ELSEVIER

Image and Vision Computing 18 (2000) 919–938

image AND vision COMPUTING

www.elsevier.com/locate/imavis

Efficient partitioning and scheduling of computer vision and image processing data on bus networks using divisible load analysis

V. Bharadwaj*, Xiaolin Li, Chi Chung Ko

Department of Electrical Engineering, The National University of Singapore,10 Kent Ridge Crescent, Singapore, Singapore 119260

Received 29 March 1999; received in revised form 1 December 1999; accepted 3 December 1999

Abstract

We investigate the data partitioning, distribution, and scheduling problem for minimizing the total processing time of computer vision and image processing (CVIP) data on bus networks. Using the recently evolved divisible load paradigm (DLT) [V. Bharadwaj, D. Ghose, V. Mani, T.G. Robertazzi, Scheduling divisible loads in parallel and distributed systems, IEEE Computer Society Press, Los Almitos, California, 1996] for processing loads that are computationally intensive, we design and analyze a scheduler that optimally partitions the CVIP data and assigns it to the processors in the network in such a way that the total processing time is a minimum. In addition to the transmission delay in the network, we consider all the overhead components that penalize the time performance in the problem formulation. With this formulation, we derive closed-form solutions for the optimal processing time when the CVIP data distribution follows a fixed sequence. We then derive a *necessary and sufficient* condition for the existence of an optimal processing time. We then prove an *optimal sequence theorem* that identifies a sequence that gives rise to an optimal processing time among all possible load distribution sequences, whenever such a sequence of load distribution exists. The performance of the strategy proposed is also analyzed with respect to *speed-up* and *processor utilization or efficiency* metrics. Several illustrative examples are shown for the ease of understanding.

Keywords: Computer vision data; Image processing; Divisible loads; Data partitioning; Communication delays; Bus networks

1. Introduction

Scheduling computer vision and image processing (CVIP) data on a network of processors always offers a ponderous challenge in designing efficient strategies to minimize the total processing time of the entire data. While *divide-and-conquer* being a general solution to such problems, the question of how to achieve this goal in the best possible way, if not optimally, still remains open. In the existing literature, most of the research was focussed on analyzing and testing the processing time behavior on several computer network architectures [1,2]. Also, in general, the thrust was on designing strategies that could achieve this objective [3]. By and large, in all these works, both program and data parallelism are largely exploited to achieve the objective.

A good compilation of interesting applications and a discussion on their performance with respect to architectures such as, SIMD and MIMD can be found in Ref. [3]. In fact, Ref. [3] also discusses the advantages offered by these architectures. In general, the CVIP data to be processed is partitioned equally among all the processors in the network and processed in a concurrent fashion. In doing so, a processor participating in this computation needs some additional information pertaining to the data residing in other processors [3–8] for processing its share of the data. This is usually taken care by letting the processors communicate among themselves during the computational phase. However, to reduce the overheads incurred during this operation, the communication of this additional data among the processors is executed in a concurrent fashion [1,3,7]. All the research so far adopts either partitioning of the data in a row-wise fashion, or column-wise fashion, or in terms of blocks of data. In Refs. [5,6], a judicious choice on these partition schemes is made while processing a sequence of tasks arranged in a linear order. Thus, in general, the approach is to either choose one of these partitioning schemes or a hybrid of these in achieving the best possible schedule. We shall present a brief literature survey later in this section.

A recent paradigm, which facilitates scheduling of such processing loads onto a network of processors, offers a new ‘calculus’ to attack the problem of load sharing. The body of literature that deals with this scheduling is referred to as

* Corresponding author.

0262-8856/00/$ - see front matter
PII: S0262-8856(99)00085-2

divisible load theory (DLT) [9]. The theory uses a well-accepted and experimentally verified linear model [9–13] in modeling the computation and communication delays present in the processors and in the network. The research in this domain is of recent origin, from 1988 [14] and has gathered a considerable attention among researchers in the field of load sharing and balancing. The main attraction is due to its simplicity in adopting a linear model and its ability to track the behavior of the time performance analytically. In fact, this modeling and analysis is shown to tackle complex load scheduling strategies and provide elegant solutions [15]. The loads that are considered in this domain of research are referred to as *arbitrarily divisible*, which means that the load to be processed can be divided into any number of small fractions and can be assigned to the processors in the network. Also, these load fractions that are assigned to the processors can be processed independently, and there is no data dependency or causal precedence between the load fractions. Thus, data parallelism is fully exploited in achieving optimal solutions to schedule large volumes of data that can be partitioned and processed on the processors. CVIP data are natural candidates to this domain, as they require lots of CPU time to process and allows us to exploit data parallelism to the fullest possible extent.

However, what is needed to be taken care of is in supporting inter-processor communication that is required for processing. For instance, processing a set of rows assigned to a particular processor needs the information about the rows above and below its first and the last rows, when row partitioning is effected [7]. These additional data that are residing on the other processors needs to be sent to this processor for processing. This can be done in two different ways. In the first approach, explicit communication can be carried out concurrently between the processors [3] and in another approach, we can over-supply [7] the required data to the respective processors at the distribution phase itself. In fact, we will show that in this paper, for the architecture that is considered, the second approach is the natural choice. In this paper, we shall use this divisible load paradigm and demonstrate the time performance behavior in comparison with the conventional approaches present in the literature.

It may be noted that the DLT does not propose any new partitioning schemes such as row, column, and block partitioning [16], instead, determines a suitable size distribution among the processors in the network. This is done by taking into account all the delays present during the communication and the computation phases. Thus, if we wish to divide the data in a row wise fashion (similarly column wise or block wise), DLT will recommend the number of rows to be assigned to each processor in the network. In fact, research in the literature on CVIP data processing using linear models invariably use an equal partitioning of data among all the processors in the network [5,6] and analyze the performance with respect to different architectures [3]. This is because, by and large, the communication delays during the initial data distribution phase are never considered explicitly in all these problems. In fact, most of the literature on CVIP examines the data processing phase, however, assumes that the data is already available on the processors [5,7]. Thus, equal partitioning becomes the natural choice in these problems. However, when processing time minimization is to be carried out, we have to account for all possible delays that affect the time performance starting from the initial load distribution phase. The approach that DLT takes is generic, and considers the problem from the distribution phase to the processing phase and makes no assumptions on the topology of the network. Thus, we seek a scheduler that could optimize the time performance by an apt choice on the size distribution of the processing data among the processors. The design of such a scheduler carrying out the optimization is imperative to applications that are sensitive to the existing delays in the system. Unless the problem is formulated in such a way that the delays are explicitly considered, the latent effects of these delays and the associated overheads are difficult to track and poses difficulty in studying certain trade-off relationships. Also, the problem formulation must provide a fine tuning control on the size of the data that can be assigned to the processors.

We shall now present a brief review on the related work in the area of CVIP data processing. In Ref. [3], the mapping of several CVIP algorithms, such as histogramming, smoothing and FFT onto SIMD/MIMD systems were demonstrated. The authors proposed parallel processing schemes for these operations in detail under the assumption that the communication between adjacent processors can take place concurrently and all the processors are *identical*. However, the actual distribution of the entire data among the processors was not considered in the analysis. Hence, equal partitioning becomes the natural choice for optimal performance. Parallel algorithms on other new CVIP applications were also studied extensively in this work. An excellent analytical treatment on concurrent processing of various algorithms is discussed in Ref. [3]. In Refs. [5,6], some strategies were presented to map a sequence of processing CVIP operations onto a distributed memory system with different data partitioning and processor allocation policies for each task to be scheduled. Since this involves a hybrid mix of strategies, the authors explicitly considered the overheads incurred while redistributing the data between operations from one strategy to the other. However, the entire data was partitioned equally among all the processors. In Ref. [1], for feature extraction using MIMD architectures, parallel algorithms were presented. Data parallelism on SIMD architectures for image correlation operations was explored in Ref. [7]. In this paper, the authors employed two different strategies for distributing the data to be processed. As mentioned earlier, strategies in which we over-supply and explicit communication of the additional data are considered and it was shown that the asymptotic complexity was $(1/N)$, when compared to the serial algorithms, where N is the number of processors in the

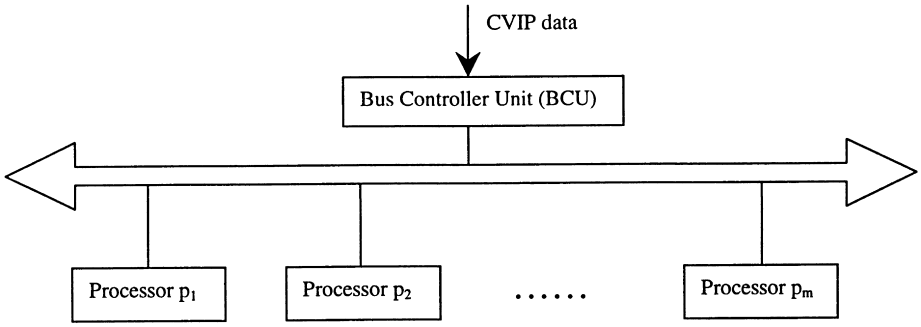


Fig. 1. Distributed bus architecture with *m*-processors and a BCU.

system. In Ref. [8], two different parallel image template matching algorithms on a 3-D processor array were proposed and optimal speed-up was achieved.

Our contributions in this paper are multifold and are as follows. Using the divisible load paradigm, on a network of processors connected by a bus, we first seek the optimal size distribution of the data. We derive a closed-form solution by including all the overhead components that penalize the time performance. Using this closed-form solution we derive *necessary and sufficient* conditions for the existence optimal processing time using all the processors in the network. We show the restricted monotonicity of the processing time behavior with respect to the number of processors, and identify the maximal set of processors to be used for achieving optimal processing time. The above mentioned optimization is carried out for a *fixed* sequence of load distribution. We introduce the concept of equivalent processor for the purpose of investigating the effect of sequencing on the time performance. We identify an optimal sequence in a space of all possible load distribution sequences, whenever such a sequence exists. To prove this result, we derive a number of interesting intermediate results. Finally, we propose a *greedy* algorithm to offer a sub-optimal solution, when an optimal sequence fails to exist. We extend the analysis for the case of homogeneous system of processors, and derive similar results. We then conduct a rigorous performance study on a variety of cases with varying parameters to see the performance. At every stage, we compare the performance with the existing equal partitioning scheme in the literature and demonstrate the advantage offered by DLT. We will also study the performance with respect to the speed-up and *utilization* or *efficiency* of the system. We present numerical examples for the ease of understanding at the required places. The contributions presented in this paper lays a strong theoretical basis and elicits a number of non-trivial observations in understanding the problem in depth. Also, this work serves as guidance to a series of experimental papers to appear in this area.

The organization of the paper is as follows. Section 2 elucidates the problem formulation and some definitions. Section 3 presents the derivation of the closed-form solution and derives a necessary and sufficient condition for the optimal load sharing. In Section 4, we propose the concept of sequencing and prove an optimal sequence theorem. Section 5 presents a detailed discussion on the results obtained and studies the behavior under different performance metrics. Section 6 concludes the paper.

2. Problem formulation, definitions and some remarks

The system model we envisage is as shown in Fig. 1. The CVIP data to be processed is assumed to arrive at the bus control unit (BCU). The BCU could be a general-purpose scheduler that computes a scheduling calendar that assigns the exact proportion of the data to the respective processors. In general, the network may have a dedicated control processor [9] to distribute the load among the processors or it may not have any dedicated BCU. In this paper, we shall consider the case when the network has a dedicated BCU to distribute the entire load optimally among the *m* processors $p_1, \ldots, p_m$.

2.1. Data partitioning and distribution strategy

The data partitioning and distribution strategy is described as follows. Consider an image of size $n \times n$ and *m* be the number of processors in the system. In the literature [16], the following are the types of partitioning of an image that are proposed.

1. *Row partition:* The *i*th row of the image data is allocated to processor $p_{R(i)}$, where $R(i) = \lfloor i/(n/m) \rfloor$, $i = 0, \ldots n-1$.
2. *Column partition:* The *j*th column of the image data is allocated to processor $p_{C(j)}$, where $C(j) = \lfloor j/(n/m) \rfloor$, $j = 0, \ldots n-1$.
3. *Block partition:* The pixel (i,j) is allocated to processor $p_{B(i,j)}$, where, $B(i,j) = \lfloor j/(n/\sqrt{m}) \rfloor + \lfloor i/(n/\sqrt{m}) \rfloor$, if $\lfloor j/(n/\sqrt{m}) \rfloor$ is even; $B(i,j) = \lfloor j/(n/\sqrt{m}) \rfloor \sqrt{m} + \lfloor j/(n/\sqrt{m}) \rfloor - \lfloor i/(n/\sqrt{m}) \rfloor$, if $\lfloor j/(n/\sqrt{m}) \rfloor$ is odd.
4. *Row-cyclic partition:* The *i*th row of the image data is allocated to processor $p_{RC(i)}$, where $RC(i)$ is $\lfloor i/b \rfloor$ mod $m, i = 0, \ldots, n-1$, and *b* is a *blocking factor* indicating

the number of consecutive rows to be packed together in a processor.

5. *Column-cyclic partition:* The jth column of the image data is allocated to processor $p_{CC(i)}$, where $CC(i)$ is $\lfloor j/b \rfloor \bmod m, j = 0, \ldots, n-1$, and b is a *blocking factor* indicating the number of consecutive columns to be packed together in a processor.
6. *Block-cyclic partition:* The pixel (i,j) is allocated to processor $p_{BC(i,j)}$, where $BC(i,j)$ is $(\lfloor i/b \rfloor + \lfloor j/b \rfloor) \bmod m$, and b is a *blocking factor* indicating the number of elements to be packed in a block.

These types of partitioning of the data yield optimal, if not the best possible time performance only under the following situations. Firstly, the data to be partitioned and processed is relatively small in size that the processing can be done on a fewer number of processors. Secondly, the communication delays and the associated overhead delays incurred are negligible. Thirdly, when homogeneous cluster of processors are to be utilized. Also, all the above mentioned partition schemes differ in the manner in which the load is partitioned, but do not mention on the specific size of the individual portions that will be assigned to the processors. Although analysis of load partitioning schemes can be carried out under the above-mentioned situations, in practice, these are too unrealistic and highly restrictive. Specifically, when the communication delays are non-zero, equal partition will not fetch the optimal processing time. This will be demonstrated in Section 3. Further, we shall relax all the above mentioned assumptions and use a mathematical model that captures the realistic scenario to distribute the data.

As mentioned in Section 1, CVIP data is the right candidate to exploit data parallelism, and hence, inherently *divisible* in nature. The only restrictive aspect while processing these data is when the individual processors demand some additional information from other processors to do processing on the data assigned to them. For instance, when a processor p_i is assigned a set of rows of an image starting from k to $k+r$, for some $k > 0$, $r > 0$ and $k + r \leq n$, in practice, depending on the image processing application [2,3], processor p_i may need the $(k-1)$th and $(k+r+1)$th rows for it to perform computation on its assigned rows. To carry out this processing, in the literature, it is a standard practice to allow the processors to communicate among themselves for the respective data exchange. This data exchange takes place concurrently among the respective pairs of processors [3]. However, in our strategy to be proposed, we assign such additional data required by the individual processors right at the initial communication phase. Thereafter, the processors will perform computations on the respective assigned portions by utilizing the additionally supplied data. We will demonstrate these aspects in our study. Thus, if all the *causal precedence* relationships are taken into account, the data can be infinitely divisible and hence, viable to the modeling and analysis proposed by DLT literature [9,15]. In fact, the contributions in this paper convincingly reflect this attempt to model such CVIP data and shows an incredible improvement in the overall performance with respect to some standard performance metrics available in the literature [17].

Hereafter, we shall refer to the CVIP data as simply a *load* for the ease of referencing. Our load distribution strategy is as follows. The BCU divides the load into m load fractions (portions) and distributes among all the m-processors in a particular order or sequence say $p_1, \ldots, p_m$, one after other. The problem is to determine the optimal size of the individual load portions assigned to the processors to minimize the total processing time. It has been rigorously proved in the literature [9] that, *for optimal processing time of a divisible load, it is* necessary and sufficient *that all the processors that are participating in the computation must stop computing at the same time instant*. It is beyond the scope of this paper to reproduce the proof. However, the proof is intuitive from the fact that, if the processors do not stop at the same time, redistribution of the data from the processors having excess loads to the processors having lighter loads will decrease the processing time. Here too, we shall use this optimality criterion to analyze the processing time performance of the system. As mentioned earlier, we assume that the individual load fractions assigned to the processors also contain the additional data required for them to process their assigned data. Example 1 provides considerable motivation for the research presented in this paper. Finally, it may be noted that the emphasis in this current research is neither on presenting an alternative load partitioning scheme such as row, and column, as mentioned above, nor on any specific CVIP operations, but instead, our focus is to derive the optimal size of the portions of the CVIP data that each individual processor should be assigned and to optimize the total processing time of the entire CVIP data. Thus, without any loss of generality, the results in this paper assume the row-partitioning scheme.

We now introduce some useful notations [9] that will be used throughout the paper.

α_i: Fraction of the load assigned to processor p_i.
w_i: The inverse of the computation speed of processor p_i.
T_{cp}: Time taken to process a unit load by the standard processor when $w_i = 1$.
θ_{cp}: An additive computation overhead component that includes the sum of all delays associated with the computation process.
z: The inverse of the communication speed of the link.
T_{cm}: Time taken to transmit a unit load by the standard communication link when $z = 1$.
θ_{cm}: An additive communication overhead component that includes the sum of all delays associated with the communication process.

Thus, $\alpha_i w_i T_{\mathrm{cp}}$ denotes the computation time of this load fraction by p_i, and $\alpha_i z T_{\mathrm{cm}}$ denotes the communication time

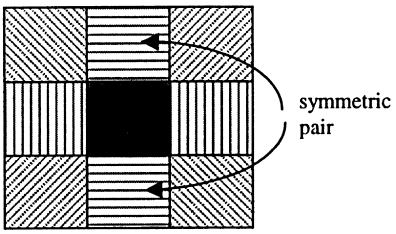


Fig. 2. Symmetric pairs of pixels for SNF.

of a fraction of the load α_i. We shall denote the product $w_i T_{cp}$ as E_i and zT_{cm} as C, respectively, throughout the paper.

2.2. Mathematical modeling

We use the following widely accepted and experimentally demonstrated linear model for communication and computation of the links and the processors. As in the literature [5,6,9,11], we assume that the communication time of a given load over the link is proportional to the amount of load that the link carries, plus a constant additive communication overhead component. Similarly, the computation time of a given load by processor p_i is proportional to the amount of load assigned to that processor, plus a constant additive computation overhead component. Thus, using the above notations, we see that the communication time of a fraction of the load α_i is given by $\alpha_i C + \theta_{cm}$ and the computation time of this load fraction by p_i is given by, $\alpha_i E_i + \theta_{cp}$. Although this kind of mathematical modeling with the overhead components was considered earlier in the literature [5,6,9,11], rigorous analysis on the size distribution and the processing time performance was not attempted using this realistic model. In this paper, we shall show that the presence of such overheads, which are inherently present in any realistic system, will drastically affect the time performance and may lead to different design decisions. This is crucial whenever the optimality of the solution and related trade-off studies are important in the design. A detailed discussion on this is presented in Section 5.

We shall now define the following terminology that will be used frequently in this paper. Without loss of generality, we let the total load to be unity, i.e. $L = 1$ in our analysis, and we refer to this load as a normalized load.

1. *Load distribution*, denoted as α for a m-processor system, is defined as a m-tuple $(\alpha_1 \ldots \alpha_m)$ such that $0 \leq \alpha_i \leq 1$ and $\sum_{k=1}^{m} \alpha_k = 1$. The equation $\sum_{k=1}^{m} \alpha_k = 1$ is referred to as *normalization equation*. Further, let us denote the space of all possible load distributions as Γ.
2. *Finish time* of processor p_i, denoted by $T_i(\alpha,m)$, is the time difference between the instant at which the ith processor stops computing and the time instant at which the BCU initiates the process.
3. *Processing time*, denoted as $T(\alpha,m)$, is the time at which the entire load is processed, i.e. $T(\alpha, m) = \max\{T_i(\alpha, m)\}$ $i = 1, 2, \ldots, m$, where T_i is the finish time of processor p_i.
4. *Optimal processing time*, denoted as $T^*(\alpha^*,m)$, which is the minimum processing time to finish the entire load assigned on the system, i.e. $T^*(\alpha^*, m) = \min_{\alpha \in \Gamma}\{T(\alpha, m)\}$.

In the following sections, we use the criterion that the optimal processing time is achieved when all the processors that are involved in the computation of the load will stop computing at the same time. In fact, this criterion has been shown to be necessary and sufficient to achieve the optimal processing time in the literature [9] without taking into account the additional overhead components. We shall present a rigorous analysis using the model which considers all the overhead components explicitly and show how this clearly elicits some of the significant trade-off relationships between the processing time and the utilization of the number of processors in the system. We also show the performance in terms of speedup factor and efficiency.

2.3. Motivating example

Example 1. Consider an input image of size 1024×1024 pixels to be processed on a $m = 4$ processor bus network shown in Fig. 1. The speed parameters are, $z = 0.5$, $w = 1.1$, and the overhead components are $\theta_{cm} = 0.1$ and $\theta_{cp} = 0.1$. For the purpose of demonstration, we consider an image enhancement operation, called symmetric neighborhood filter (SNF) [18], to be performed on each pixel in this image. SNF is an efficient edge-preserving smoothing filter. We shall give a brief description of SNF algorithm. SNF filter compares each pixel to its eight-connected neighbors. The neighbors are grouped into four symmetric pairs. Fig. 2 shows a 3×3 mask with the symmetric pairs having identical textures. In this algorithm, from each pair, a pixel that has a closer gray level to the center pixel is selected. Then, the corresponding gray levels for all these four selected pixels (from the four symmetric pairs) are averaged together. Finally, the center pixel is replaced by the mean of the above-calculated average and its current gray level. For the purpose of demonstration, let one row of the input image be a unit load, and normalize the time measure of T_{cm} and T_{cp} as 1 unit time. In fact, T_{cm} may be in the order of 100 ms for some well-known architectures such as iPSC/2, CM-5, and nCube [2,11]. Note that, for SNF enhancement operation, we have to supply the boundary rows to each processor in an overlapped manner. This is the additional data we are referring to earlier in this section. As mentioned, we prefer an *over-supply* strategy in which we assign the required rows on the boundary from the adjacent sub-images. However, it should be noted that no operation will be performed on these additional rows by this processor.

We may adopt two partition strategies. In the first case,

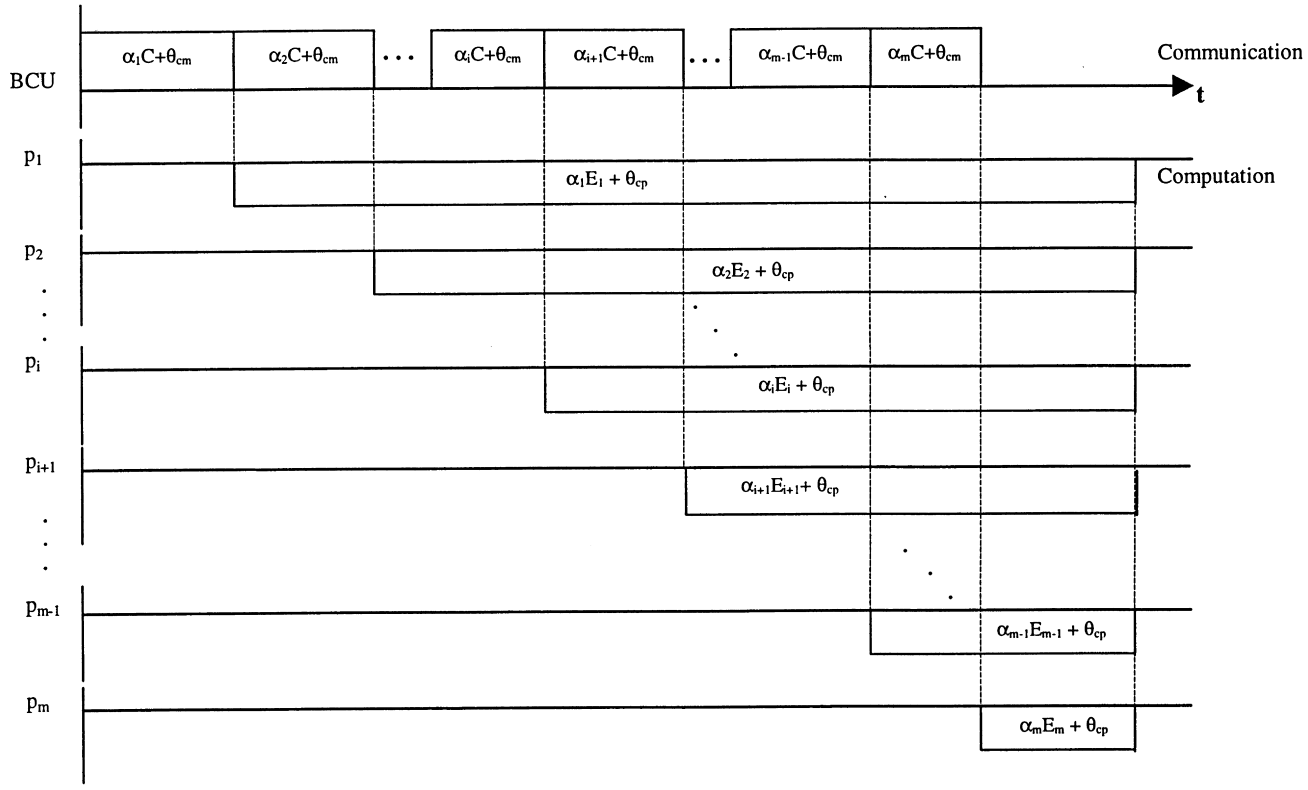


Fig. 3. Timing diagram for processing the divisible load with *m*-processors.

referred to as P_{eq}, we partition the image into four sub-images of equal size, each one with 256 rows of pixels, and in the other case, referred to as P_{uneq}, we partition the image into four sub-images of unequal size. For P_{eq}, the load distribution is $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = (257, 258, 258, 257)$ (the real values have been rounded up). Note that the load distribution has included the six rows of over-supplied information. In this case, the processing time is 797.1 units. For P_{uneq}, we choose the load distribution as $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = (415, 285, 196, 134)$. It may be verified that the processing time in this case is 663.1 units. Compared to P_{eq}, in the case of P_{uneq}, the performance gain on the total processing time is about 16.8%.

Thus, it is apparent that by employing a judicious strategy to partition the given data, an optimal processing time could be achieved. When a multiprocessor system is used for such CVIP data processing applications, one of the standard and useful performance measures is the *speed-up* [17] achieved. In our case, with equal partitioning (P_{eq}), we have a speed-up of 2.05 and using the load distribution we have considered (P_{uneq}), we have a speed-up given by, 2.47, a 20.5% improvement in the gain achieved. Similarly, one can also compute *efficiency* or *utilization* defined as *Efficiency* $= 100\% \times$ *Speedup/number of processors*. In our case, these are, 51.25% for P_{eq} and 61.7% for P_{uneq}, respectively. Our research contribution formalizes these observations.

3. Closed-form solutions for the processing time

In this section, we shall derive a closed-form expression for the optimal processing time by assuming that the sequence of load distribution is from p_1 to p_m in that order. As in the previous section, we refer to this sequence as a *fixed sequence* and the load distribution strategy as the *proposed strategy* throughout this paper.

3.1. Recursive equations and optimal solution

From Fig. 3, we obtain the following recursive equations:

$$\alpha_i E_i + \theta_{cp} = \alpha_{i+1}(E_{i+1} + C) + \theta_{cp} + \theta_{cm}, \quad i = 1, \ldots, m-1. \tag{1}$$

We rewrite Eq. (1) as,

$$\alpha_i = \alpha_{i+1} f_{i+1} + \beta_i, \quad i = 1, \ldots, m-1, \tag{2}$$

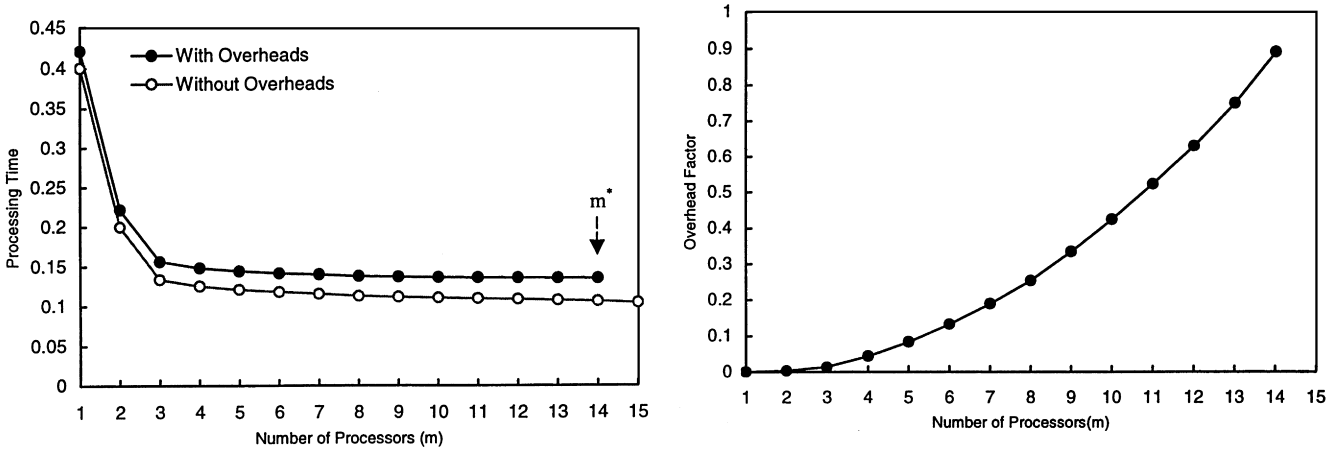
where

$$\frac{(E_{i+1} + C)}{E_i} = f_{i+1}$$

and

$$\left(\frac{\theta_{cm}}{E_i}\right) = \beta_i$$

for all $i = 1, \ldots, m-1$. Now, expressing each of these load



Speed Parameters:

C	θ_{cm}	θ_{cp}	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	E_{11}	E_{12}	E_{13}	E_{14}	E_{15}
0.1	0.001	0.02	0.3	0.2	0.1	0.4	0.6	0.7	0.8	0.5	0.9	1.1	1.3	1	0.6	0.5	0.3

Fig. 4. Behavior of the processing time and the overhead factor for the proposed strategy with respect to m (heterogeneous system).

fractions in terms of α_m, we obtain,

$$\alpha_i = \alpha_m M_i + N_i, \quad i = 1, \ldots, m-1 \tag{3}$$

where

$$M_i = \prod_{j=i+1}^{m} f_j, \quad i = 1, \ldots, m-1, \tag{4}$$

$$N_i = \sum_{p=i}^{m-1} \beta_p \left(\prod_{j=i+1}^{p} f_j \right), \quad i = 1, \ldots, m-1, \tag{5}$$

Thus, from Eq. (3) we have $(m-1)$ linear equations with m variables, and together with the normalization equation, we have m equations. These equations can be solved to obtain the individual load fractions. Now, using Eq. (3) in the normalization equation, we obtain α_m as,

$$\alpha_m = \left(\frac{1 - \eta(m))}{\Omega(m)} \right), \tag{6}$$

where

$$\eta(m) = \sum_{i=1}^{m-1} \sum_{p=i}^{m-1} \beta_p \left(\prod_{j=i+1}^{p} f_j \right), \tag{7}$$

$$\Omega(m) = \left(1 + \sum_{i=1}^{m-1} \prod_{j=i+1}^{m} f_j \right), \tag{8}$$

Substituting Eq. (6) in Eq. (3), we obtain the individual load fractions. From Fig. 3, we obtain the expression for the processing time as

$$T(\alpha, m) = \alpha_1 (C + E_1) + \theta_{cm} + \theta_{cp}, \tag{9}$$

Using $i = 1$ in Eq. (3) and substituting in Eq. (9), we obtain,

$$T(\alpha, m) = (\alpha_m M_1 + N_1)(C + E_1) + \theta_{cm} + \theta_{cp}, \tag{10}$$

where, α_m, M_1, and N_1 are as defined above.

It is worth mentioning at this juncture that given an m-processor system, with the inclusion of all the overheads, it may not be necessary that an optimal solution exists when one attempts to utilize all the m-processors. This behavior can be seen from Fig. 4(a). Here, we observe that as we tend to increase the number of processors, the processing time decreases. Also, we have shown the influence of $\eta(m)$ (given by Eq. (7)) which we will refer to as the *overhead factor*, in Fig. 4(b). From this figure, we see that the overhead factor $\eta(m)$ also increases as m increases. Also, we observe that for the speed parameters chosen, the maximum number of processors that can be utilized with the given sequence of load distribution is $m^* = 14$. Thus, beyond this m^*, optimal solution ceases to exist. This is due to the fact that the value of the overhead factor becomes *greater than one*, and hence, there will not be any gain in the time performance even if we attempt to utilize more processors beyond m^*. Therefore, as long as the value of $\eta(m)$ is less than one, we can utilize all the m processors to process the entire load in a minimum amount of time with this fixed sequence.

Thus, we see that, for the existence of an optimal load distribution using all the m-processors, a necessary and sufficient condition is given by,

$$\eta(m) = \sum_{i=1}^{m-1} \sum_{p=i}^{m-1} \beta_p \left(\prod_{j=i+1}^{p} f_j \right) < 1 \tag{11}$$

What we observe here is a restricted monotonic nature of the

processing time behavior. It may be verified that when the overheads are neglected in the model, the processing time monotonically decreases, as we tend to increase the number of processors. However, employing a realistic model, as in our analysis, we see a restricted monotonic behavior. Thus, given a fixed sequence, whenever an optimal solution using a k-processor system $p_1 \ldots p_k$ ceases to exist, then we may utilize a maximal subset of k', $k' < k$ processors in the same sequence involving processors $p_1 \ldots p_{k'}$ to obtain the optimal processing time. Of course, again we have to check the condition (11) for this set of k' processors for the existence of an optimal solution.

Another observation (Fig. 4(a)) is on the rate at which the processing time decreases. Beyond $m = 7$ or 8, we see that the rate of decrease of the processing time is not significant for an acceptable solution that is close to the optimal solution given by m^* processors. This in a way reduces the overhead processing by the system considerably, as it can be seen by the increase in the values of $\eta(m)$ for $m = 7$ and 8, respectively.

3.2. Concept of equivalent processor

Now, we shall introduce the concept of equivalent processor, which will be useful to prove a number of important results in the subsequent sections. The concept of equivalent processor has been introduced in the theory of scheduling divisible loads in a variety of situations [9]. We shall first derive a closed-form expression for the equivalent processor from the timing diagram shown in Fig. 5. Here, the basic idea is to collapse a set of processors and to form a single equivalent processor such that analysis of load distribution strategies is easier to perform.

Consider determining an equivalent processor p_{eq} for a set of processors $p_{q+1}, \ldots, p_{q+r}$. Let the total load assigned to this set of processors be σ_q. Let us denote the equivalent speed of these processors as $E(q+1, \ldots, q+r)$. From Fig. 5, we obtain,

$$\alpha_{q+r} E_{q+r} + \sigma_q C + r\theta_{\text{cm}} + \theta_{\text{cp}} = \sigma_q (E(q+1, \ldots, q+r) + C) + r\theta_{\text{cm}} + \theta_{\text{cp}} \tag{12}$$

which implies that,

$$E(q+1, \ldots, q+r) = \left(\frac{\alpha_{q+r}}{\sum_{i=q+1}^{q+r} \alpha_i} \right) E_{q+r}, \tag{13}$$

This can be simplified by using Eq. (3) to Eq. (8) with $i = q+1, \ldots, q+r$, to obtain a closed-form expression for the equivalent processor speed as,

$$E(q+1, \ldots, q+r) = E_{q+r} \left(\frac{1 - \frac{1}{\sigma_q} \sum_{i=1}^{r-1} N_{q+i}}{1 + \sum_{i=1}^{r-1} M_{q+i}} \right) \tag{14}$$

where

$$\sigma_q = \sum_{i=1}^{r} \alpha_{q+i}, \tag{15}$$

$$M_{q+i} = \prod_{j=i+1}^{r} f_{q+j}, \quad i = 1, 2, \ldots, r-1, \tag{16}$$

$$N_{q+i} = \sum_{k=i}^{r-1} \prod_{j=i+1}^{k} \beta_{q+k} f_{q+j}, \quad i = 1, 2, \ldots, r-1. \tag{17}$$

Now using the concept of equivalent processor we prove the following.

Lemma 1. *In an m-processor system with a fixed sequence of load distribution as $p_1, \ldots p_m$, if α_m given by Eq. (6) is strictly greater than zero, then α'_{m-1} given by Eq. (6) is also strictly greater than zero for the $(m-1)$ processor system, using processors from p_1 to p_{m-1} in the above fixed sequence.*

Proof. Since $\alpha_m > 0$, we have, $\eta(m) < 1$, which means that $\sum_{i=1}^{m-1} N_i < 1$. For a $(m-1)$-processor system with the same fixed sequence, we have, $\alpha'_{m-1} = (1 - \eta'(m-1))/(\Omega(m-1))$, where $\eta'(m-1) = \sum_{i=1}^{m-2} N'_i$. Rewriting $\eta(m)$ as,

$$\eta(m) = \eta'(m-1) + \beta_{m-1} \left(\sum_{i=1}^{m-2} \prod_{q=i+1}^{m-1} f_q + 1 \right), \tag{18}$$

we see that, $\eta'(m-1) < \eta(m) < 1$. Therefore, $\alpha'_{m-1} > 0$. Hence the proof. □

The above lemma demonstrates the fact that whenever an m-processor optimal solution exists, then the optimal solution for any k-processor, $k < m$ system also exists.

Now, using the result of Lemma 1, we shall now demonstrate the monotonic behavior of the processing time for an m-processor fixed sequence, when it exists.

Theorem 1. *In an m-processor system with a fixed sequence of load distribution as $p_1, \ldots, p_m$, if α_m given by Eq. (6) is greater than zero, then $T(\alpha, m) < T(\alpha', m-1)$, where $T(\alpha, m)$ and $T(\alpha', m-1)$ are the optimal processing times in m and $(m-1)$ processors systems, respectively.*

Proof. Since $\alpha_m < 1$, from Lemma 1, we know that the optimal solutions for m-processor and $(m-1)$-processor

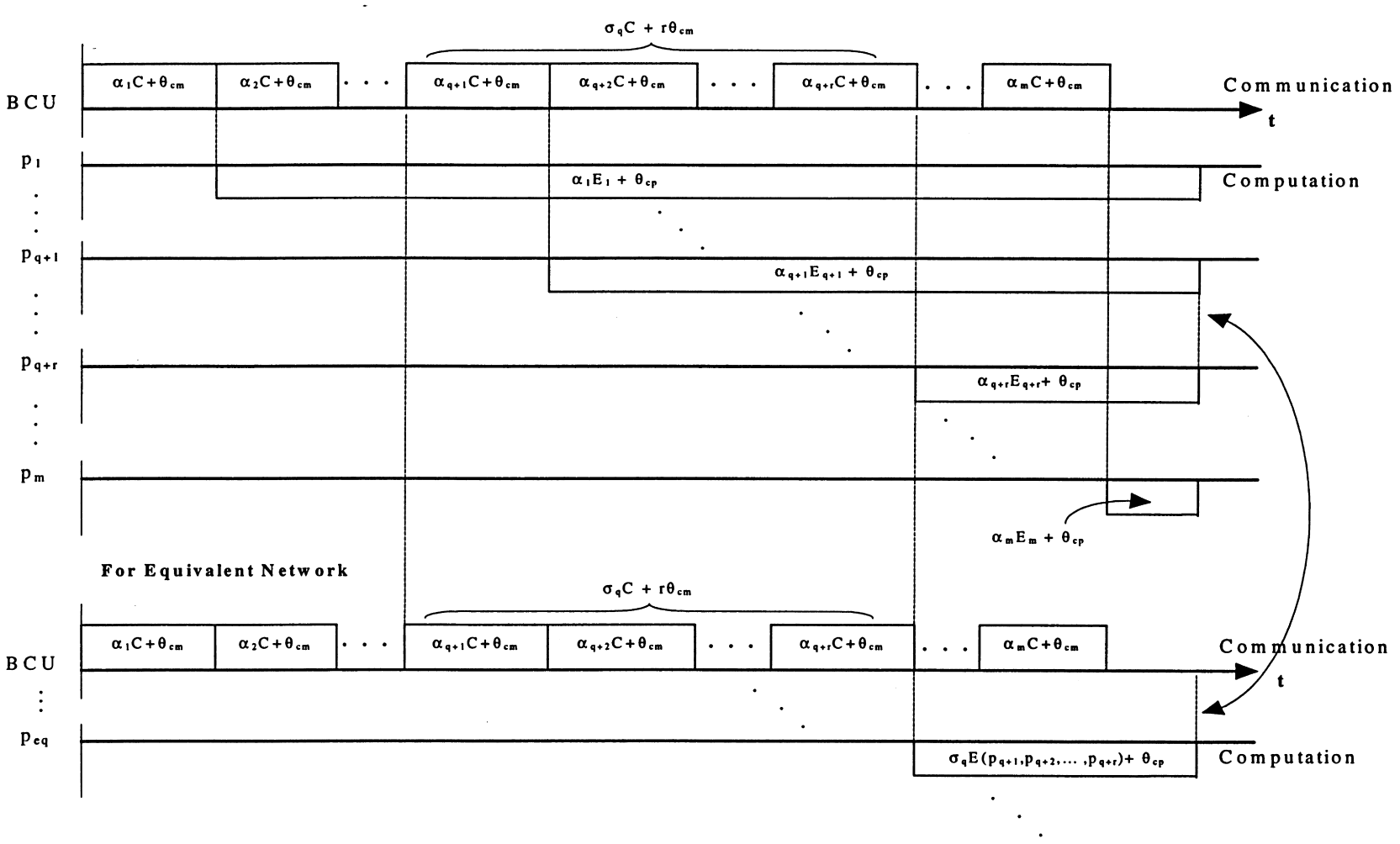
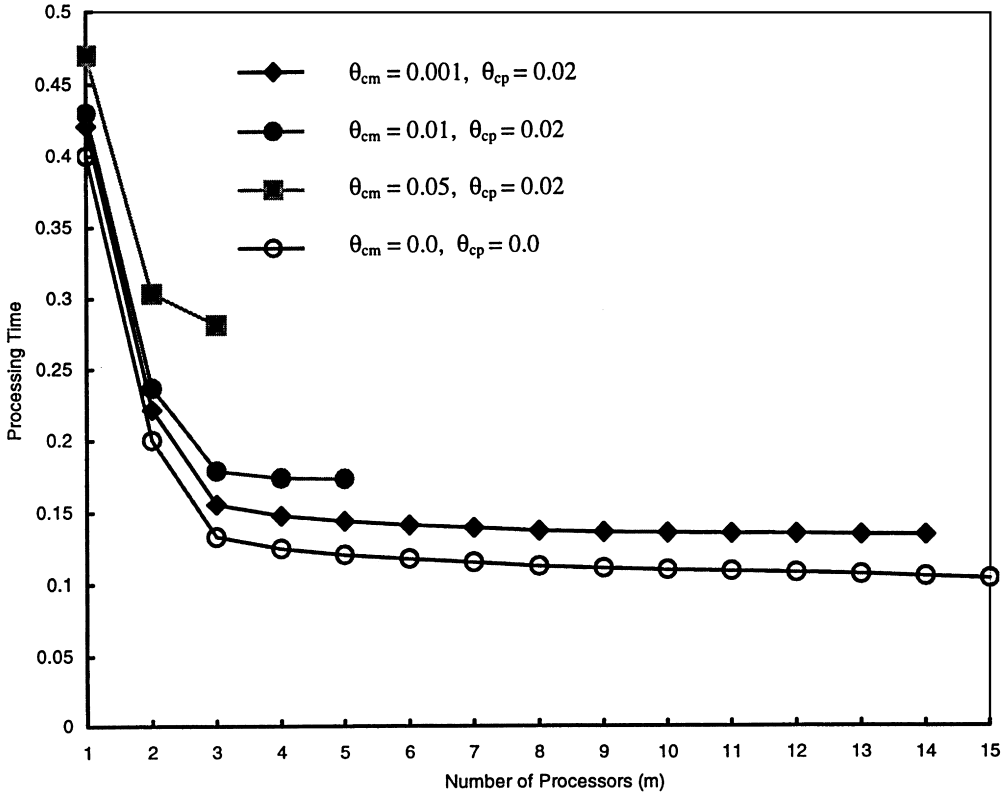


Fig. 5. Timing diagram showing the processor equivalence.



Speed Parameters:

C	0.1
E_1	0.3
E_2	0.2
E_3	0.1
E_4	0.4
E_5	0.6
E_6	0.7
E_7	0.8
E_8	0.5
E_9	0.9
E_{10}	1.1
E_{11}	1.3
E_{12}	1
E_{13}	0.6
E_{14}	0.5
E_{15}	0.3

Fig. 6. Processing time vs. number of processors for different overhead parameters (heterogeneous system).

systems exist. Let us denote the fixed m-processor sequence $p_1,\ldots,p_m$, as (p_{eqI},p_m), where p_{eqI} is the equivalent processor for the set of processors $(p_1,\ldots,p_{m-1})$. Let the speed parameter $E(p_1,p_2,\ldots,p_{m-1})$ of p_{eqI} be denoted as E_{eqI}. For the same fixed sequence with $(m-1)$ processors we denote the equivalent processor for the set $(p_1,p_2,\ldots,p_{m-1})$ as (p'_{eqI}), and denote its speed parameter asE'_{eqI}. The expressions for the $T(\alpha,m)$ and $T(\alpha',m-1)$ are given by,

$$T(\alpha,m)=(1-\alpha_m)(E_{\text{eqI}}+C)+\theta_{\text{cp}}+(m-1)\theta_{\text{cm}}, \quad (19)$$

$$T(\alpha',m-1)=(E'_{\text{eqI}}+C)+\theta_{\text{cp}}+(m-1)\theta_{\text{cm}} \quad (20)$$

Now, evaluating

$$T(\alpha,m)-T(\alpha',m-1)=(1-\alpha_m)E_{\text{eqI}}-E'_{\text{eqI}}-\alpha_m C \quad (21)$$

Now, using Eq. (14) for the respective equivalent processor speeds, we immediately observe that $(1-\alpha_m)E_{\text{eqI}}<E'_{\text{eqI}}$. Therefore, $T(\alpha,m)<T(\alpha',m-1)$. Hence the proof. □

The above theorem signifies the fact that whenever the value of the overhead factor is less than one for an m-processor system with a given sequence, then the optimal processing time by the m-processor system is less than the optimal processing time with $(m-1)$-processor system following the same sequence. Thus, it clearly demonstrates the monotonic behavior of the processing time.

3.3. Homogeneous system

As a special case, for a homogeneous system ($w_i=w$, and hence, $E_i=E$, $i=1,\ldots,m$), the overhead factor $\eta(m)$ given by Eq. (7) can be written as,

$$\eta(m)=\frac{\beta}{(1-f)^2}[(1-f)(m-1)+f^m-f] \quad (22)$$

and the value of the denominator $\Omega(m)$ in Eq. (8) can be written as,

$$\Omega(m)=\frac{1-f^m}{1-f} \quad (23)$$

where, $f=(E+C)/E$, $\beta=\theta_{\text{cm}}/E$. Substituting Eqs. (22) and (23) in Eq. (10), we obtain,

$$T(\alpha,m)=(C+E)\left[f^{m-1}\left(\frac{1-\eta(m)}{\Omega(m)}\right)+\beta\left(\frac{1-f^{m-1}}{1-f}\right)\right]+\theta_{\text{cm}}+\theta_{\text{cp}} \quad (24)$$

In the homogeneous case, we see that the *necessary and sufficient* condition for the existence of an m-processor optimal processing time, given by Eq. (11), can be simplified using Eq. (22) to yield the following condition:

$$g(f)<1-\beta(m-1) \quad (25)$$

where, $g(f)=\beta f^m-f^2+(2-m\beta)f$. Fig. 6(a) shows the

behavior of the processing times for two cases, using the proposed strategy and the equal partitioning strategy, when we increase the number of processors. Similar to the heterogeneous case, here, we note that beyond $m^* = 15$, the optimal processing time ceases to exist, as the value of $\eta(16) > 1$, as shown in Fig. 6(b). It may be verified that the results of Lemma 1 and Theorem 1 hold for this case too.

Lemma 2. *In a homogeneous system with m-processors, under the proposed strategy, the optimal processing time utilizing all the m-processors, if it exists, is strictly less than total processing time obtained by using the equal partition strategy.*

Proof. For equal partition scheme, the total load is partitioned into m identical portions of size L/m each. Therefore, from the timing diagram, the processing time using equal partition scheme is, $T_{eq} = LC + m\theta_{cm} + ((EL)/m) + \theta_{cp}$. In our proposed strategy, we distribute the load portions among the processors $(p_1, p_2, \ldots p_m)$ in this order. It can be seen that the individual size of the load fractions distributed among the processors strictly decrease from p_1 to p_m, as the system is homogeneous. Now, in the case of equal partition strategy, it may be noted that the processor p_m contributes to the total processing time, and hence it is sufficient to consider the finish time of p_m. The optimal load distribution given by our proposed strategy, is such that $\alpha_i > \alpha_m, \forall i = 1, \ldots, m-1$. Summing up the LHS and RHS for all i, we obtain, $m\alpha_m < \sum_{i=1}^{m} \alpha_i$, which implies that $\alpha_m < L/m$. Therefore, the processing time using our strategy is, $LC + m\theta_{cm} + \alpha_m E + \theta_{cp}$. Because $\alpha_m < L/m$, the result follows. Hence the proof. □

4. Processing time minimization using the concept of sequencing

In this section, we shall first demonstrate through a numerical example that the time performance of the system can be altered by changing the sequence of load distribution. This behavior was never exhibited by this system when the modeling did not take into account of the overheads [9]. Clearly, it would be interesting to determine which sequence would be optimal, if it exists, among all possible load distribution sequences.

4.1. Motivating example

Example 2. Consider a bus network with $m = 5$ processors, with the following speed parameters: $C = 0.200$, $E_1 = 0.1$, $E_2 = 0.2$, $E_3 = 2.1$, $E_4 = 3.7$, $E_5 = 2.9$, and let the total load $L = 1$.

Let $\theta_{cm} = 0.010$, $\theta_{cp} = 0.010$. We first distribute load in the following order: $(p_1, p_2, p_3, p_4, p_5)$, where the load is first assigned to p_1, then to p_2, and so on till p_5 in the above order. Now, using the closed-form solution (10), we obtain the optimal processing time as $T(\alpha, 5) = 0.262$, where the optimal load distribution is given by, $\alpha = (0.808, 0.177, 0.011, 0.003, 0.001)$. Alternatively, we may distribute load to the processors in the following order: $(p_5, p_4, p_3, p_2, p_1)$. That is, the load is first assigned to p_5 and then to p_4, and so on till p_1 in the above order. Now, again using the closed-form solution (10), we obtain the optimal finish time for this sequence of load distribution as $T(\alpha', 5) = 0.288$, where the corresponding load distribution is given by, $\alpha' = (0.086, 0.062, 0.095, 0.474, 0.283)$. Thus, we observe that the processing time depends on the *sequence* or the *order* in which the load is distributed among the processors. Also, note that $T(\alpha, 5) < T(\alpha', 5)$.

Thus, we see that the concept of sequencing is applicable only when we include the overheads in the model and is more relevant to practical situations. We observe that due to the presence of overheads, when the system has m-processors, we have $m!$ different load distribution sequences possible. However, from the results of the previous section, we note that not all m-processor sequences need to exist. This is due to the presence of very large overhead factors. We represent the space of all m-processor sequences as Ψ_m. In general, we can partition the entire space Ψ consisting of all possible sequences into m different sub-spaces, each represented by Ψ_k, $k = 1, \ldots, m$, where Ψ_k is the space of all k-processor sequences. We refer to the space Ψ as the *sequence space* and refer to a space Ψ_k as a *k-dimensional* space. It may be observed that the total number of sequences with k-processors in an m-processor system, is given by, $\binom{m}{k}k!$. The factor $\binom{m}{k}$ comes due to the fact that out of m-processors we can choose k-processors in $\binom{m}{k}$ ways.

For the purpose of notational convenience, we introduce the following. Note that this change of notation is to include the influence of the load distribution sequence on the processing time and also for the clarity in the analysis. It may be recalled that the expressions for the following quantities are as derived in Section 3.

1. Let $\psi_j(k)$ indicate the jth k-processor sequence in some specified order of load distribution. We represent such a sequence as $\psi_j(k) = (p_1, p_2, \ldots, p_k)$, where the first load fraction α_1 is assigned to p_1, the second load fraction α_2 is assigned to $p_2, \ldots$, and the last portion of load is assigned to the kth processor.
2. Let $T(\alpha, k, \psi_j(k))$ denote the optimal processing time for the sequence $\psi_j(k)$, where $\psi_j(k) \in \Psi_k$, is as defined above.
3. Let $\eta(\psi_j(k), k)$ denote the overhead factor with k-processors involved in processing and $\psi_j(k) \in \Psi_k$.

The expression for $T(\alpha, k, \psi_j(k))$ is given by Eq. (10). The expression for $\eta(\psi_j(k), k)$ is given by Eq. (7) and the necessary and sufficient condition for this k-processor sequence to exist is given by Eq. (11), with $m = k$. Also,

we denote $\beta_{i,j}(m) = \theta_{cm}/E_j$ and $f_{i,j}(m) = (C + E_j)/E_{j-1}$ in the ith m-processor sequence.

In the subsequent sections, we present some important non-trivial properties of the sequencing that will be used to identify an optimal sequence, if it exists in Ψ.

4.2. Optimal sequencing

In this section, we shall first present some important intermediate results that are useful in identifying an optimal sequence. The proofs of the lemmas and the theorems can be found in the Appendix A.

Lemma 3. *Let* $\psi_1(m-1) = (p_1, p_2, \ldots, p_k, p_{k+1}, \ldots, p_{m-1})$, *and denote the new sequence obtained by swapping* p_k *and* p_{k+1}, *as* $\psi_2(m-1) = (p_1, p_2, \ldots, p_{k+1}, p_k, \ldots, p_{m-1})$. *Then* $\eta(\psi_1(m), m) - \eta(\psi_2(m), m) = \eta(\psi_1(m-1), m-1) - \eta(\psi_2(m-1), m-1)$, *for* $k = 1, 2, \ldots, m-3, m-2$, *where* $\psi_1(m) = (\psi_1(m-1), p_m)$ *and* $\psi_2(m) = (\psi_2(m-1)p_m)$, *respectively*.

This is an important result that will be used in our proof of an optimal sequencing theorem.

Lemma 4. *Consider the sequences* $\psi_1(m)$ *and* $\psi_2(m)$ *in Lemma 3. Let* $m \geq 2$, *and* $E_k < E_{k+1}$, *i.e. the processor* p_k *is faster than processor* $p_{k=1}$. *Then,* $\eta(\psi_1(m), m) > \eta(\psi_2(m), m)$.

The lemma signifies the fact that when a pair of adjacent processors in an increasing order of speeds are swapped in a sequence, the resulting sequence has an overhead factor that is strictly less than the overhead factor for the original sequence.

Lemma 5. *If* $\eta(\psi_j(m), m) < 1$, *where* $\psi_j(m) = (p_1, p_2, \ldots, p_m)$ *such that* $E_l < E_{l+1}$ *for all* $l = 1, 2, \ldots, m-1$, *then* $\eta(\psi_i(k), k) < 1$, *where* $k \leq m$, *for all* $\psi_i(k) \in \Psi$.

The lemma shows that, if the fast sequence exists, then the overhead factor for the fast sequence has the maximum value than the overhead factor for all other sequences in Ψ. This important property of the fast sequence will be used in proving the optimal sequence theorem later.

Lemma 6. *Consider* $\psi_1(m)$ *and* $\psi_2(m)$ *as defined in Lemma 3. The respective load distributions are,* α *and* α' *given by,* $\alpha = \sum_{i=1}^{m} \alpha_i$ *and* $\alpha' = \sum_{i=1}^{m} \alpha'_i$. *Denote,* $\alpha_{\mathrm{I}} = \sum_{i=1}^{k-1} \alpha_i$, $\alpha_{\mathrm{II}} = \alpha_k + \alpha_{k+1}$, $\alpha_{\mathrm{III}} = \sum_{i=k+2}^{m} \alpha_i$, $\alpha'_{\mathrm{I}} = \sum_{i=1}^{k-1} \alpha'_i$, $\alpha'_{\mathrm{II}} = \alpha'_k + \alpha'_{k+1}$, $\alpha'_{\mathrm{III}} = \sum_{i=k+2}^{m} \alpha'_i$. *If* $\alpha_{\mathrm{II}} < \alpha'_{\mathrm{II}}$ *and* $E_k < E_{k+1}$, *then* $E(p_k, p_{k+1})\alpha_{\mathrm{II}} < E'(p_{k+1}, p_k)\alpha'_{\mathrm{II}}$, *where* $E(p_k, p_{k+1})$ *is the equivalent speed of processors* p_k *and* p_{k+1}, *and* $E'(p_k, p_{k+1})$ *is the equivalent speed of processors* p_{k+1} *and* p_k.

Proof. Using the closed-form for the equivalent processor (14) for the respective distributions α and α', and evaluating the expression $E(p_k, p_{k+1})\alpha_{\mathrm{II}} - E(p_{k+1}, p_k)\alpha'_{\mathrm{II}}$, we see that the proof follows immediately. Hence the proof. □

Theorem 2. *If* $\eta(\psi_j(m), m) < 1$, *then* $T(\alpha, m, \psi_j(m)) < T(\alpha', m, \psi_{j'}(m))$, *where* $\psi_j(m) = (p_1, p_2, \ldots, p_k, p_{k+1}, \ldots, p_m)$, *and* $\psi_{j'}(m) = (p_1, p_2, \ldots, p_{k+1}, p_k, \ldots, p_m)$, *and* $E_k < E_{k+1}, k \in \{1, 2, \ldots, m-1\}$.

This theorem shows that the optimal processing time using sequence with a fast pair of processors p_k and p_{k+1} is less than the optimal processing time when the processors p_k and p_{k+1} in that sequence are swapped. This property is critical and will be used to prove the following *optimal sequence theorem*.

Theorem 3 (Optimal sequence theorem). *If* $\eta(\psi_j(m), m) < 1, T(\alpha, m, \psi_j(m)) = \min_{\psi_i(k) \in \Psi}\{T(\alpha, k, \psi_i(k))\}$, *where the sequence* $\psi_j(m) = (p_1, p_2, \ldots, p_k, p_{k+1}, \ldots, p_m), E_i < E_{i+1}$, *for all* $i = 1, 2, \ldots, m-1$, *and* $\psi_i(k) \in \Psi_k, k = 1, \ldots, m$. *The sequence* $\psi_j(m)$ *defined above is referred to as fast sequence.*

The result of this theorem is an important contribution to this domain of research. The importance of the theorem lies in eliciting the property of sequencing in the case of bus networks. It shows that, given an m-processor system, if all the processors can be utilized by following a sequence in which the processor speed decrease then such a sequence results in an optimal time performance. A brief discussion on this result is presented in Section 6. Since the presence of the overhead factor affects processing time, in general, it may be possible that an m-processor fast sequence may not exist at all. In the following numerical example, we demonstrate this non-trivial observation.

Example 3. Consider a system with $m = 5$ processors and the following speed parameters: $C = 0.4$, $\theta_{cp} = 0.005$, $\theta_{cm} = 0.01$, $E_1 = 0.1$, $E_2 = 0.3$, $E_3 = 0.4$, $E_4 = 0.7$, $E_5 = 1.0$. First, we choose the sequence $\psi_1(5) = (p_1, p_2, p_3, p_4, p_5)$ to distribute. Note that $\psi_1(5)$ is a fast sequence. But, $\eta(\psi 1(5),5)$ given by Eq. (7) is greater than one, and hence, we observe that a fast sequence with all the five processors does not exist.

Now, consider the sequence $\psi_2(5) = (p_3, p_5, p_4, p_1, p_2)$ and it may be verified that $\eta(\psi_2(5), 5) < 1$ and using Eq. (10) we obtain the optimal processing time as, $T(\alpha, 5, \psi_2(5)) = 0.4575$. Instead, we may choose the fast sequence with only $m-1$ processors $\psi_1(4)$ to distribute the load. Using Eq. (7), we observe that $\eta(\psi_1(4), 4) = 0.925 < 1$, and obtain $T(\alpha', 4, \psi_1(4)) = 0.4458$. Note that $T(\alpha', 4, \psi_1(4)) < T(\alpha, 5, \psi_2(5))$. Therefore, we can draw the conclusion that $T(\alpha, m, \psi(m))$, where $\psi(m)$ is not the fast sequence in m-dimension, need not be less than

$T(\alpha', (m-1), \psi'(m-1))$, where $\psi'(m-1)$, is the fast sequence in Ψ_{m-1}.

5. Discussions

The contributions in this paper are multifold and are novel to this domain of research. Scheduling CVIP data onto a network of processors employing a realistic modeling has been presented. So far in the literature pertaining to parallelizing the CVIP data, focus was in evaluating the speed-up achieved on different parallel architectures, such as hypercubes, and meshes, for a wide variety of image processing and computer vision data processing applications. These attempts used data partitioning schemes such as row, column, and block, to schedule the data onto the processors in a given architecture. As mentioned before, most of the work in this parallel processing of CVIP data focuses on the processing phase alone rather than addressing the problem starting from the initial data distribution phase to the final processing phase.

This research essentially addresses this question and as a case study, considers a bus topology for evaluating the performance of the strategy proposed in this paper. All the literature in the domain of processing of image data on multiprocessors attempt to employ a partition scheme that tends to balance the load among all the processors. This modeling will perform superior and is a natural choice, as the effect of communication delays and the associated overheads are insignificant to consider in the case of parallel processing architectures. However, in a distributed system, the delays associated with the communication processing and overhead processing cannot be neglected [4,11] and have significant influence on the time performance. Figs. 4 and 6–9 reveal this behavior. The closed-form solution for the processing time demonstrates the influence of the overheads involved and Theorem 1 presents a restricted monotonic behavior of the processing time. A necessary and sufficient condition to utilize a given set of *m*-processors is derived. Thus, for a given set of processors, there is an upper bound on the number of processors that should be utilized to obtain the optimal processing time, when the overhead components are included.

For the case when the overheads are negligible, we apply the closed-form solution (10) by setting the $\theta_{cm} = \theta_{cp} = 0$, and obtain,

$$T(\alpha, m) = \frac{M_1(C + E_1)}{\Omega(m)} \tag{26}$$

where, M_1 and $\Omega(m)$ are as defined by Eqs. (4) and (8).

Fig. 7 shows the behavior of the processing time when we alter the overhead parameters, including the case without overheads. Apart form the closed-form solution for the processing time, the concept of sequencing is introduced. It was demonstrated that when the overheads are present, processing time is influenced when the sequence of data distribution is varied. This behavior was seldom exhibited when the model neglects the associated overhead components $\theta_{cm} = \theta_{cp} = 0$). A number of non-trivial properties related to this concept of sequencing were derived. Based on these preliminary properties, we prove that, if the overhead factor for the *m*-processor *fast sequence* (as defined in Section 4) is less than one, then this fast sequence yields the optimal processing time. In case, the overhead factor of the fast sequence violates the condition (11), we may choose some other sequences that satisfy condition (11) to use all the *m*-processors. Note that the search space Ψ_m is very large (*m*! sequences), when *m* is large. However, from Example 3, we observe that when the fast sequence in Ψ_m does not exist, we note that some distribution sequences using all the *m*-processors may be inferior to the fast sequence in Ψ_{m-1}. Further, this does not guarantee that the fast sequence using $(m-1)$ processors will yield an optimal processing time. This is a very crucial observation to make, especially if the design requirements depend on the utilization of certain number of processors. This behavior of the processing time actually prevents from making naive decisions on the choice of the number of processors and the sequence in which the load to be distributed, as these two quantities behave in a somewhat "push–pull" fashion.

Under these circumstances, it may be possible to design a number of greedy algorithms to yield sub-optimal solutions. However, unless such a greedy algorithm converges to a sub-optimal solution in a polynomial time, the search could be CPU time intensive. Also, the sub-optimal solution that is generated by these algorithms must lie within the acceptable limits. We suggest the following greedy algorithm that generates a sub-optimal solution that is suited for most of the applications. The rationale behind this algorithm lies in making use of a maximal set of processors in the order of their decreasing speeds.

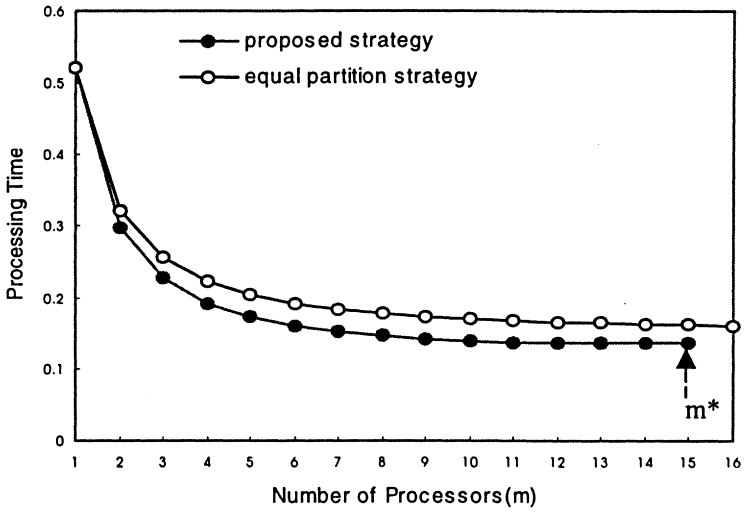
5.1. Greedy algorithm

Let Ψ_{fast} be the set of all fast sequences in Ψ. Note that the number of fast sequences in Ψ is *m*, as in each Ψ_k, $k = 1, \ldots, m$, we have one fast sequence.

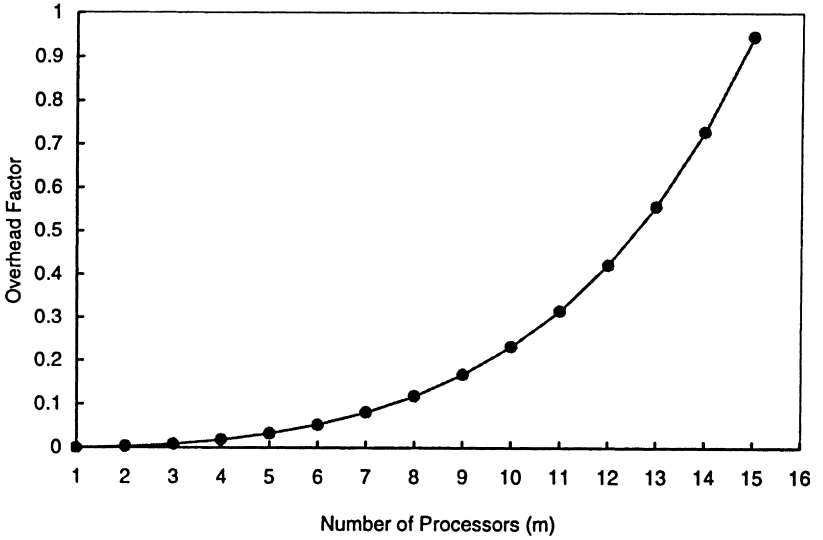
For $(k = m; k <= 1; k = k - 1)$
{
Let $\psi_{fast}(k) \in \Psi_{fast}$ and let $\eta(\psi_{fast}(k), k)$ be the associated overhead factor.
Then,
$\eta(\psi_{fast}(k), k) = \sum_{i=1}^{k-1} \sum_{p=i}^{k-1} \beta_p(\Pi_{j=i+1}^{p} f_i)$;
If $(\eta(\psi_{fast}(k), k) < 1)$
{
$r = k$;
break;
}
}

Sub_optimal_solution= $T(\alpha, r, \psi_{fast}(r))$;

Note that the values of $\eta(\psi_{fast}(k), k)$ for every $k \in$



(a)

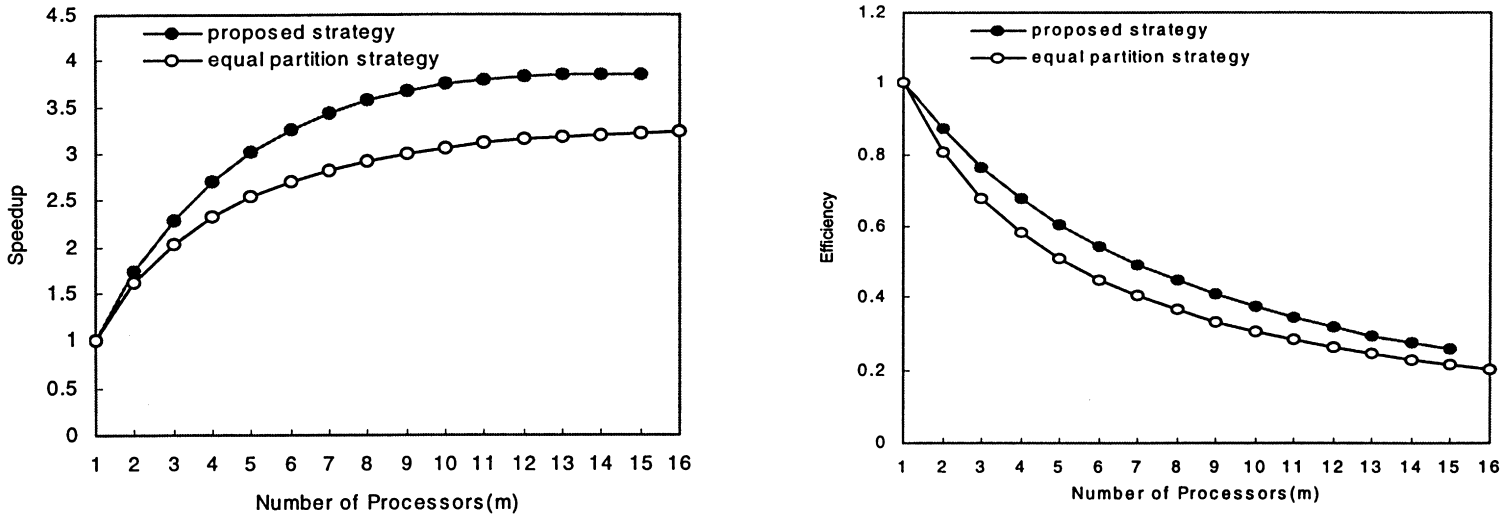


(b)

Speed Parameters:

C	E	θ_{cm}	θ_{cp}
0.1	0.4	0.001	0.02

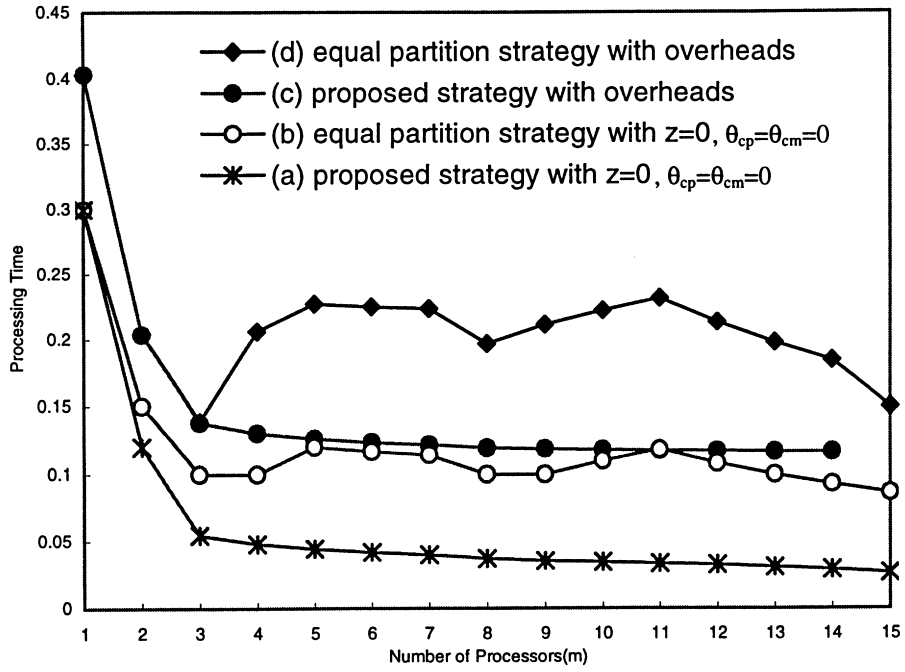
Fig. 7. Behavior of the processing time and the overhead factor with respect to *m* (homogeneous system).



Speed parameters:

C	E	θ_{cm}	θ_{cp}
0.1	0.4	0.001	0.02

Fig. 8. Behavior of speedup and efficiency with respect to *m* (homogeneous system).



Speed parameters:

C	0.1
θ_{cm}	0.001
θ_{cp}	0.02
E_1	0.3
E_2	0.2
E_3	0.1
E_4	0.4
E_5	0.6
E_6	0.7
E_7	0.8
E_8	0.5
E_9	0.9
E_{10}	1.1
E_{11}	1.3
E_{12}	1
E_{13}	0.6
E_{14}	0.5
E_{15}	0.3

Fig. 9. Behavior of the processing time with respect to m (heterogeneous system).

$\{1, 2, \ldots, m\}$ can be precomputed. Hence, we see that the complexity of the algorithm is $O(m)$. Further, the algorithm considers only the fast sequences in every Ψ_k, $k = 1, \ldots, m$. When condition (11) is satisfied for a fast sequence in some dimension $r \leq m$, the algorithm terminates, thus limiting our search to at most m iterations. The concept of sequencing can also be used to make wise decisions on the choice of the network and the number of processors on which the load may be processed. Even if one uses a large network having very low average overhead delays, an inadvertent choice of a sequence, when a fast sequence fails to exist, may degrade the time performance than choosing a small sized network with large average delays. This is one of the important trade-off relationships between the number of processors to be used, the combined influence of the overhead components and the sequence of load distribution.

We shall now demonstrate the behavior of the strategy with respect to some of the standard performance metrics [17], such as the speedup factor, denoted by $S(m)$, and the efficiency (or utilization), denoted by $E(m)$. These are as defined below

$$S(m) = \frac{\text{time to finish a task in a uniprocessor}}{\text{time to finish a task in an } m \text{ processors system}}, \tag{27}$$

$$E(m) = \frac{S(m)}{m} \times 100\%. \tag{28}$$

Fig. 8(a) and (b) shows the behavior of speedup and efficiency with respect to the number of processors in a *homogeneous system*. We clearly observe that the performance of the proposed strategy is superior with respect to these metrics. However, such a promise does not take us far when these overheads tend to dominate. Thus, with the inclusion of overheads, the speedup for the proposed strategy is restricted. From Fig. 8(a), we note that the speedup for the proposed strategy reaches its peak value at $m^* = 15$. Also, it may be observed that as we tend to use more number of processors (close to m^*), the rate of increase in the speed-up is not significant, and the performance tends to saturate. However, from Fig. 8(b), we observe that the rate of decrease in efficiency is quite drastic when we move close to m^*. Nevertheless, from our analysis with realistic modeling, we observe that this performance is acceptable and shows a superior performance when compared to an equal partition strategy.

In Fig. 9, we show the behavior of the processing time with respect to the number of processors in a *heterogeneous system* when the sequence of load distribution is fixed. As expected, when communication delays and the associated overheads are ignored, due to the heterogeneity of the system, the time performance using the equal partition scheme fluctuates (see curve (b)), as the performance depends on sequencing. However, the time performance using the proposed strategy (see curve (a)) is much better than that of using equal partition strategy even when the communication delays and the associated overheads are neglected. Note that to obtain curve (a) we use the closed-form solution (ten) and substituting $C = 0$, $\theta_{cm} = \theta_{cp} = 0$. Also, we observe that, with more processors involved in

processing, the processing time monotonically decreases. When we consider curves (c) and (d) (in which we account for all the delays and the associated overheads), the time performance exhibited by these strategies has the same behavior as that of case (a) and (b), however, the magnitudes are higher in the present case, as expected. Also, under the influence of these overheads, using the strategy proposed, there exists an m^*, beyond which the optimal processing time ceases to decrease with respect to m.

6. Conclusions

The problem of scheduling CVIP data on bus-based architectures is studied. It was argued that the CVIP is a right candidate to exploit data parallelism, and hence, the recently evolved *divisible load paradigm* was used to schedule these data for processing in a minimum amount of time. The use of divisible load analysis promises a richer set of results and tackles the problem of scheduling under realistic constraints. We use a widely accepted mathematical mode that captures all the delay and processing components and the associated overheads. With the aid of this model, we propose a load distribution strategy and derived the closed-form solution for the optimal processing time. Using the closed-form solutions, we determine a necessary and sufficient condition for the existence of the optimal processing time using all the processors in the system. A restricted monotonic behavior of the time performance with respect to the number of processors is demonstrated for both the homogeneous and the heterogeneous systems.

The influence of the overhead components on the processing time when the load distribution sequence is altered is then rigorously studied. A number of useful and interesting properties have been derived and we proved an *optimal sequencing* theorem. All possible cases have been compared against the commonly used load partitioning schemes in the literature [3,5,7] under some standard performance metrics. Although one of the significant contributions in this paper is the processing time minimization through the concept of sequencing, it still remains an open problem to be solved when an optimal sequence fails to exist. We proposed a greedy algorithm to seek a sub-optimal sequence that delivers an acceptable performance. The greedy approach suggests a solution that is given by a *fast sequence* using a maximal set of $k \leq m$ processors in the system.

The impact of divisible load paradigm, although is demonstrated for processing CVIP data, in general, can suit applications in which large volume of data needs to be processed with high speed requirements. Some applications cited in Ref. [3] form natural candidates for this kind of treatment. With the problem formulation presented in this paper, it would be interesting to conduct a rigorous study on load distribution problems on other networks that are commonly found in practice.

Appendix A

We present the proofs for all the Lemma 3–5 and Theorems 2 and 3, stated in Section 4.2.

Proof of Lemma 3. From Eq. (7), we obtain,

$$\eta(\psi_1(m), m) = \eta(\psi_1(m-1), m-1) + \sum_{i=1}^{m-2} (\beta_{1,m-1}(m) \prod_{q=i+1}^{m-1} f_{1,q}(m)) + \beta_{1,m-1}(m) \tag{A1}$$

Similarly,

$$\eta(\psi_2(m), m) = \eta(\psi_2(m-1), m-1) + \sum_{i=1}^{m-2} (\beta_{2,m-1}(m) \prod_{q=i+1}^{m-1} f_{2,q}(m)) + \beta_{2,m-1}(m) \tag{A2}$$

Note that when $k = m-2$, $\beta_{1,m-1}(m) \neq \beta_{2,m-1}(m)$, and when $k \neq m-2$, $\beta_{1,m-1}(m) = \beta_{2,m-1}(m)$. Now, we shall consider these two cases separately.

Case $k = 1, 2, \ldots, m-3$: We denote,

$$M_{1,i}(m) = \prod_{q=i+1}^{m-1} f_{1,q}(m), \text{ and } M_{2,i}(m) = \prod_{q=i+1}^{m-1} f_{2,q}(m)$$

for $i = 1, 2, \ldots, k, k+1, \ldots, m-1$

and note that $M_{1,i}(m) = M_{2,i}(m)$, for $i = 1, 2, \ldots, k-1$, $k+2, \ldots, m-2$. When $i = k$ and $i = k+1$, we have,

$$M_{1,k}(m) = \prod_{q=k+1}^{m-1} f_{1,q}(m) = f_{1,k+1}(m) f_{1,k+2}(m) M_{1,k+3}(m) \tag{A3}$$

$$M_{1,k+1}(m) = \prod_{q=k+2}^{m-1} f_{1,q}(m) = f_{1,k+2}(m) f_{1,k+3}(m) \tag{A4}$$

$$M_{2,k}(m) = \prod_{q=k+1}^{m-1} f_{2,q}(m) = f_{2,k+1}(m) f_{2,k+2}(m) M_{2,k+3}(m) \tag{A5}$$

$$M_{2,k+1}(m) = \prod_{q=k+2}^{m-1} f_{2,q}(m) = f_{2,k+2}(m) M_{2,k+3}(m). \tag{A6}$$

From the above equations, we obtain, $M_{1,k}(m) + M_{1,k+1}(m) = M_{2,k}(m) + M_{2,k+1}(m)$. Therefore, $\sum_{i=1}^{m-2} M_{1,i}(m) = \sum_{i=1}^{m-2} M_{2,i}(m)$. Since $k \neq m-1$ and $k \neq m-2$, $\beta_{1,m-1}(m) = \beta_{2,m-1}(m)$. Subtracting $\eta(\psi_2(m), m)$ from $\eta(\psi_1(m), m)$, the

result follows. That is,

$$\eta(\psi_1(m),m)-\eta(\psi_2(m),m)$$
$$=\eta(\psi_1(m-1),m-1)-\eta(\psi_2(m-1),m-1)$$
$$+\beta_{1,m-1}(m)\left(\sum_{i=1}^{m-2}M_{1,i}(m)+1\right)$$
$$-\beta_{2,m-1}(m)\left(\sum_{i=1}^{m-2}M_{2,i}(m)+1\right)$$
$$=\eta(\psi_1(m-1),m-1)-\eta(\psi_2(m-1),m-1)$$

Case $k=m-2$: We observe that,

$$\beta_{1,m-1}(m)f_{1,m-2}(m)f_{1,m-1}(m)=\beta_{2,m-1}(m)f_{2,m-2}(m)f_{2,m-1}(m), \tag{A7}$$

$$\beta_{1,m-1}(m)f_{1,m-1}(m)+\beta_{1,m-1}(m)$$
$$=\beta_{2,m-1}(m)f_{2,m-1}(m)+\beta_{2,m-1}(m) \tag{A8}$$

And also,

$$\prod_{q=i+1}^{m-3}f_{1,q}(m)=\prod_{q=i+1}^{m-3}f_{2,q}(m),\quad i=1,2,\ldots,m-4. \tag{A9}$$

Thus, we obtain,

$$\sum_{i=1}^{m-2}\left(\beta_{1,m-1}(m)\prod_{q=i+1}^{m-1}f_{1,q}(m)\right)+\beta_{1,m-1}(m)$$
$$=\sum_{i=1}^{m-2}\left(\beta_{2,m-1}(m)\prod_{q=i+1}^{m-1}f_{2,q}(m)\right)+\beta_{2,m-1}(m). \tag{A10}$$

Hence, from Eqs. (A1), (A2) and (A10), we obtain $\eta(\psi_1(m),m)-\eta(\psi_2(m),m)=\eta(\psi_1(m-1),m-1)-\eta(\psi_2(m-1),m-1)$. Hence the proof. □

Proof of Lemma 4. We shall prove the lemma by using induction. When $m=2$, we have, $\psi_1(2)=(p_1,p_2)$, $\psi_2(2)=(p_2,p_1)$, and $E_1<E_2$. From Eq. (7), we obtain, $\eta(\psi_1(2),2)=\theta_{cm}/E_1$, and $\eta(\psi_2(2),2)=\theta_{cm}/E_2$. Since $E_1<E_2$, we obtain $\eta(\psi_1(2),2)>\eta(\psi_2(2),2)$. Hence the claim is true. Let the claim be true for $m=r$, which means, $\eta(\psi_1(r),r)>\eta(\psi_2(r),r)$, where, $\psi_1(r)=(p_1,p_2,\ldots,p_k,p_{k+1},\ldots,p_r)$, and $\psi_2(r)=(p_1,p_2,\ldots,p_{k+1},p_k,\ldots,p_r)$, respectively. We shall show that our claim holds for $m=r+1$. Following are the two cases to consider.

Case $k=1,2,\ldots,r-1$: We denote $\psi_1(r+1)=(\psi_1(r),p_{r+1})$, and $\psi_2(r+1)=(\psi_2(r),p_{r+1})$. From Lemma 3, we observe that $\eta(\psi_1(r+1),r+1)-\eta(\psi_2(r+1),r+1)=\eta(\psi_1(r),r)-\eta(\psi_2(r),r)$. Since $\eta(\psi_1(r),r)>\eta(\psi_2(r),r)$, the result follows.

Case $k=r$: This means that $\psi_1(r+1)=(p_1,p_2,\ldots,p_r,p_{r+1})$ and $\psi_2(r+1)=(p_1,p_2,\ldots,p_{r+1},p_r)$. From Eq. (A1), we know,

$$\eta(\psi_1(r+1),r+1)$$
$$=\eta(\psi_1(r),r)+\sum_{i=1}^{r-1}\left(\beta_{1,r}(r+1)\prod_{q=i+1}^{r}f_{1,q}(r+1)\right)$$
$$+\beta_{1,r}(r+1), \tag{A11}$$

$$\eta(\psi_2(r+1),r+1)$$
$$=\eta(\psi_{2'}(r),r)+\sum_{i=1}^{r-1}\left(\beta_{2,r}(r+1)\prod_{q=i+1}^{r}f_{2,q}(r+1)\right)$$
$$+\beta_{2,r}(r+1) \tag{A12}$$

where, $\psi_{2'}(r)=(p_1,p_2,\ldots,p_{r-1},p_{r+1})$. Now, using the closed-form Eq. (7) for the overhead factors $\eta(\psi_1(r),r)$ and $\eta(\psi_{2'}(r),r)$ in the above set of equations, and using the fact that $E_r<E_{r+1}$, with little algebraic manipulation, the result follows. Hence the proof. □

Proof of Lemma 5. We shall first present the proof when $k=m$, i.e. the claim is true in the m-dimensional space. We shall refer to the sequence $\psi_j(m)$ defined in the statement of the lemma as a fast sequence. We shall now prove that $\eta(\psi_j(m),m)=\max_{\psi_i(m)\in\Psi_m}\{\eta(\psi_i(m),m)\}$, by using contradiction. Suppose the above claim is not true. This means that there exists a $\psi_r(m)\in\Psi_m$, which is not the fast sequence and $\eta(\psi_r(m),m)=\max_{\psi_i(m)\in\Psi_m}\{\eta(\psi_i(m),m)\}$. Because $\psi_r(m)$ is not a fast sequence, there is at least one adjacent pair of processors p_q and p_{q+1} such that $E_q>E_{q+1}$. Swapping these two processors, we obtain another sequence $\psi_{r'}(m)$. Now, from the result of Lemma 4, we obtain $\eta(\psi_{r'}(m),m)>\eta(\psi_r(m),m)$. Thus, we arrive at a contradiction to our assumption on $\eta(\psi_r(m),m)$. Therefore, $\eta(\psi_j(m),m)=\max_{\psi_i(m)\in\Psi_m}\{\eta(\psi_i(m),m)\}$, which means that the overhead factor for the *fast sequence* is greater than the overhead factor for all sequences in the m-dimensional space.

Now, we shall prove the lemma in the space Ψ_q, $q=1,\ldots,m-1$. From the above proof, we observe that the claim is true in every dimension, i.e. $\eta(\psi_j(k),k)=\max_{\psi_i(k)\in\Psi_k}\{\eta(\psi_i(k),k)\}$, where $\psi_j(k)=(p_1,p_2,\ldots,p_{k-1},p_k)$ such that $E_q<E_{q+1}$, for all $q=1,\ldots,k$. Now, from Eq. (A1), for a fixed sequence $\psi_i(k)$, we observe that $\eta(\psi_i(k),k)>\eta(\psi_{i'}(k-1),k-1)$, where $\psi_i(k)=(\psi_{i'}(k-1),p_k)$. This completes the proof that $\eta(\psi_j(m),m)=\max_{\psi_i(k)\in\Psi}\{\eta(\psi_i(k),k)\}$, where $k=1,2,\ldots,m$. Hence the proof. □

Proof of Theorem 2. We shall prove this lemma using contradiction. Suppose the above claim is not true. Then, this means that $T(\alpha,m,\psi_j(m))\geq T(\alpha',m,\psi_{j'}(m))$. Since

$\eta(\psi_j(m), m) < 1$, from Lemma 4, $\eta(\psi_{j'}(m), m) < 1$. This means that $\psi_{j'}(m)$ exists. Let us denote $\alpha_{\mathrm{I}} = \sum_{i=1}^{k-1} \alpha_i$, $\alpha_{\mathrm{II}} = \alpha_k + \alpha_{k+1}$, and $\alpha_{\mathrm{III}} = \sum_{i=k+2}^{m} \alpha_i$. Similarly, for the load distribution α', we denote $\alpha'_{\mathrm{I}} = \sum_{i=1}^{k-1} \alpha'_i$, $\alpha'_{\mathrm{II}} = \alpha'_k + \alpha'_{k+1}$, and $\alpha'_{\mathrm{III}} = \sum_{i=k+2}^{m} \alpha'_i$.

It may be noted that the optimal processing time using the load distributions α and α' have the property that all the processors stop computing at the same instant in time. Using the concept of equivalent processor presented in Section 3.2, for the sequence $\psi_j(m)$, we transform the m-processor system into a three-processor system, denoted as p_{eqI}, p_{eqII}, and p_{eqIII}, and their speed parameters as E_{eqI}, E_{eqII}, and E_{eqIII}, respectively. In the above definition, $p_{\mathrm{eqI}} = (p_1, p_2, \ldots, p_{k-1})$, $p_{\mathrm{eqII}} = (p_k, p_{k+1})$, and $p_{\mathrm{eqIII}} = (p_{k+2}, p_{k+3}, \ldots, p_m)$. Similarly, for the sequence $\psi_{j'}(m)$, we transform the m-processor system into a three processor system, denoted as p'_{eqI}, p'_{eqII}, and p'_{eqIII}, and their speed parameters as E'_{eqI}, E'_{eqII}, and E'_{eqIII}, respectively. For this sequence, we have, $p'_{\mathrm{eqI}} = (p_1, p_2, \ldots, p_{k-1})$, $p'_{\mathrm{eqII}} = (p_{k+1}, p_k)$, and $p'_{\mathrm{eqIII}} = (p_{k+2}, p_{k+3}, \ldots, p_m)$. From our assumption, $T(\alpha, m, \psi_j(m)) \geq T(\alpha', m, \psi_{j'}(m))$, using the equivalent processor representation, we obtain,

$$E_{\mathrm{eqI}}\alpha_{\mathrm{I}} + \theta_{\mathrm{cp}} + \alpha_{\mathrm{I}}C + (k-1)\theta_{\mathrm{cm}} \geq E'_{\mathrm{eqI}}\alpha'_{\mathrm{I}} + \theta_{\mathrm{cp}} + \alpha'_{\mathrm{I}}C + (k-1)\theta_{\mathrm{cm}} \quad \text{(A13)}$$

Using the closed-form solution for p_{eqI} and p'_{eqI} given by Eq. (14), we obtain $\alpha_{\mathrm{I}} \geq \alpha'_{\mathrm{I}}$ and $\alpha_{\mathrm{III}} \geq \alpha'_{\mathrm{III}}$. Since, $\alpha_{\mathrm{I}} + \alpha_{\mathrm{II}} + \alpha_{\mathrm{III}} = \alpha'_{\mathrm{I}} + \alpha'_{\mathrm{II}} + \alpha'_{\mathrm{III}} = 1$, we have,

$$\alpha'_{\mathrm{II}} \geq \alpha_{\mathrm{II}} \quad \text{(A14)}$$

and,

$$\alpha'_{\mathrm{I}} + \alpha'_{\mathrm{II}} \geq \alpha_{\mathrm{I}} + \alpha_{\mathrm{II}} \quad \text{(A15)}$$

Further,

$$T(\alpha, m, \psi_j(m)) = E_{\mathrm{eqII}}\alpha_{\mathrm{II}} + (\alpha_{\mathrm{I}} + \alpha_{\mathrm{II}})C + (k+1)\theta_{\mathrm{cm}} + \theta_{\mathrm{cp}}, \quad \text{(A16)}$$

$$T(\alpha', m, \psi_{j'}(m)) = E'_{\mathrm{eqII}}\alpha'_{\mathrm{II}} + (\alpha'_{\mathrm{I}} + \alpha'_{\mathrm{II}})C + (k+1)\theta_{\mathrm{cm}} + \theta_{\mathrm{cp}} \quad \text{(A17)}$$

and from Lemma 6 we have the following relation:

$$E_{\mathrm{eqII}}\alpha_{\mathrm{II}} < E'_{\mathrm{eqII}}\alpha'_{\mathrm{II}} \quad \text{(A18)}$$

Thus,

$$T(\alpha, m, \psi_{j'}(m)) - T(\alpha', m, \psi_{j'}(m)) = E_{\mathrm{eqII}}\alpha_{\mathrm{II}} - E'_{\mathrm{eqII}}\alpha'_{\mathrm{II}} + C + (\alpha_{\mathrm{I}} + \alpha_{\mathrm{II}} - \alpha'_{\mathrm{I}} - \alpha'_{\mathrm{II}}) \quad \text{(A19)}$$

From Eqs. (A15) and (A18), we immediately conclude $T(\alpha, m, \psi_j(m)) < T(\alpha', m, \psi_{j'}(m))$, which contradicts our assumption. Hence the proof. □

Proof of Theorem 3. Since $\eta(\psi_j(m), m) < 1$, from Lemma 5, $\eta(\psi_r(k), k) < 1$, for all $\psi_r(k) \in \Psi_k$ and $k = 1, \ldots, m$. From Theorem 1, we know that for a fixed sequence $\psi_j(k)$, $T(\alpha, k, \psi_j(k)) < T(\alpha, k-1, \psi_j(k-1))$, $k = 1, 2, \ldots, m$, where $\psi_j(k) = (\psi_j(k-1), p_k)$. Let us prove the theorem when $k = m$ using contradiction. Suppose the above claim is not true. This means that there exists another sequence $\psi_r(m) \in \Psi_m$ which is not the fast sequence and $T(\alpha, m, \psi_r(m)) = \min_{\psi_i(m) \in \Psi_m}\{T(\alpha, m, \psi_i(m))\}$. Since $\psi_r(m)$ is not the fast sequence, there exists at least one pair of processors p_j and p_{j+1} such that $E_j > E_{j+1}$. Swapping these two processors, we obtain another sequence $\psi_{r'}(m)$. From Theorem 2, we know that $T(\alpha, m, \psi_{r'}(m)) < T(\alpha, m, \psi_r(m))$. This contradicts our former assumption that $T(\alpha, k, \psi_r(k))$ is the minimum. Therefore, $T(\alpha, k, \psi_j(k)) = \min_{\psi_i(k) \in \Psi}\{T(\alpha, k, \psi_i(k))\}$, where $\psi_j(k)$ is the fast sequence in the m-dimensional space.

Now, since the above proof holds in every Ψ_k, $k = 1, \ldots, m$, and using Theorem 1, we immediately see that the result follows. Hence the proof. □

References

[1] Y. Chung, V.K. Prasanna, Parallelizing image feature extraction on coarse-grain machines, IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (12) (1998) 1389–1998.

[2] D.A. Bader, J. JáJá, D. Harwood, L.S. Davis, Parallel algorithms for image enhancement and segmentation by region growing with an experimental study, TR, University of Maryland, College Park, 1995.

[3] H.J. Siegel, J.B. Armstrong, D.W. Watson, Mapping computer-vision-related tasks onto reconfigurable parallel-processing systems, IEEE Computer 25 (2) (1992) 54–63.

[4] D.P. Bertsekas, J.N. Tsitsiklis, Parallel and Distributed Computation: Numerical Methods, Prentice Hall, Englewood Cliffs, NJ, 1989.

[5] C. Lee, Y.-F. Wang, T. Yang, Static global scheduling for optimal computer vision and image processing operations on distributed-memory multiprocessors, TRC 94-23, University of California, Santa Barbara, 1994.

[6] C. Lee, Y.-F. Wang, T. Yang, Global optimization for mapping parallel image processing tasks on distributed-memory multiprocessors, TRCS 96-28, University of California, Santa Barbara, 1996.

[7] L.J. Siegel, H.J. Siegel, A.E. Feather, Parallel processing approaches to image correlation, IEEE Transactions on Computers 31 (3) (1982) 208–218.

[8] H.R. Tsai, S.J. Horng, S.S. Tsai, S.S. Lee, T.W. Kao, C.H. Chen, Optimal speed-up parallel image template matching algorithms on processor arrays with a reconfigurable bus system, Computer Vision and Image Understanding 71 (3) (1998) 393–412.

[9] V. Bharadwaj, D. Ghose, V. Mani, T.G. Robertazzi, Scheduling Divisible Loads in Parallel and Distributed Systems, IEEE Computer Society Press, Los Almitos, CA, 1996.

[10] A. Gerasoulis, T. Yang, On the granularity and clustering of directed acyclic task graphs, IEEE Transactions on Parallel and Distributed Systems 4 (6) (1993) 66–701.

[11] V. Sarkar, J. Hennessy, Compile-time partitioning and scheduling of parallel programs, Proceedings of the SIGPLAN 86 Symposium on Compiler Construction, ACM Press, New York, 1986 (pp. 17–26).

[12] T.H. Dunigan, Performance of a Second Generation Hypercube, ORNL/TM-10 881, Oak Ridge National Laboratory, TN, 1988.

[13] M.T. Heath, C.H. Romine, Parallel solution of triangular systems on

distributed memory multiprocessors, SIAM Journal of Science, Statistics and Computers 9 (1988) 558–588.
[14] Y.C. Cheng, T.G. Robertazzi, Distributed computation with communication delays, IEEE Transactions on Aerospace and Electronic Systems 24 (1988) 700–712.
[15] V. Bharadwaj, D. Ghose, V. Mani, Multi-installment load distribution in tree networks with delays, IEEE Transactions on Aerospace and Electronic Systems 31 (1995) 555–567.
[16] C.H. Koelbel, D.B. Loveman, R.S. Schreiber, G.L. Steele Jr., M.E. Zosel, The High Performance Fortran Handbook, MIT Press, Cambridge, MA, 1994.
[17] S. Sahni, V. Thanvantri, Performance metrics: keeping the focus on runtime, IEEE Parallel and Distributed Technology Spring (1996) 43–56.
[18] D. Harwood, M. Subbarao, H. Hakalahti, L.S. Davis, A new class of edge-preserving smoothing filters, Pattern Recognition Letters 6 (1987) 155–162.