication) of the second job is started after the completion of the first job's computation (T_{f_1}). Therefore, the time that each processor waits for its processing load to be received – the transmission time during which the control processor transmits the processing load to each processor – is wasted. That is, the durations $\hat{\beta}_1 Z T_{cm_2}, (\hat{\beta}_1 + \hat{\beta}_2) Z T_{cm_2}, \ldots, (\hat{\beta}_1 + \hat{\beta}_2 + \cdots + \hat{\beta}_N) Z T_{cm_2}$ are wasted in processor 1, processor 2, ...,processor N, respectively.

On the other hand, since the load distribution of the second job can be started right after the control processor finishes the transmission of first job's load, those processors which received their processing load prior to the completion of the first job's computation (T_{f_1}) can start their computation immediately after T_{f_1} in the multi-job scheme. Hence, in the multi-job scheme one can significantly reduce the overall processing time for the second job compared to the single-job scheme since the processing load is transmitted immediately following the completion of the previous job's transmission.

Now, the closed-form solutions for finding $\hat{\alpha}_i$ (or $\hat{\beta}_i$) for the optimal load allocation for each processor in the single-job scheme which appears in [15][16] will be reviewed. Following this, the recursive algorithm to find β_i for the optimal load allocation in the multi-job scheme will be examined in the following subsections.

A. *Closed-Form Solution to Find $\hat{\alpha}_i$ in the Single-Job Scheme*

Since distributing the processing load is done in the same fashion for every job in the single-job scheme and for the first job in the multi-job scheme, the closed-form solutions to find $\hat{\alpha}_i$, $\hat{\beta}_i$ and α_i are the same in both cases as well. The case of finding $\hat{\alpha}_i$ for the first job in the single-job scheme will thus solely be reviewed.

As proved in [15][16], the minimum time solution occurs when all processors finish their computation at the same time. Based on that fact, one can set up the following set of equations (Fig. 2). These equate the computation time of the ith processor to the transmission plus computation time of the $i+1$st processor.

$$\hat{\alpha}_1 w_1 T_{cp_1} = \hat{\alpha}_2 Z T_{cm_1} + \hat{\alpha}_2 w_2 T_{cp_1} \quad (1)$$

$$\hat{\alpha}_2 w_2 T_{cp_1} = \hat{\alpha}_3 Z T_{cm_1} + \hat{\alpha}_3 w_3 T_{cp_1} \quad (2)$$

$$\hat{\alpha}_3 w_3 T_{cp_1} = \hat{\alpha}_4 Z T_{cm_1} + \hat{\alpha}_4 w_4 T_{cp_1} \quad (3)$$

$$\vdots$$

$$\hat{\alpha}_{N-1} w_{N-1} T_{cp_1} = \hat{\alpha}_N Z T_{cm_1} + \hat{\alpha}_N w_N T_{cp_1} \quad (4)$$

In these timing diagrams communication time appears above the axis and computation time appears below the axis. These equations can be solved as follows:

$$\hat{\alpha}_2 = \frac{w_1 T_{cp_1}}{Z T_{cm_1} + w_2 T_{cp_1}} \hat{\alpha}_1 = k_1 \hat{\alpha}_1 \quad (5)$$

$$\hat{\alpha}_3 = \frac{w_2 T_{cp_1}}{Z T_{cm_1} + w_3 T_{cp_1}} \hat{\alpha}_2 = k_2 \hat{\alpha}_2 = k_2 k_1 \hat{\alpha}_1 \quad (6)$$

$$\hat{\alpha}_4 = \frac{w_3 T_{cp_1}}{Z T_{cm_1} + w_4 T_{cp_1}} \hat{\alpha}_3 = k_3 \hat{\alpha}_3 = k_3 k_2 k_1 \hat{\alpha}_1 \quad (7)$$

$$\vdots$$

$$\hat{\alpha}_N = \frac{w_{N-1} T_{cp_1}}{Z T_{cm_1} + w_N T_{cp_1}} \hat{\alpha}_{N-1} = k_{N-1} \hat{\alpha}_{N-1} = k_{N-1} k_{N-2} \cdots k_1 \cdot \hat{\alpha}_1 \quad (8)$$

where

$$k_i = \frac{\hat{\alpha}_{i+1}}{\hat{\alpha}_i} = \frac{w_i T_{cp_1}}{Z T_{cm_1} + w_{i+1} T_{cp_1}} \qquad 1 \le i \le N-1 \quad (9)$$

Here k_i can be directly computed from the system parameters $(w_i, Z, T_{cm_1}, T_{cp_1})$. Since the fractions of the total processing load should sum to one, $\hat{\alpha}_1$ can be obtained by the normalization equation.

$$\begin{aligned} 1 &= \hat{\alpha}_1 + \hat{\alpha}_2 + \hat{\alpha}_3 + \cdots + \hat{\alpha}_N \\ &= (1 + k_1 + k_1 k_2 + \cdots + k_1 k_2 \cdots k_{N-1})\hat{\alpha}_1 \end{aligned} \tag{10}$$

Once $\hat{\alpha}_1$ is found, one can easily calculate the rest of the load allocation fractions $(\hat{\alpha}_2, \hat{\alpha}_3, \ldots, \hat{\alpha}_N)$, that the originating processor (the control processor) should calculate in order to achieve the minimal solution time when distributing load, as all those fractions are function of $\hat{\alpha}_1$ as seen in Eq.(5) to Eq.(8). Note that the single-job scheme is optimal if only a single job is to be solved. When multiple jobs are involved it is not optimal, as the following section will demonstrate.

B. *Algorithm to Find β_i in the Multi-Job Scheme*

Case 1: When $T_{f_1} \geq ZT_{cm_1} + ZT_{cm_2}$

Let us first consider the easiest case that the transmission of the second job's processing load is started immediately after the control processor finishes the transmission of the first job's processing load and is finished before the completion of the first job's computation (T_{f_1}) as shown in Fig. 4. In other words, the location of T_{f_1} is placed after the time that the transmission of the second job's processing load is finished. In this case, the algorithm to find the β_i's is very simple.

Since all the processors have received their processing load prior to the finish time of the previous job (T_{f_1}) they can simultaneously start their computation right after T_{f_1} and finish their computation at the same time for a minimal solution time. Therefore, the processing

time in each processor is the same. That is,

$$\beta_1 w_1 T_{cp_2} = \beta_2 w_2 T_{cp_2} = \beta_3 w_3 T_{cp_2} = \cdots = \beta_N w_N T_{cp_2} \tag{11}$$

and those equations can be further expressed as a function of β_1:

$$\beta_2 = \frac{w_1}{w_2}\beta_1 \tag{12}$$

$$\beta_3 = \frac{w_1}{w_3}\beta_1 \tag{13}$$

$$\beta_4 = \frac{w_1}{w_4}\beta_1 \tag{14}$$

$$\vdots$$

$$\beta_N = \frac{w_1}{w_N}\beta_1 \tag{15}$$

As the normalization equation states that the sum of the fractions of the total processing load is one, β_1 can be obtained.

$$\sum_{i=1}^{N} \beta_i = \sum_{i=1}^{N} \frac{w_1}{w_i}\beta_1 = 1 \tag{16}$$

Therefore, one can derive a simple closed-form solution:

$$\beta_1 = (\sum_{i=1}^{N} \frac{w_1}{w_i})^{-1} \qquad 1 \le i \le N \tag{17}$$

$$\beta_i = \frac{w_1}{w_i}\beta_1 \tag{18}$$

Note that since there is no wasted time between the consecutive jobs in every processor, the total required processing time for the second job $(T_{f_2} - T_{f_1})$ is minimized compared to any other cases.

Case 2: When $T_{f_1} < ZT_{cm_1} + ZT_{cm_2}$

This subsection will consider the case that T_{f_1} is located somewhere in the transmission period of the second job's load. That is, some of the processors which have received their processing load before T_{f_1} can start their computation immediately after T_{f_1} while the other

processors which did not receive their processing load yet have to wait some period of time for their processing load to be delivered. In Fig. 5, the processing finish time of the first job's load (T_{f_1}) is in the transmission of the nth fraction of the second job's load. Let I_i denote the time interval between T_{f_1} and the time that the ith processor receives its processing load. Thus processor 1 to processor n-1 can start their computation immediately after T_{f_1} while processor n, processor n+1, ...,processor N have to wait $I_n, I_{n+1}, \ldots, I_N$ amounts of time, respectively, to receive their processing load and start their computation at time $T_{f_1} + I_n, T_{f_1} + I_{n+1}, \ldots, T_{f_1} + I_N$, respectively.

Now let us derive the algorithm to find β_i's which minimizes the total solution time. We will assume that all processors must stop at the same time. Intuitively this is because otherwise the solution could be improved by transferring load from one processor to another. The processing time of processor 1 to processor n-1 is the same since they simultaneously start their computation and finish it at the same time.

$$\beta_1 w_1 T_{cp_2} = \beta_2 w_2 T_{cp_2} = \cdots = \beta_{n-1} w_{n-1} T_{cp_2} \tag{19}$$

Therefore

$$\beta_i = \frac{w_1}{w_i}\beta_1 \qquad\qquad 1 \le i \le n-1 \tag{20}$$

The next step is to find an expression for I_i, which is the time interval between T_{f_1} and the time ith processor receives its processing load.

$$I_i = (\beta_1 + \beta_2 + \cdots + \beta_i) Z T_{cm_2} - \alpha_N w_N T_{cp_1} \qquad\qquad n \le i \le N \tag{21}$$

The processing time of the ith processor to compute its processing load is then

$$\begin{aligned} \beta_i w_i T_{cp_2} &= \beta_1 w_1 T_{cp_2} - I_i \\ &= \beta_1 w_1 T_{cp_2} - (\sum_{k=1}^{i} \beta_k) Z T_{cm_2} + \alpha_N w_N T_{cp_1} \qquad\qquad n \le i \le N \end{aligned} \tag{22}$$

Now consider the processing time taken by the nth processor to compute its processing load, i.e., the processing time when $i = n$.

$$\beta_n w_n T_{cp_2} = \beta_1 w_1 T_{cp_2} - (\beta_1 + \beta_2 + \cdots + \beta_{n-1} + \beta_n) Z T_{cm_2} + \alpha_N w_N T_{cp_1} \tag{23}$$

Here β_n can be expressed as a function of β_1:

$$(Z T_{cm_2} + w_n T_{cp_2})\beta_n = [w_1 T_{cp_2} - (1 + \frac{w_1}{w_2} + \cdots + \frac{w_1}{w_{n-1}}) Z T_{cm_2}]\beta_1 + \alpha_N w_N T_{cp_1} \tag{24}$$

or

$$\begin{aligned} \beta_n &= \frac{w_1 T_{cp_2} - (\sum_{k=1}^{n-1} \frac{w_1}{w_k}) Z T_{cm_2}}{Z T_{cm_2} + w_n T_{cp_2}} \beta_1 + \frac{\alpha_N w_N T_{cp_1}}{Z T_{cm_2} + w_n T_{cp_2}} \\ &= p_n \beta_1 + q_n \end{aligned} \tag{25}$$

where p_n and q_n are constants which are functions of system parameters $(Z, w_i, T_{cp_1}, T_{cp_2}, T_{cm_2})$ and α_N [1].

$$p_n = \frac{w_1 T_{cp_2} - (\sum_{k=1}^{n-1} \frac{w_1}{w_k}) Z T_{cm_2}}{Z T_{cm_2} + w_n T_{cp_2}} \tag{26}$$

$$q_n = \frac{\alpha_N w_N T_{cp_1}}{Z T_{cm_2} + w_n T_{cp_2}} \tag{27}$$

Let us consider now one more case when $i = n + 1$.

$$\beta_{n+1} w_{n+1} T_{cp_2} = \beta_1 w_1 T_{cp_2} - (\beta_1 + \beta_2 + \cdots + \beta_{n-1} + \beta_n + \beta_{n+1}) Z T_{cm_2} + \alpha_N w_N T_{cp_1} \tag{28}$$

Here β_{n+1} can also be expressed as a function of β_1.

$$\begin{aligned} & (Z T_{cm_2} + w_{n+1} T_{cp_2})\beta_{n+1} \\ = \; & \beta_1 w_1 T_{cp_2} - [(1 + \frac{w_1}{w_2} + \cdots + \frac{w_1}{w_{n-1}})\beta_1 + (p_n \beta_1 + q_n)] Z T_{cm_2} + \alpha_N w_N T_{cp_1} \\ = \; & [w_1 T_{cp_2} - (\sum_{k=1}^{n-1} \frac{w_1}{w_k} + p_n) Z T_{cm_2}]\beta_1 + \alpha_N w_N T_{cp_1} - q_n Z T_{cm_2} \end{aligned} \tag{29}$$

[1] Here α_N is assumed to be a known value because this is the fraction found when the first job is distributed.

Hence

$$\begin{aligned}\beta_{n+1} &= \frac{w_1 T_{cp_2} - (\sum_{k=1}^{n-1} \frac{w_1}{w_k} + p_n) Z T_{cm_2}}{Z T_{cm_2} + w_{n+1} T_{cp_2}} \beta_1 + \frac{\alpha_N w_N T_{cp_1} - q_n Z T_{cm_2}}{Z T_{cm_2} + w_{n+1} T_{cp_2}} \\ &= p_{n+1} \beta_1 + q_{n+1} \end{aligned} \tag{30}$$

where p_{n+1} and q_{n+1} are also constants since p_n and q_n have been found previously.

One can see that this procedure can be continued up to the case where $i = N$. Then every fraction (β_i's) that the originating processor should calculate to achieve the minimum solution time and which is a function of β_1 is found.

Once all β_i's have been found, β_1 can then be calculated from the normalization equation.

$$\begin{aligned} 1 &= (\beta_1 + \beta_2 + \cdots + \beta_{n-1}) + (\beta_n + \beta_{n+1} + \cdots + \beta_N) \\ &= (1 + \frac{w_1}{w_2} + \cdots + \frac{w_1}{w_{n-1}})\beta_1 + (p_n \beta_1 + q_n) + (p_{n+1}\beta_1 + q_{n+1}) + \cdots + (p_N \beta_1 + q_N) \\ &= (\sum_{k=1}^{n-1} \frac{w_1}{w_k} + \sum_{k=n}^{N} p_k)\beta_1 + \sum_{k=n}^{N} q_k \end{aligned} \tag{31}$$

As a summary of this section, let us rephrase the algorithm to find β_i's when $T_{f_1} < Z T_{cm_1} + Z T_{cm_2}$.

$$(i) \qquad p_i = \begin{cases} \frac{w_1 T_{cp_2} - (\sum_{k=1}^{n-1} \frac{w_1}{w_k}) Z T_{cm_2}}{Z T_{cm_2} + w_n T_{cp_2}} & i = n \\ \\ \frac{w_1 T_{cp_2} - (\sum_{k=1}^{n-1} \frac{w_1}{w_k} + \sum_{k=n}^{i-1} p_k) Z T_{cm_2}}{Z T_{cm_2} + w_i T_{cp_2}} & n+1 \le i \le N \end{cases} \tag{32}$$

$$q_i = \begin{cases} \frac{\alpha_N w_N T_{cp_1}}{Z T_{cm_2} + w_n T_{cp_2}} & i = n \\ \\ \frac{\alpha_N w_N T_{cp_1} - (\sum_{k=n}^{i-1} q_k) Z T_{cm_2}}{Z T_{cm_2} + w_i T_{cp_2}} & n+1 \le i \le N \end{cases} \tag{33}$$

$$(ii) \qquad \beta_1 = \frac{1 - \sum_{k=n}^{N} q_k}{\sum_{k=1}^{n-1} \frac{w_1}{w_k} + \sum_{k=n}^{N} p_k} \tag{34}$$

$$(iii) \qquad \beta_i = \begin{cases} \frac{w_1}{w_i} \beta_1 & 1 \le i \le n-1 \\ p_i \beta_1 + q_i & n \le i \le N \end{cases} \tag{35}$$

Case 3: When the position of T_{f_1} is changed due to the new values of β_i's

Reading along so far, one should understand that an unrealizable situation has been described as one can ask how can one determine in which fraction of transmission T_{f_1} is located even before the actual values of β_i's are calculated. Actually it may seem impossible to determine the location of T_{f_1} because to do that, all values of β_i's should be calculated first and to calculate the values of β_i's, the location of T_{f_1} should be decided, and so on.

But this dilemma can be resolved in the following recursive fashion: First, identify the location of T_{f_1} by using the values of $\hat{\beta}_i$ calculated in the single-job scheme. Since only the values of the system parameters are required to calculate $\hat{\beta}_i$'s, one can easily compute $\hat{\beta}_i$'s by the closed-form solution described in section II.A. Next the location of T_{f_1} can be decided. Once the location of T_{f_1} is known, *tentative* β_i's can be computed by the algorithm described in the previous section.

In the next step, one investigates the location of T_{f_1} as in Fig. 6. If the location of T_{f_1} is changed from $\hat{\beta}_n$ to β_{n+1} or even to $\beta_{n+2}, \beta_{n+3}, \ldots$, then one simply increases the value of n to the corresponding new value, $n+1, n+2, \ldots$, and calculates the values of the β_i's again. The process is repeated until the location of T_{f_1} is not changed when comparing the location of T_{f_1} with the previous values of β_i and with the new values of β_i at the time of each iteration.

Note that when it is said that the position or location of T_{f_1} is changed it means that the fraction in which T_{f_1} is located is changed. Actually, T_{f_1} itself is not changed and depends only on the previous job's load.

Let us give an example. Suppose that there is a bus network with 5 processors. For the second job, first calculate $\hat{\beta}_i$'s by the closed-form solution with the single-job scheme $(\hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3, \hat{\beta}_4, \hat{\beta}_5)$. If T_{f_1} is located in the transmission of the second fraction $(\hat{\beta}_2)$, set $n = 2$ and calculate the new values of β_i's by the algorithm of the multi-job scheme $(\beta_1, \beta_2, \beta_3, \beta_4, \beta_5)$. Now if T_{f_1} is located in the transmission of the fourth fraction (β_4) when

reidentifying the location, reset $n = 4$ and recalculate β_i's. If the location of T_{f_1} is still in the transmission of β_4 after recalculation and reidentification, then stop. In practice, however, the case such that the location of T_{f_1} is changed after the second iteration is rare.

Note that the value of n never decreases in each iteration. It always increases. This is because the earlier parts of the fractions become smaller (e.g., β_1, β_2) and the latter parts of the fractions become larger (e.g., β_{N-1}, β_N) with each iteration.

III. Architecture 2: No Control Processor, Processors with Front-End Processors

The bus network to be examined in this section is one without control processor as in Fig. 7. As with the case of the network with a control processor, arriving jobs first enter a queuing system and wait for service. One of the waiting jobs in the queuing system is delivered directly to one of the N processors. Without loss of generality, we will assume that is processor 1. Processor 1 distributes the processing load of the received job among the other processors for parallel processing. Each processor is equipped with a front-end processor for communications off-loading. That is, the processors can communicate and compute at the same time. It is also assumed here that each processor may have a different computing speed.

As one might guess, the solution procedure to find the best fractions for optimal load allocation is very similar to the case where the network has a control processor. In particular, the closed-form solutions to find $\hat{\alpha}_i$, $\hat{\beta}_i$ (single-job scheme) and α_i are exactly the same. So the closed-form solutions will not be discussed again. The algorithm to find β_i's for optimal load allocation will be examined in the next subsection.

A. *Algorithm to Find β_i in the Multi-Job Scheme*

Case 1: When $T_{f_1} \geq (1-\alpha_1)ZT_{cm_1} + (1-\beta_1)ZT_{cm_2}$

Consider the case where the transmission of the second job's load starts right after the completion of the first job's load and finishes before the completion of computation for the first job's load as in Fig. 8. This is the same situation as in case 1, section II.B. Since all the processors receive their processing load for the second job before they finish their computation of the first job's load, they can start and finish their computation simultaneously to achieve the minimal solution time. Therefore, the processing time of the second job's load in each processor is the same and this yields the same result for β_i's as in case 1, section II.B. That is,

$$\beta_1 = (\sum_{i=1}^{N} \frac{w_1}{w_i})^{-1} \qquad 1 \leq i \leq N \tag{36}$$

$$\beta_i = \frac{w_1}{w_i}\beta_1 \tag{37}$$

However, there is one major difference from the case where the network has a control processor. In the case where the network has a control processor, the total amount of the transmission time for one job was independent of how the control processor assigns the fractions (β_i's) or where T_{f_1} is located. That is, the total amount of the transmission time for the second job was a fixed value (ZT_{cm_2}).

But in the case where there is no control processor, it is a different story. Since the total amount of the transmission time where there is no control processor is $(\beta_2 + \beta_3 + \cdots + \beta_N)ZT_{cm_2} = (1-\beta_1)ZT_{cm_2}$, there is a strong dependency on the value of β_1. In other words, the smaller the value of β_1, the larger the total amount of the transmission time, and vice versa.

Therefore, it is possible for the following case to exist: when first determining the location of T_{f_1} by using the values of the load allocation fractions in the single-job scheme $(\hat{\beta}_2, \hat{\beta}_3, \ldots, \hat{\beta}_N)$, T_{f_1} is located after the time that processor 1 finishes the transmission of the

second job's load. So one might calculate the new values of the fractions (β_i's) simply by using Eq.(36) and Eq.(37). But when identifying the location of T_{f_1} again after calculation, it is possible that the new location of T_{f_1} is placed before the end of the transmission as in Fig. 9. This is because, as mentioned before, the early part of the fractions becomes smaller and the latter part of the fractions becomes larger in the multi-job scheme compared to the single-job scheme. If this happens, the situation becomes the case that $T_{f_1} < (1-\alpha_1)ZT_{cm_1} + (1-\beta_1)ZT_{cm_2}$. This case is discussed in the next subsection.

One more thing that must be pointed out is that one might think that it takes less time to transmit the second job's load in the single-job scheme than in the multi-job scheme because the multi-job's total transmission time is longer than the single-job's. It does. But the total processing time $(T_{f_2} - T_{f_1})$ to compute the second job's load in the single-job scheme takes longer than that in the multi-job scheme because for the load allocation fractions calculated in the single-job scheme, for jobs after the first job, each processor does not finish its computation at the same time. This is because the load fractions are based on the assumption that there is waiting time. In fact if all the processors have already received their load when computation starts, there is no waiting time and processors will not stop at the same time.

Case 2: When $T_{f_1} < (1-\alpha_1)ZT_{cm_1} + (1-\beta_1)ZT_{cm_2}$

Again the case where the network does not have a control processor and where processors are equipped with front-end processors is very similar to the case where the network has a control processor as in section II. The only difference is that the originating processor (processor 1) in the network without control processor and with front-end processors does not need to transmit its own fraction (α_1, β_1) of the load. The timing diagram for a bus network with no control processor and where processors are equipped with front-end processors when $T_{f_1} < (1-\alpha_1)ZT_{cm_1} + (1-\beta_1)ZT_{cm_2}$ is depicted in Fig. 10.

Since processor 1 to processor n-1 received their processing load before they finish their computation of the first job's load, they can start and finish their computation of the second job's load at the same time.

$$\beta_1 w_1 T_{cp_2} = \beta_2 w_2 T_{cp_2} = \cdots = \beta_{n-1} w_{n-1} T_{cp_2} \tag{38}$$

Hence

$$\beta_i = \frac{w_1}{w_i}\beta_1 \qquad\qquad 1 \leq i \leq n-1 \tag{39}$$

which is exactly the same result as in case 2, section II.B.

The time interval I_i, the time between T_{f_1} and the time ith processor receives its processing load, is defined in the same fashion as before.

$$I_i = (\beta_2 + \beta_3 + \cdots + \beta_i) Z T_{cm_2} - \alpha_N w_N T_{cp_1} \tag{40}$$

The processing time the ith processor needs to compute its own processing load is then

$$\begin{aligned} \beta_i w_i T_{cp_2} &= \beta_1 w_1 T_{cp_2} - I_i \\ &= \beta_1 w_1 T_{cp_2} - (\sum_{k=2}^{i} \beta_k) Z T_{cm_2} + \alpha_N w_N T_{cp_1} \qquad\qquad n \leq i \leq N \end{aligned} \tag{41}$$

Now, $\beta_n, \beta_{n+1}, \ldots, \beta_N$ can be expressed as a function of β_1 as was done previously.

Consider the first case when $i = n$.

$$\beta_n w_n T_{cp_2} = \beta_1 w_1 T_{cp_2} - (\beta_2 + \beta_3 + \cdots + \beta_{n-1} + \beta_n) Z T_{cm_2} + \alpha_N w_N T_{cp_1} \tag{42}$$

so

$$(Z T_{cm_2} + w_n T_{cp_2})\beta_n = [w_1 T_{cp_2} - (\frac{w_1}{w_2} + \frac{w_1}{w_3} + \cdots + \frac{w_1}{w_{n-1}}) Z T_{cm_2}]\beta_1 + \alpha_N w_N T_{cp_1} \tag{43}$$

or

$$\begin{aligned} \beta_n &= \frac{w_1 T_{cp_2} - (\sum_{k=2}^{n-1} \frac{w_1}{w_k}) Z T_{cm_2}}{Z T_{cm_2} + w_n T_{cp_2}} \beta_1 + \frac{\alpha_N w_N T_{cp_1}}{Z T_{cm_2} + w_n T_{cp_2}} \\ &= p_n \beta_1 + q_n \end{aligned} \tag{44}$$

where p_n and q_n are constants

$$p_n = \frac{w_1 T_{cp_2} - (\sum_{k=2}^{n-1} \frac{w_1}{w_k}) Z T_{cm_2}}{Z T_{cm_2} + w_n T_{cp_2}} \tag{45}$$

$$q_n = \frac{\alpha_N w_N T_{cp_1}}{Z T_{cm_2} + w_n T_{cp_2}} \tag{46}$$

Next, consider the case when $i = n + 1$.

$$\beta_{n+1} w_{n+1} T_{cp_2} = \beta_1 w_1 T_{cp_2} - (\beta_2 + \beta_3 + \cdots + \beta_{n-1} + \beta_n + \beta_{n+1}) Z T_{cm_2} + \alpha_N w_N T_{cp_1} \tag{47}$$

or

$$\begin{aligned} & (Z T_{cm_2} + w_{n+1} T_{cp_2}) \beta_{n+1} \\ = \; & \beta_1 w_1 T_{cp_2} - [(\frac{w_1}{w_2} + \frac{w_1}{w_3} + \cdots + \frac{w_1}{w_{n-1}}) \beta_1 + (p_n \beta_1 + q_n)] Z T_{cm_2} + \alpha_N w_N T_{cp_1} \\ = \; & [w_1 T_{cp_2} - (\sum_{k=2}^{n-1} \frac{w_1}{w_k} + p_n) Z T_{cm_2}] \beta_1 + \alpha_N w_N T_{cp_1} - q_n Z T_{cm_2} \end{aligned} \tag{48}$$

Hence

$$\begin{aligned} \beta_{n+1} &= \frac{w_1 T_{cp_2} - (\sum_{k=2}^{n-1} \frac{w_1}{w_k} + p_n) Z T_{cm_2}}{Z T_{cm_2} + w_{n+1} T_{cp_2}} \beta_1 + \frac{\alpha_N w_N T_{cp_1} - q_n Z T_{cm_2}}{Z T_{cm_2} + w_{n+1} T_{cp_2}} \\ &= p_{n+1} \beta_1 + q_{n+1} \end{aligned} \tag{49}$$

And this procedure can be continued up to the case when $i = N$. Then β_1 can be computed by the normalization equation.

$$\begin{aligned} 1 &= (\beta_1 + \beta_2 + \cdots + \beta_{n-1}) + (\beta_n + \beta_{n+1} + \cdots + \beta_N) \\ &= (1 + \frac{w_1}{w_2} + \cdots + \frac{w_1}{w_{n-1}}) \beta_1 + (p_n \beta_1 + q_n) + (p_{n+1} \beta_1 + q_{n+1}) + \cdots + (p_N \beta_1 + q_N) \\ &= (\sum_{k=1}^{n-1} \frac{w_1}{w_k} + \sum_{k=n}^{N} p_k) \beta_1 + \sum_{k=n}^{N} q_k \end{aligned} \tag{50}$$

As one can see, the algorithm to find β_i's is the same as in case 2, section II.B. The only difference is that the summation term in the numerator of p_i is the sum of $\frac{w_1}{w_k}$ from $k = 2$ to $n - 1$ ($\sum_{k=2}^{n-1} \frac{w_1}{w_k}$) in the case where the network has no control processor and where processors

are equipped with front-end processors while the summation term in the numerator of p_i is the sum of $\frac{w_1}{w_k}$ from $k = 1$ to $n-1$ ($\sum_{k=1}^{n-1} \frac{w_1}{w_k}$) in the case where the network has a control processor.

As a conclusion to this section, the algorithm to find the β_i's is summarized below:

$$(i) \qquad p_i = \begin{cases} \frac{w_1 T_{cp_2} - (\sum_{k=2}^{n-1} \frac{w_1}{w_k}) Z T_{cm_2}}{Z T_{cm_2} + w_n T_{cp_2}} & i = n \\ \\ \frac{w_1 T_{cp_2} - (\sum_{k=2}^{n-1} \frac{w_1}{w_k} + \sum_{k=n}^{i-1} p_k) Z T_{cm_2}}{Z T_{cm_2} + w_i T_{cp_2}} & n+1 \leq i \leq N \end{cases} \tag{51}$$

$$q_i = \begin{cases} \frac{\alpha_N w_N T_{cp_1}}{Z T_{cm_2} + w_n T_{cp_2}} & i = n \\ \\ \frac{\alpha_N w_N T_{cp_1} - (\sum_{k=n}^{i-1} q_k) Z T_{cm_2}}{Z T_{cm_2} + w_i T_{cp_2}} & n+1 \leq i \leq N \end{cases} \tag{52}$$

$$(ii) \qquad \beta_1 = \frac{1 - \sum_{k=n}^{N} q_K}{\sum_{k=1}^{n-1} \frac{w_1}{w_k} + \sum_{k=n}^{N} p_k} \tag{53}$$

$$(iii) \qquad \beta_i = \begin{cases} \frac{w_1}{w_i} \beta_1 & 1 \leq i \leq n-1 \\ p_i \beta_1 + q_i & n \leq i \leq N \end{cases} \tag{54}$$

Case 3: When the position of T_{f_1} is changed due to the new values of β_i's

Again, it is possible for the case such that the position of T_{f_1} is changed due to the new values of the β_i's to exist. If this happens, as discussed before, we can solve this problem by (1) identifying the location of T_{f_1} by comparing the values of $\hat{\beta}_i$'s calculated in the single-job scheme and β_i's calculated in the multi-job scheme (2) if the location of T_{f_1} is changed, assign the value of n to the corresponding new value (3) calculate the values of β_i's again and compare the location of T_{f_1} associated with the old β_i's and the new value associated with the new β_i's until the location of T_{f_1} is not changed.

IV. Architecture 3: No Control Processor, Processors without Front-End Processors

The network configuration to be discussed in this section is similar to the previous section in that it involves a bus network with no control processor. But in this section, the processors are not equipped with front-end processors. Let's take a look Fig. 11, the timing diagram where there is no control processor and where processors are not equipped with front-end processors. For convenience, the originating processor which distributes the processing load to the other processors to achieve the benefits of parallel processing is assigned to be processor N.

However, there is a problem when one tries to find the best values of the fractions for optimal load allocation that occurs at processor N. In Fig. 11, it seems that processor N can start its computation when it finishes the transmission of the second job's load. But imagine what happens if there is a third job waiting in the queue. The processor N then has to transmit the third job's load when it finishes the transmission of the second job's load instead of computing the last fraction of the second job's load. So the last fraction of the second job's load is dependent on the presence of the next or future job. That is, if there is no job available after the second job, processor N has to assign some nonzero value to β_N. On the other hand, if there is a next job, the value of β_N will be zero. Therefore, we cannot determine the value of β_N because we are dealing with a purely deterministic system.

One simple solution would be if processor N is used only for transmitting the processing load no matter whether there is a next job or not, i.e., processor N acts as a control processor. Then the network configuration becomes a bus network with a control processor and N-1 processors.

V. Performance Evaluation

Based on the previous results, some performance evaluation results were simulated in various cases – in the single-job scheme and in the multi-job scheme for the network with a control processor and the network with no control processor but where processors are equipped with front-end processors. In each plot, the total solution finish time is drawn against the number of jobs. In this section, it is simulated when the total number of presented jobs is 10 and there are five processors in the network, and it is assumed that the values of the channel speed, the computing speed of each processor and communication load for every job are assigned to unity (Z, all w_i's, all T_{cm}'s = 1).

- In Fig. 12, the total solution finish time is plotted against the number of jobs when the computational load of every job is 6 (all T_{cp}'s = 6) which is a relatively medium load in comparison with communication load which is set to one and with five processors. Since the computational load for every job is the same, the total solution finish time is linear in the single-job scheme. Clearly, the the multi-job scheme is superior to the single-job scheme in terms of the total solution finish time. Even in the case where processors are not equipped with front-end processors for the multi-job scheme, the total solution time is significantly reduced compared to the case where the processors are equipped with front-end processors in the single-job scheme.

- Fig. 13 and Fig. 14 plots the solution time when the computational load of the first half of 10 jobs is 10 ($T_{cp_{1,2,3,4,5}} = 10$, heavy computational load) and that of the second half of 10 jobs is 3 ($T_{cp_{6,7,8,9,10}} = 3$, light computational load), and vice versa for the single-job scheme and for the multi-job scheme for the network with a control processor and for the network without control processor but where processors are equipped with front-end processors. Naturally, the total solution finish time of the last job in the single-job scheme is the same no matter what order the computational load is assigned

although there is some difference in the total solution finish time in the middle of the processing of the given jobs. However, for the multi-job scheme assigning the job with heavy computational load first and the one with light computational load last causes reduction of the total solution finish time at the end of processing. This is because serving the heavy computational load first results in a timing diagram like Fig. 4 or Fig. 8 rather than Fig. 5 or Fig. 10. If this is repeated several times, the load distribution will be done much ahead of the time of its computation and this means there is no wasting time or less waiting time between the consecutive jobs. Therefore, if it is possible to assign the order of service at one's convenience, i.e., the service discipline is not restricted to FIFO or LIFO, assigning heavy computational load first can reduce the overall solution finish time.

VI. Conclusion

In this paper, a recursive algorithm to find an efficient strategy for optimal load allocation for three types of bus network with multiple jobs is discussed. It is found that the multi-job scheme is superior in terms of the total solution finish time over a single-job scheme. A problem left open by this research is the case where the network has no control processor and where processors are without front-end processors.

References

[1] R.R. Tenney and N.R. Sandell.Jr., "Detection with distributed sensors," *IEEE Transaction on Aerospace and Electronic Systems*, vol. AES-17, pp. 501–510, July 1981.

[2] C.Y. Chong, E. Tse, and S. Mori, "Distributed estimation in networks," presented at the American Control Conference, San Franciso, 1983.

[3] S.H. Bokhari, *Assignment Problems in Parallel and Distributed Computing*, Boston: Kluwer Academic Publishers, 1987.

[4] H.S. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," *IEEE Transaction on Software Engineering*, vol. SE-3, no. 1, pp. 85–93, Jan. 1977.

[5] J. Du and J.Y.T. Leung, "Complexity of scheduling parallel task systems," *SIAM Journal on Discrete Mathematics*, pp. 473–487, Nov. 1989.

[6] J. Blazewicz, M. Drabowski, and J. Weglarz, "Scheduling multiprocessor tasks to minimize schedule length," *IEEE Transactions on Computers*, vol. C-35, pp. 389–398, May 1986.

[7] W. Zhao, K. Ramamritham, and J.A. Stankovic, "Preemptive scheduling under time and resource constraints," *IEEE Transactions on Computers*, vol. C-36, pp. 949–960, Aug. 1987.

[8] Y.C. Cheng and T.G. Robertazzi, "Distributed computation with communication delays," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 24, no. 6, pp. 700–712, Nov. 1988.

[9] Y.C. Cheng and T.G. Robertazzi, "Distributed computation for tree network with communication delays," *IEEE Transactions on Aerospace and Systems*, vol. 26, no. 3, pp. 511–516, May 1990.

[10] S. Bataineh and T.G. Robertazzi, "Distributed computation for a bus networks with communication delays," *Proceedings of the 1991 Conference on Information Sciences and Systems*, The Johns Hopkins University, Baltimore, pp. 709–714, March 1991.

[11] S. Bataineh and T.G. Robertazzi, "Bus oriented load sharing for a network of sensor driven processors," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 21, no. 5, Sept. 1991.

[12] S. Bataineh and T.G. Robertazzi, "Ultimate performance limits for networks of load sharing processors," *Proceedings of the 1992 Conference on Information Sciences and Systems*, Princeton, NJ, pp. 794–799, March 1992.

[13] S. Bataineh and T.G. Robertazzi, "Closed form solutions for bus and tree networks of processors load sharing a divisible job," *SUNY at Stony Brook College of Engineering and Applied Science Technical Report*, no. 627, May 1992. (Available from T. Robertazzi).

[14] T.G. Robertazzi, "Processor equivalence for load sharing processor daisy chains," *accepted by the IEEE Transactions on Aerospace and Electronic Systems for Oct. 1993 issue.*

[15] J. Sohn and T.G. Robertazzi, "Optimal load sharing for a divisible job on bus network," *Proceedings of the 1993 Conference on Information Sciences and Systems*, The Johns Hopkins University, Baltimore, March 1993.

[16] J. Sohn and T.G. Robertazzi, "Optimal load sharing for a divisible job on bus networks," *SUNY at Stony Brook College of Engineering and Applied Science Technical Report*, Dec. 1992. (Available from T. Robertazzi).

[17] H.J. Kim, G.I. Jee, and J.G. Lee, "Optimal load distribution for tree network processors," submitted for publication.

[18] D. Ghose and V. Mani, "Distributed computation in a linear network: closed form solution and computational techniques," submitted for publication.

[19] D. Ghose, and V. Mani, "Distributed computation with communication delays: Asymptotic performance analysis," submitted for publication.

[20] V. Bharadwaj, D. Ghose, and V. Mani, "Closed form solutions for optimal processing time in distributed single-level tree networks with communication delays," submitted for publication.

[21] V. Bharadwaj, D. Ghose, and V. Mani, "A new strategy of load distribution in a distributed single-level tree network with communication delays," submitted for publication.

[22] V. Bharadwaj, D. Ghose, and V. Mani, "An efficient load distribution strategy for a distributed linear network of processors with communication delays," submitted for publication.

Affiliation of Author

Authors are with the Department of Electrical Engineering, SUNY at Stony Brook, Stony Brook, N.Y. 11794

Acknowledgment

The research in this paper was supported in part by the SDIO/IST and managed by the U.S. Office of Naval Research under grant no. N00014-91-J4063.

Figure Captions

Fig 1. Bus network with a control processor and queueing system.

Fig 2. Timing diagram for bus network with control processor in single-job scheme.

Fig 3. Timing diagram for bus network with control processor in multi-job scheme.

Fig 4. Timing diagram for bus network with control processor in multi-job scheme when $T_{f_1} \geq ZT_{cm_1} + ZT_{cm_2}$.

Fig 5. Timing diagram for bus network with control processor in multi-job scheme when $T_{f_1} < ZT_{cm_1} + ZT_{cm_2}$.

Fig 6. Transmission timing diagram for bus network with control processor in single-job scheme and in multi-job scheme.

Fig 7. Bus network without control processor.

Fig 8. Timing diagram for bus network with front-end processors in multi-job scheme when $T_{f_1} \geq (1-\alpha_1)ZT_{cm_1} + (1-\beta_1)ZT_{cm_2}$.

Fig 9. Transmission timing diagram for bus network with front-end processors in single-job scheme and in multi-job scheme.

Fig 10. Timing diagram for bus network with front-end processors in multi-job scheme when $T_{f_1} < (1-\alpha_1)ZT_{cm_1} + (1-\beta_1)ZT_{cm_2}$.

Fig 11. Timing diagram for bus network without control processor, processors without front-end processors in multi-job scheme.

Fig 12. Total solution time when all T_{cp}'s= 6.

Fig 13. Total solution time for network with control processor when $T_{cp} = 10$ for the first five jobs and $T_{cp} = 3$ for the last five jobs and vice versa.

Fig 14. Total solution time for network with front-end processors when $T_{cp} = 10$ for the first five jobs and $T_{cp} = 3$ for the last five jobs and vice versa.

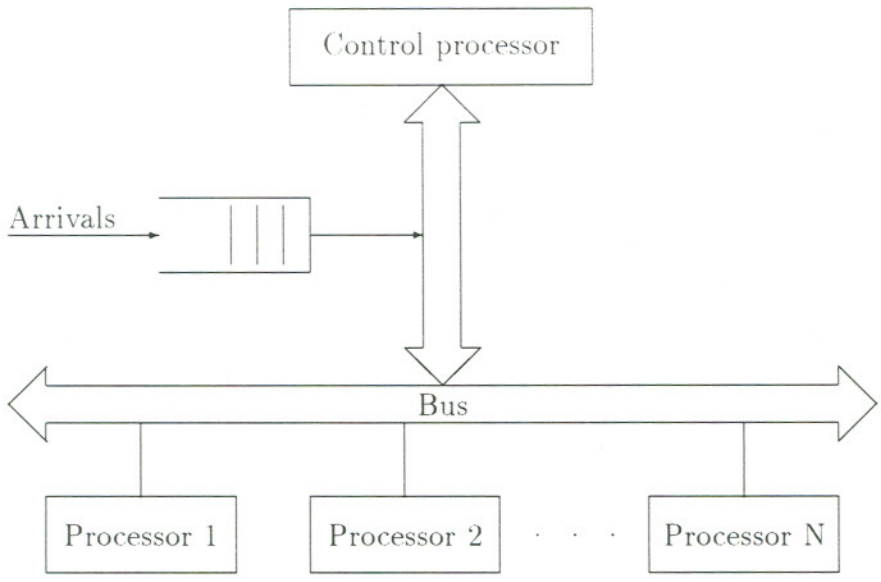

Figure 1. Bus network with a control processor and queueing system.

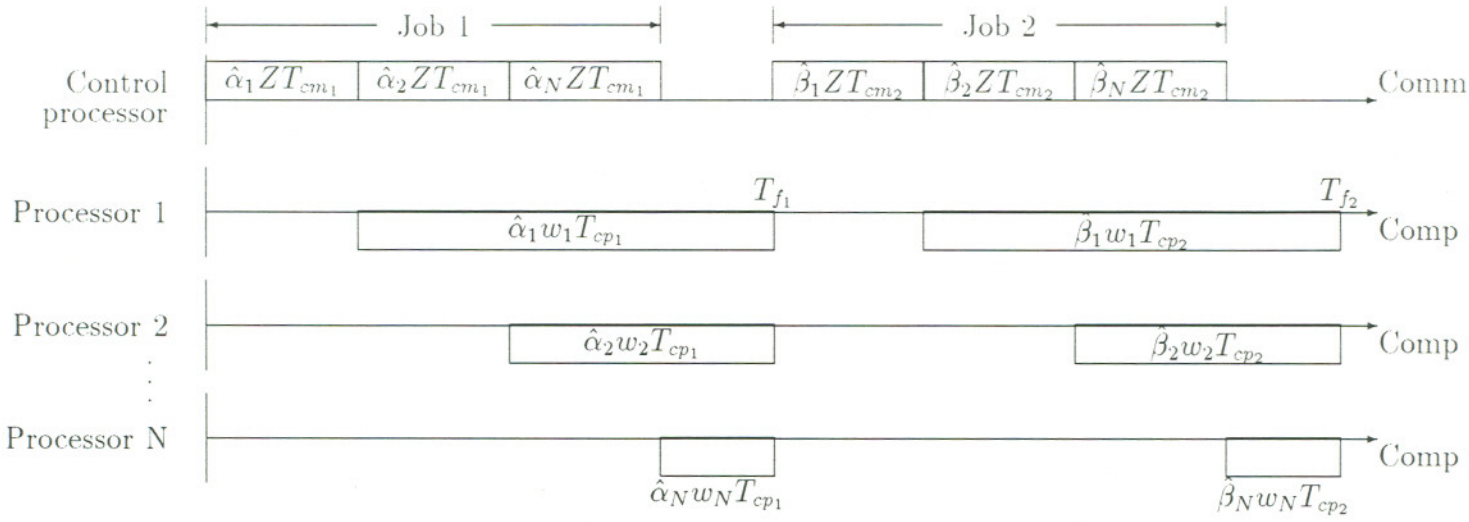

Figure 2. Timing diagram for bus network with control processor in single-job scheme.

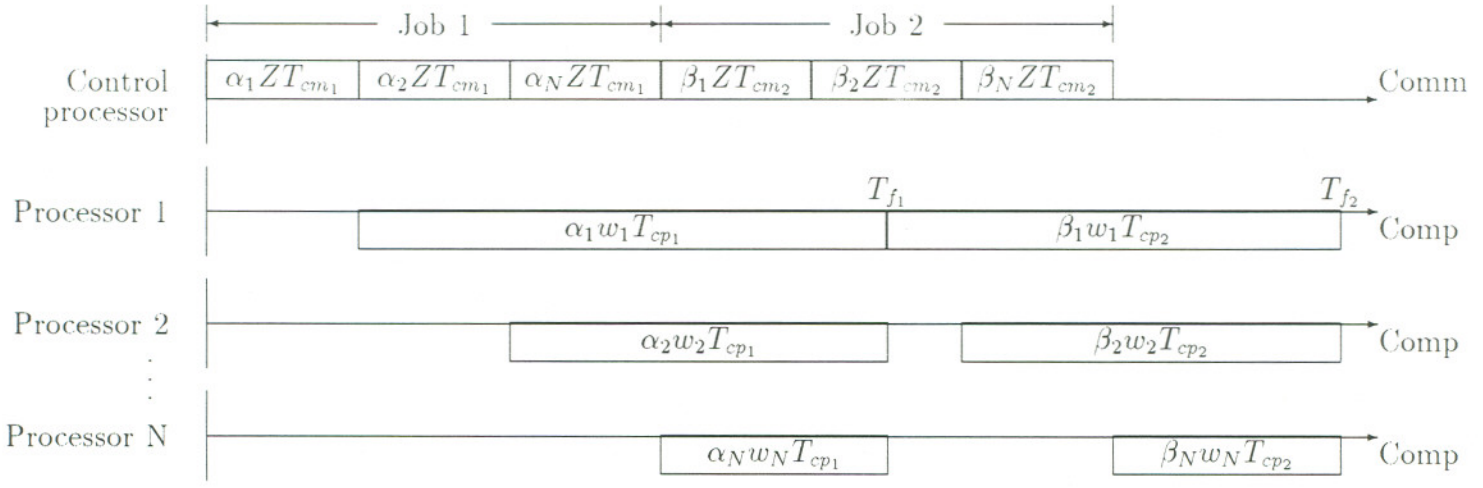

Figure 3. Timing diagram for bus network with control processor in multi-job scheme.

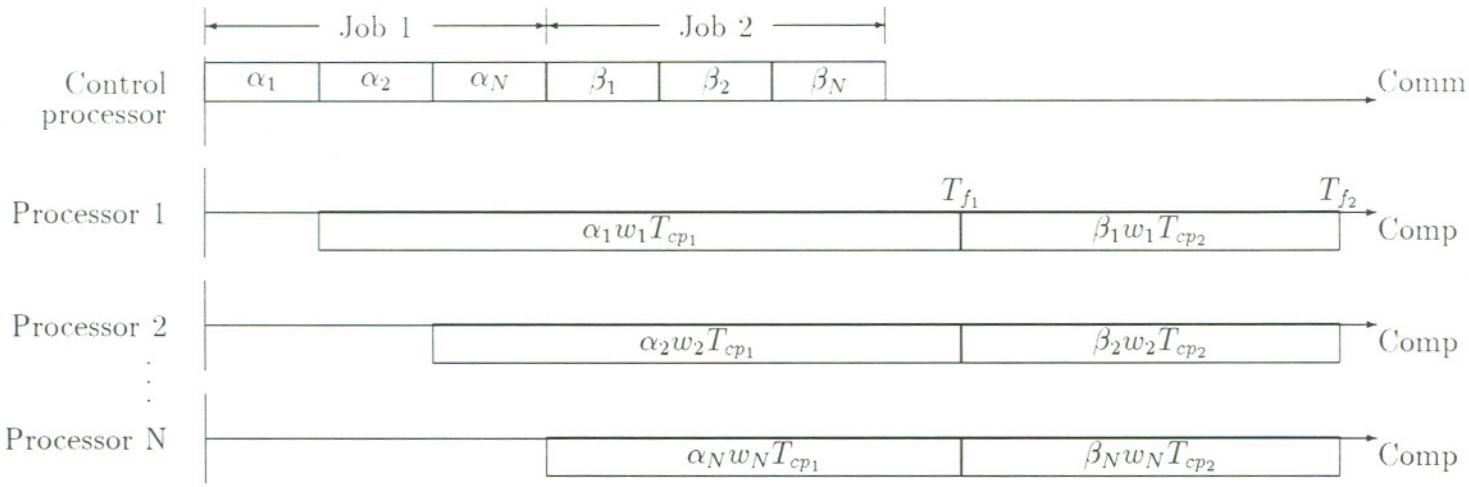

Figure 4. Timing diagram for bus network with control processor in multi-job scheme when $T_{f_1} \geq ZT_{cm_1} + ZT_{cm_2}$.

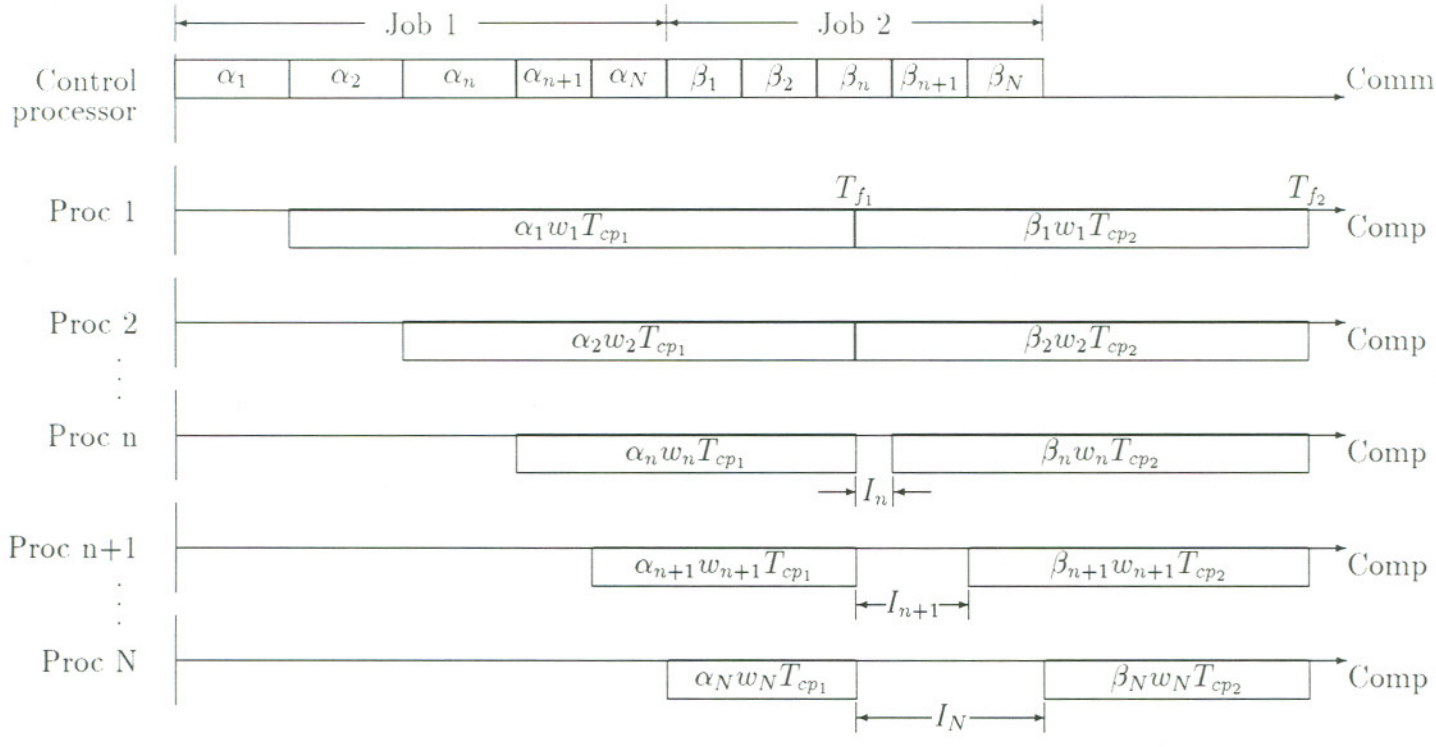

Figure 5. Timing diagram for bus network with control processor in multi-job scheme when $T_{f_1} < ZT_{cm_1} + ZT_{cm_2}$.

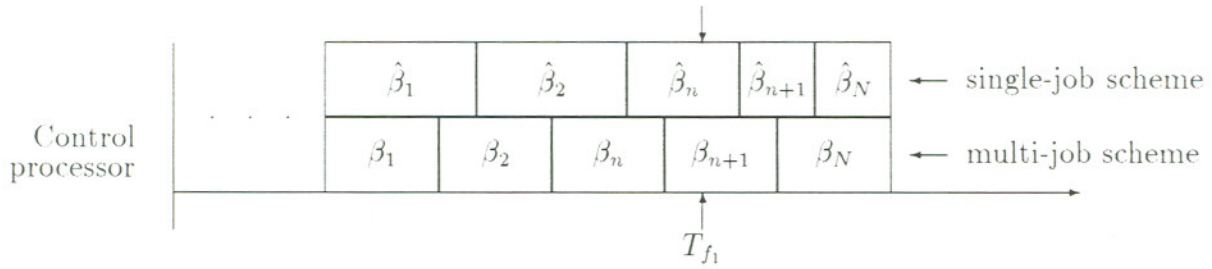

Figure 6. Transmission timing diagram for bus network with control processor in single-job scheme and in multi-job scheme.

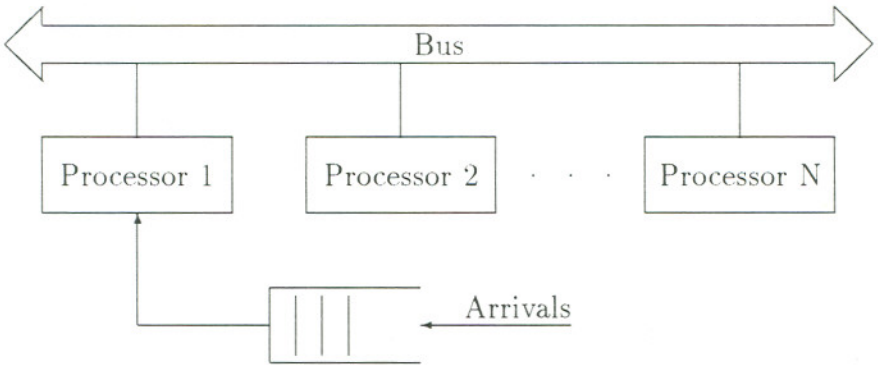

Figure 7. Bus network without control processor.

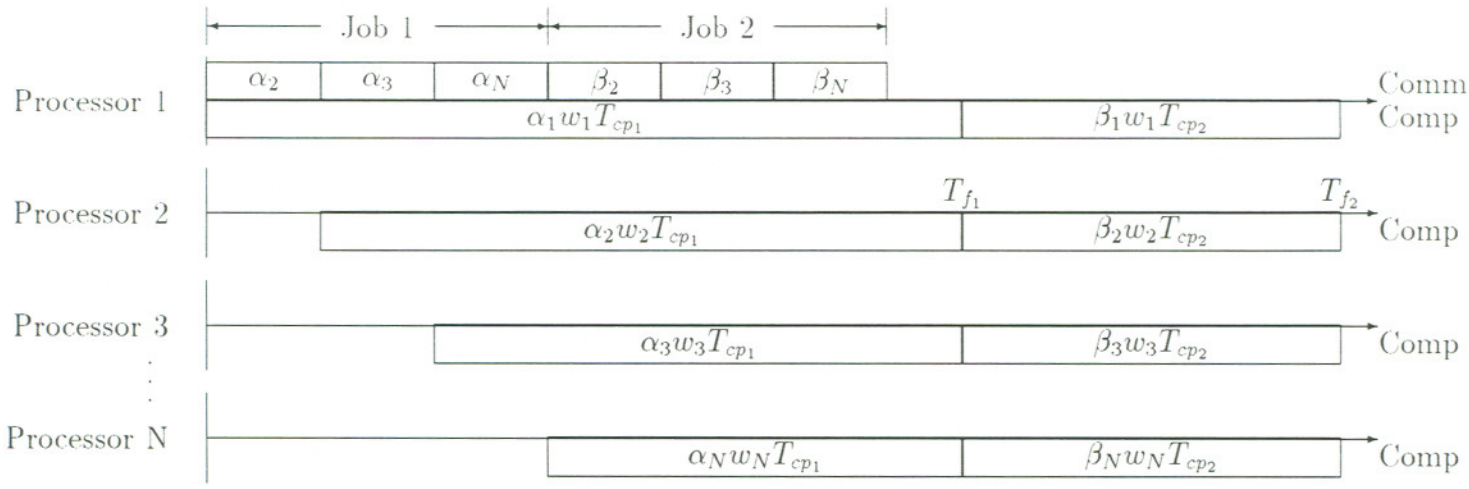

Figure 8. Timing diagram for bus network with front-end processors in multi-job scheme when $T_{f_1} \geq (1-\alpha_1)ZT_{cm_1} + (1-\beta_1)ZT_{cm_2}$.

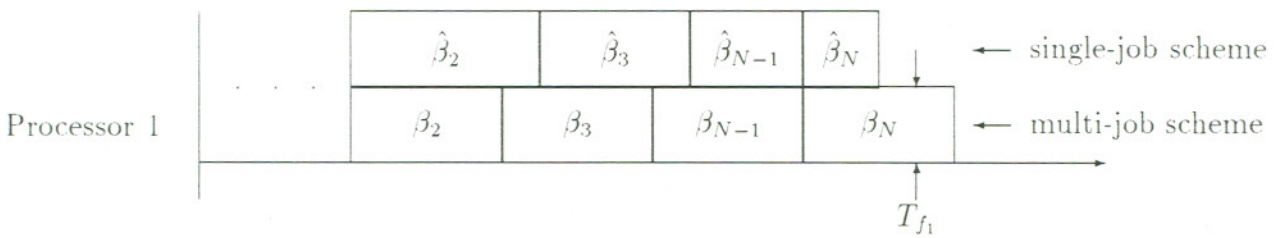

Figure 9. Transmission timing diagram for bus network with front-end processors in single-job scheme and in multi-job scheme.

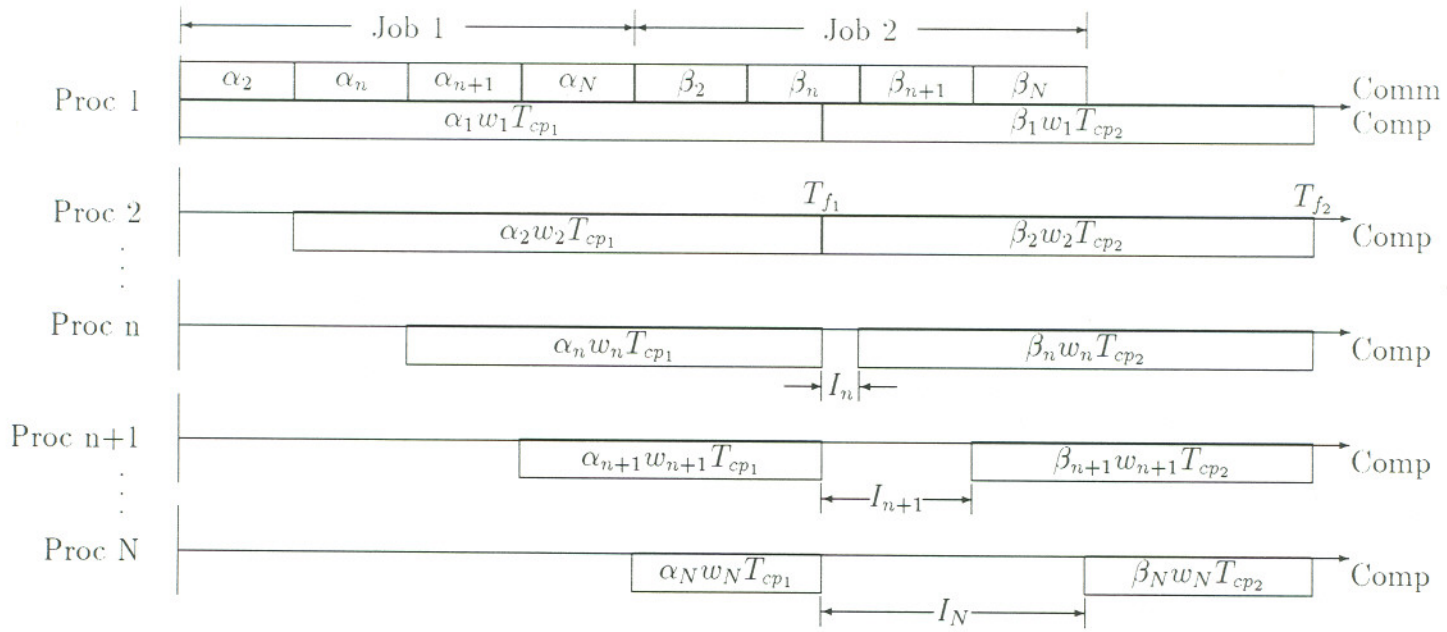

Figure 10. Timing diagram for bus network with front-end processors in multi-job scheme when $T_{f_1} < (1-\alpha_1)ZT_{cm_1} + (1-\beta_1)ZT_{cm_2}$.

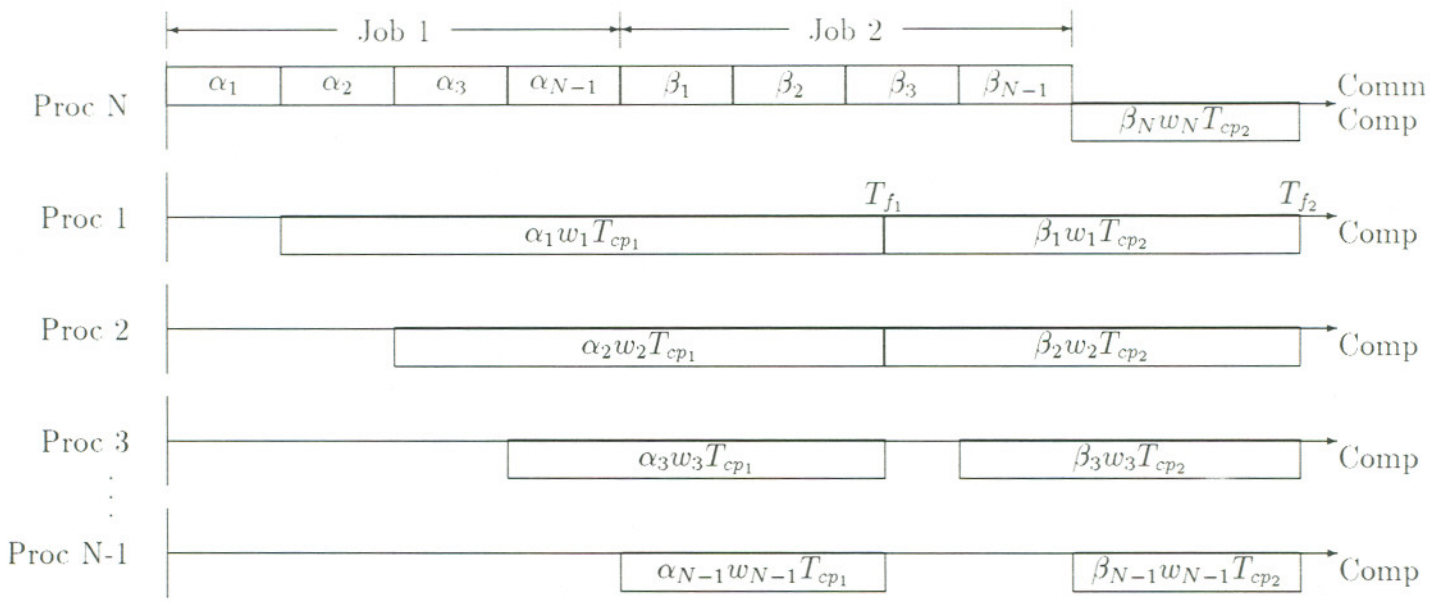

Figure 11. Timing diagram for bus network without control processor, processors without front-end processors in multi-job scheme

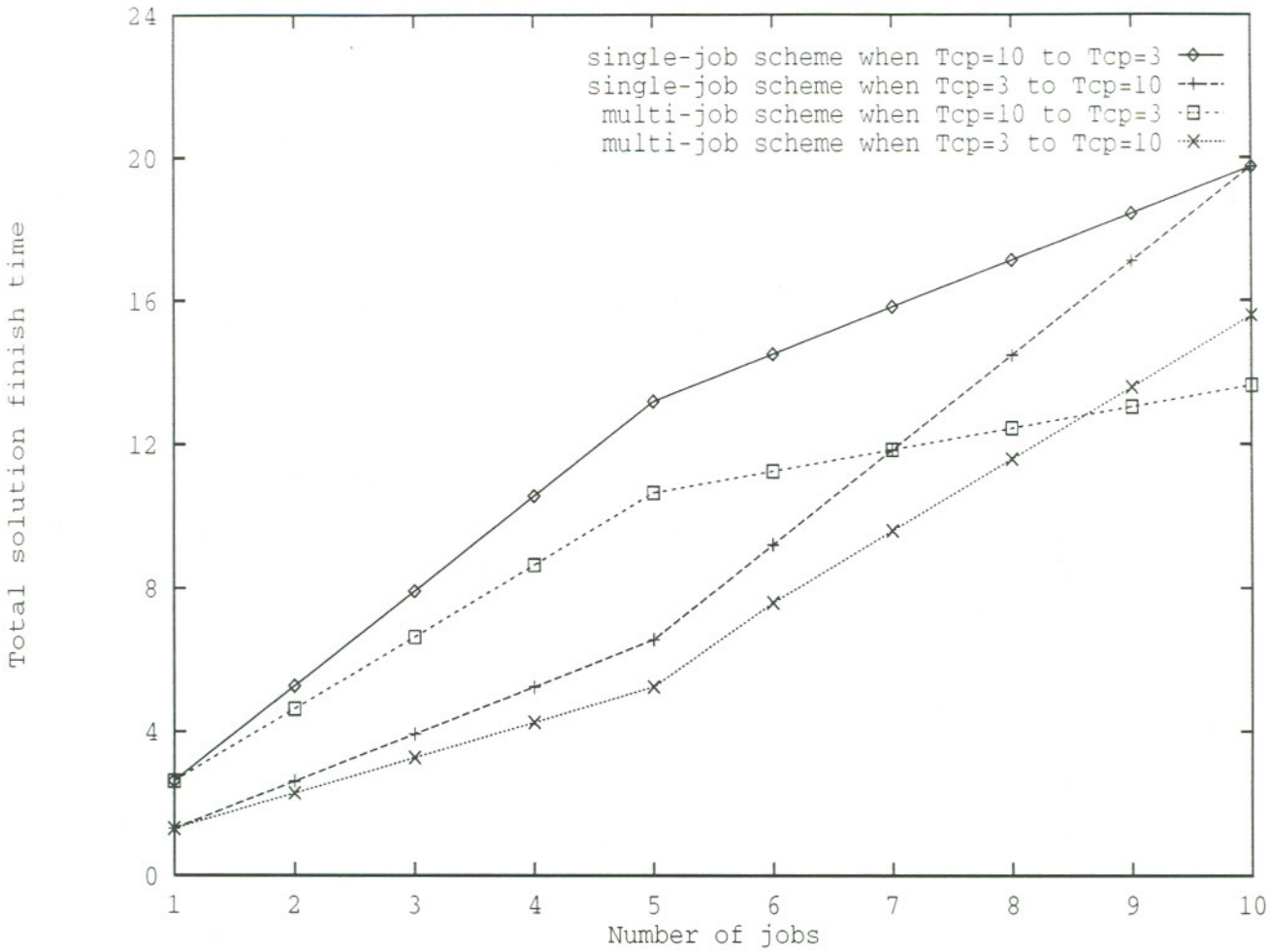

Figure 13. Total solution time for network with control processor when $T_{cp} = 10$ for the first five jobs and $T_{cp} = 3$ for the last five jobs and vice versa.

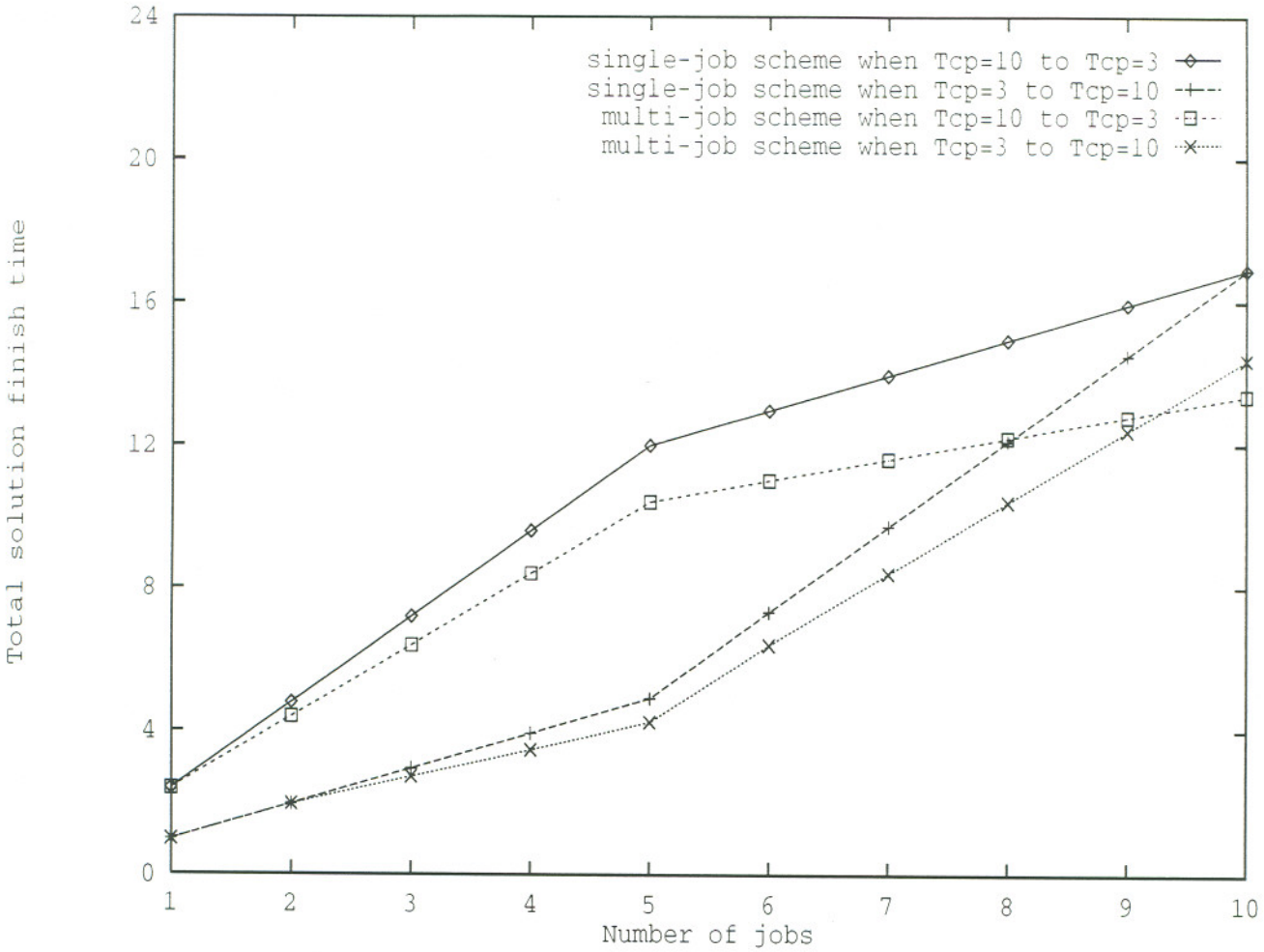

Figure 14. Total solution time for network with front-end processors when $T_{cp} = 10$ for the first five jobs and $T_{cp} = 3$ for the last five jobs and vice versa.