Received: 25 August 2018 | Revised: 22 July 2019 | Accepted: 27 July 2019
DOI: 10.1002/cpe.5498

SPECIAL ISSUE PAPER

Divisible load scheduling of image processing applications on the heterogeneous star and tree networks using a new genetic algorithm

Sahar Nikbakht Aali | Nader Bagherzadeh

University of California, Irvine, Irvine, California

Correspondence
Sahar Nikbakht Aali, University of California, Irvine, Irvine, CA 92697.
Email: snaali@uci.edu

Summary

The divisible load scheduling of image processing applications on the heterogeneous star and multi-level tree networks is addressed in this paper. In our platforms, processors and network links have different speeds. In addition, computation and communication overheads are considered. A new genetic algorithm for minimizing the processing time of low-level image applications using divisible load theory is introduced. The closed-form solution for the processing time, the image fractions that should be allocated to each processor, the optimum number of participating processors, and the optimal sequence for load distribution are derived. The new concept of equivalent processor in tree network is introduced and the effect of different image and kernel sizes on processing time and speed up are investigated. Finally, to indicate the efficiency of our algorithm, several numerical experiments are presented.

KEYWORDS

divisible load scheduling, equivalent processor, genetic algorithm, image fractions, load distribution sequence, local operation

1 | INTRODUCTION

Divisible load scheduling is a special class of data parallelization methods that can be used in those applications that are divided into any number of independent fractions. These fractions can be processed in parallel on different processors. Big dataset processing, matrix computation, signal and image processing, Hough transform, and experimental data processing are examples of these applications. Divisible Load Scheduling (DLS) has been studied extensively in the last two decades because of its simplicity and analytical tractability.[1]

Image processing applications need extensive computational power that cannot be provided by one processor.[2] DLS is a good option for exploiting data parallelism in image applications[3,4] because most image processing applications are divisible in nature. There are three kinds of operations in image processing, ie, pixel operations in which each pixel can be processed independently; local operations, which are most common operators in image application, and the value of each output pixel is a function of the value of that pixel plus some neighboring pixels; and global operators that need the information of the whole image to process the value of one pixel. Pixel and local operations are good candidates for data parallelism. In this paper, we focus on local operators, as the most common operators in image applications.

We consider the star and tree networks as the target platforms. In these networks, the master processor holds the entire image and the kernel. It does not participate in image processing and just partitions the workload and distributes it to other slave processors. In a star topology, slave processors begin to compute their image fractions after receiving their workload from the master processor completely. Parent nodes in tree topology have the capability of computing and communicating at the same time. They are equipped with front-end processors in our model. The objective of using DLS for data parallelism in image applications in our model is minimizing the processing time which is the most common criterion in related papers. It has been proved that AFS policy (where All nodes Finish Simultaneously) is the necessary and sufficient condition to obtain the minimum processing time in heterogeneous bus and star networks.[5] It implies that, if the load is distributed in a way that all processors finish their process at the same time, then the optimum scheduling is achieved. By introducing a new concept of equivalent processor in the multi-level tree, we obtain the same result for tree topology.

The homogenous architectures were considered in the first studies of DLS. Bharadwaj et al[6] used divisible load scheduling on the bus network. They obtained the closed-form expression, the optimal processing time and the workload fractions for slave processors. Then the DLS theory was applied to other platforms such as star, multi-level tree[7,8] and linear networks. Different cost models and distribution algorithms for star and tree topologies were introduced[9,10] and multi-objective methods for scheduling workload on tree topology were presented.[7] The cheating problem in scheduling was considered in the work of Ghanbari and Othman.[11] It was shown that adaptive methods decrease communication and computation cheating-rate considerably. In the work of Chen,[12] it was proved that the communication and computation overheads have significant effects on the processing time and these parameters need to be considered for achieving more accurate results. In the work of Wang et al,[13] the effect of other parameters such as release time was considered. It has been proven in the work of Veeravalli et al[14] that it is not necessary that all processors participate in processing and adding more processors will not always increase the processing time. There is an optimum number for participating processors that give the minimum processing time. Moreover, it has been indicated that the order of load distribution to slave processors can affect the processing time considerably and there is an optimum sequence of load distributing that will reduce the processing time significantly.[5] In the work of Mingsheng,[15] it was proved that the optimum processing time without considering computation and communication overheads is independent of the sequence of load distribution. It has been shown that by considering different speeds for communication links, the non-decreasing order of communication speeds gives the minimum processing time in a star network.[16] In the work of Suresh et al,[17] the influence of constant start-up overhead on processing time for bus network with the same communication speeds was considered. It was proved that the non-decreasing order of computation speed is the optimum sequence for decreasing the processing time. It was indicated in the work of Bharadwaj et al[16] that, by considering different values for communication overheads and omitting computation overheads, the optimum sequence will be dependent on both computation and communication speeds. A new solution for distribution of divisible nonlinear workload on the heterogeneous star topology was introduced in the work of Chen and Chu[18] and the optimum number of slave processors and the number of installments were obtained. In the work of Wang et al,[19] a genetic algorithm for allocating a large-scale astronomical load to the heterogeneous processors was introduced. This proposed strategy decreases the processing time significantly.

To use a more realistic model, we consider different communication and computation overheads in our heterogeneous platforms. To the best of our knowledge, finding the optimal sequence for load distribution by considering different values for computation and communication speeds and various values of computation and communication overheads is still a challenging problem.

In this paper, first, a closed-form solution for obtaining the optimum processing time, the number of image fractions that will be assigned to each processor, and the number of participating processors is presented. Then, the amount of additional pixels that each processor needs to compute its load fraction is obtained. In order to represent the entire multi-level tree as a star topology, a new concept of equivalent processor is introduced as well. A new genetic algorithm is introduced to find the optimal load sequence for workload distribution and, finally, the influence of kernel size on processing time and speed up is investigated.

The rest of this paper is arranged as follows. The problem definition and some notations and remarks are presented in Section 2. Section 3 indicates the new genetic algorithm for obtaining the optimum sequence of load distribution. The experimental results for verifying the effectiveness of our algorithm in both networks are presented in Section 4. The conclusion of this paper and the future works are presented in Section 5.

2 | PROBLEM DEFINITION, NOTATIONS, AND CLOSED-FORM SOLUTION

2.1 | System model of star topology

The first platform considered for data parallel processing of low-level image applications is the heterogeneous star topology, which is denoted by $S = (P,L,E,C)$. $P = [P_0, \dots, P_m]$ is the set of $(m+1)$ processors. The entire image will be partitioned into different parts by master processors, P_0, and then these fractions will be assigned to slave processors based on load distribution order. P_0 is only responsible for partitioning and allocating workload to other slave processors and does not participate in load processing. $L = [L_1, \dots, L_m]$ is the set of communication links between master and slave processors.$E = [E_1, \dots, E_m]$ is the set of m computation parameters and $C = [C_1, \dots, C_m]$ is the set of m communication parameters of links between master and slaves. Figure 1A shows a heterogeneous star topology with five processors. In the rest of this paper, the blocking mode of communication is considered[13] and a single-port and no-overlap model is followed by all the processors. In this model, the master processor can communicate only with one slave processor at each time. Moreover, simultaneous communication and computation between processors are not allowed.

2.2 | System model of multi-level tree topology

The next topology is a heterogeneous multi-level tree network in which the load is passed through each level from the top of the tree to bottom. This network consists of one master and a set of slave processors. Slave processors can be parents or leaves. A slave that has processors below it is considered as a parent and otherwise is considered as a leaf processor. A 3-level tree network is depicted in Figure 1B. Like star topology, the master does not compute any load fraction and it only partitions the load and allocates them to its children.

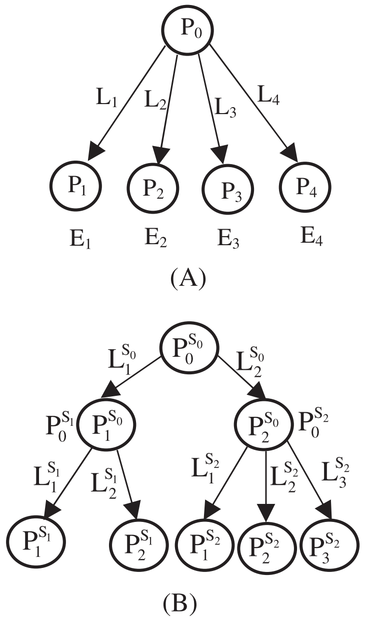

FIGURE 1 A, A heterogeneous star network with five processors; B, A 3-level tree network with eight processors

The parent nodes in multi-level tree topology have front-end processors, which means they can compute and communicate simultaneously. Therefore, the no-overlap assumption that was considered for star network had to be changed slightly. This assumption for multi-level tree network ensures that any concurrent communication between the immediate upper and lower level in the network by the same parent is not possible. Therefore, the parent nodes can perform computation and communicate to only one of their children at a time. It is a single-port model, so only one communication can occur between two processors at a time. It is possible that the intermediate stars, which are at the same level of the tree solve their load in parallel, which can reduce the processing time.

The multi-level tree network is made up of n+1 star network $S = [S_0, \dots, S_n]$ and any star in this topology, $S_k \in S,\ 0 \le k \le n$ has $m_{S_k} + 1$ processors. The system parameters for star topology are introduced in the previous section. $P_0^{S_0}$ is the master processor in tree network, which just schedules image fractions to its children in the first level. Parent nodes (excluding the master) are given two different labels, $P_0^{S_k}$, which is a parent node in the star network; S_k, $1 \le k \le n$ could be a child node in the star network S_j, $j \ne k$, $j < k$ (as shown in Figure 1B). Each node in star network S_k, $0 \le k \le n$ is denoted by $P_i^{S_k}$, $1 \le i \le m_{S_k}$.

$L^{S_k} = [L_1^{S_k}, \dots, L_{m_{S_k}}^{S_k}], 0 \le k \le n$ is the set of links that connect each parent processor in star S_k to its children. The computation and communication parameters for each star topology will be equal to $E^{S_k} = [E_1^{S_k}, \dots, E_{m_{S_k}}^{S_k}]$ and $C^{S_k} = [C_1^{S_k}, \dots, C_{m_{S_k}}^{S_k}]$, respectively.

2.3 | Image partitioning

Using data parallelism in the processing of low-level image applications is a good option to reduce the processing time considerably, but the only obstacle in using data parallelism in different heterogeneous topologies is the dependency between image partitions. By partitioning the image into different fractions and allocating them to each processor, each processing element needs additional pixels from the neighboring processors to compute its fraction. These additional pixels will be prepared for individual processors in two different ways. In the first method, the neighboring processors can exchange the additional pixels simultaneously at computation phase and, in the second method, which is more common, the required data will be assigned to individual processors at the beginning of the computation phase. As mentioned before, in our model, only two processors can communicate with each other at a time, so the additional data is prepared at the initial communication phase and before computation. Thus, the processors do not need to communicate to each other during the computation phase.

To exploit data parallelism in image processing applications, at first, the image is partitioned into different blocks, and then the appropriate number of these blocks will be assigned to processors. There are different methods for image partitioning in the literature such as column, raw, block, column-cyclic, row-cyclic, and block-cyclic.[3] The number of additional pixels that should be assigned to individual processors is based on the method of partitioning and kernel size.

The kernel is a small matrix applied to each pixel and its neighbors in the most of low-level image applications. These kinds of applications are based on a convolution of the image with a kernel with dimension $M \times M$. It is the process of adding each value of image pixels to its local neighbors, weighted by the kernel. A kernel operates on the pixel's values to construct a new image, which is dependent upon the kind of kernel. We suppose that the values of the kernel are transmitted by the master processors to other processors along with the required date since they need them to compute their workload.

2.4 | Closed-form solution for optimum scheduling

The closed-form solutions for star and tree networks are presented in this section. The timing diagrams by considering a fixed sequence of load distribution for star and tree networks are depicted in Figure 2A and 2B, respectively. By considering the timing diagram, a closed-form expression for optimal scheduling of the local operators of image processing applications and the number of image blocks that should be allocated to each processor will be obtained. Some notations that will be used in the rest of this paper are introduced next:

$N_1 \times N_2$	Size of the two-dimensional image;
$M \times M$	Size of the Kernel;
$\alpha_i, \alpha_i^{S_k}$	Partition of the image assigned to P_i and $P_i^{S_k}$;
Δ	Number of required additional pixels;
$C_i, C_i^{S_k}$	The inverse of communication speed of L_i and $L_i^{S_k}$;
$E_i, E_i^{S_k}$	The inverse of computation speed of P_i and $P_i^{S_k}$;
$Z_i, Z_i^{S_k}$	Computation overhead parameter, which consists of all delays during the computation phase;
$O_i, O_i^{S_k}$	Communication overhead parameter, which consists of all delays during the communication phase.

Star network:

Figure 2A presents two different phases, the communication phase between master and slave processors and the computation phase of each slave processor. Each slave processor starts to compute its image fraction right after receiving its load completely from the master processor. The master processor takes the responsibility of partitioning the image with dimensions $N_1 \times N_2$ into m parts and distributing them to slave processors (according to the distribution order). The master processor also provides the required pixels and kernel to all the slave processors. Thus, the slave processors do not need to communicate to each other during the computation time.

Table 1 shows the required additional pixels, Δ, for each processor based on different ways of image partitioning. In the rest of this paper, the column partitioning method in which each image block consists of a number of image columns is considered. $\alpha = [\alpha_1, \dots, \alpha_m], 1 \le \alpha_i \le N_2$ is the load distribution vector and $\sum_{i=1}^{m} \alpha_i = N_2$ is the normalization equation. The kernel is transmitted to all processors by the master at the communication phase. $N_1\alpha_i$ is used to show the number of pixels that will be assigned to processor P_i. For example, consider $N_1 \times N_2 = 1024 \times 1000$ and $\alpha_i = 20$, so the number of pixels that will be computed by P_i is 20×1024.

A linear model for computation and communication times is considered. This model is a very common and proven model in most of the DLS literature[13] and the results obtained from this model can be extended to other cost models as well. In this model, the communication time is proportional to the amount of workload that is carried by the network links. This workload consists of image blocks, kernel pixels, additional data, and communication overhead (such as start-up delay). Thus, the communication time is given by $(N_1\alpha_i + M^2 + \Delta)C_i + O_i$.

The computation time is proportional to the amount of workload assigned to slave processors. A single local operation on a pixel involving all the M^2 neighboring pixels and it needs M^2 multiplications to compute each pixel. Thus, processor P_i needs $M^2N_1\alpha_iE_i$ time units to process

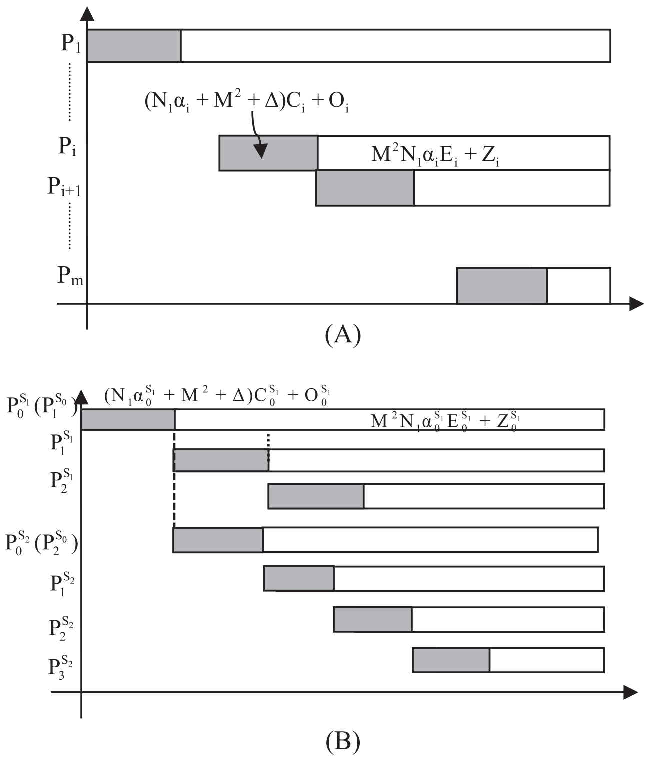
P_1
$(N_1\alpha_i + M^2 + \Delta)C_i + O_i$
P_i
$M^2N_1\alpha_iE_i + Z_i$
P_{i+1}
P_m
(A)
$P_0^{S_1}(P_1^{S_0})$
$(N_1\alpha_0^{S_1} + M^2 + \Delta)C_0^{S_1} + O_0^{S_1}$
$M^2N_1\alpha_0^{S_1}E_0^{S_1} + Z_0^{S_1}$
$P_1^{S_1}$
$P_2^{S_1}$
$P_0^{S_2}(P_2^{S_0})$
$P_1^{S_2}$
$P_2^{S_2}$
$P_3^{S_2}$
(B)

FIGURE 2 Timing diagram for image divisible load processing on (A) heterogonous star network and (B) tree-level network presented Figure 1B

Image partitioning methods	Δ
Row Partitioning	$N_2(M - 1)$
Column Partitioning	$N_1(M - 1)$
Block partitioning Block size: $b_1 \times b_2$ $Mod(N_1/b_1) = 0$ $Mod(N_2/b_2) = 0$	$(M - 1)(b_1+b_2)+(M - 1)^2$
Row-Cyclic Partitioning Blocking factor: p	$\frac{N_1}{bp}\left((M-1)-N_2\right)$
Column-Cyclic Partitioning Blocking factor: p	$\frac{N_2}{bp}\left((M-1)-N_1\right)$
Block-Cyclic Partitioning Block size: $b_1 \times b_2$ Blocking factor: p	$\frac{N_1N_2}{b_1b_2p}\left((M-1)(b_1+b_2)+(M-1)^2\right)$

TABLE 1 The required additional pixels for different types of image partitioning

one image partition α_i on processor P_i. Therefore, the computation time is equal to this value plus the computation overhead, which includes all delays in the computation phase, such as start-up overhead. The computation time is given by $M^2N_1\alpha_iE_i+Z_i$.

From Figure 2, the communication time of P_i is equal to the communication and computation time of P_{i+1} such as

$$M^2N_1\alpha_iE_i + Z_i = \left(N_1\alpha_{i+1} + M^2 + \Delta\right) C_{i+1} + O_{i+1} + M^2N_1\alpha_{i+1}E_{i+1} + Z_{i+1}. \tag{1}$$

Therefore,

$$\alpha_i = \alpha_{i+1}F_i + H_i, \tag{2}$$

where

$$F_i = \frac{\left(N_1C_{i+1} + M^2N_1E_{i+1}\right)}{M^2N_1E_i}, H_i = \frac{\left(M^2 + \Delta\right) C_{i+1} + O_{i+1} + Z_{i+1}-Z_i}{M^2N_1E_i}. \tag{3}$$

If we consider the normalization equation, $\sum_{i=1}^{m} \alpha_i = N_2$ and Equation (2), we will have m variables in m equations, so each workload for slave processors will be obtained. By considering these two equations, the workload of the last processor, P_m, and other slave processors α_i, $1 \le i \le m - 1$ will be obtained

$$\alpha_m = \frac{N_2- \sum\limits_{i=1}^{m-1} G_i}{1 + \sum\limits_{i=1}^{m-1} Q_i} \tag{4}$$

$$\alpha_i = \alpha_m \prod_{j=i}^{m-1} F_j + G_i, \tag{5}$$

where

$$Q_i = \prod_{j=i}^{m-1} F_j \quad , \quad G_i = \sum_{k=i}^{m-1} H_k \left(\prod_{j=i+1}^{k} F_{j-1}\right). \tag{6}$$

With substituting Equation (4) in (5), different workload fractions for each individual processor is obtained. According to Figure 2A, the processing time, T(α, m), is equal to the communication and computation time of the first slave processor. Thus, it is denoted by Equation (7)

$$T(\alpha, m) = \left(N_1\alpha_1 + M^2 + \Delta\right) C_1 + O_1 + M^2N_1\alpha_1E_1 + Z_1. \tag{7}$$

Using $i = 1$ in Equation (5), we will have

$$T(\alpha, m) = \left(\alpha_m \prod_{j=1}^{m-1} F_j + G_1\right) \left(N_1C_1 + M^2N_1E_1\right) + \left(M^2 + \Delta\right) C_1 + O_1 + Z_1, \tag{8}$$

where α_m, G_1 and F_j are defined before.

In Equation (4), α_m is the last image partition that is allocated to the last processor. This value is greater than zero, otherwise adding more processors does not reduce the processing time. Thus, the necessary and sufficient condition to obtain an Optimum Number of Processors (ONP) is

$$\sum_{i=1}^{m-1} G_i \le N_2. \tag{9}$$

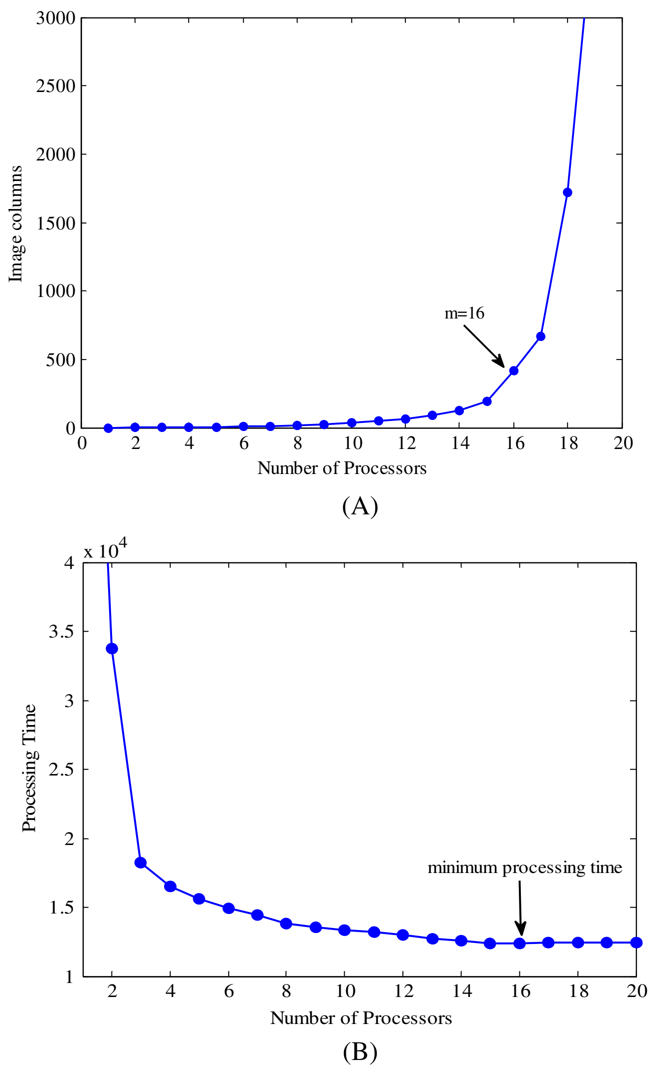

FIGURE 3 (A) ONP factor and (B) the processing time for different number of processors

For $N_2 = 520$, the ONP factor, $\sum_{i=1}^{m-1} G_i$, for different numbers of processors is illustrated in Figure 3A. Clearly, for m = 16, the overhead factor is equal to the number of image columns and as shown in Figure 3B, with this number of processors, the minimum processing time is achieved. Adding more processors cannot reduce the processing time.

Multi-Level Tree Network:

Proposition 1. *There is an optimal solution for scheduling divisible load on the heterogeneous multi-level tree network, in which all the slave processors finish computation tasks simultaneously.*

Proof. Proposition 1 has been proven for bus and star topologies.[12-16] To prove it for the tree network, the concept of equivalent processor, which presents every star network of multi-level tree topology as one processing node, is used. By replacing star topologies $S^k, 1 \leq k \leq n$, in tree network with one equivalent processor, the single level tree is obtained and all the results from the previous section are applied to the multi-level tree network. As it was mentioned before, the order of load distribution can reduce the processing time significantly. In the simulation results, it will be shown that the communication speed of processors has a considerable effect on processing time. Therefore, by introducing the following equivalent processor, we keep the order of load distribution based on communication speed. To the best of our knowledge, obtaining an equivalent processor for heterogeneous multi-level tree network by considering computation and communication latencies, is new. The fixed sequence for load distribution in the network was assumed and, at each iteration, only two processors were selected to combine. □

This process begins from S_n star network in the multi-level tree and ends to S_1. First, the initial two processors of star network S_n were considered, and then these two processors are replaced by one equivalent processor. In the next iteration, the equivalent processor and next processor (according to the load distribution order) were selected and this process was repeated until all processors in S_n were considered. At the subsequent iteration, star network S_{n-1} was considered and, in the last step, a single level tree network will remain. The communication and computation of the equivalent processor are obtained according to Figure 4. As it is clear, the processing time is equal to T (the optimal processing time of S_k). Therefore, replacing two processors with one single processor does not change the processing time. In addition, the time

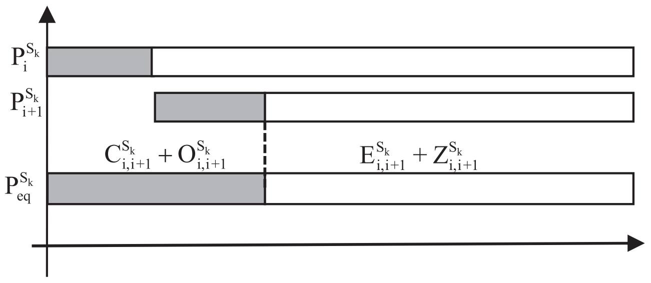
$P_i^{S_k}$
$P_{i+1}^{S_k}$
$C_{i,i+1}^{S_k}+O_{i,i+1}^{S_k}$
$E_{i,i+1}^{S_k}+Z_{i,i+1}^{S_k}$
$P_{eq}^{S_k}$

FIGURE 4 Equivalent processor for two processors

spent on load allocation is unchanged. From Figure 4, the communication, computation, and overhead parameters of the equivalent processor will be expressed as

$$C_{i,i+1}^{S_k}=\frac{C_i^{S_k}C_{i+1}^{S_k}\left(1+M^2\left(\rho_i^{S_k}+\rho_{i+1}^{S_k}\right)\right)}{C_{i+1}^{S_k}+M^2\left(E_i^{S_k}+E_{i+1}^{S_k}\right)}, \tag{10}$$

where $\rho_i^{S_k}=\frac{E_i^{S_k}}{C_i^{S_k}}$

$$E_{i,i+1}^{S_k}=\frac{M^2E_i^{S_k}E_{i+1}^{S_k}}{C_{i+1}^{S_k}+M^2\left(E_i^{S_k}+E_{i+1}^{S_k}\right)} \tag{11}$$

$$Z_{i,i+1}^{S_k}=Z_{i+1}^{S_k},\ O_{i,i+1}^{S_k}=O_i^{S_k}+O_{i+1}^{S_k}. \tag{12}$$

It can be verified that this representation of the equivalent processor keeps the order of best load distribution found by genetic algorithm in each star. According to Figure 4, $\alpha_i^{S_k}C_i^{S_k}+\alpha_{i+1}^{S_k}C_{i+1}^{S_k}=N_1C_{i,i+1}^{S_k}$ and $\alpha_i^{S_k}+\alpha_{i+1}^{S_k}=N_2$, so we will have $\alpha_{i+1}^{S_k}(C_{i+1}^{S_k}\text{-}C_i^{S_k})+N_2C_i^{S_k}=N_2C_{i,i+1}^{S_k}$. Since by definition,$1\le\alpha_{i+1}^{S_k}\le N_2$, it is clear that $C_i^{S_k}\le C_{i,i+1}^{S_k}\le C_{i+1}^{S_k}$. It is indicated that, if we distribute the workload based on the communication parameters, the value of communication parameter of the equivalent processor is between the communication values of two combined processors.

Similarly, from the definition of the equivalent processor in this model, $E_{i,i+1}^{S_k}=\frac{\alpha_{i+1}^{S_k}E_{i+1}^{S_k}}{N_1}=\frac{M^2\alpha_i^{S_k}E_i^{S_k}-\alpha_i^{S_k}C_i^{S_k}}{M^2N_1}$. Since by definition $1\le\alpha_i^{S_k},\alpha_{i+1}^{S_k}\le N_2$, it follows that $E_{i,i+1}^{S_k}\le E_i^{S_k},E_{i+1}^{S_k}$, which indicate the computation parameter of equivalent processor is less than two combined processors. Therefore, the order of load distribution will remain unchanged.

The Gantt chart for obtaining the closed-form solution for multi-level tree network is presented in Figure 2B. Both phases are shown in this figure. Each node receives the load fractions from its parent completely before beginning computation. Parent nodes begin computing their fraction and communicating to one of their children at the same time.

The master allocates the image, the kernel, and additional pixels to all of its immediate children at level 1 and these children schedule parts of their load to their children at level 2 and so on. This timing diagram is considered for a fixed sequence of load distribution in each star $S_k, 0\le k\le n$, which is denoted by a vector, $\alpha^{S_k}=[\alpha_1^{S_k},\ \dots\ ,\alpha_{m_{S_k}}^{S_k}]$ such that $1\le\alpha_i^{S_k}\le N_2$. Note that parent nodes parameters have two different labels, eg, in Figure 2B, $\alpha_0^{S_1}$ and $\alpha_1^{S_0}$are labels of the load allocated to $P_0^{S_1}(P_1^{S_0})$. The normalization equation for this network will be equal to $\sum_{k=0}^{n}\sum_{i=1}^{m_{S_k}}\alpha_i^{S_k}=N_2$. As in the previous section, a linear model for computation and communication is considered. In the following, a closed-form model for multi-level tree network regarding Figure 2B will be obtained.

The computation time of $P_i^{S_k}$ will be equal to the communication and computation time of $P_{i+1}^{S_k}$

$$\begin{aligned}M^2N_1\alpha_i^{S_k}E_i^{S_k}+Z_i^{S_k}&=\left(N_1\alpha_{i+1}^{S_k}+M^2+\Delta\right)C_{i+1}^{S_k}+O_{i+1}^{S_k}\\&\quad+M^2N_1\alpha_{i+1}^{S_k}E_{i+1}^{S_k}+Z_{i+1}^{S_k}\ ,\ 0\le k\le n, 1\le i\le m_{S_k},\end{aligned} \tag{13}$$

and in each star network, $S_k, 0\le k\le n$ of multi-level tree, the computation time of the parent node is equal to the communication and computation time of the first child

$$\begin{aligned}M^2N_1\alpha_0^{S_k}E_0^{S_k}+Z_0^{S_k}&=\left(N_1\alpha_1^{S_k}+M^2+\Delta\right)C_1^{S_k}+\\&\quad O_1^{S_k}+M^2N_1\alpha_1^{S_k}E_1^{S_k}+Z_1^{S_k}\ ,1\le k\le n.\end{aligned} \tag{14}$$

Therefore,

$$\alpha_i^{S_k}=\alpha_{i+1}^{S_k}F_i^{S_k}+H_i^{S_k} \tag{15}$$

$$\alpha_0^{S_k}=\alpha_1^{S_k}F_0^{S_k}+H_0^{S_k}. \tag{16}$$

F_i and H_i for the star topology are defined before in Equation (3). These parameters are considered for each star network, S_k, $0\le k\le n$ of multi-level tree.

First, the load allocated to the last processor in S_0, $P^{S_0}_{m_{S_0}}$, by considering Equation (15) and the normalization equation $\sum_{k=0}^{n}\sum_{i=1}^{m_{S_k}}\alpha_i^{S_k} = N_2$ is achieved according to

$$\alpha^{S_0}_{m_{S_0}} = \frac{N_2 - \sum\limits_{i=1}^{m_{S_0}-1} G_i^{S_0}}{1 + \sum\limits_{i=1}^{m_{S_0}-1} Q_i^{S_0}}. \tag{17}$$

Then, the workload that need to be assigned to other processors in S_0 will be achieved

$$\alpha_i^{S_0} = \alpha^{S_0}_{m_{S_0}} Q_i^{S_0} + G_i^{S_0}. \tag{18}$$

Q_i and G_i for the star network are defined before in Equation (6). At next step, the load scheduled to each processor from star $S_1$to S_n can be obtained through Equation (19). The parent nodes compute part of the load allocated to them and schedule other partitions to their children

$$\alpha^{S_k}_{m_{S_k}} = \frac{\alpha_0^{S_k} - \sum\limits_{i=0}^{m_{S_k}-1} G_i^{S_k}}{1 + \sum\limits_{i=1}^{m_{S_k}-1} Q_i^{S_k}} \tag{19}$$

$$\alpha_i^{S_k} = \alpha^{S_k}_{m_{S_k}} Q_i^{S_k} + G_i^{S_k}, 0 \le i \le m_{S_k}-1. \tag{20}$$

According to Figure 2B, the processing time, $T(\alpha, m)$, is equal to the communication and computation time of the parent of the first star, $P_0^{S_1}$, and it will be expressed by Equation (21)

$$T(\alpha, m) = \left(N_1\alpha_1^{S_0} + M^2 + \Delta\right) C_1^{S_0} + O_1^{S_0} + M^2 N_1 \alpha_1^{S_0} E_1^{S_0} + Z_1^{S_0}. \tag{21}$$

3 | A NEW GENETIC ALGORITHM FOR DLS OF IMAGE APPLICATIONS (IDLS-GA)

The scheduling problem is one of the hardest optimization problems. It is mentioned before that in DLS, finding the optimum processing time by considering different values for computation and communication speeds and overheads on the heterogeneous platforms is intractable.

The processing time of each image fraction in divisible load scheduling method onto heterogeneous topologies is dependent on a few parameters including the number of participating processing elements, fraction of workload that allocated to individual processors, and the order of load distribution. Thus, by finding the optimum sequence of load distribution, the optimum number of participating processors, as well as each load fraction, will be obtained. A genetic algorithm could be a promising technique to find this sequence among all other sequences. Genetic algorithms have been used in divisible load scheduling problems before.[20-25] At first, we need to make some assumptions, ie, each chromosome contains a load distribution sequence and each gene defines each participating processor. Only one instance of a processor should exist in one chromosome and omission and duplication of processors are not allowed for an individual. The new genetic algorithm for finding the best distribution order in a star network is described next.

First step is generation of the initial population. We consider N chromosomes as the initial population. Each chromosome defines a sequence of load distribution. $C^i = (c^i_1, c^i_2, \dots, c^i_m), 1 \le i \le N$ is a specified chromosome, in which each gene has a value between 1 and m. For example, consider the platform in Figure 1A; one chromosome for this topology can be denoted as $C = (3,4,2,1)$. It shows that the first image fraction should be scheduled to P_3 and the second fraction to P_4 and so on. After the creation of the initial population, we evaluate each individual of the initial generation according to Algorithm 1.[20] The fitness function selects the best sequences of load distribution for crossover and mutation operators. The objective of the divisible load scheduling is considered as the evaluation criteria. This objective is minimizing the processing time according to the timing diagram presented in Figure 2A. At this step, the processing time, load fractions for slave processors, and optimum numbers of processors will be achieved. At the next step, the roulette wheel for selecting good parents among all sequences of the first population is used. We give each individual a percentage fitness value equal to $\frac{T_i(\alpha,m)}{\sum_{i=1}^{N} T_i(\alpha,m)}$, and then by using roulette wheels, the parents for the next section will be selected. Roulette wheel gives good sequences more chance to remain in the next generation. It also gives the weak sequences a chance to appear as parents, since sometimes reordering of these sequences by crossover and mutation operators leads to better sequences. Next, the crossover operator is applied to selected chromosomes and reorders them to generate better sequences.

We choose parents with the probability of P_{cros} from chromosomes that are obtained as a result of roulette wheels. Different methods of crossover have been introduced so far[21,22]; since we first assumed that there should not be any duplication or omission in each chromosome, we have a limitation in our model. Thus, common crossover methods such as N-point crossover cannot be used here because each chromosome presents a sequence of processors. Therefore, combination of two parents violates our first assumptions. At this situation, some processors can appear twice or more and some of them can be omitted. To solve this problem, we use another way for generating offspring.

Algorithm 1 Fitness function (evaluation of chromosomes)

Input: The target platform, The image with dimensions $N_1 \times N_2$, a kernel with dimensions $M \times M$ and one chromosome.

Output: The processing time regarding the sequence of load distribution (chromosome), the load fraction for each processor, number of participating processor.

```
1.  for i=1 to m-1 do
2.      calculate $F_i$ and $H_i$ by Equation (3)
3.  end for
4.  for i=1 to m-1
5.      calculate $M_i$ and $G_i$ by Equation (3)
6.  end for
7.  if $\sum_{i=1}^{m-1} G_i > N$
8.          then let m=m-1 and return to step 4
9.  else
10. compute $\alpha_m$ according to (4)
11. compute processing time according to (7)
12. for i=1 to m-1
13.     calculate $\alpha_i$ according to (5)
14. end for
15. end if
```

Most of the chromosomes that have been selected as parents for crossover operators are appropriate sequences based on our objective function. Therefore, to keep the order of good genes, a new method for generating offspring is described in Algorithm 2.[20] In this method, two parents, C^j and C^k,$1 \leq j, k \leq N$,$j \neq k$, are selected randomly. The optimum number of processors in each of them is J and K, respectively. A weight value is assigned to each processor in a chromosome. This value is proportional to the order of processors in an individual. These weight values are from m to 1. As it is clear in Figure 3, the initial processors have to compute more image fractions in comparison to others. If one considers the first processor, its computation time is equal to the communication and computation time of the second processor, which indicates this processor computes more load in comparison with others. Thus, the role of these processing elements in computing workload is more important than others and more weight value need to be assigned to them. We clarify our method by using an example.

Algorithm 2 Crossover operator

Input: Two parents, $C^j = (c_1^j, c_2^j, \ldots, c_J^j)$ and $C^k = (c_1^k, c_2^k, \ldots, c_K^k)$, P_{Cros}

Output: One offspring, $C^z = (c_1^z, c_2^z, \ldots, c_m^z)$

```
1.  Give a weigh value, from m, m-1,...,1 to each gene in individuals. If one sequence do not contain a processor, the weight of that gene will be
    considered zero.
2.  for i = 1 to m
3.      find weight of $P_i$ in both parents
4.  end for
5.  for i = 1 to m
6.      add weight of $P_i$ in parent 1 with weight of $P_i$ in parent 2
7.  end for
8.  sort $P_i$, $1 \leq i \leq m$ decreasing order of their weight values
9.  if weight value of $P_i$ is equal to weight value of $P_j$ for $i \neq j$, then we have to compare weight values of $P_i$ and $P_j$ in both parents $C^j$ and $C^k$.
    Each of them that have more weight in one parents should be appeared in front position in offspring. If both processors have the same
    position in parents, we can put them in any arbitrary order in offspring.
```

Consider C^i = (4,1,3,7,5),C^j = (4,3,1,2,7,5,8) and m = 8. The number of participating processors in C^i and C^j is 5 and 7, respectively. The weight assigned to P_1 to P_8 in C^i and C^j are (7,0,6,8,4,0,5,0) and (6,5,7,8,3,0,4,2), respectively. In C^i, the first load is assigned to P_4. Therefore, it has the most weight value and, since P_5 computes the last partition of the workload, it has the lowest weight value. The weight value is assigned to each processor according to its workload.

By taking their sum and sorting them in the decreasing order, we will have ($P_4, P_1, P_3, P_7, P_5, P_2, P_8, P_6$). Therefore, the new sequence is given by (4,1,3,7,5,2,8,6). The offspring keep the distribution order of their parents. At the next step, the individuals with probability P_{mut} from previous offspring will be selected according to Algorithm 3.[20] In this step, two genes are randomly selected in each chromosome and evaluated by the fitness function; if the resulted processing time is less than the one without mutation, the new offspring will be retained for the next step. Otherwise, two other genes are chosen for repeating the process until local optimum is reached.

Algorithm 3 Mutation operator

```
Input: The previous offspring, $C^z = (c_1^z, c_2^z, \dots, c_m^z)$, $P_{mut}$
Output: The new offspring, $C^h = (c_1^h, c_2^h, \dots, c_m^h)$
1.  select a value for r , $0 \le r \le 1$
2.  if  $r \le P_{mut}$ then
3.      obtain the fitness function, $T_{C^z}(\alpha, m)$, for chromosome $C^z$
4.  t=0
5.  swap tow genes in $C^z$ randomly and create a new sequence, $C^h = (c_1^h, c_2^h, \dots, c_m^h)$
6.  obtain the fitness function, $T_{C^h}(\alpha, m)$, for new chromosome $C^h$
7.  if  $T_{C^h}(\alpha, m) > T_{C^z}(\alpha, m)$
8.      t=t+1;
9.      if  t>m stop, otherwise return to step 5
10.     end if
11. else
12.     return  $C^h = (c_1^h, c_2^h, \dots, c_m^h)$ as a new offspring
13. end if
```

At the last step, the N best chromosomes from the set of initial population, offspring resulted from crossover operator, and sequences obtained from mutation operator are selected to generate the next population. This process will be continued until the termination condition has reached. At this step, the best sequence for load distribution will be obtained. This chromosome gives the minimum processing time of scheduling of local image operators onto the mentioned platforms. If the termination condition is not satisfied, then we need to back to the third step, using the roulette wheel again to select the parents for crossover operator.

Using a genetic algorithm to find the best sequence for load distribution on a multi-level tree network is similar to the star topology. The process of the proposed genetic algorithm on tree topology is presented next.

1. **Generation of the initial population:** randomly generate N chromosomes denoted by $C^i = (C_1^{S_0}, \dots, C_{m_{S_0}}^{S_0}, C_1^{S_1}, \dots, C_{m_{S_n}}^{S_n})$. Every processor in each star is considered once from S_0 to S_n in each chromosome. $t = 0$ is the generation number.
2. **Evaluation:** the chromosomes generated in Step 1 are evaluated based on their processing time. Algorithm 1 will be considered for every star network $S^k, 1 \le k \le n$ in a multi-level tree. For each sequence of load distribution, two processors, as described before, will be replaced by one equivalent processor and the processing time will be obtained for tree network.
3. **Roulette wheel:** some parents based on the fitness value will be selected for the next steps. Although, this step gives sequences with lower processing time more chance to appear in the next step, other sequences will still have a chance to participate in the crossover step.
4. **Crossover:** parents with probability P_{cros} from the population of Step 3 will be selected. A new offspring will be generated according to the importance of processors. A weight value is allocated to each gene based on its workload. In this new way, there is more chance that good genes will be appeared in the offspring. A new set of population is formed by the resulted offspring, expressed by N_{cros}.
5. **Mutation:** parents with probability P_{mut} will be selected from N_{cros}. For generation of the better sequences, the values of the two genes will be exchanged. The mutation operator is applied to each parent and generates new offspring denoted by N_{mut}.
6. **Selection:** The best chromosomes from $N \cup N_{cros} \cup N_{mut}$, the initial population, and sequences from resulted crossover and mutation, respectively, are maintained for the next generation, $t = 1$.
7. **Termination criteria:** If the stop condition holds, the best sequence for load distribution should be maintained as the optimal solution; otherwise, we should back to Step 3.

4 | SIMULATION RESULTS AND ANALYSIS

The efficiency of our previous results will be investigated with several simulation experiments in this section. Figure 5 presents the effects of different kernel sizes on the processing time and the speed up for different numbers of processors on star and three-level tree networks. For this experiment, we consider $N_1 \times N_2 = 520$ and the parameters that are used for the platforms were given by Mingsheng.[15]

The effects of using different kernel sizes on the processing time in star and tree networks are shown in Figures 5A and 5B, respectively. Results show that, by increasing the kernel size, the processing time will increase as well. Clearly, when we increase the size of kernel, the number of multiplications that each individual processor needs to compute its fraction will increase as well, which leads to increasing the computational time. As it is shown in these figures, the kernel size has more effects on the processing time when a fewer number of processors are considered. Moreover, it is shown that by using larger kernel sizes, adding more processors can reduce the processing time considerably in comparison to smaller kernel sizes.

By increasing the size of kernel, the ratio of computation time to communication time will increase too. This leads to an increase in the computational demand. The effect of kernel size on speed up for different number processors is depicted in Figure 5C. Speed up is equal to the

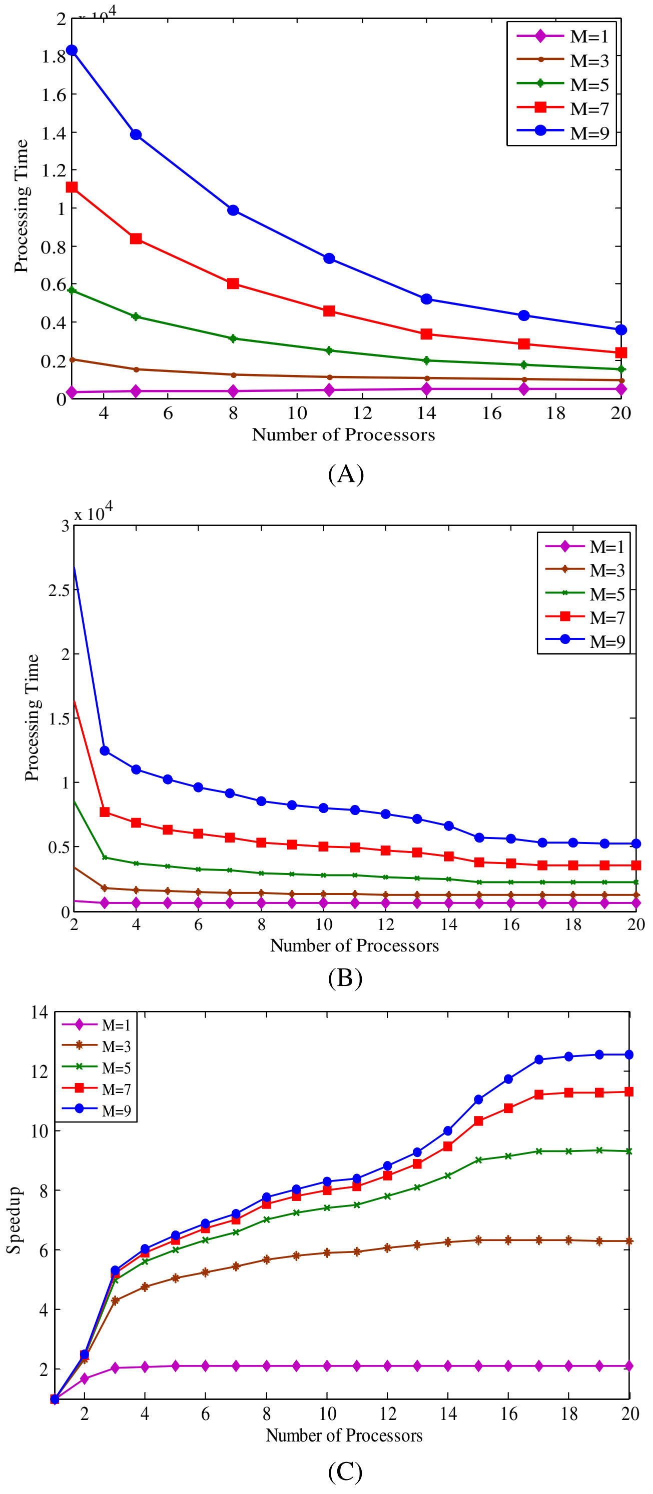

FIGURE 5 Effect of different kernel sizes on (A) processing time for different numbers of processors on star and (B) tree networks and (C) speed up for different numbers of processors on star topology

ratio of processing time using one processor to processing time using m processors and it is given by

$$S = \frac{N_1 N_2 M^2 E}{T(\alpha, m)}. \tag{22}$$

It is clear in Figure 5C that, when the kernel size increases, speed up will increase as well. In addition, speed up increases for larger kernel sizes in comparison to smaller ones. The larger kernel sizes are more dependent on the number of processors.

Based on our knowledge, there is not an optimal method for divisible load scheduling of a constant workload in an arbitrary time onto a heterogeneous topology by considering different values for computation and communication overheads. Since the order of load distribution has a considerable effect on the processing time, we use the genetic algorithm to obtain the best order of load distribution. Most of the parameters used in our simulation were given by Mingsheng.[15] Other parameters are as follows: $N_1 = [100, \ldots, 1000]$, $M = 3$, $N = 100$, $P_{cros} = 0.6$, $P_{mut} = 0.02$, and stop criterion $t = 1500$. To reduce the search space, 100 chromosomes are considered as the initial population and, by using roulette wheels method, we have tried to reduce the number of generations also.

The performance of our new algorithm is compared with three well-known algorithms in this area. IC, IE, and ICE indicate the algorithms in which the workload is distributed in a non-decreasing order of C_i,[13] non-decreasing order of E_i[14] and non-decreasing order of $E_i \times C_i$,[15] respectively. IDLS-GA (Image Divisible Load Scheduling by Genetic Algorithm) indicates our new algorithm. These algorithms were selected because they are dependent on the communication time, computation time, and both of them, respectively. IDLS-GA can be dependent on all the parameters, so finding the optimum sequence for load distribution is a complex issue and cannot be obtained by common methods of load allocations. In addition, by comparing the new algorithms to these methods, the effectiveness of our algorithms will be verified.

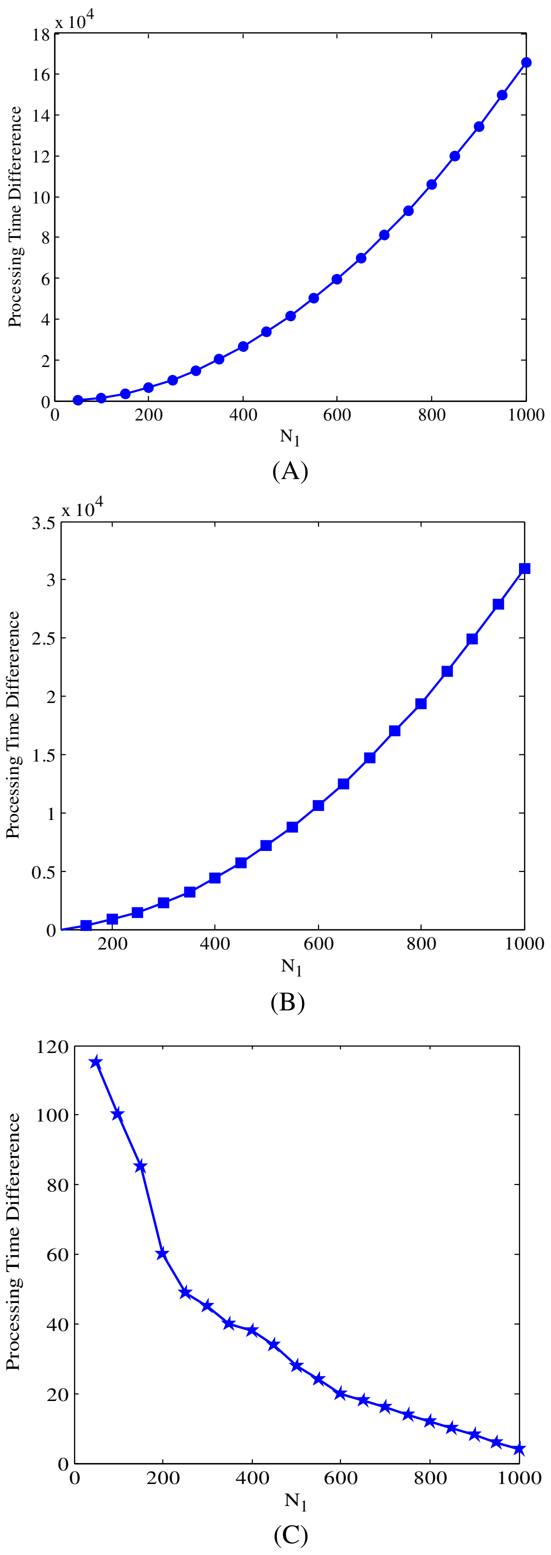

FIGURE 6 The processing time difference between IDLS-GA and (A) IE, (B) ICE, and (C) IC on star network

The processing time difference between the IDLS-GA and the other three algorithms for star topology is presented in Figure 6. In Figure 6A, the time differences between IDLS-GA and IE is presented and this difference is considerable. Clearly, by increasing the number of image columns, the processing time difference between the two algorithms will increase as well. The time difference between IDLS-GA and ICE is depicted in Figure 6B. The results show that the processing time difference between these two algorithms is significant and, by increasing the workload, this value will increase as well. Obviously, the time difference between ICE and IDLS-GA is less than IDLS-GA and IE. This shows that the influence of communication speed of links on the processing time in such heterogeneous platforms is significant. The time difference between IC and our new algorithm is presented in Figure 6C. The new algorithm has lower processing time in comparison with IC, especially for smaller images. By increasing the number of image columns, the time difference between these two algorithms decreases. This result shows that, when we use a large image, the decreasing order of communication speeds is the best sequence. Thus, if the load is distributed according to this order, the minimum processing time will be obtained.

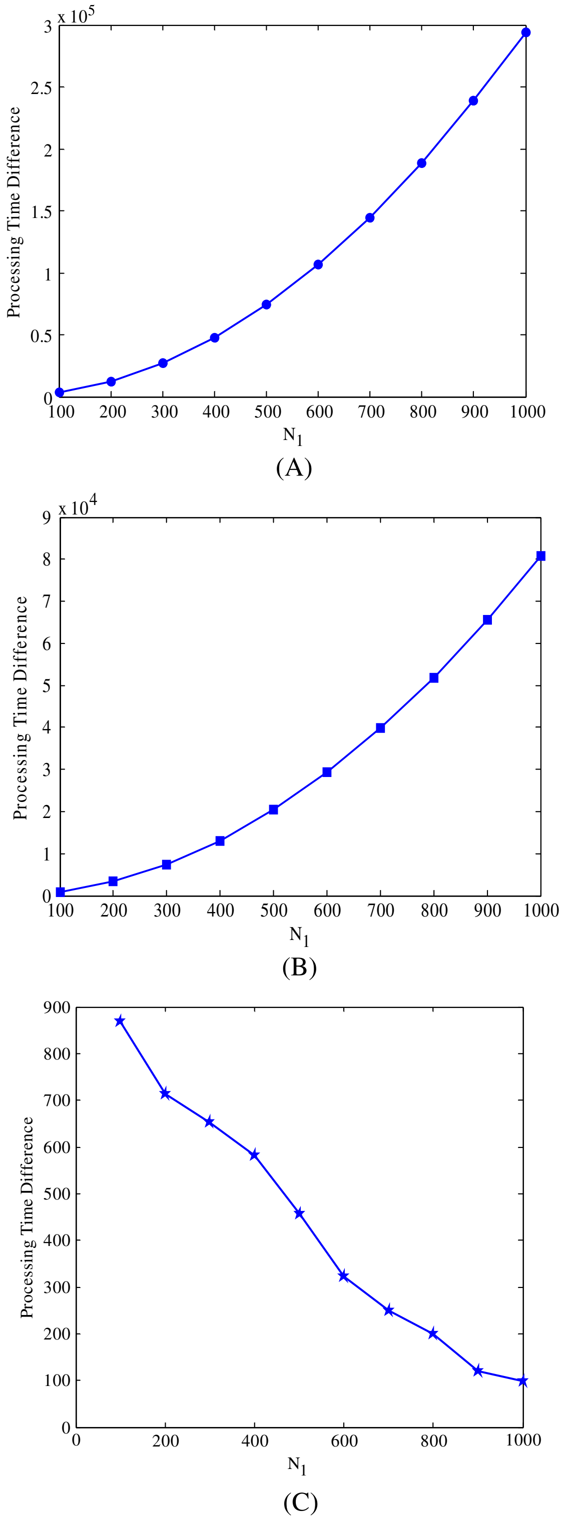

FIGURE 7 The processing time difference between IDLS-GA and (A) IE, (B) ICE, and (C) IC on 3-level tree network

Similarly, the performance of our algorithm is compared with IE, ICE, and IC for three-level tree network in Figure 7A to 7C, respectively. IDLS-GA performs better than others, especially for large images. The processing time difference between the new algorithm and IE is considerable which shows distributing load according to the computation parameter on the mentioned platform leads to more processing time. The new algorithm outperforms the ICE as well according to Figure 7B. The time difference between IDLS and IC in Figure 7C is less considerable for large images. Therefore, IDLS-GA outperforms other three algorithms by finding optimum order of load distribution. Moreover, the results show that the new algorithm has better performance on the multi-level tree platform in comparison with the star network. The processing time difference between the new algorithm and IE, ICE, and IC algorithms is more on the three-level tree than on the star network.

5 | CONCLUSION

In this paper, a new method for optimal scheduling of image processing applications on heterogeneous star and tree networks by using a genetic algorithm has been developed. We use divisible load scheduling for exploiting data parallelism for local image operations on two platforms. In our model, processors and communication links have different speeds and overheads. A closed-form solution for obtaining the processing time, the number of image partitions that should be assigned to each processor, and the optimum number of processors was achieved. A new concept for equivalent processor in order to keep the sequence of load distribution introduced.

A new genetic algorithm, IDLS-GA, was introduced to obtain an efficient sequence of load distribution. The effect of different kernel sizes on processing time and speed up was investigated as well. The performance of our new algorithm on processing time was compared with three common algorithms in this area. Results show that the new algorithm reduces the processing time considerably, especially on multi-level tree topology.

The results indicate that, for large images, the performance of the new algorithm is relatively close to IC, which implies that if the workload is large enough the load distribution sequence follows the decreasing order of communication speed.

ORCID

Sahar Nikbakht Aali https://orcid.org/0000-0003-3126-591X

REFERENCES

1. Ghanbari S, Othman M. Comprehensive review on divisible load theory: concepts, strategies, and approaches. *Math Probl Eng*. 2014;2014:1-13.
2. Lee C, Hamdi M. Parallel image processing applications on a network of workstations. *Parallel Comput*. 1995;21(1):137-160.
3. Bharadwaj V, Li X, Ko CC. Efficient partitioning and scheduling of computer vision and image processing data on bus networks using divisible load analysis. *Image Vis Comput*. 2000;18(11):919-938.
4. Li X, Veeravalli B, Ko CC. Distributed image processing on a network of workstations. *Int J Comput Appl*. 2003;25(2):1-10.
5. Barlas GD. Collection-aware optimum sequencing of operations and closed-form folutions for the distribution of a divisible load on arbitrary processor trees. *IEEE Trans Parallel Distributed Syst*. 1998;9(5):429-441.
6. Bharadwaj V, Ghose D, Robertazzi T. Divisible load theory: a new paradigm for load scheduling in distributed systems. *Clust Comput*. 2003;6(1):7-17.
7. Ghanbari S, Othman M, Baker M, Leong W. Multi-objective method for divisible load scheduling in multi-level tree network. *Future Gener Comput Syst*. 2016;54:132-143.
8. Veeravalli B, Yao J. Divisible load scheduling strategies on distributed multi-level tree networks with communication delays and buffer constraints. *Comput Commun*. 2004;27(1):93-110.
9. Beaumont O, Casanova H, Legrand A, Robert Y, Yang Y. Scheduling divisible loads on star and tree networks: results and open problems. *IEEE Trans Parallel Distributed Syst*. 2005;16(3):207-218.
10. Beaumont O, Legrand A, Robert Y. Optimal algorithms for scheduling divisible workload on heterogeneous systems. In: Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03); 2003; Nice, France.
11. Ghanbari S, Othman M. Time cheating in divisible load scheduling: sensitivity analysis, results and open problems. *Procedia Comput Sci*. 2018;125:935-943.
12. Chen C-Y. Scheduling divisible loads on heterogeneous linear networks using pipelined communications. Paper presented at: 17th World Congress of International Fuzzy Systems Association and 9th International Conference on Soft Computing and Intelligent Systems (IFSA-SCIS); 2017; Otsu, Japan.
13. Wang X, Wang Y, Lai J. Divisible load scheduling for network-based computing systems with processor startup overheads and release times. Paper presented at: 12th International Conference on Computational Intelligence and Security (CIS); 2016; Wuxi, China.
14. Veeravalli B, Li X, Ko CC. On the influence of start-up costs in scheduling divisible loads on bus networks. *IEEE Trans Parallel Distrib Syst*. 2000;11(12):1288-1305.
15. Mingsheng S. Optimal algorithm for scheduling large divisible workload on heterogeneous system. *App Math Model*. 2008;32(9):1682-1695.
16. Bharadwaj V, Ghose D, Mani V, Robertazzi TG. *Scheduling Divisible Load in Parallel and Distributed Systems*. Los Almitos, California: IEEE Computer Society Press; 1996.
17. Suresh S, Mani V, Omkar SN. The effect of start-up delays in scheduling divisible load on bus networks: an alternate approach. *Comput Math Appl*. 2003;46(1-11):1545-1557.
18. Chen C-Y, Chu C-P. Divisible nonlinear load distribution on heterogeneous single-level trees. *IEEE Trans Aerosp Electron Syst*. 2018. In press.
19. Wang X, Veeravalli B, Rana O. An optimal task-scheduling strategy for large-scale astronomical workloads using in-transit computation model. *Int J Comput Intell Syst*. 2018;11(1):600-607.

20. Aali SN, Shahhoseini H, Bagherzadeh N. Divisible load scheduling of image processing applications on the heterogeneous star network using a new genetic algorithm. Paper presented at: 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP); 2018; Cambridge, UK.
21. Wang X, Wang Y, Meng K. Optimization algorithm for divisible load scheduling on heterogeneous star networks. *J Softw*. 2014;9(7):1757-1766.
22. Brandão J, Noronha T, Resende M, Ribeiro C. A biased random-key genetic algorithm for scheduling heterogeneous multi-round systems. *Int Trans Oper Res*. 2017;24(5):1061-1077.
23. Wang X, Veeravalli B. A genetic algorithm based efficient static load distribution strategy for handling large-scale workloads on sustainable computing systems. In: Sangaiah A, Abraham A, Siarry P, Sheng M, eds. *Intelligent Decision Support Systems for Sustainable Computing*. Vol. 705. Cham, Switzerland: Springer; 2017:7-13.
24. Kardi RL, Boctor FF. An efficient genetic algorithm to solve the resource-constrained project scheduling problem with transfer times: the single mode case. *Eur J Oper Res*. 2018;265(2):454-462.
25. Haowei Z, Xie J, Ge J, Zhang Z, Zong B. A hybrid adaptively genetic algorithm for task scheduling problem in the phased array radar. *Eur J Oper Res*. 2019;272(3):868-878.

How to cite this article: Nikbakht Aali S, Bagherzadeh N. Divisible load scheduling of image processing applications on the heterogeneous star and tree networks using a new genetic algorithm. *Concurrency Computat Pract Exper*. 2019;e5498. https://doi.org/10.1002/cpe.5498