논문-05-10-1-09

Parallel Video Processing Using Divisible Load Scheduling Paradigm

S. Suresh[a)‡], V. Mani[a)], S. N. Omkar[a)], and H.J. Kim[b)]

Abstract

The problem of video scheduling is analyzed in the framework of divisible load scheduling. A divisible load can be divided into any number of fractions (parts) and can be processed/computed independently on the processors in a distributed computing system/network, as there are no precedence relationships. In the video scheduling, a frame can be split into any number of fractions (tiles) and can be processed independently on the processors in the network, and then the results are collected to recompose the single processed frame. The divisible load arrives at one of the processors in the network (root processor) and the results of the computation are collected and stored in the same processor. In this problem communication delay plays an important role. Communication delay is the time to send/distribute the load fractions to other processors in the network, and the time to collect the results of computation from other processors by the root processors. The objective in this scheduling problem is that of obtaining the load fractions assigned to each processor in the network such that the processing time of the entire load is a minimum. We derive closed-form expression for the processing time by taking into consideration the communication delay in the load distribution process and the communication delay in the result collection process. Using this closed-form expression, we also obtain the optimal number of processors that are required to solve this scheduling problem. This scheduling problem is formulated as a linear programming problem and its solution using neural network is also presented. Numerical examples are presented for ease of understanding.

Keywords: Parallel computing, distributed computing system, divisible load scheduling, video scheduling, neural network.

I. Introduction

In this paper, we address the problem of scheduling a single processing load that originates at one of the processors, in a distributed computing network. This processing load is considered to be divisible. Divisible load has the property that all the elements in the load require the same type of processing. So, a divisible load can be divided into any number of fractions, and can be processed independently on the processors in the network, as there is no precedence relationship.

Digital video processing in multimedia applications requires considerable amount of computation. In a frame by frame processing of a digital video, a frame originates at a processor in the network. This frame can be split into tiles and can be processed independently in parallel on the available processors in a distributed computing network. Once the tiles are processed in the processors, the results of individual tiles can be collected at the originating processor to recompose the single processed frame [1]. In recent years, there is a considerable amount of interest in scheduling divisible loads in distributed computing systems [2-4]. In these studies the communication delay (i.e.,, time to communicate a load fraction to a processor) is taken into account in the scheduling problem. The initial application of divisible load scheduling was motivated by the objective of integrating the communication and computation in distributed sensor networks [5]. In digital video processing, the frame can be considered as a divisible load (split into tiles) and can be

a) Department of Aerospace Engineering, Indian Institute of Science, Bangalore, India.
b) Department of Control and Instrumentation Engineering Kangwon National University, Chunchon 200-701, Korea.
※ This paper was in part supported by the Media Services Research Center (MSRC-ITRC), Ministry of Information and Communications, Korea.

processed independently in the processors in the network. Hence, the studies on divisible load scheduling can be used for this digital video processing problem with modifications.

First, we will briefly describe the divisible load scheduling problem and then show its relationship to digital video processing, and the modifications necessary to use the divisible load scheduling methodology.

In divisible load scheduling problem the processing load (divisible) originates at one of the processors in the network. The load originating (root) processor may or may not participate in the computation process. The root processor partitions the load into many fractions, keeps one of the fractions for itself to compute, and sends the remaining fractions to other processors in the network. The objective in this problem is that of finding the load fractions assigned to each processor in the network, so that the processing time of the entire processing load is a minimum. This problem of scheduling divisible loads with communication delays is addressed in the context of distributed sensor networks [5] for a linear network of processors. The methodology from [5], is extended to a single-level tree network and a bus network in [6, 7]. In the case of a single-level tree network, a closed-form expression for the processing time is derived in [8-10]. Using the closed-form expression for processing time, some important results on optimal sequence and optimal arrangement of processors and links in the network is presented in [8].

In divisible load scheduling studies it is assumed that the results of computation should be stored locally. Or in other words, the time to gather these results by the originating processor is not considered. Hence, in these studies, the load fractions are obtained by assuming that all the processors involved in the computation process must stop computing at the same time instant. This assumption has been shown to be a necessary and sufficient condition to obtain the optimal processing time, for the case of linear networks using the concept of processor equivalence in [11] and for a bus network in [12]. However, it has been rigorously proved that this condition is true only in a restricted sense [2] for the case of a single level tree network.

In the digital video processing problem, the divisible processing load (a single video frame) originates at one of the processors in the network. This load originating processor divides the load into many fractions and sends these fractions to other processors in the network for processing/computation so that these fractions can be processed in parallel. After processing, the results of the computation (of these load fractions) from the processors in the network are collected by the originating processor to recompose the single processed video frame.

In the divisible load scheduling problem, there is only one communication delay. The communication delay to send/distribute the load fractions to other processors in the network. But, in the video processing there are two communication delays. The communication delay to send the load fractions to the processors in the network, and the communication delay in collecting the results of computation from the other processors in the network by the originating processor. In divisible load scheduling, the communication delay to collect the results of computation is not considered, and hence, the load fractions assigned to the processors are obtained, based on the assumption, that all the processors participating in the computation process, should stop computing at the same time instant. In video processing problem the results of the computation are collected in the originating processor, and hence the condition that all the processors stop computing at the same time instant is not valid.

In the video scheduling problem, the load fractions assigned to the processors in the network, are obtained by maximum utilization of the communication channels and the processors [1]. In this study [1], two scheduling algorithms namely Parallel Recursive (PR) and Parallel Interlaced (PI) are presented for frame by frame processing. Also in this study [1], the load distribution time and the results collection times are considered as read and write times of respective tiles of a single video frame.

This paper is organized as follows. An overview of related works in divisible load theory is given in Section II. In Section III, we describe the problem, the architecture, the recursive solution procedure, and its relationship with divisible load scheduling for the two

scheduling algorithms (PR and PI). We formulate these two scheduling algorithms as a linear programming problem in Section IV, and in Section V, we propose neural network method to solve the linear programming formulation. Section VI concludes the paper.

Ⅱ. Related Work

The problem of scheduling divisible loads in dis tributed computing systems started in 1988 [5] and has generated considerable amount of interest among researchers. Initially, the research in divisible load scheduling evolved from the processing of large volume of data that arrive in a distributed sensor network. In the earlier studies, the architecture of distributed computing systems considered is a linear, bus and single level tree network. Also in these studies, the processors in the network may or may not be equipped with front ends (communication co-processor). When a processor is equipped with a front end, it can simultaneously compute its load fraction and communicate the load fraction to other processors. The objective in divisible load scheduling is to partition the entire processing load in to smaller load fractions and process them independ ently in parallel on the available processors in the network. This involves mathematical modelling of the network parameters, speed parameters of the processors in the network and the size of the load fractions. The mathematical model describing the time to communicate a load fraction over a link and the time to compute the load fraction in a processor are assumed to be linear. In other words, the mathematical models adopted is such that the time to communicate a load fraction over a link is proportional to the size of the load fraction, and the time to compute a load fraction is proportional to the size of the load fraction. In the earlier studies in divisible load scheduling, the objective is to find the optimal load fractions assigned to each processor in the system such that the processing time of the entire load is a minimum. In order to achieve this objective, many researchers proposed many scheduling strategies for this scheduling problem in different architectures [23, 24].

Another important aspect studied in divisible load scheduling literature is asymptotic performance an- alysis of the load distribution process. This study is useful in obtaining the cost benefit analysis and trade off relation-ships between the cost of enhancing the capability of each element in an existing network and the corres-ponding reduction in processing time. The asymptotic performance results on linear, tree and bus networks are given in [11, 25]. The asymptotic performance results in two dimensional network and hypercube network are given in [26, 27], respectively. The problem of scheduling divisible load in a three-dimensional mesh of processors is studied in [28]. The parallel time and speedup for processing divisible load in a linear array, mesh and hypercube architecture are presented in [29, 30].

Research works in divisible load scheduling were extended to various scenarios with practical constraints, such as fault-tolerance constraints [31], time-varying channels [32], and optimizing computing costs [33]. The processors in the network may not have sufficient memory to store the load fractions assigned to that processor. This problem of limited memory/buffer is discussed in [34, 35]. Another important practical aspect is the availability of processors in the network. In some situations the processors in the network will be available for processing only after a specific time (release time). So, the scheduling of divisible loads with arbitrary proc-essor release times is presented in [36] and the combined effect of buffer restriction and processor release times is discussed [37]. A new communication model in which the divisibility property is further exploited known as "non blocking mode of communi-cation" is presented [38]. In all the earlier studies, the process of communicating the load fractions is the "blocking mode of communi-cation." In the blocking mode, the processor will start the computation process only after the front end has received all the load fractions assigned to that processor. In the non-blocking mode, the processor will start the communication while its front-end is receiving the pro-cessing load fraction assigned to it. Optimal sequencing and arrangement of processors in the network using non-blocking mode of communication is studied in [39].

From a practical point of view, the applicability of divisible load scheduling methodologies for large matrix-vector products and trade-off study are presented in [40, 41].

Another important aspect considered in scheduling divisible loads is the effect of communication and computation overhead components (start-up delays) that could penalize the performance of the system in addition to communication and computation delays. Results on this aspect are presented in [42-44]. With the inclusion of the start-up delays the communication model is not a linear model. This is the model used in [1] for video scheduling problem. A linear programming model in the context of parallel programming is discussed as a partitioning technique problem for large grained parallelism in [45] and this study has many similarities with divisible load scheduling problem. A generalized linear programming approach to optimal divisible load scheduling is presented in [46]. The advantages and the reasons for use of divisible load theory are given in [47].

Ⅲ. Problem Statement

Consider a bus network consist of $m+1$ processors connected to a common bus as shown in Figure 1. The processor p_0 is referred as a control processor in which the processing load is resident and the final results of the computation is expected to be stored. We assume that the control processor will not participate in computation process. So the control processor only distributes the load fractions to other processors in the network, and collects the results of computation from other processors in the network. The processors p_1, p_2 $\cdots$ p_m are called child processors. We shall call this network as m- processor bus network meaning that m-child processors connected to a control processor. The communication between the control processor and child processors take place through a common bus. The following assumptions are made on a bus network:

(1) The bus network is *homogeneous* in the sense that all the child processors are identical which means that they have the same processing speed or computation speed.
(2) At any given time-instant, only one child processor can communicate with control processor.
(3) The child processor can either receive the load fractions from the control processor or send the results to the control processor. This model is called one-way commu- nication model or client-server model.
(4) No communication occurs between the child processors.

The control processor divides the processing load into m fractions (α_1, α_2 $\cdots$ α_m) and distributes these load fractions (α_1, α_2 $\cdots$ α_m) to the child processors (p_1, p_2 $\cdots$ p_m) one after another for computation. The child processors start computation immediately after receiving their respective load fractions. The child processors (p_1, p_2 $\cdots$ p_m) after computing their respective load fractions send the results of computation to the control processor one after another. The load fractions communicated to the child processors and results of computations collected from the child processors may be different for different

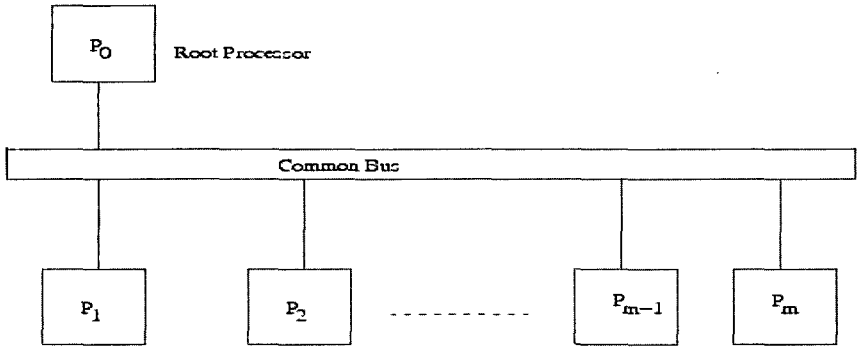

Figure 1. Schematic Diagram of Bus network.

applications. The results of computations may be greater than or less than or equal to the load fractions communicated to the child processors. Let the results of computation collected from any child processor p_i is K times the load fraction (α_i) sent to the child processor p_i, i.e., if α_1, $\alpha_2 \cdots \alpha_m$ be load fractions communicated to the child processors p_1, $p_2 \cdots p_m$ then $K\alpha_1$, $K\alpha_2 \cdots K\alpha_m$ are the results of computation received from child processors p_1, $p_2 \cdots p_m$. Hence, the time to communicate the load fractions to the child processors by the control processor may be different from the time to communicate the results by the child processors to the control processor. Based on the value of K, we have the following three cases:

- Load fractions communicated and results of computation are of same size (K = 1).
- The load fraction communicated to the child processors is greater than the results of computation communicated by the child processors (K < 1).
- The load fraction communicated to the child processors is less than the results of computation communicated by the child processors (K > 1).

1. Definitions:

In this paper, we follow the standard notations and definitions used in the divisible load scheduling literature [2].

- Load Distribution defined as an m-tuple (α_1, $\alpha_2 \cdots \alpha_m$) such that $0 \le \alpha_1 \le 1$. The equation $\sum_{i=1}^{m} \alpha_i = 1$ is the normalization equation, and the space of all possible load distribution is denoted as Γ.
- Processing Time denoted by $T(m)$, is the difference between the time at which the control processor start the load distributions process and the time at which the control processor receives all the results from the child processors.

2. Notations:

- α_1: Fraction of the processing load assigned to processor p_1.
- w_1: ratio of the time taken by processor pi to compute a given load to the time taken by a standard processor to compute the same load.
- z: ratio of the time taken by communication bus to communicate a given load to the time taken by a standard bus to communicate the load.
- T_{cp}: time taken to process a unit load by the standard processor.
- T_{cm}: time taken to communicate a unit load by the standard communication bus.
- σ : ratio of communication time to processing time for a given load in a standard communication bus and processor (i.e., $\sigma = T_{cm} / T_{cp}$).

Standard processor or standard link is any processor or link which is used as a reference. From the above definitions, we see that $\alpha_1 w_i T_{cp}$ is the time to process/compute the load fraction α_1 of the entire processing load, by the processor p_1. In the same way, $\alpha_1 z T_{cm}$ is the time to communicate the load fraction α_1 of the entire processing load to the processor p_1, over the common bus and $K\alpha_1 z T_{cm}$ is the time to communicate the results of the load fractions α_1 to the originating/control processor by the processor p_1, over the common bus.

In this paper, we consider two video scheduling algorithms referred to as parallel interlaced (PI) and parallel recursive (PR) discussed in [1]. Both these two algorithms discussed in [1] can be easily rep- resented in the framework of divisible load sched- uling problem by including the time taken to collect the results by the control processor. The difference between these two algorithms is in the sequence of results collection by the control processor. In PI, the sequence of load distribution is p_1, $p_2 \cdots p_m$, and the sequence of results collection is p_1, $p_2 \cdots p_m$. In the PR, the sequence of load distribution is p_1, $p_2 \cdots p_m$, and the sequence of results collection is p_m, $p_{m-1} \cdots p_1$.

The objective in the video scheduling is to obtain the load fractions assigned to each processor in the network such that the processing time of the entire load is a

minimum, and with the constraint that the utilization the processors and the communication bus is a maximum.

3. Parallel Interlaced Scheduling (PI)

Let p_1, p_2 ⋯ p_m be the sequence of load distribution and the sequence of results collection by the control processor. This means that, the control processor divides the total processing load into m parts and distributes the load fractions α_1, α_2 ⋯ α_m to the child processors p_1, p_2 ⋯ p_m in the sequence p_1, p_2 ⋯ p_m one after another. Each processor in the network starts computing immediately upon receiving its load fraction and continues to do so until its load fraction is exhausted. The control processor after distributing the load fractions to the processor, starts collecting the results of the computation $K\alpha_1$, $K\alpha_2$ ⋯ $K\alpha_m$ from the child processors p_1, p_2 ⋯ p_m in the same sequence of load distribution.

In divisible load scheduling literature, timing diagram (a pictorial representation) is the usual way of representing the load distribution and computation process. In the timing diagram the communication is shown above the time axis and the computation is shown below the time axis for all the processors in the network. For the video scheduling problem also, we follow the timing diagram representation. The timing diagram for the PI scheduling is shown in Figure 2. In this timing diagram we have included the time taken by the control processor to collect the results from the child processors in the network.

We consider the case of a homogeneous network, where all the processors are identical: i.e., $w_i = w$. From this timing diagram, the recursive load dis- tribution equations for PI scheduling, are obtained as

$$\alpha_i w T_{cp} = \alpha_{i+1} z T_{cm} + \alpha_{i+1} w T_{cp} - k\alpha_i z T_{cm}, \quad i = 1,2,...,m\text{-}1 \quad (1)$$

Since the load originating at the control processor is assumed to be normalized to a unit load, the fractions of the total load processing load should sum to one.

$$\sum_{j=1}^{m} \alpha_j = 1 \quad (2)$$

This above equation is referred to as normalization equation in divisible load scheduling literature. The above recursive equations (1) for PI scheduling, are obtained from the following condition.

The computation time of the load fraction α_i and the time to communicate the results of computation $K\alpha_1$ to the control processor, by the processor p_1 is equal to the time to communicate the load fraction α_{i+1} to the processor p_{i+1} by the control processor, and the computation time of the load fraction α_{i+1} by the processor p_{i+1}. We also see from equation (1) that these equations along with the normalization equation form a system of m linear equations in m unknowns. These can be solved by exploiting the recursive nature of the equations as follows:

Equation (1) can be rewritten as

$$\alpha_i = \alpha_{i+1} g, \quad i = 1,2,...,m\text{-}1 \quad (3)$$

where

$$g = \frac{wT_{cp} + zT_{cm}}{wT_{cp} + KzT_{cm}} = \frac{1+\gamma}{1+K\gamma} \quad (4)$$

We can express all the α_i, $(i = 1, 2, , m-1)$ in terms of α_m as

$$\alpha_i = \alpha_m \left(g^{m-i}\right) \quad i = 1,2, ...,m\text{-}1 \quad (5)$$

and the value of α_m is obtained from the normalization equation as

$$\alpha_m = \frac{1}{1 + \sum_{j=1}^{m-1} g^j} \quad (6)$$

The load fraction assigned to processor p_1 is α_i and is

$$\alpha_i = \alpha_m g^{m-i} \quad i = 1,2, ..., m\text{-}1 \quad (7)$$

In video scheduling problems the processing time, i.e., $T(m)$, is the time difference between the time at which

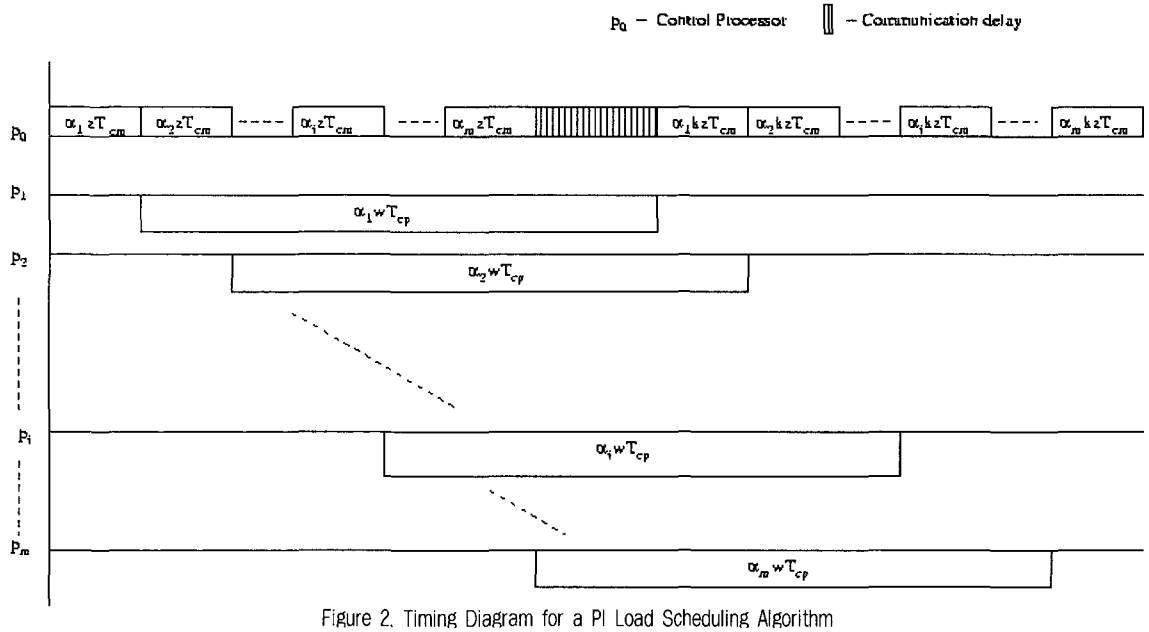

Figure 2. Timing Diagram for a PI Load Scheduling Algorithm

the control processor starts the load distribution process and the time at which the control processor collects the results of computation from all the child processors. Thus, from the timing diagram shown in Figure 2, it can be seen that $T(m)$ is the addition of time to communicate the load fraction α_1 to the child processor p_1 and the processing/computation time of the child processor p_1 and the time to collect the results from all the child processors. Hence,

$$T(m) = \alpha_1 zT_{cm} + \alpha_1 wT_{cp} + \sum_{i=1}^{m} K\alpha_i zT_{cm} \tag{8}$$

The above equation can be rewritten as

$$T(m) = \{\alpha_1(1+\gamma) + K\gamma\} wT_{cp} \tag{9}$$

By substituting the value of load fraction 1 in equation (9), we can obtain the closed-form expression for processing time as

$$T(m) = \left(\frac{g^{m-1}(1+\gamma)}{1 + \sum_{j=1}^{m-1} g^{j}} + K\gamma \right) wT_{cp} \tag{10}$$

and this above equation can be further simplified as

$$T(m) = \left(\frac{g^{m}(1+K\gamma)}{1 + \sum_{j=1}^{m-1} g^{j}} + K\gamma \right) wT_{cp} \tag{11}$$

The expression for processing time T(m) is true for all values of K but it is valid only when the child processors send their results of computation immediately after the computation process is over. Or in other words, the communication bus is available for the child processors to send their results to the control processor immediately after the computation process is finished.

In general, in this video scheduling problem of load distribution and results collection the following two situations will arise.

- Situation 1: The bus is idle for some time duration. In this situation, after communicating the load fractions (α_1, α_2 ··· α_m) to the child processors (p_1, p_2 ··· p_m), the bus waits till the first child processor p_1 completes its computation process and ready to send the results. The idle time of the bus is $\alpha_1 wT_{cp} - \sum_{i=2}^{m} \alpha_i zT_{cm}$ and is shown in

Figure 2. In other words, here the child processor p1 has not completed its com- putation, but the control processor p0has completed the load dis- tribution process to the other child processors. Hence, the bus has to wait till the first processor to complete its computation process, to start the results collection process. Because of the waiting of the bus, the utilization of the bus is not a maximum. The processing time $T(m)$ for this situation is given by equation (11).

Situation 2: The child processors are idle for some time duration. In this situation, the child processors, after completing their computation process waits for the bus to send their results to the control processor. This is due to the fact that the control processor has not completed the load distribution processor the other processors in the network. The idle time of the first child processor is $\sum_{i=2}^{m}\alpha_i zT_{cm}-\alpha_1 wT_{cp}$. In general the idle time of the child processor p_i is $\sum_{j=i+1}^{m}\alpha_j zT_{cm}+\sum_{j=1}^{i-1}K\alpha_j zT_{cm}-\alpha_i wT_{cp}$. Here the child processor p_i has completed its computation process and it can send the results only after the control processor completes the load distribution to all other processors (p_{i+1}, p_{i+2} $\cdots$ p_m) and the results collection process of the preceding processors (p_1, p_2 $\cdots$ p_{i-1}). Note that here because of the idle time of the processors the utilization of the processors is not maximum. In this situation, the processing time given by equation (11) is not valid.

In order to obtain the processing time and to an- alyze the Situation 2, consider a bus network with m_1 child processors. The load distribution sequence is (p_1, p_2 $\cdots$ p_{m1}) and the results collection sequence is also (p_1, p_2 $\cdots$ p_{m1}). The timing diagram for load distribution and results collection process is shown in Figure 3. From timing diagram, we can easily see that the child processors after completing their computation wait for the bus to communicate the results to the control processor. In this case, the processing time $T(m_1)$ can be obtained from the timing diagram as

$$T(m_1)=\sum_{i=1}^{m_1}\alpha_i zT_{cm}+\sum_{j=1}^{m_1}K\alpha_j zT_{cm} \tag{12}$$

By the normalization equation $T(m_1)$ is obtained as

$$T(m_1)=(1+K)\alpha_1 wT_{cp} \tag{13}$$

The processing time $T(m_1)$ is obtained as the sum of processing load distribution time to the child processors and the result collection time by the control processor. Note that the processing time given in equation (8) is different from the processing time given by equation (13). The reason for this is that the child processors wait after completing their computation process for the bus to send the results of computation.

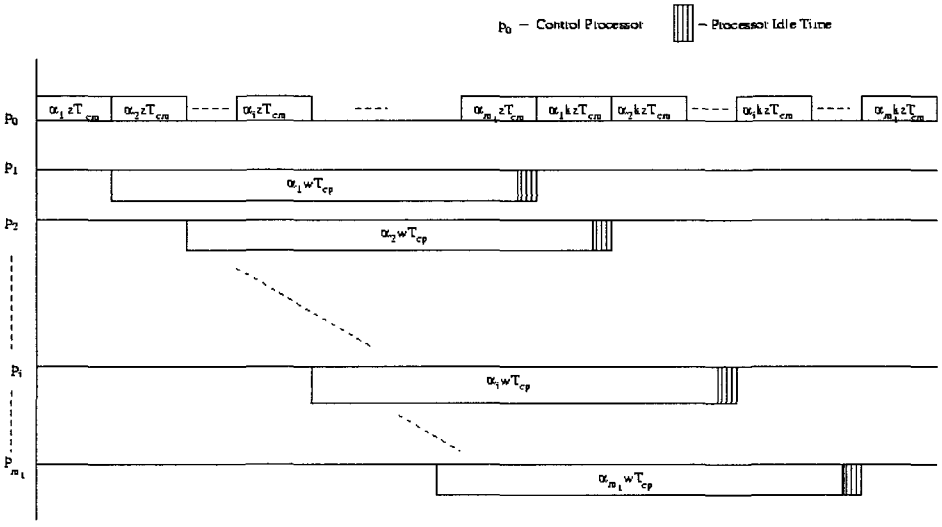

Figure 3. Timing Diagram for a PI Load Scheduling Algorithm with m1 Processors.

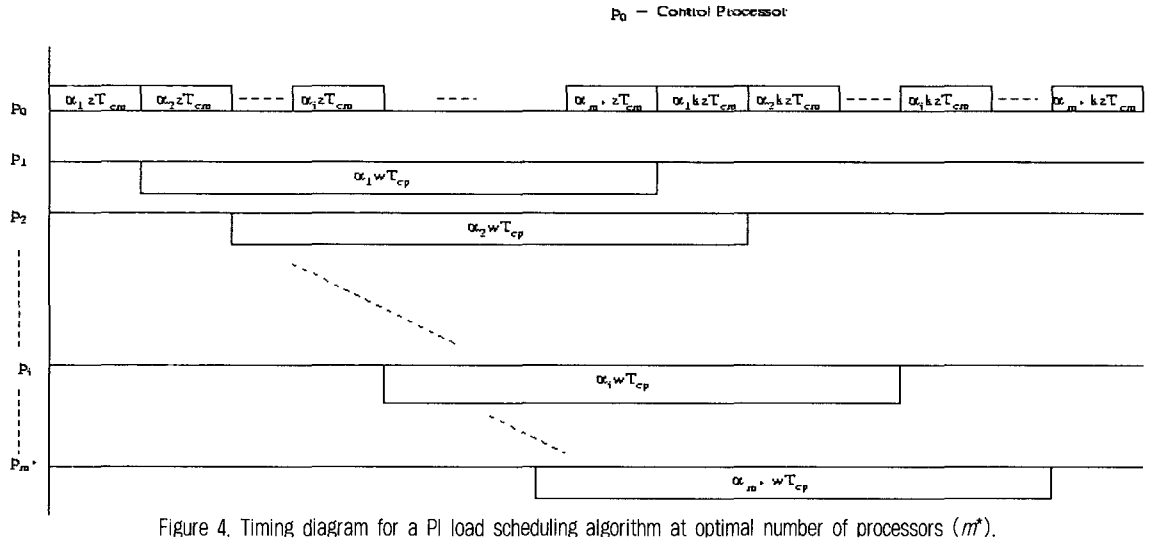

Figure 4. Timing diagram for a PI load scheduling algorithm at optimal number of processors (m^*).

Form the above equation we can see that the processing time is independent of the number of processors m. But, from equation (11) we see that the processing time is a function of the number of processors. From equations (11) and (13), we see that there exist an optimal number of processors (m^*) for a given load distribution and results collection sequence, up to which the processing time will decrease with increase in number of processors and beyond the optimal number of processors, the processing time remains constant. This optimal number of processors (m^*) satisfies the condition that the processing time is minimum and the utilization of the processors and the communication bus is maximum. Now, we will find the closed-form expression for the optimal number of processors.

4. Optimal Number of Processors

From timing diagram shown in Figure 2, we can observe the following: the communication bus is idle for some time duration after distributing the load fractions to the child processors and before start collecting the results from the child processors. This idle time can be reduced by imposing the condition that the computation time of first child processor should be greater than or equal to the sum of load distribution time for child processors p_2, $p_3 \cdots p_m$. The condition is

$$\alpha_1 wT_{cp} \geq \sum_{i=2}^{m} \alpha_i zT_{cm} \tag{14}$$

In general for any child processor p_i, the com- putation time $\alpha_1 wT_{cp}$ should be greater than or equal to the sum of the load distribution time of the successor processors and the results collection time of the preceding processors. Hence the condition is

$$\alpha_i wT_{cp} \geq \sum_{j=i+1}^{m} \alpha_j zT_{cm} + \sum_{j=1}^{i-1} K\alpha_j zT_{cm},\ i = 2,3,\Lambda\ m \tag{15}$$

If this condition is not satisfied, then the first child processor (p_1) after completing its computation process will wait for some time to send the results of computation to the control processor. The reason for this is that the control processor is busy in sending the load fractions to the child processors. The control processor will start collecting the results from the child processors only after completing the load distribution process. The processor waiting times are shown in Figure 3. On the other hand, if the computation time ($\alpha_1 wT_{cp}$) of the first child processor is greater than the sum of all load distribution time of its successors (i.e., the condition is satisfied) then there exists an idle time in the communication channel.

This idle time is shown in Figure 2. This idle time can be reduced by increasing number of processors.

Thus, there exists an optimal number of processors for a given load distribution and results collection sequence and is determined by the effective utilization of communication bus. Effectively, we mean that the idle time of the communication channel is a minimum. Let m^* be the optimal number of processors, such that the idle time in the communication bus is minimum. The timing diagram for the load distributions sequence (p_1, p_2 $\cdots$ p_{m*}) and results collection sequence (p_1, p_2 $\cdots$ p_{m*}) at optimal number of processors is shown in Figure 4. At m^*, the communication bus is available for all the child processors immediately after their respective computation process without being idle. The computation time for the child processor p_1 is equal to the sum of load distribution time for all its successor child processors. At m^* number of processors, the condition given in equation (15) is satisfied with strict equality. At this condition, the load assigned to the first child processor p_1 is

$$\alpha_1 w T_{cp} = (1-\alpha_1) z T_{cm} \tag{16}$$

The above equation can be further simplified as

$$\alpha_1 = \frac{\gamma}{1+\gamma} \tag{17}$$

By substituting the value of 1 from equation (7) in equation (17), we get the following condition

$$\frac{g^{m-1}}{1+\sum_{j=1}^{m-1} g^j} = \frac{\gamma}{1+\gamma} \tag{18}$$

This above equation is reduced to

$$g^{m^*-1} = \gamma\left(1+g+\Lambda+g^{m^*-2}\right) \tag{19}$$

From the timing diagram shown in Figure 4, the load processing time is

$$T(m^*) = \sum_{i=1}^{m^*} \alpha_i z T_{cm} + \sum_{i=1}^{m^*} K\alpha_i z T_{cm} \tag{20}$$

The above equation is simplified as

$$T(m^*) = (1+K)\gamma w T_{cp} \tag{21}$$

From the above expression we see that the optimal number of processors depends on the value of network parameters and K. Using the equation (19) we will find closed-form expression for optimal number of processors at different values of K.

Case (i): Distribution time equal is to result collection time (k=1).

Following our analysis in the previous section, g = 1 when K = 1. Substituting the values of g and K in equation (6), we obtain the load fraction m as

$$\alpha_m = \frac{1}{m} \tag{22}$$

and from equation (7), we can see that since g = 1, the values of load fraction i assigned to the processor p_i is,

$$\alpha_i = g^{m-i}\alpha_m = \frac{1}{m}, \quad i = 1,2,\Lambda\ m-1 \tag{23}$$

We see that the load fractions assigned to the processors in the network are of the same size ($\alpha_i = 1/m$). Hence, processing time of the entire load is given as

$$T(m) = \frac{1+\gamma(1+m)}{m} w T_{cp} \tag{24}$$

This equation (24) is valid only when the condition given in equation (15) is satisfied. So this expression for processing time is valid up to the optimal number of processors. The optimal number of processors is obtained by substituting the value of g in equation (19),

$$1 = \gamma(m^* - 1) \tag{25}$$

The above equation can be rewritten as,

$$m^* = 1 + \frac{1}{\gamma} \tag{26}$$

Since the optimal number of processor is an integer value, we can either use ceiling function $\lceil m^* \rceil$ (to round off to nearest integer that is not smaller than the m^*) or $\lfloor m^* \rfloor$, i.e., floor function (to give largest integer less than or equal to the m^*). If we use the floor function for the optimal number of processors, then the load fractions assigned to the child processors are less than the optimal load fractions given in equation (15) and the communication bus is idle for some time. If we use ceil function for optimal number of processors, then the load fractions assigned to the child processors are greater than the optimal load fractions given in equation (15) and the child processors are idle for some time. We know that the processing time will decrease if the communication bus is utilized effectively and the processing time will not increase if there is an idle time in the child processors. Therefore, the optimal number of processors is obtained using ceil function as,

$$m^* = \left\lceil 1 + \frac{1}{\gamma} \right\rceil \tag{27}$$

The processing time of the entire processing load with optimal number of processors ($m = m^*$) is obtained from equation (12) as

$$T(m^*) = 2zT_{cm} \tag{28}$$

Case (i): Distribution time is greater than the results collection time ($K < 1$).

From the equation (6), the expression for load fraction α_m can be rewritten as,

$$\alpha_m = \frac{g-1}{g^m - 1} \tag{29}$$

and any load fraction α_i assigned to processor p_i is,

$$\alpha_i = \frac{g-1}{g^m - 1} g^{m-i} \quad i = 1,2, .., m\text{-}1 \tag{30}$$

The processing time for the entire processing load is given as

$$T(m) = \left(\frac{g-1}{g^m - 1} g^m (1 + K\gamma) + K\gamma \right) wT_{cp} \tag{31}$$

In order to obtain closed-form expression for optimal number of processors, the equation (19) can be simplified as

$$g^{m-1} = \gamma \left(\frac{g^{m-1} - 1}{g - 1} \right) \tag{32}$$

The above equation can be further simplified as

$$m^* = \left\lceil \frac{-\log(K)}{\log(g)} \right\rceil \tag{33}$$

Since k is less than 1, the term $-\log(k)$ is always positive and g is greater than one. Hence, the optimal number of processors is always a positive quantity. The processing time of the entire processing load with optimal number of processors ($m = m^*$) is

$$T(m^*) = (1 + K) zT_{cm} \tag{34}$$

Case (iii) : Distribution time is less than the results collection time ($K > 1$).

From the equation (6), the expression for load fraction α_m can be rewritten as,

$$\alpha_m = \frac{1-g}{1-g^m} \tag{35}$$

and any load fraction α_i assigned to processor p_i is

$$\alpha_i = \frac{1-g}{1-g^m} g^{m-i} \quad i = 1,2, .., m\text{-}1 \tag{36}$$

The processing time for the entire load is given as

$$T(\alpha,m)=\left(\frac{1-g}{1-g^{m}}g^{m}(1+K\gamma)+K\gamma\right)wT_{cp} \quad (37)$$

In order to obtain closed-form expression for optimal number of processors, the equation (19) can be simplified as

$$g^{m^*-1}=\gamma\left(\frac{1-g^{m^*-1}}{1-g}\right) \quad (38)$$

The above equation can be further simplified and the optimal number of processors is obtained as

$$m^*=\left\lceil\frac{-\log(K)}{\log(g)}\right\rceil \quad (39)$$

Since K is greater than one, the term $\log(g)$ is always negative and $\log(k)$ is positive. Hence, the optimal number of processors is always positive. The processing time of the entire processing load with optimal number of processors ($m=m^*$) is

$$T(m^*)=(1+K)zT_{cm} \quad (40)$$

From equations (39) and (33) we can see that the closed-form expressions for optimal number of processors at $K<1$ and $K>1$ conditions are the same. But the values of m^* depends on the value of K.

5. Parallel Recursive Scheduling (PR)

The parallel recursive (PR) scheduling algorithm is different from the PI scheduling algorithm only in the sequence of results collection. Let us consider the same m-processor bus network. The control processor divides the processing load into m fractions (α'_1, α'_2 $\cdots$ α'_m) and distributes these load fractions to the m child processors in a sequence (p_1, p_2 $\cdots$ p_m) one after another. The child processors start computing the load fraction immediately after receiving their respective load fractions. After computing the results the child processors send the results of computation ($k\alpha'_1$, $k\alpha'_2$ $\cdots$ $k\alpha'_m$) to the control processor in a sequence (p_m, p_{m-1} $\cdots$ p_1) one after another. The timing diagram for the above load distribution and results collection sequence is shown in Figure 5. From the timing diagram, the recursive load distribution equations are obtained as,

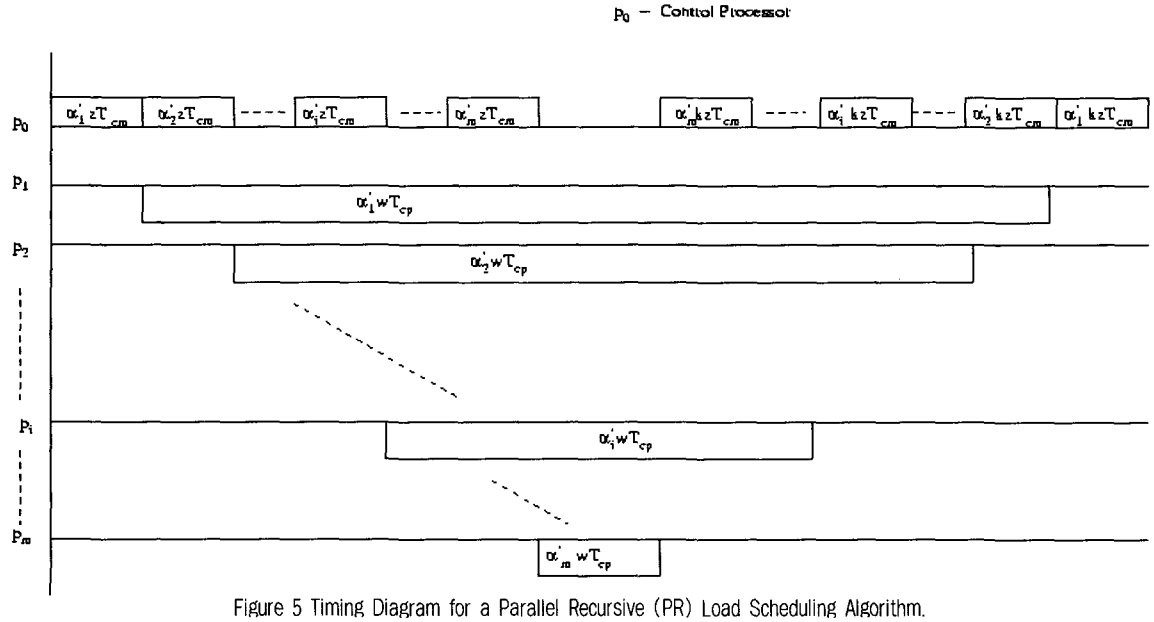

Figure 5 Timing Diagram for a Parallel Recursive (PR) Load Scheduling Algorithm.

$$\alpha'_i wT_{cp} = \alpha'_{i+1} zT_{cm} + \alpha'_{i+1} wT_{cp} + K\alpha'_{i+1} zT_{cm} \quad i = 1, 2, \ldots, m\text{-}1 \tag{41}$$

The normalization equation is

$$\sum_{i=1}^{m} \alpha'_i = 1 \tag{42}$$

The above recursive equations are obtained from the following condition: The computation time of the load fraction α'_i by the processor p_i is equal to the sum of the time to communicate the load fraction α'_{i+1} to the processor p_{i+1} by the control processor, the computation time of the load fraction α'_{i+1} by the processor p_{i+1} and the time to communicate the results of computation $k\alpha'_1$ to the control processor by the processor p_i. Here also we see that the recursive load distribution equation (41) along with the normalization equation form a system of m linear equations with m unknowns. These can be solved by exploiting the recursive nature of the equations as in the PI scheduling algorithm. Therefore, equation (41) can be rewritten as

$$\alpha'_i = \alpha'_{i+1} f, \quad i = 1,2,\ldots,m\text{-}1 \tag{43}$$

where

$$f = \frac{wT_{cp} + (1+K)zT_{cm}}{wT_{cp}} = 1 + (1+K)\gamma \tag{44}$$

These m recursive equations can be solve by expressing any load fraction α'_i in terms of α'_m as

$$\alpha'_i = f^{m-i}\alpha'_m, \quad i = 1,2,\Lambda\ ,m-1 \tag{45}$$

From the normalization equation, the value of load fraction α'_m is obtained as

$$\alpha'_m = \frac{1}{1 + \sum_{j=1}^{m-1} f^j} \tag{46}$$

The above equation can be reduced to

$$\alpha'_m = \frac{f-1}{f^m - 1} \tag{47}$$

From the timing diagram the processing time of the entire load is given as

$$T'(m) = \alpha'_1 zT_{cm} + \alpha'_1 wT_{cp} + K\alpha'_1 zT_{cm} \tag{48}$$

Above equation can be further simplified as

$$T'(m) = \alpha'_1(1 + \gamma + K\gamma)wT_{cp} \tag{49}$$

By substituting the value of α'_1 in equation (49) we can obtain closed-form expression for processing time as

$$T'(m) = \frac{f-1}{f^m - 1} f^m wT_{cp} \tag{50}$$

The closed-form expression for processing time clearly indicates that the processing time decreases with increase in number of processors. In this PR scheduling there is no restriction on the number of processors. So we can use infinite number of processors in PR scheduling. In such situation the control processor will assign infinity small load fraction to the last child processor and collect the corresponding results of computation. In this approach the idle time of the processors are quite high. This can be clearly seen from the timing diagram as is shown in Figure 5. Though we can use a large number of processors in this approach, the processing time for PR scheduling algorithm will converge to the minimum processing time obtained in PI scheduling approach. We will prove this fact using asymptotic analysis.

6. Asymptotic Analysis

Now, we prove that the minimum processing time in PR scheduling algorithm will converge to the minimum processing time in PI scheduling algorithm. For this purpose, we will examine the behavior of the processing time $T'(m)$ in PR algorithm as $m \to \infty$.

$$\lim_{m\to\infty} T'(\alpha,m)=\frac{f-1}{f^m-1} f^m w T_{cp} \to (f-1) w T_{cp} \qquad (51)$$

In the abovementioned processing time equation the term $f^m/(f^m-1)$ will be equal or converge to one when $m \to \infty$. Hence, the processing time is reduced to

$$T'(m) = (1+K)\gamma w T_{cp} \qquad (52)$$

In the above equation we observe the following: The processing time is the same as obtained in PI scheduling with optimal number of processors (See equation (21)). So, in PR scheduling algorithm even if we use a large number of processors the processing time will not be less than the processing time for PI scheduling scheme with optimal number of processors. We will present numerical examples to show the performance of PI and PR scheduling algorithms.

7. Numerical Example

Consider a homogeneous bus network with thirty child processors (m = 30). The computation speed parameter of the processors in the network is w = 1.0 and the communication speed parameter of the common bus is z = 0.15. Let T_{cm} = 1.0 and T_{cp} = 1.0. Now, we will consider different values of K for PI and PR load scheduling algorithms.

Case (i) K = 1: The processing times for the PI and PR scheduling algorithms are obtained from the closed-form expressions obtained earlier. The optimal number of processors for PI scheduling algorithm is obtained from equa- tion (27). The optimal number of processors for the PI algorithm is 6. The processing times for PI and PR al- gorithms are plotted against the number of processors in Figure 6. From Figure 6, we can observe that the proc- essing time for PI algorithm decreases with the increase in number of processors up to optimal number of processor (m^* = 6) and remains a constant after the optimal number of processors. The processing time for PR algorithm asymptotically reaches the minimum processing time of PI algorithm. Also, we can see from the Figure 6 that the processing time for PI algorithm is less than or equal to the processing time of PR algorithm for any number of processors.

Case (ii) K = 0.2: The optimal number of processors for PI scheduling algorithm is 12.

Case (iii) K = 1.8: The optimal number of processors for PI scheduling algorithm is 5.

For the cases (ii) and (iii) the processing time for PI and PR algorithms are also shown in Figure 6.

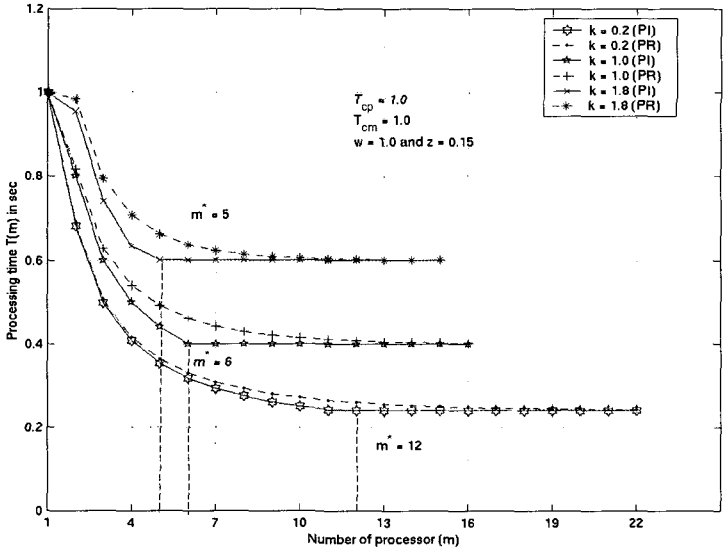

Figure 6. The Processing Time Vs Number of Processors for PI and PR Load Scheduling Algorithms: At Different Values of K.

Ⅳ. Linear Programming Formulation for Video Scheduling

In this section the video scheduling problem dis- cussed in the earlier section is formulated as a linear programming problem. We will consider both Parallel Interlaced (PI) and Parallel Recursive (PR) algorithms in this formulation. Consider the following linear programming problem in the standard form.

$$\text{Minimize } c^T X \quad \text{subject to } A^* X \geq b,\ X \geq 0. \tag{53}$$

In the video scheduling problem the objective of the problem is to find the load fractions assigned to the processors in the network so that the processing time (T) of the entire processing load is a minimum. Consider the bus network shown in Figure 1. There are m processors in the network participating in the processing. Hence, the decision variables (X) in the above linear programming formulation, are the processing time (T), and the load fractions (1, 2, ..., m) assigned to the processors in the network. So the vectors $X^T = [T, \alpha_1, \alpha_2, \Lambda, \alpha_m]$, and $c^T = [1, 0, 0, \Lambda, 0]$ are of dimension (m+1). The constraints matrix A and the vector b will be different for PI and PR algorithms.

PI algorithm : The constraints matrix A and the vector b are obtained from the timing diagram. For example, the processing time (T) should be greater than or equal to the sum of communication time of load fraction 1 to processor p_1, the processing time of load fraction 1 by the processor p_1, and the results communication time of all the processors in the network as given in equation (8). Similarly, the condition for any other processors can be obtained from the timing diagram. From the timing diagram we can also see that the processing time (T) is greater than or equal to the sum of load distribution time and results collection time by the control processors. The normalization condition is also included to restrict the sum of load fractions assigned to the processors to unity. Hence, the constraint matrix $\boldsymbol{A}$ and the vector $\boldsymbol{b}$ are obtained from the following equations.

$$T - \sum_{j=1}^{i} \alpha_j z T_{cm} - \alpha_i w T_{cp} - \sum_{j=i}^{m} K \alpha_j z T_{cm} \geq 0, \quad i = 1,2,\Lambda\, m \tag{54}$$

$$T - \sum_{i=1}^{m} (1+K) \alpha_i z T_{cm} \geq 0 \tag{55}$$

$$\alpha_1 + \alpha_2 + \Lambda + \alpha_m \geq 1 \tag{56}$$

Hence, in linear programming formulation of PI video scheduling problem, we have the matrix $\boldsymbol{A}$ is of dimension $\Re^{(m+2)\times(m+1)}$. Note that here in this formulation the entire load processing time is denoted as T. Also note that in PI scheduling, it is shown that there exist an optimal number of processors to process the entire processing load. In the solution of Linear Programming formulation the load fractions assigned to the processors are greater than zero up to the optimal number of processors and the load fractions assigned to processors beyond the optimal number of processors are zero. We will show this in the next section via a numerical example.

PR algorithm : For this algorithm the A matrix is ob- tained based on the condition that the processing time (T) should be greater than or equal to the finish time of all the (m) processors in the network. Finish time of the processor p_i, is the time difference between the instant at which the results from the processor p_i reach the root processor and the instant at which the root processor initiates the load distribution process. For example, if the root processor initiates the load distribution process at time zero, then the finish time of the processor p_i is the sum of communication time of the load fractions α'_1, α'_2 ⋯ α'_{i-1} to the child processors p_1, p_2 ⋯ p_{i-1} and computation time of the load fraction α'_1 for the processor pi, and the communication time of results of computation $K\alpha'_1$, $K\alpha'_2$ ⋯ $K\alpha'_{i-1}$ from processors $p\,p_1$, p_2 ⋯ p_{i-1}, to the control processor. The normalization equation is also included in the constraints. Hence, the constraints for PR algorithm are

$$T - \sum_{j=1}^{i} \alpha_j z T_{cm} - \alpha_i \left(w T_{cp} + z T_{cm} \right) \geq 0, \geq 0, \ i = 1,2,\Lambda\, ,m \tag{57}$$

$$\alpha_1 + \alpha_2 + \Lambda + \alpha_m \geq 1 \quad (58)$$

Hence in linear programming formulation of PR video scheduling problem we have $m+1$ constraints and $m+1$ decision variables. These above linear programming formulations of video scheduling problem are solved using neural network approach described in the following section.

V. Neural Network for Linear Programming Problem

Artificial neural networks (ANN's) have been applied to several classes of constrained optimization problems and have shown potential for solving such problems efficiently. Among such networks, the first one for solving linear programming problems is proposed by Tank and Hopfield [13], where linear programming (LP) problem is mapped onto a recurrent neural network. When the constraint violation occurs, the magnitude and the direction of violation are sent back to adjust the states of the neurons in the network such that the overall energy in the network is always decreasing until it reaches a minimum. When the energy attains its minimum, the states of the neurons are the solution of the original problem. The advantage of this model [13] is that it can be implemented using analog electrical components. The analog circuit is designed so as to model the basic components of a biological neural network and also provides very fast solutions. This study has generated a lot of interest and many researchers attempted to solve optimization and related problems using recurrent neural networks. It may not be possible to discuss all the studies in this aspect and so we mention a few studies in this direction [14-20]. From scheduling point of view, application of neural network approach to job-shop scheduling and process scheduling in real-time communication systems are discussed in [21-22], respectively.

There are two possible methods to formulate the LP problem in terms of artificial neural networks. One method is to construct an appropriate energy function (Lyapunov function) so that the lowest energy state will correspond to the desired (optimal) solution. This derivation of energy function enables us to transform the minimization problem into a set of ordinary differential equations on the basis of which we can design artificial neural networks with connection weights and non-linear activation function [14-16]. Another method is to construct a set of ordinary differential equations and then find an appropriate Lyapunov function, such that all the trajectories of the system converges to some equilibrium points which corresponds to the desired (optimal) solution [18-20]. A comparative study of solving linear programming problems with neural networks is given in [17]. Recently, a simple neural network model based on differential equation method that solves both the primal and dual problems presented in [20] is used in this paper for the video scheduling problem in distributed computing system because of its faster convergence. Also, this method does not depend on any parameter selection.

Problem formulation : Consider the following linear programming problem in the standard form:

$$\begin{aligned} &\text{Minimize the cost function } c^T X \\ &\text{subject to linear constraints, } AX \geq b,\ X \geq 0 \end{aligned} \quad (59)$$

The dual problem of the above primal problem is

$$\begin{aligned} &\text{Maximize } b^T Y \\ &\text{subject to linear constraints, } a^T Y \leq c,\ Y \geq 0 \end{aligned} \quad (60)$$

Network architecture : The recurrent network architecture presented in [20] has neurons arranged in a two- dimensional array of size $\Re^{n \times m}$, where n is the number of primal variable and m is the number of dual variables. The output of each neuron is solution of the linear programming problem. The constant external input to the primal neuron layer is array c and to the dual neuron array is vector b. A new feature introduced in this study [20] is that the inputs to the primal neurons are the outputs of the dual neurons and their derivatives. In the same way, the inputs to the dual neurons are the outputs of the primal neurons and their derivatives. This feature introduces non-linearity into the system. The output

of the neurons passes through an activation function. The activation function enforces the non-negativity constraints on primal and dual variables. The basic requirements of the activation function are non-negativity and nondecreasing monotonicity, i.e., $g(u) \geq 0$ and $g'(u) \geq 0$. A simple activation function which satisfies the above property is used in this approach. It can be defined as $g(u) = 0$ if $u > 0$, otherwise g(u) = 0. The following nonlinear dynamical system describes the outputs of the primal and dual neurons.

$$\frac{dx}{dt} = -c + A^T\left(y + k\frac{dy}{dt}\right), \quad x \geq 0 \tag{61}$$

$$\frac{dy}{dt} = b - A\left(x + \frac{dx}{dt}\right), \quad y \geq 0 \tag{62}$$

The above dynamical system equations are also e quivalent to a system of second order differential e quations. From the dynamical system equations (61) and (62), we can see that the weight connections and bias (or external inputs) are expressed in terms of the real coefficients of the linear programming problem. In order to simulate the neural network architecture numerically we can transform the set of differential equations (61) and (62) into a system of difference equations.

$$dx^{p+1} = \max\left(x^p + dt\left[-c + A^T\left(y^p + \gamma dy^p\right)\right], 0\right) - x^p \tag{63}$$

$$x^{p+1} = x^p + dx^{p+1} \tag{64}$$

$$dy^{p+1} = \max\left(y^p + dt\left[b - A\left(x^p + \gamma dx^p\right)\right], 0\right) - y^p \tag{65}$$

$$y^{p+1} = y^p + dy^{p+1} \tag{66}$$

The difference equations (63) and (65) will cal culate the changes in the primal and dual design var iables and also performs characteristics of activation function. This neural network can be easily realized using analog circuit realization [20]. Based on ex periments it is shown in [20] that the system always converges to a stable state if $r > 0$ and the speed of convergence depends on the value of r. The per formance improvement of this new model is due to its ability to handle larger discrete time step (dt) without becoming unstable. For numerical simulation, the solution process is stopped when the results are converged to equilibrium value (optimal value), i.e., the simulation is terminated when the outputs of all primal and dual neurons do not change by more than a threshold value ($\eta = 10^{-5}$) from one iteration to the next. For ease of understanding the neural network architecture, we will present a few illustrative examples in the following section.

1. Neural Networks for Video Scheduling Problems

The neural network solution to the video scheduling problems is achieved by defining the neural network architecture, weight connections between the neurons in different layer, and bias (or external) input to each neuron. The neural network architecture (i.e., number of neuron in each layer and external inputs to neurons in each layer) depends on the size of the problem. Now, we present formal description of the neural network architecture used for video scheduling problems. From the linear programming formulation of PI and PR scheduling algorithm and the same example (numerical example), we will demonstrate the advantage of neural network based scheduling approach. Let us consider four ($m = 4$) processors system.

PI Algorithm : Using the linear programming formulation for PI algorithm, the constraint matrix A, vectors X, b and c are given below:

$$A=\begin{bmatrix} 1.0 & -0.4 & -0.4 & -0.4 & -0.4 \\ 1.0 & -1.4 & -0.2 & -0.2 & -0.2 \\ 1.0 & -0.2 & -1.4 & -0.2 & -0.2 \\ 1.0 & -0.2 & -0.2 & -1.4 & -0.2 \\ 1.0 & -0.2 & -0.2 & -0.2 & -1.4 \\ 0 & 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix}, b=\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, c=[1\ 0\ 0\ 0\ 0]^T \tag{67}$$

From the above equations, we can observe that the neural network has six dual neurons and five primal neurons. The external input to the first primal neuron is -1 and sixth dual neuron is 1. The neural network is

simulated numerically with zero initial conditions. The processing time (T), and the load fractions (α_1, α_2, α_3, α_4) assigned to the processors in the network obtained from the neural network approach are T = 0.5, α_1 = 0.25, α_2 = 0.25, α_3 = 0.25, and α_4 = 0.25.

PR Algorithm : For the linear programming formulation for PR algorithm the constraint matrix A, vectors X, b and c are given below:

$$A=\begin{bmatrix} 1.0 & -1.4 & 0 & 0 & 0 \\ 1.0 & -0.4 & -1.4 & & 0 \\ 1.0 & -0.4 & -0.4 & -1.4 & 0 \\ 1.0 & -0.4 & -0.4 & -0.4 & -1.4 \\ 0 & 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix}, b=\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, c=[1\ 0\ 0\ 0\ 0]^T \quad (68)$$

From the above equations, we can observe that the neural network has five primal and dual neurons. The external input to the first primal neuron is -1 and fifth dual neuron is 1. Let us assume zero initial conditions for the outputs and the derivatives of the primal and dual neurons in the network. The processing time (T), and the load fractions (α_1, α_2, α_3, α_4) assigned to the processors in the network obtained from the neural network approach are T = 0.5408, α_1 =0.3863, α_2 = 0.2759, α_3 = 0.1971, and α_4 = 0.1408. The results obtained using neural network approach for PI and PR scheduling algorithms exactly matches with the closed-form solutions.

VI. Conclusions

The applicability of divisible load scheduling methodology to the problem of video scheduling is presented. The time to collect the results of computation by the originating processor is included in the divisible load scheduling to obtain solution to the video scheduling problem. Closed-form expressions for the processing time are presented and using this closed-form expression the optimal number of processors is also derived. The video scheduling problem is also formulated as a linear programming problem and its solution using neural network is presented.

References:

[1] D. T. Altilar and Y. Paker, "Optimal Scheduling Algorithms for Communication Constrained Parallel Processing," *Lecture Notes in Computer Science*, LNCS, Vol. 2400, pp. 197-206, Euro-Para, 2002, Springer-Verlag, Berlin Hidelberg, 2002.

[2] V. Bharadwaj, D. Ghose, V. Mani, and T. G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems*, Los Alamitos, California, IEEE Computer Society Press, 1996.

[3] Special Issue on: Divisible Load Scheduling, *Cluster Computing*, Vol. 6, Jan 2003.

[4] http://www.ee.sunysb.edu/tom/dlt.html

[5] Y. C. Cheng and T. G. Robertazzi, "Distributed computation with communication delay," *IEEE Trans. Aerospace and Electronic Systems*, Vol. 24, pp. 700-712, Nov. 1988.

[6] Y. C. Cheng and T. G. Robertazzi, "Distributed computation for a tree network with communication delay," *IEEE Trans. Aerospace and Electronic Systems*, Vol. 26, pp.511-516, May 1990.

[7] S. Bataineh and T.G. Robertazzi, "Bus oriented load sharing for a network of sensor driven processors," *IEEE Trans. Systems Man Cybernetics*, Vol. 21, pp. 1202-1205, Sep-Oct, 1991.

[8] V. Bharadwaj, D. Ghose, and V. Mani, "Optimal sequencing and arrangement in single-level tree networks with communication delays," *IEEE Trans. Parallel and Distributed Systems*, Vol. 5, pp. 968-976, 1994.

[9] H. J. Kim, G. I. Lee, and J. G. Lee, "Optimal load distribution for tree network of processors," *IEEE Trans. Aerospace and Electronic Systems*, Vol. 32, no. 2, pp.607-612, 1996.

[10] S. Bataineh, T. Hsiung, and T. G. Robertazzi, "Closed form solutions for bus and tree networks of processors sharing a divisible job," *IEEE Trans on Computers*, Vol. 43, pp. 1184-1196, 1994.

[11] T.G. Robertazzi, "Processor equivalence for daisy chain load sharing processors," *IEEE Trans. Aerospace and Electronic Systems*, Vol. 29, pp. 1216-1221, Oct 1993.

[12] J. Sohn, and T. G. Robertazzi, "Optimal divisible job load sharing on bus networks," *IEEE Trans. Aerospace and Electronic Systems*, Vol. 32, pp. 34-40, Jan 1996.

[13] D. W. Tank and J. J. Hopfield, "Simple 'neural' optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit," *IEEE Trans. Circuits and Systems*, Vol. CAS-33, pp. 533-541, May 1986.

[14] A. Cichocki and R. Unbehauen, "Neural networks for solving systems of linear equations and related problems," *IEEE Trans. Circuits and Systems-I: Fund. Theory App.*, Vol. 39, pp. 580-594, Feb. 1992.

[15] M. P. Kennedy and L. O. Chua, "Neural networks for nonlinear programming," *IEEE Trans Circuits and Systems*, Vol. 35, pp. 554-562, May 1988.

[16] A. Rodriguez-Vazquez, R. Dominguer-Castro, A. Rueda, J.L.

Huertas, and E. Sanchez-Sinencio, "Nonlinear switched capacitor 'neural' networks for optimization problems," *IEEE Trans. Circuits and Systems*, Vol. 3 7, pp. 384-397, March 1990.

[17] S. H. Zak, V. Upatising, and S. Hui, "Solving linear programming problems with neural networks: A comparative study," *IEEE Trans. Neural Networks*, Vol. 6, pp. 94-104, Jan. 1995.

[18] Y. Xia, and J. Wang, "Neural networks for solving linear programming problems with bounded variables," *IEEE Trans Neural Networks*, Vol. 6, pp. 515-519, March 1995.

[19] Y. Xia, "A new neural network for solving linear programming problems and its application," *IEEE Trans Neural Networks*, Vol. 7, pp. 525-529, March 1996.

[20] K. V. Nguyan, "A nonlinear neural network for solving linear programming problems," ISMP-2000, International Symposium on Mathematical Programming, Aug. 7-11, Atlanta, GA, USA, 2000.

[21] D. N. Zhou, V. Cherkassky, T. R. Baldwin, and D. E. Olson, "A neural network approach to job-shop scheduling," *IEEE Trans. Neural Networks*, Vol. 2, pp. 175-179, Jan. 1991.

[22] S. Cavalieri and O. Mirabella, "Neural networks for process scheduling in real-time communication systems," *IEEE Trans. Neural Networks*, Vol. 7, pp.1272-1285, Sep. 1996.

[23] V. Bharadwaj, D. Ghose, and V. Mani, "Multi installment load distribution in tree networks with delays," *IEEE Trans. on Aerospace and Electronic Systems*, Vol. 31, pp. 555-567, 1995.

[24] V. Bharadwaj, D. Ghose, V. Mani, "An efficient load distribution strategy for a linear network of processors with communication delays," *Computers and Mathematics with Applications*, Vol. 29, pp. 95-112, 1995.

[25] D. Ghose, and V. Mani, "Distributed computation with communication delays: Asymptotic performance analysis," *Journal of Parallel and Distributed Computing*, Vol. 23, pp. 293-305, 1994.

[26] J. Blazewicz, and M. Drozdowski, "The performance limits of a two-dimensional network of load sharing processors," *Foundations of Computing and Decision Sciences*, Vol. 21, pp. 3-15, 1996.

[27] J. Blazewicz, and M. Drozdowski, "Scheduling divisible jobs on hypercube," *Parallel Computing*, Vol. 21, pp. 1945-1956, 1995.

[28] W. Glazek, "A multistage load distribution strategy for three dimensional meshes," *Cluster Computing*, Special Issue on: Divisible Load Scheduling, Vol. 6, pp. 31-39, 2003.

[29] Keqin Li, "Parallel processing of divisible loads on partitionable static interconnection networks," *Cluster Computing*, Special Issue on: Divisible Load Scheduling, Vol. 6, pp. 47-55, 2003.

[30] Keqin Li, "Improved methods for divisible load distribution on k-dimensional meshes using pipelined communications," IEEE Transactions on Parallel and Distributed Systems, Vol. 14, pp. 1250-1261, 2003.

[31] S. Bataineh, and M. Alibraham, "Effect of fault tolerance and communication delay on response time in a multiprocessor system with a bus topology," *Computer Communications*, Vol. 17, 1994.

[32] J. Sohn, and T. G. Robertazzi, "Optimal time-varying load sharing divisible jobs," *IEEE Trans. on Aerospace and Electronic Systems*, Vol. 34, 1988.

[33] J. Sohn, T. G. Robertazzi, and S. Luryi, "Optimizing computing costs using divisible load analysis," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 9, pp. 225-234, 1998.

[34] X. Li, V. Bharadwaj, and C. C. Ko, "Divisible load scheduling on single-level tree networks with buffer constraints," *IEEE Trans. on Aerospace and Electronic Systems*, Vol. 36, pp.1298-1308, 2000.

[35] M. Drozdowski, and P. Wolniewicz, "Divisible load scheduling in systems with limited memory," *Cluster Computing*, Special Issue on: Divisible Load Scheduling, Vol. 6, pp. 19-29, 2003.

[36] V. Bharadwaj, H. F. Li, and T. Radhakrishnan, "Scheduling divisible loads in bus networks with arbitrary processor release times," *Computers and Mathematics with Applications*, Vol.32, pp. 57-77, 1996.

[37] V. Bharadwaj, and G. Barlas, "Scheduling divisible loads with processor release times and finite size buffer capacity constraints in bus networks," *Cluster Computing*, Special Issue on: Divisible Load Scheduling, Vol. 6, pp. 63-74, 2003.

[38] H. J. Kim, "A novel optimal load distribution algorithm for divisible loads," *Cluster Computing*, Special Issue on: Divisible Load Scheduling, Vol. 6, pp. 41-46, 2003.

[39] H. J. Kim, and V. Mani, "Divisible load scheduling in single level tree networks: Optimal sequencing and arrangement in the non-blocking mode of communication," *Computers and Mathematics with Applications*, Vol. 46, pp. 1611-1623, 2003.

[40] S. K. Chan, V. Bharadwaj, and D. Ghose, "Large matrix-vector products on distributed bus networks with communication delays using the divisible load paradigm: Performance analysis and simulation," *Mathematics and Computer Simulation*, Vol. 58, pp. 71-92, 2001.

[41] D. Ghose, and H. J. Kim, "Load partitioning and trade off study for large matrix-vector computations in multicast bus networks with communication delays," *Journal of Parallel and Distributed Computing*, Vol. 55, pp. 32-59, 1998.

[42] J. Blazewicz, and M. Drozdowski, "Distributed processing of divisible jobs with communication startup costs," *Discrete Applied Mathematics*, Vol. 76, pp. 21-41, 1997.

[43] V. Bharadwaj, X. Li, and C. C. Ko, "On the influence of start-up costs in scheduling divisible loads on bus networks," *IEEE Trans. Parallel and Distributed Systems*, Vol. 11, pp. 1288-1305, 2000.

[44] S. Suresh, V. Mani, and S. N. Omkar, "The effect of start-up delays in scheduling divisible loads on bus networks: An alternate approach," *Computers and Mathematics with Applications*, Vol. 46, pp. 1545-1557, 2003.

[45] R. Agrawal, and H. V. Jagadish, "Partitioning techniques for large-grained parallelism," *IEEE Trans. Computers*, Vol. 37, pp. 1627-1634, 1988.

[46] D. Ghose, and H.J. Kim, "A generalized linear programming approach to optimal divisible load scheduling," Technical Report KNU/CI/MSL/001/2004, Kangwon National University, Department of Control and Instrumentation, Chunchon, 200-701, Korea, 2004.

[47] T. G. Robertazzi, "Ten reasons to use divisible load theory," *IEEE Computer*, pp. 63-68, May 2003.

저 자 소 개

S. Suresh

S. Sureshreceived the B.E. degree in electrical and electronics engineering from Bharathiyar University in 1999, and the M.E. degree in aerospace engineering from Indian Institute of Science, Bangalore in 2001. At present, he is working towards the Ph.D. degree in the Department of Aerospace Engineering at the Indian Institute of Science, Bangalore. His research interests include parallel and distributed computing, intelligence flight control, data mining, genetic algorithm and neural network.

V. Mani

V. Mani received the B.E. degree in civil engineering from Madurai University in 1974, the M.Tech. degree in aero- nautical engineering from Indian Institute of Technology, Madras, in 1976, and the Ph.D. degree in engineering from Indian Institute of Science (IISc), Bangalore, in 1986. From 1986 to 1988, he was a Research Associate at the School of Computer Science, University of Windsor, Windsor, ON, Canada, and from 1989 to 1990 at the Department of Aero- space Engineering, Indian Institute of Science. Since 1990, he has been with IISc, Bangalore, where he is presently a Professor in the Department of Aerospace Engineering. His research interests include distributed computing, queuing networks, evolutionary computing, neural computing and mathematical modeling. He is the co-author of a book, Scheduling Divisible Loads in Parallel and Distributed Systems (Los Alamitos, CA: IEEE Computer Society Press).

S. N. Omkar

S. N. Omkar received B.E. in mechanical engineering from University Viswesvarayya College of Engineering in 1985, the M.Sc (Engg) in aerospace engineering from Indian Institute of Science in 1992, and the Ph.D. degree in aerospace engineering from Indian Institute of Science, Bangalore, in 1999. He joined the Department of aerospace engineering at Indian Institute of Science, Bangalore where he is presently a Senior Scientific Officer. His research interest includes helicopter dynamics, neural network, fuzzy logic and parallel computing.

H. J. Kim

H. J. Kim received his B.S. in Electrical Engineering from Seoul National University in 1978, and M.S. and Ph.D. degrees in Control and Instrumentation Engineering from Seoul National University in 1986 and 1989, respectively. He joined Department of Control and Instrumentation Engineering, Kangwon National University in 1989, where he is currently a Professor. He was a visiting scholar at the University of Southern California, during 1992-1993. He was a Prime Investigator of the iPCTV national project during 1997-2000 to develop MHP, ATVEF, and DASE data broadcasting platform. During 2000-2005, he was the Prime Investigator of the iMS national project to develop an interactive broadcasting system based on DASE and ACAP, personalized broadcasting system over TV-Anytime specification, and TVN for home networking solution over digital television. He was the founder of the International Workshop on Digital Watermarking (IWDW). Currently, he is the Chairman of the Korea e-Learning Society, Editor-in-Chief of the Korean Society of Broadcast Engineers, and Head of the Media Service Research Center (MSRC-ITRC).