

Divisible Non-linear Load Distribution on Heterogeneous Single-Level Trees

Chi-Yeh Chen and Chih-Ping Chu

Abstract

This work studies the divisible non-linear load distribution problem on heterogeneous single-level tree networks with the collective communication model. The goal is to find a feasible distribution that minimizes the parallel processing time. The classical model of non-linear computational loads omits many processing steps, and yields only an approximate solution to distribute fractional loads. This work considers a new model of non-linear computational loads that includes all of processing steps of the load. This model can simplify recursive equation for the size of fractional loads and yield a practical solution to distribute fractional loads. This work proposes two new methods which incorporates a new non-linear computational model to distribute a divisible non-linear load on heterogeneous single-level tree networks. Closed-form expressions for the parallel processing time and speed-up for single-level tree networks are derived. This work demonstrates that the asymptotic speed-up of the proposed algorithm is $m + 1$ where m is the number of child processors in a single-level tree network. We show that our algorithm improved the previous method in terms of speed-up.

Index Terms

divisible load theory, non-linear loads, single-level tree, load distribution, multi-installment.

I. INTRODUCTION

The divisible load theory (DLT) was proposed by Cheng and Robertazzi [19] in 1988. A divisible load can be arbitrarily partitioned and can be independently processed on any processor in the network. The goal is to determine the optimal fractions of the load to be assigned to the processors for minimizing the total processing time. The divisible load can be applied in many practical applications such as linear algebra [31], image processing [52], multimedia applications [50], database searching, large-scale data file processing [42], data intensive applications [43], numerical computing [59], biomedicine and bioinformatics [53], and Internet packet scheduling [35].

Numerous interconnection topologies with scheduling policies have been studied, such that bus [35], [67], linear array [14], tree [3], [6], [7], [15], [18], [49], hypercube [9], [12], [16], [51], mesh [10], [11], [12], [14], [22],

The authors are with the Department of Computer Science and Information Engineering, National Cheng Kung University, No. 1, University Road, Tainan, Taiwan, ROC.

E-mail: chency@csie.ncku.edu.tw, chuap@csie.ncku.edu.tw.

[32], [46], [48], partitionable networks [45], [47], arbitrary networks [68], [69], clusters [25], [63], grids [65], and networks of workstations [4], [52]. The scheduling policies include multiple loads [23], limited memory [26], [61], simultaneous distribution [36], [55], simultaneous start [41], detailed parameterizations and solution time optimization [1], combinatorial schedule optimization [28] and multi-installment processing. Suresh *et al.* [56] design a scheduling strategy for heterogeneous computing resources with shared data banks in a compute cloud system. Wang and Robertazzi [66] proposed a scheduling model for single level tree networks with various distribution policies in which the communication time from the root to each node is nonlinear in the size of the load. Kyong and Robertazzi [44] applied the divisible load theory to the problem of signature search time evaluation in flat file databases.

The single-level tree network is a popular topology for the master-worker style computations. A master-worker computation can be easily implemented and deployed on computing platforms ranging from small commodity clusters to computational grids [30], [5]. This work considers heterogeneous single-level tree networks with the collective communication model. In the collective broadcast model, the root processor has separate ports and can distribute the load fractions simultaneously [33]. The multi-installment processing can minimize the parallel processing time, in which at least one processor receives at least two fractional loads. Numerous multi-installment divisible load algorithms for chains, stars, and trees can be found elsewhere [7], [6], [21], [24], [67].

A. Related Work

Much research has dealt with real-time modeling and simulation of complex systems which include nuclear modeling, aircraft/spacecraft simulation, biological system, bio-physical modeling, genome search, etc. In these areas, many algorithms require non-linear computational complexity, that is, the computational time of the given load is non-linear in the size of load. For example, the Hough transform [27], the 2D hidden Markov mode (HMM) [54], the learning vector quantization neural network [40] and the block tri-diagonalization of real symmetric matrices [2], which have second-order computational time complexity.

The classical Hough transform can be used to identify lines within an image, but later the Hough transform has been extended to use for the detection of regular curves such as lines, circles, ellipses, etc. Guil *et al.* [34] shown the parallelization of the Hough transform for multi-processors with shared and distributed memory. Carlson *et al.* [13] consider how well a Hough transform detector with binary integration improves the performance of a typical surveillance radar. Using the Hough transform, Zhang *et al.* [70] proposed an imaging method for moving targets with rotating parts. The separable 2D Hidden Markov Model (HMM) was proposed by Othman and Aboulnasr [54] for the problem of face recognition. This model allows the state transition to be separated into vertical and horizontal state transitions. This separation of state transitions brings the complexity of the hidden layer of the proposed model from the order of $O(L^3k)$ to the order of $O(L^2k)$, where L is the number of the states in the model and k is the total number of observation blocks in the image. Vakanski *et al.* [62] proposed a method for trajectory learning and generation using a robot PbD approach. The idea is to relate the transitions between different types of movements that were used for encoding the relevant features of the trajectories. By constructing a HMM predicting the probability

of residing in each motion primitive, Field et al. [29] presented an approach for learning robust models of humanoid robot trajectories from demonstration. The learning vector quantization neural network was developed by Kohonen et al. [40] for pattern recognition. Inggs and Robinson [39] investigated the classification of ship targets using low resolution down-range radar profiles together with preprocessing and neural networks. The block tri-diagonalization was proposed by Bai and Ward [2] for computing eigensystems. This method is a critical pre-processing step for the block tri-diagonal divide-and-conquer algorithm and is useful for many algorithms desiring the efficiencies of block structure in matrices.

The non-linear cost function was firstly used in the literatures [20], [38]. Drozdowski and Wolniewicz [20] considered distributed systems which have both the hierarchical memory model and a piecewise linear dependence of the processing time on the size of the assigned load. Hung and Robertazzi [38] proposed a distribution algorithm in which the computational loads require non-linear processing time depending on the size of load fractions. Hung and Robertazzi [37] considered a scheduling model for a tree network where the computation time for each node is non-linear in the size of the assigned load. Suresh *et al.* [57] presented a distribution algorithm for non-linear computational loads in a single level tree network with the collective communication model. Suresh *et al.* [60] developed a distribution algorithm for distributing the second-order non-linear load in a master-slave paradigm with non-blocking mode of communication. In other words, the master processor distributes the load fractions one-by-one to the slave processors. The model of non-linear load in [57], [60] has the following problems. It omits many processing steps for the load, and yields only an approximate solution to distribute fractional loads. Therefore, Chen and Chu [17] proposed a novel computational model of non-linear loads that includes complete steps for processing them. This model solves the problem of the classical model; whose performance degrades by separating the load. They also proposed two algorithms to distribute a non-linear divisible load on a homogeneous linear network.

B. Contribution

This work uses a novel computational model of non-linear loads that is proposed by Chen and Chu [17]. This model can simplify recursive equation for the size of fractional loads and yield a practical solution to distribute fractional loads. An algorithm \mathbb{S} (Single-installment) is presented to distribute a non-linear load on single-level trees. The closed-form expressions for the parallel processing time and speed-up for single-level tree networks are derived (see Theorems 3, 4, 6 and 9). This paper shows that the asymptotic speed-up of the proposed algorithm is $m+1$ where m is the number of child processors in a homogeneous single-level tree network. The performance of algorithm \mathbb{S} improves the previous method in [57] since the performance of previous method degrades by separating the load. This work also proposes an algorithm \mathbb{M} (Multi-installment) that uses multi-installment processing to reduce the initial distribution time and to improve upon the algorithm \mathbb{S} . The affine cost model is also considered. In this model, the communication time and the computation time of a load L are $\theta_{cm} + LG$ and $\theta_{cp} + LA$, respectively, where θ_{cm} is the communication start-up cost, θ_{cp} is the computation start-up cost, G is the time to transmit a unit of load and A is the time to process a processing step. The communication start-up cost θ_{cm} due to protocol processing delays, unavailability of certain internal and external communication resources, queuing delays at intermediate sites,

etc., [58], [64]. The computation start-up cost θ_{cp} due to delay in layered protocol, extracting the data, processor initialization, etc., [58], [64]. When the computation and communication start-up costs are considered, this work finds two ranges for searching an optimal number of installments and an optimal number of child processors.

C. Organization

The rest of this paper is organized as follows: Section II presents the model of divisible non-linear load distribution in static interconnection networks. Section III reviews the classic method for non-linear divisible loads distribution. Section IV describes and analyzes the algorithm for distributing divisible non-linear loads on single-level trees. Section V describes and analyzes the algorithm that uses multi-installment to distribute divisible non-linear loads on single-level trees. Section VI compares the performance of the classical method with that of the proposed methods. Section VII draws conclusions.

II. MODEL

This work considers a heterogeneous single-level tree network. A single-level tree is composed of $m+1$ processors P_0, P_1, \dots, P_m , that connected by a static interconnection network. The processor P_0 has m neighbors P_1, \dots, P_m that are linked by communication links (l_1, l_2, \dots, l_m) , as shown in Fig. 1. The root processor P_0 is assumed to be the initial processor that transmits fractional loads to other processors for processing. All of the links have different communication speeds and bandwidths. All of the processors have different processing capacities. The root processor P_0 has separate ports to communicate with all neighbors. Therefore, the root processor can simultaneously send messages to all of its neighbors. A processor sends a fractional load to a neighbor and then can proceed with other computational and communicative activities without waiting for the completion of the sending of the fractional load. Thus, it can perform computation and communication simultaneously. However, a neighbor begins processing its fractional load only after it receives the entire fractional load from its predecessor. The time to return the results is assumed to be negligible if it is so short in comparison with the load distributing and processing steps. It can be observed in the DLT papers [17], [46], [57], [60]. In this assumption, the schedule for a load is the shortest when all processors finish computing at the same time. Table I presents the notation and terminology that are used herein.

In this work, a divisible non-linear load is considered as a matrix type computation. A matrix can be partitioned into a list of sub-matrices (or subsets). The model and algorithm design are also based on the matrix type computation. The non-linear computational load is explained in Definition 1.

Definition 1. *An entire load can represent a nonempty data set \mathcal{S} that comprises L elements. A γ^{th} -order computational load with a data set \mathcal{S} requires to process a γ -dimension data set \mathcal{S}^γ . A fraction of load with a subset \mathcal{S}_i requires to process a γ -dimension data set $\mathcal{S}_i \times \mathcal{S}^{\gamma-1}$.*

A partition of a nonempty set \mathcal{S} is a list $\mathcal{S}_0, \mathcal{S}_1, \dots, \mathcal{S}_m$ of subsets of \mathcal{S} such that each element of \mathcal{S} appears in one and only one subset in the list. Let $F(\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_\gamma)$ be the number of processing steps (instructions) for a

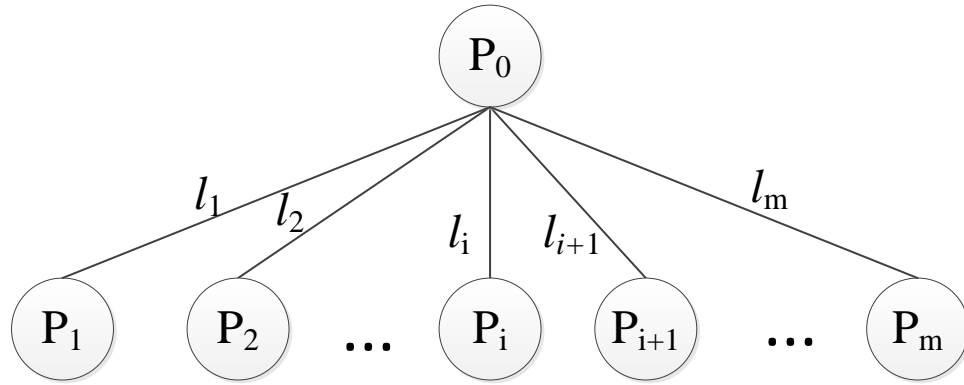


Fig. 1: A single-level tree network.

general γ^{th} -order computational load $\mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_\gamma$ where \mathcal{X}_j is a subset of \mathcal{S} for $1 \leq j \leq \gamma$. A processing step can be defined as a basic operation in a specific algorithm. Each processing step takes the same time under the same computing capability. The data set can be arbitrarily partitioned and the computation times for each fraction of data set are non-linear in the size of the data sets. For example, a second-order complexity load that comprises L elements require L^2 steps; then $F(\mathcal{S}, \mathcal{S}) = L^2$ (steps). For a subset \mathcal{S}_1 with size of αL elements, the number of processing steps is

$$F(\mathcal{S}_1, \mathcal{S}_1) = (\alpha L)^2 = \alpha^2 L^2 = \alpha^2 F(\mathcal{S}, \mathcal{S}).$$

If a fraction α is partitioned into α_1 and α_2 , then the number of processing steps can be rewritten as follows.

$$\begin{aligned} F(\mathcal{S}_1, \mathcal{S}_1) &= (\alpha L)^2 = \alpha^2 L^2 = (\alpha_1 + \alpha_2)^2 L^2 \\ &= (\alpha_1^2 + \alpha_1 \alpha_2 + \alpha_2 \alpha_1 + \alpha_2^2) L^2 \\ &= F(\mathcal{S}_{1,1}, \mathcal{S}_{1,1}) + F(\mathcal{S}_{1,1}, \mathcal{S}_{1,2}) + F(\mathcal{S}_{1,2}, \mathcal{S}_{1,1}) + F(\mathcal{S}_{1,2}, \mathcal{S}_{1,2}) \end{aligned}$$

where \mathcal{S}_1 is partitioned into $\mathcal{S}_{1,1}$ and $\mathcal{S}_{1,2}$, which correspond to a fraction α_1 and a fraction α_2 , respectively.

The set \mathcal{S} can be partitioned into a list $\mathcal{S}_0, \mathcal{S}_1, \dots, \mathcal{S}_m$ of subsets with the corresponding fractions $\alpha_0, \alpha_1, \dots, \alpha_m$. The number of processing steps for the data set \mathcal{S} can be partitioned as

$$\begin{aligned} F(\mathcal{S}, \mathcal{S}) &= \sum_{i=0}^m F(\mathcal{S}_i, \mathcal{S}) \\ &= \sum_{i=0}^m F(\mathcal{S}_i, \mathcal{S}_i) + F(\mathcal{S}_i, \mathcal{S} \setminus \mathcal{S}_i) \end{aligned}$$

where $F(\mathcal{S}_k, \mathcal{S}_k) = \alpha_k^2 L^2$ and $F(\mathcal{S}_k, \mathcal{S} \setminus \mathcal{S}_k) = \alpha_k (1 - \alpha_k) L^2$ for $0 \leq k < m$.

Careful consideration of the above equation reveals that the first term is a summation of the processing steps for the two-dimension data set $\mathcal{S}_i \times \mathcal{S}_i$, while the second term is the number of processing steps for the two-dimension

TABLE I: Notations and Terminology

\mathcal{P}	The set of child processors in a multicomputer system.
m	The number of child processors in a multicomputer system.
L	The total load to be processed.
G_i	The time to transmit a unit of load along the link l_i .
A_i	The time to process a processing step on the processor P_i .
β_i	The computation-to-communication ratio for the processor P_i . $\beta_i = A_i/G_i$
θ_{cp}	The computation start-up costs in terms of delay time.
θ_{cm}	The communication start-up cost in terms of delay time.
γ	Integer constant depends on the nature of the algorithm used for processing the load. For example, this value is 2 for second-order nonlinear system.
α_i	Fraction of the processing load assigned to processor P_i , $i = 0, 1, \dots, m$.
$\alpha_{i,j}$	j^{th} fraction of the processing load assigned to processor P_i in the second step, $i = 0, 1, \dots, m$.
\mathcal{S}	The data set of the entire load, where $ \mathcal{S} = L$.
$\mathcal{S}_0, \mathcal{S}_1, \dots, \mathcal{S}_m$	A partition of the data set \mathcal{S} where the fraction of each load is denoted by $\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_m$ and P_0 sends \mathcal{S}_i to P_i , $i = 0, 1, \dots, m$ in the first step.
ρ	The number of installments in the proposed algorithm \mathbb{M} .
n_i^*	The minimum number of communications required to transfer the entire data set to P_i .
$T_{m+1}^{\mathbb{A}}, T_{m+1,\rho}^{\mathbb{A}}$	The parallel processing time of L units of load by using the load distribution algorithm \mathbb{A} in a system of $m+1$ processors. For algorithm with multi-installment, we use $T_{m+1,\rho}^{\mathbb{A}}$
$Speedup_{m+1}^{\mathbb{A}}$	The ratio of the sequential processing time to the parallel processing time in a system of $m+1$ processors, namely $Speedup_{m+1}^{\mathbb{A}} = T_1/T_{m+1}^{\mathbb{A}}$.
C_i^N	$C_i^N = N(N-1) \times \dots \times (N-i+1)/i!$ which is the number of combinations of i components selected from a set of N components.

data set $\mathcal{S}_i \times (\mathcal{S} \setminus \mathcal{S}_i)$. This work requests that each processor P_k can obtain a data set \mathcal{S}_k as first communication process. Then, P_k can compute the data set $\mathcal{S}_k \times \mathcal{S}_k$. While P_k is computing the data set $\mathcal{S}_k \times \mathcal{S}_k$, it receives the remaining data set $\mathcal{S} \setminus \mathcal{S}_k$. Finally, P_k computes the data set $\mathcal{S}_k \times (\mathcal{S} \setminus \mathcal{S}_k)$. In conclusion, P_k computes $F(\mathcal{S}_k, \mathcal{S}) = F(\mathcal{S}_k, \mathcal{S}_k) + F(\mathcal{S}_k, \mathcal{S} \setminus \mathcal{S}_k)$ processing steps for the data set $\mathcal{S}_k \times \mathcal{S} = (\mathcal{S}_k \times \mathcal{S}_k) \cup (\mathcal{S}_k \times (\mathcal{S} \setminus \mathcal{S}_k))$. Example 1 shows an application of non-linear computational load.

Example 1. The discrete cosine transform (DCT) is often used in signal and image processing. An image can represent a $a \times b$ matrix \mathcal{S} . Thus, the transform of \mathcal{S} is given by

$$D(u, v) = \frac{C(u)C(v)}{\sqrt{ab}} \sum_{y=0}^{b-1} \sum_{x=0}^{a-1} \mathcal{S}(x, y) \cos\left(\frac{(2x+1)u\pi}{2a}\right) \cos\left(\frac{(2y+1)v\pi}{2b}\right)$$

for $u = 0, 1, \dots, a-1$ and $v = 0, 1, \dots, b-1$ where $C(i) = 1/\sqrt{2}$ for $i = 0$ and $C(i) = 1$ for $i \neq 0$. A processing step of DCT is a basic operation

$$\frac{C(u)C(v)}{\sqrt{ab}} \mathcal{S}(x, y) \cos\left(\frac{(2x+1)u\pi}{2a}\right) \cos\left(\frac{(2y+1)v\pi}{2b}\right).$$

The matrix \mathcal{S} can be partitioned into a list of sub-matrices $\mathcal{S}_0, \mathcal{S}_1, \dots, \mathcal{S}_m$. Let \mathcal{R}_k be the coordinates of \mathcal{S}_k within

\mathcal{S} . Let \mathcal{R} be the coordinates of \mathcal{S} . After processor P_k receives the sub-matrix \mathcal{S}_k , it computes

$$D(u, v) = \frac{C(u)C(v)}{\sqrt{ab}} \sum_{(x,y) \in \mathcal{R}_k} \mathcal{S}(x, y) \cos\left(\frac{(2x+1)u\pi}{2a}\right) \cos\left(\frac{(2y+1)v\pi}{2b}\right)$$

for all $(u, v) \in \mathcal{R}_k$. After P_k receives the data $\mathcal{S} \setminus \mathcal{S}_k$, it performs

$$D(u, v) = D(u, v) + \frac{C(u)C(v)}{\sqrt{ab}} \sum_{(x,y) \in \mathcal{R} \setminus \mathcal{R}_k} \mathcal{S}(x, y) \cos\left(\frac{(2x+1)u\pi}{2a}\right) \cos\left(\frac{(2y+1)v\pi}{2b}\right)$$

for all $(u, v) \in \mathcal{R}_k$.

A γ^{th} -order computational data set \mathcal{S}^γ can be partitioned into a list of subsets $\mathcal{S}_0 \times \mathcal{S}^{\gamma-1}, \mathcal{S}_1 \times \mathcal{S}^{\gamma-1}, \dots, \mathcal{S}_m \times \mathcal{S}^{\gamma-1}$. Each processor P_k deals with the γ -dimension data set $\mathcal{S}_k \times \mathcal{S}^{\gamma-1}$ for $0 \leq k \leq m$. Each data set $\mathcal{S}_k \times \mathcal{S}^{\gamma-1}$ can be partitioned into $2^{\gamma-1}$ subsets $\{\mathcal{S}_k \times \mathcal{X}_1 \times \mathcal{X}_2 \times \dots \times \mathcal{X}_{\gamma-1}\}$ where $\mathcal{X}_j = \mathcal{S}_k$ or $\mathcal{X}_j = \mathcal{S} \setminus \mathcal{S}_k$ for $1 \leq j \leq \gamma-1$. Each processor P_k obtains the data set \mathcal{S}_k at the first communication process, then computes the subset \mathcal{S}_k^γ . While P_k is computing, it iteratively receives the data set $\mathcal{S} \setminus \mathcal{S}_k$ and computes the data set $\mathcal{S}_k \times \mathcal{S}^{\gamma-1} - \mathcal{S}_k^\gamma$.

Let $F_k(s_1 s_2 \dots s_\gamma) = F(\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_\gamma)$ be a part of the number of processing steps for the subset \mathcal{S}_k where $\mathcal{X}_j = \mathcal{S}_k$ if $s_j = 1$ or $\mathcal{X}_j = \mathcal{S} \setminus \mathcal{S}_k$ if $s_j = 0$ for $1 \leq j \leq \gamma$. For convenience of specification of the algorithms, the binary number $s_1 s_2 \dots s_\gamma$ can be converted into a decimal number. For example, $F_k(14) = F_k(1110) = F(\mathcal{S}_k, \mathcal{S}_k, \mathcal{S}_k, \mathcal{S} \setminus \mathcal{S}_k)$. Each processor P_k computes

$$F\left(\mathcal{S}_k, \overbrace{\mathcal{S}, \dots, \mathcal{S}}^{\gamma-1}\right) = \sum_{j=2^{\gamma-1}}^{2^\gamma-1} F_k(j)$$

processing steps for the data set $\mathcal{S}_k \times \mathcal{S}^{\gamma-1}$.

The following property is used in the analysis.

Proposition 1. Suppose that x_1 is the largest real root to the equation $f(x) = \sum_{i=1}^n a_i x^i = 0 (a_n \neq 0)$. If $a_n > 0$, then $f(x) > 0$ for $x \in (x_1, \infty)$. If $a_n < 0$, then $f(x) < 0$ for $x \in (x_1, \infty)$.

III. THE CLASSICAL METHOD

The classical divisible load distribution method [57] on a single-level tree network of $m+1$ processors divides the entire load into $m+1$ fractions. The processor P_0 is assumed to be the initial processor that begins the computation and communication. In the first step, P_0 computes a fractional load and simultaneously transmits another fractional load to each child. In the second step, processors P_1, P_2, \dots, P_m compute a fractional load. Figure 2 presents the load distribution diagram of classical method. The classical method assumes that the number of processing steps for a fractional load with size $\alpha_i L$ in the processor P_i is only $F_i(2^\gamma - 1) = (\alpha_i L)^\gamma$ [38], [37], [57], [60]. The remaining processing steps are the post-processing steps. However, as number of fractional loads increases, the number of the post-processing steps increases. Therefore, it performs poorly. Hung and Robertazzii [37] and Suresh *et al.* [57] show that their asymptotic speed-up for a second-order computational load is $(m+1)^2$. That is because the post-processing steps are not taken into account. For example, assume that the transmitting time is neglect and the time to process a processing step is one. We have $m+1$ units of load that distribute to $m+1$ processors. Then,

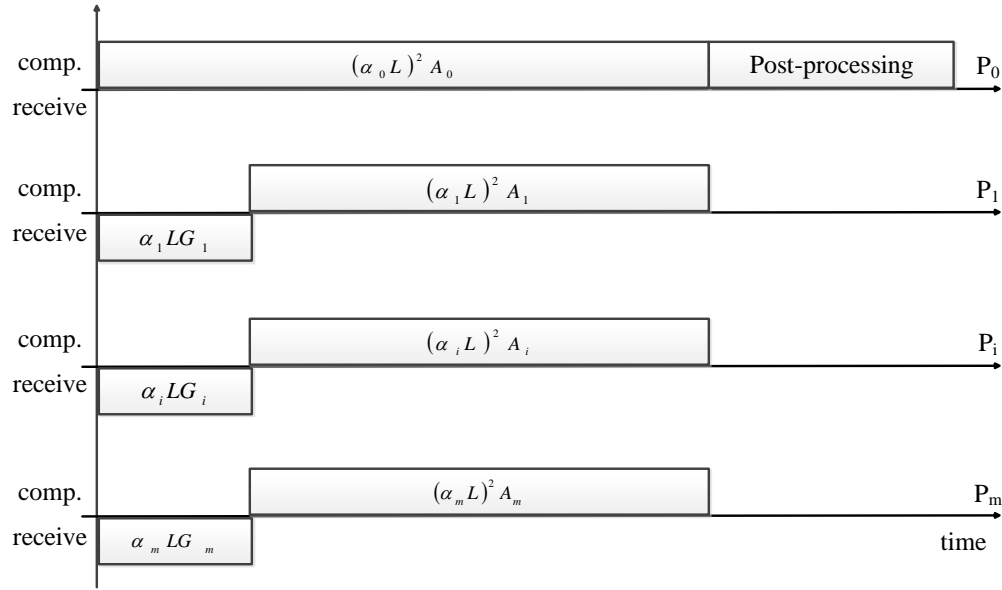


Fig. 2: The load distribution diagram of classical method.

each processor has a unit of load. In this case, the papers [37], [57] show the parallel processing time is one and the speed-up is $(m+1)^2$. However, the number of the post-processing steps is $(m+1)^2 - (m+1)$. The parallel processing time should be $1 + (m+1)^2 - (m+1)$ and the speed-up should be $\frac{(m+1)^2}{1+(m+1)^2-(m+1)}$. In fact, the optimal solution is that each processor evenly computes the same size of loads and all processors finish computing at the same time. Therefore, each processor computes $m+1$ processing steps evenly and finishes at the same time is optimal and the speed-up is $m+1$. In other words, the maximal speed-up of $m+1$ processors for any distribution method is at most $m+1$. Additionally, the classical model suffers from following problems: (i) it omits many of the processing steps for the load; (ii) it exists high-order functions that are difficult to find general solution for the size of fractional loads and yield only an approximate solution to distribute fractional loads. In this work, the number of processing steps of the novel method for a fractional load with size $\alpha_i L$ is $\sum_{j=2^{\gamma-1}}^{2^{\gamma}-1} F_i(j) = \alpha_i L^{\gamma}$ that includes complete processing steps. According to this method, we only need to solve a low-order recursive equation and can easily find general solution.

Since the classical model yield only an approximate solution, exact closed-form expressions for the parallel processing time and speed-up cannot be obtained. However, if the computation speed is significantly less than the communication link speed, the communication time for transmitting load can be neglected and the load can be evenly distributed over all processors. Therefore, the following theorem shows the parallel processing time and speedup of algorithm \mathbb{C} (Classical method) with the classical model.

Theorem 1. *If the computation speed is significantly less than the communication link speed (at least of the order*

of 10, $\beta_i \geq 10$), the parallel processing time of algorithm \mathbb{C} in [57] is

$$T_{m+1}^{\mathbb{C}} = \left(1 - \sum_{i=1}^m \alpha_i^\gamma\right) L^\gamma A_0$$

where $\alpha_i = \alpha_0 \prod_{k=1}^i f_k^{1/\gamma}$, $\alpha_0 = \frac{1}{1 + \sum_{i=1}^m \prod_{k=1}^i f_k^{1/\gamma}}$ and $f_k = \frac{A_{k-1}}{A_k}$. If a homogeneous single-level tree network is considered, then the speedup is given as

$$\text{Speedup}_{m+1}^{\mathbb{C}} = \frac{L^\gamma A}{T_{m+1}^{\mathbb{C}}} = \frac{(1+m)^\gamma}{(1+m)^\gamma - m}$$

where $A_i = A$ and $G_i = G$ for all i .

Since algorithm \mathbb{C} yields only an approximate solution to distribute fractional loads, this work compares with algorithm \mathbb{C} in Theroem 1.

IV. ALGORITHM

This section presents an algorithm \mathbb{S} (Single-installment) whose goal is using the new model with the single-installment technique to distribute a non-linear load in a single-level tree. The pseudocode of the proposed method is given in algorithm \mathbb{S} . We call a load distribution is feasible if the processor can receive the entire data set and no processor is idle after it has received the first subset. The following lemma shows a sufficient condition for a feasible load distribution.

Lemma 2. A load distribution is feasible if $\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i \geq 1 - \alpha_i$ for $1 \leq i \leq m$.

Proof: In the proposed algorithm, the processor P_i receives a fraction with size of $\alpha_i \sum_{j=0}^{n_i-1} \left(\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i\right)^j$ after n_i communication steps (describe in Sections IV-A and IV-B.) A load distribution is feasible if the processor can receive the entire data set and no processor is idle after it has received the first subset. Therefore, the following inequality must be confirmed

$$1 \leq \alpha_i \sum_{j=0}^{n_i-1} \left(\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i\right)^j \quad (1)$$

for some $n_i \in \mathbb{N}$. According to the inequality (1), two cases must be considered: either $\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i \geq 1$ or $\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i < 1$. Here, in the case of $\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i \geq 1$, inequality (1) can easily be verified for a certain $n_i \in \mathbb{N}$. For example, $n_i = \lceil 1/\alpha_i \rceil$ in the case of $\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i = 1$. For $\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i < 1$, the processor P_i can receive a fraction with size of

$$\lim_{n_i \rightarrow \infty} \alpha_i \sum_{j=0}^{n_i-1} \left(\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i\right)^j = \frac{\alpha_i}{1 - \alpha_i^{\gamma-1} L^{\gamma-1} \beta_i}.$$

Therefore, the processor P_i can receive the entire data set if the following constraint is satisfied

$$1 \leq \frac{\alpha_i}{1 - \alpha_i^{\gamma-1} L^{\gamma-1} \beta_i}$$

yielding the constraint $\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i \geq 1 - \alpha_i$. ■

TABLE II: ALGORITHM PE (Processor Elimination)

Algorithm 1 ALGORITHM PE (Processor Elimination)

Output: A set of child processors \mathcal{P} .

- 1: $m = |\mathcal{P}|$
 - 2: $\alpha_0 = \frac{1}{1 + \sum_{i=1}^m \frac{L^\gamma A_0}{L^\gamma A_i + L G_i}}$.
 - 3: $\alpha_i = \frac{\alpha_0 L^\gamma A_0}{L^\gamma A_i + L G_i}$, where $1 \leq i \leq m$.
 - 4: **repeat**
 - 5: **if** $\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i < 1 - \alpha_i$ for some $i \in \{1, 2, \dots, m\}$ **then**
 - 6: find a processor P_i with minimal $\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i + \alpha_i$.
 - 7: remove the processor P_i from \mathcal{P} and reorder $P_{j+1}, A_{j+1}, G_{j+1}$ to P_j, A_j, G_j for $i \leq j \leq m$.
 - 8: $m = |\mathcal{P}|$
 - 9: $\alpha_0 = \frac{1}{1 + \sum_{i=1}^m \frac{L^\gamma A_0}{L^\gamma A_i + L G_i}}$.
 - 10: $\alpha_i = \frac{\alpha_0 L^\gamma A_0}{L^\gamma A_i + L G_i}$, where $1 \leq i \leq m$.
 - 11: **until** $\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i \geq 1 - \alpha_i$ for all $i \in \{1, 2, \dots, |\mathcal{P}|\}$
-

From Lemma 2, we call a processor P_i by useless processor if $\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i < 1 - \alpha_i$. Algorithm S uses a procedure PE (Processor Elimination) to remove useless processors. Then, algorithm S performs the following three operations: (i) P_0 partitions the entire data set into $m + 1$ fractions and distributes m data subsets to each child; (ii) while every processor computes its loads, P_0 transmits the remaining data subsets to all children until every processor has received the entire data set; (iii) every processor performs the remaining processing steps.

A. Algorithm with Single Installment for Second-Order Complexity

This section considers the second-order computational load distribution. Figure 3 presents the load distribution diagram obtained by using the proposed algorithm with $\gamma = 2$. While P_0 computes $(\alpha_0 L)^2 + \alpha_0 (1 - \alpha_0) L^2$ processing steps for processing the two-dimension data set $\mathcal{S}_0 \times \mathcal{S}$, P_0 successively transmits two data sets \mathcal{S}_i and $\mathcal{S} \setminus \mathcal{S}_i$ to P_i where $1 \leq i \leq m$. After P_i receives the data set \mathcal{S}_i with size $\alpha_i L$, it performs $(\alpha_i L)^2$ processing steps for processing the data set $\mathcal{S}_i \times \mathcal{S}_i$ and receives the data set $\mathcal{S}_{i,1} \subseteq \mathcal{S} \setminus \mathcal{S}_i$. Since P_i only can receive a fragment of load with size $(\alpha_i L)^2 \beta_i$ while it preforms $(\alpha_i L)^2$ processing steps, the size of $\mathcal{S}_{i,1}$ is at most $(\alpha_i L)^2 \beta_i$. Therefore, $|\mathcal{S}_{i,1}| = \alpha_{i,1} L = \min \{ \alpha_i^2 L \beta_i, 1 - \alpha_i \} L$ where $(1 - \alpha_i) L$ is the size of $\mathcal{S} \setminus \mathcal{S}_i$. Generally, if P_i receives the data $\mathcal{S}_{i,j}$ and has not received the complete data set (i.e. $\mathcal{S} \setminus \left(\bigcup_{k=1}^j \mathcal{S}_{i,k} \cup \mathcal{S}_i \right) \neq \emptyset$), then P_i computes $\alpha_i \alpha_{i,j} L^2$ processing steps for processing the data set $\mathcal{S}_i \times \mathcal{S}_{i,j}$ and receives the data $\mathcal{S}_{i,j+1} \subseteq \mathcal{S} \setminus \left(\bigcup_{k=1}^j \mathcal{S}_{i,k} \cup \mathcal{S}_i \right)$ with a size of $\alpha_{i,j+1} L = \min \left\{ \alpha_i \alpha_{i,j} L \beta_i, 1 - \alpha_i - \sum_{k=1}^j \alpha_{i,k} \right\} L$. We repeat this procedure until P_i has received the complete data set. In conclusion, P_i computes $\alpha_i L^2 = (\alpha_i L)^2 + \alpha_i (1 - \alpha_i) L^2$ processing steps for processing the data set $\mathcal{S}_i \times \mathcal{S}$. According to [8], the total time for processing the load is minimum only if all processors finish

TABLE III: ALGORITHM S (Single-installment)

ALGORITHM S (Single-installment)

Output: A load distribution.

```

1: call Algorithm PE.
2:  $\alpha_0 = \frac{1}{1 + \sum_{i=1}^m \frac{L^\gamma A_0}{L^\gamma A_i + LG_i}}$ .
3:  $\alpha_i = \frac{\alpha_0 L^\gamma A_0}{L^\gamma A_i + LG_i}$ , where  $1 \leq i \leq m$ .
4:  $\alpha_{i,1} = \min \{ \alpha_i^\gamma L^{\gamma-1} \beta_i, 1 - \alpha_i \}$ , where  $1 \leq i \leq m$ .
5: for all  $i = 0 \rightarrow m$  do in parallel
6:   if  $i = 0$  then
7:      $P_0$  do the following two actions simultaneously.
8:     1) computation
9:       compute  $\sum_{j=2^{\gamma-1}}^{2^\gamma-1} F_0(j)$  processing steps for the corresponding data set  $\mathcal{S}_0 \times \mathcal{S}^{\gamma-1}$ .
10:    2) communication
11:      2.1) send the data set  $\mathcal{S}_i \subseteq \mathcal{S}$  to child  $P_i$  where  $1 \leq i \leq m$  and  $|\mathcal{S}_i| = \alpha_i L$ .
12:      2.2) send the data set  $\mathcal{S}_{i,1} \subseteq \mathcal{S} \setminus \mathcal{S}_i$  to child  $P_i$  where  $1 \leq i \leq m$  and  $|\mathcal{S}_{i,1}| = \alpha_{i,1} L$ .
13:      2.3) send the rest of fractional loads to each processor until all processors receive the entire data set  $\mathcal{S}$ .
14:    else
15:      receive the data set  $\mathcal{S}_i \subseteq \mathcal{S}$  from  $P_0$  where  $|\mathcal{S}_i| = \alpha_i L$ .
16:      compute  $(\alpha_i L)^\gamma$  processing steps for the data set  $\mathcal{S}_i^\gamma$  and receive the data set  $\mathcal{S}_{i,1} \subseteq \mathcal{S} \setminus \mathcal{S}_i$  from  $P_0$ , where  $|\mathcal{S}_{i,1}| = \alpha_{i,1} L$ .
17:       $k_i = 1$ 
18:      repeat
19:        compute  $\alpha_i^{\gamma-1} \alpha_{i,k_i} L^\gamma$  processing steps for the data set  $\mathcal{S}_i^{\gamma-1} \times \mathcal{S}_{i,k_i}$ , and receive the data set  $\mathcal{S}_{i,k_i+1} \subseteq \mathcal{S} \setminus (\bigcup_{k=0}^{k_i} \mathcal{S}_{i,k} \cup \mathcal{S}_i)$ 
        from  $P_0$  where  $\alpha_{i,k_i+1} = \min \{ \alpha_i^{\gamma-1} \alpha_{i,k_i} L^{\gamma-1} \beta_i, 1 - \alpha_i - \sum_{k=1}^{k_i} \alpha_{i,k} \}$  and  $|\mathcal{S}_{i,k_i+1}| = \alpha_{i,k_i+1} L$  if  $\alpha_i + \sum_{k=1}^{k_i} \alpha_{i,k} < 1$ .
20:         $k_i = k_i + 1$ 
21:      until  $\alpha_i + \sum_{k=1}^{k_i-1} \alpha_{i,k} = 1$ 
22:      compute  $\sum_{j=2^{\gamma-1}}^{2^\gamma-3} F_i(j)$  processing steps for the corresponding data set.
```

computing at the same time. For this purpose, the recursive equations for load distribution are

$$\begin{aligned}
\alpha_0 L^2 A_0 &= (\alpha_i L)^2 A_i + \alpha_i (1 - \alpha_i) L^2 A_i + \alpha_i L G_i \\
&= \alpha_i^2 L^2 A_i + \alpha_i (1 - \alpha_i) L^2 A_i + \alpha_i L G_i \\
&= \alpha_i (\alpha_i + 1 - \alpha_i) L^2 A_i + \alpha_i L G_i \\
&= \alpha_i L^2 A_i + \alpha_i L G_i
\end{aligned}$$

which yields

$$\alpha_i = \frac{\alpha_0 L^2 A_0}{L^2 A_i + L G_i} \tag{2}$$

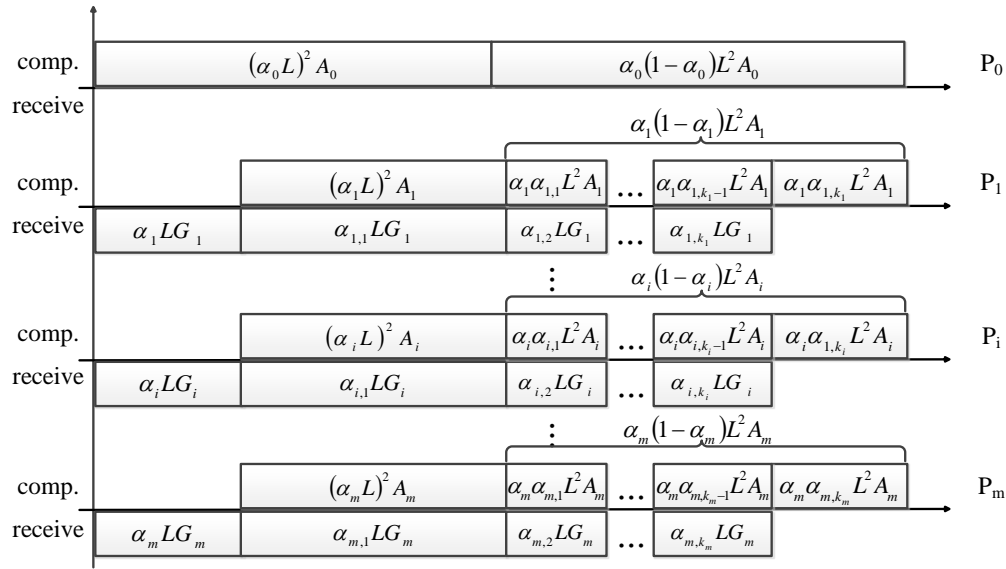


Fig. 3: The load distribution diagram of proposed algorithm with $\gamma = 2$.

for $i = 1, 2, \dots, m$. Since the set \mathcal{S} is partitioned into a list $\mathcal{S}_0, \mathcal{S}_1, \dots, \mathcal{S}_m$ of subsets and $|\mathcal{S}_i| = \alpha_i L$ for $0 \leq i \leq m$, $\sum_{i=0}^m \alpha_i L = L$. Therefore, substituting α_i from (2) in $\sum_{i=0}^m \alpha_i = 1$ gives

$$\alpha_0 = \frac{1}{1 + \sum_{i=1}^m \frac{L^2 A_0}{L^2 A_i + L G_i}}.$$

The following theorem explains the results.

Theorem 3. For a single-level tree network with the root as the initial processor, we have

$$T_{m+1}^{\mathcal{S}} = \alpha_0 L^2 A_0 = \frac{L^2 A_0}{1 + \sum_{i=1}^m \frac{L^2 A_0}{L^2 A_i + L G_i}}.$$

If a homogeneous single-level tree network is considered, then the speedup is given as

$$\text{Speedup}_{m+1}^{\mathcal{S}} = \frac{L^2 A}{T_{m+1}^{\mathcal{S}}} = 1 + \frac{m L \beta}{L \beta + 1}$$

where $A_i = A$, $G_i = G$ and $\beta_i = \beta = \frac{A}{G}$ for all i . Its asymptotic speed-up is $\lim_{L \rightarrow \infty} S_{m+1}^{\mathcal{S}} = m + 1$.

B. Algorithm with Single Installment for γ^{th} -Order Complexity

This section discusses the γ^{th} -order computational load distribution. While P_0 performs $\alpha_0 L^\gamma$ processing steps for processing the γ -dimension data set $\mathcal{S}_0 \times \mathcal{S}^{\gamma-1}$, P_0 successively transmits two data sets \mathcal{S}_i and $\mathcal{S} \setminus \mathcal{S}_i$ to each P_i where $1 \leq i \leq m$. After P_i has received the data set \mathcal{S}_i , it performs $(\alpha_i L)^\gamma$ processing steps for processing the data set \mathcal{S}_i^γ and receives the data $\mathcal{S}_{i,1} \subseteq \mathcal{S} \setminus \mathcal{S}_i$, where $|\mathcal{S}_i| = \alpha_i L$ and $|\mathcal{S}_{i,1}| = \alpha_{i,1} L = \min \{ \alpha_i^\gamma L^{\gamma-1} \beta_i, 1 - \alpha_i \} L$. If P_i receives the data $\mathcal{S}_{i,j}$ but has not yet received the entire data set, then P_i computes $\alpha_i^{\gamma-1} \alpha_{i,j} L^\gamma$ processing steps for processing the data set $\mathcal{S}_i^{\gamma-1} \times \mathcal{S}_{i,j}$ and receives the data $\mathcal{S}_{i,j+1} \subseteq \mathcal{S} \setminus \left(\bigcup_{k=0}^j \mathcal{S}_{i,k} \cup \mathcal{S}_i \right)$ with a size

of $\alpha_{i,j+1}L = \min \left\{ \alpha_i^{\gamma-1} \alpha_{i,j} L^{\gamma-1} \beta_i, 1 - \alpha_i - \sum_{k=1}^j \alpha_{i,k} \right\} L$. The iteration is continued until P_i has received the entire data set. According to the above procedure, P_i performs a set of processing steps $\{F_i(2^\gamma - 1), F_i(2^\gamma - 2)\}$. Since P_i has the entire data set, P_i performs a set of processing steps $\{F_i(2^\gamma - 3), F_i(2^\gamma - 4), \dots, F_i(2^{\gamma-1})\}$ for the corresponding γ -dimension data set, where $0 \leq i \leq m$. As determined in the above step, the number of

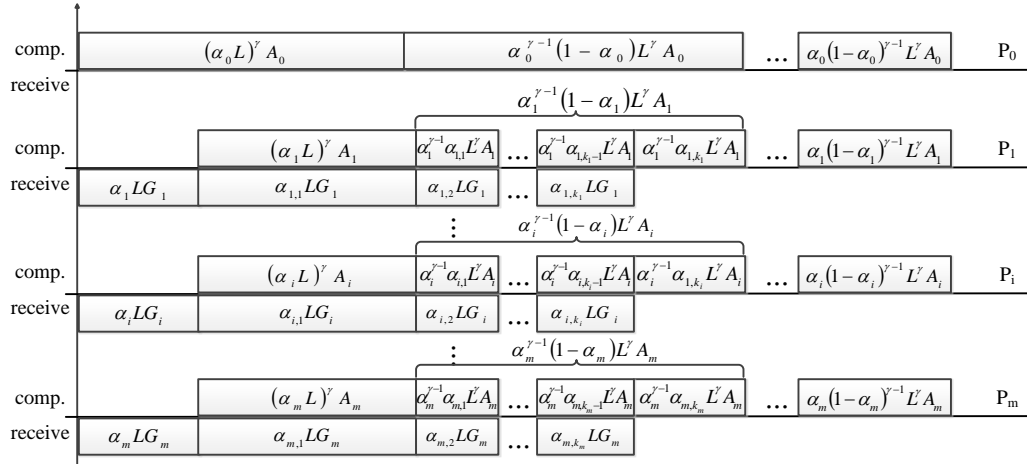


Fig. 4: The load distribution diagram of proposed algorithm for $\gamma \geq 3$.

processing steps of P_i is

$$\begin{aligned} \sum_{j=2^{\gamma-1}}^{2^\gamma-1} F_i(j) &= \alpha_i \left(\alpha_i^{\gamma-1} + C_1^{\gamma-1} \alpha_i^{\gamma-2} (1 - \alpha_i) + \dots + C_{\gamma-2}^{\gamma-1} \alpha_i (1 - \alpha_i)^{\gamma-2} + (1 - \alpha_i)^{\gamma-1} \right) L^\gamma \\ &= \alpha_i (\alpha_i + 1 - \alpha_i)^{\gamma-1} L^\gamma \\ &= \alpha_i L^\gamma. \end{aligned}$$

The total number of processing steps is $\sum_{k=0}^m \sum_{j=2^{\gamma-1}}^{2^\gamma-1} F_k(j) = L^\gamma$. Figure 4 presents the load distribution diagram of the proposed algorithm with $\gamma \geq 3$. For the purpose of minimum processing time, the recursive equations for load distribution are

$$\begin{aligned} \alpha_0 L^\gamma A_0 &= \sum_{j=2^{\gamma-1}}^{2^\gamma-1} F_i(j) A_i + \alpha_i L G_i \\ &= \alpha_i L^\gamma A_i + \alpha_i L G_i \end{aligned}$$

which yields

$$\alpha_i = \frac{\alpha_0 L^\gamma A_0}{L^\gamma A_i + L G_i} \quad (3)$$

for $i = 1, 2, \dots, m$. Substituting α_i from (3) in $\sum_{i=0}^m \alpha_i = 1$ yields

$$\alpha_0 = \frac{1}{1 + \sum_{i=1}^m \frac{L^\gamma A_0}{L^\gamma A_i + L G_i}}.$$

The following theorems explain these results.

Theorem 4. *For a single-level tree network with the root as the initial processor, we have*

$$T_{m+1}^{\mathbb{S}} = \alpha_0 L^\gamma A_0 = \frac{L^\gamma A_0}{1 + \sum_{i=1}^m \frac{L^\gamma A_0}{L^\gamma A_i + LG_i}}.$$

If a homogeneous single-level tree network is considered, then the speedup is given as

$$\text{Speedup}_{m+1}^{\mathbb{S}} = \frac{L^\gamma A}{T_{m+1}^{\mathbb{S}}} = 1 + \frac{mL^{\gamma-1}\beta}{L^{\gamma-1}\beta + 1}$$

where $A_i = A$, $G_i = G$ and $\beta_i = \beta = \frac{A}{G}$ for all i . Its asymptotic speed-up is $\lim_{L \rightarrow \infty} S_{m+1}^{\mathbb{S}} = m + 1$.

Since algorithm \mathbb{C} yield only an approximate solution, algorithm \mathbb{C} cannot compare with algorithm \mathbb{S} in a mathematical method. However, algorithm \mathbb{C} can evenly distribute the load over all processors if $\beta \geq 10$. Therefore, we have the following theorem.

Theorem 5. *For algorithms \mathbb{C} and \mathbb{S} without start-up costs, $\text{Speedup}_{m+1}^{\mathbb{S}} \geq \text{Speedup}_{m+1}^{\mathbb{C}}$ for $L, m \geq 1$ and $\beta \geq 10$ on homogeneous single-level tree networks.*

Proof: Consider the case in which $L, m \geq 1$ and $\beta \geq 10$. The inequality $LHS = 1 + \frac{mL^{\gamma-1}\beta}{L^{\gamma-1}\beta + 1} \geq \frac{(1+m)^\gamma}{(1+m)^{\gamma-m}} = RHS$ must be proved. Since RHS is decreasing in γ and LHS is increasing in L and β , only $1 + \frac{10m}{11} \geq \frac{(1+m)^2}{(1+m)^2-m}$ need to be probed. The inequality $1 + \frac{10m}{11} \geq \frac{(1+m)^2}{(1+m)^2-m}$ is equivalent to $10m^2 + 10m - 1 \geq 0$. Solving $10m^2 + 10m - 1 = 0$ yields roots of 0.0916 and -1.0916 . According to Proposition 1, the inequality $1 + \frac{10m}{11} \geq \frac{(1+m)^2}{(1+m)^2-m}$ holds if $m \geq 0.0916$. ■

V. ALGORITHM WITH MULTIPLE INSTALLMENTS

This section elucidates the γ^{th} -order computational load distribution with reference to the multiple-installments technique. Algorithm \mathbb{M} (Multi-installment) includes the pseudocode of the proposed method. Firstly, algorithm \mathbb{M} calls algorithm \mathbb{PE} with $\rho = 1$ to remove useless processors. Next, algorithm \mathbb{FON} (Finding Optimal Number) searches for the optimal number of installments ρ . Each child processor deals with ρ data sets. The entire load S can be partitioned into $\rho m + 1$ fractions $S_0, S_1, \dots, S_{\rho m}$ where $|S_{km+i}| = \alpha_i L$ for $0 \leq k < \rho$ and $1 \leq i \leq m$. The processor P_0 deals with the data set $S_0 \times S^{\gamma-1}$ and the processor P_i deals with the data sets $S_{km+i} \times S^{\gamma-1}$ for $0 \leq k < \rho$ and $1 \leq i \leq m$. As P_0 is computing $\alpha_0 L^\gamma$ processing steps for the data set $S_0 \times S^{\gamma-1}$, the proposed algorithm performs the following two steps. The first step is formally analogous to the γ^{th} -order computational load distribution in the section IV-B. Hence, each processor P_i performs $\alpha_i L^\gamma$ processing steps for the data set $S_i \times S^{\gamma-1}$ and has received the entire data set. In the second step, each processor P_i immediately computes $\sum_{k=1}^{\rho-1} \sum_{j=2^{\gamma-1}}^{2^\gamma-1} F_{km+i}(j)$ processing steps for the data set $\bigcup_{k=1}^{\rho-1} (S_{km+i} \times S^{\gamma-1})$ where $1 \leq i \leq m$. Based on the above steps, the recursive equations for load distribution are

$$\alpha_0 L^\gamma A_0 = \rho \alpha_i L^\gamma A_i + \alpha_i L G_i, i = 1, 2, \dots, m$$

which yields

$$\alpha_i = \frac{\alpha_0 L^\gamma A_0}{\rho L^\gamma A_i + L G_i} \quad (4)$$

for $i = 1, 2, \dots, m$. Since the set \mathcal{S} is partitioned into a list $\mathcal{S}_0, \mathcal{S}_1, \dots, \mathcal{S}_{\rho m}$ of subsets and $|\mathcal{S}_{km+i}| = \alpha_i L$ for $0 \leq k < \rho$ and $1 \leq i \leq m$, $\alpha_0 + \rho \sum_{i=1}^m \alpha_i = 1$. Therefore, substituting α_i from (4) in $\alpha_0 + \rho \sum_{i=1}^m \alpha_i = 1$ gives

$$\alpha_0 = \frac{1}{1 + \rho \sum_{i=1}^m \frac{L^\gamma A_0}{\rho L^\gamma A_i + L G_i}}. \quad (5)$$

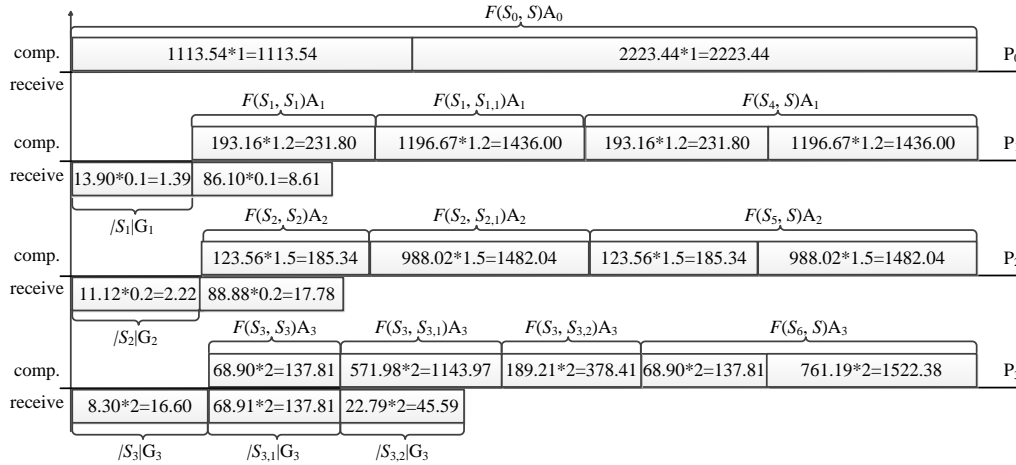


Fig. 5: An example of algorithm \mathbb{M} with parameters $L = 100$, $\gamma = 2$, $\rho = 2$, $A_0 = 1$, $A_1 = 1.2$, $A_2 = 1.5$, $A_3 = 2$, $G_1 = 0.1$, $G_2 = 0.2$, $G_3 = 2$.

Example 2. Assume that $L = 100$, $\gamma = 2$, $\rho = 2$, $A_0 = 1$, $A_1 = 1.2$, $A_2 = 1.5$, $A_3 = 2$, $G_1 = 0.1$, $G_2 = 0.2$, $G_3 = 2$. Figure 5 presents the load distribution diagram of the algorithm \mathbb{M} . According to the equations (5) and (4), the fractions of the load are $\alpha_0 = 0.3337$, $\alpha_1 = 0.1390$, $\alpha_2 = 0.1112$ and $\alpha_3 = 0.0830$. The entire load S can be partitioned into $\mathcal{S}_0, \mathcal{S}_1, \dots, \mathcal{S}_6$ where $|\mathcal{S}_0| = \alpha_0 L = 33.37$, $|\mathcal{S}_1| = |\mathcal{S}_4| = \alpha_1 L = 13.90$, $|\mathcal{S}_2| = |\mathcal{S}_5| = \alpha_2 L = 11.12$ and $|\mathcal{S}_3| = |\mathcal{S}_6| = \alpha_3 L = 8.30$. Since $\alpha_1^{\gamma-1} L^{\gamma-1} A_1 / G_1 = 166.8 > 1 - \alpha_1 = 0.86$, $\alpha_2^{\gamma-1} L^{\gamma-1} A_2 / G_2 = 83.4 > 1 - \alpha_2 = 0.89$ and $\alpha_3^{\gamma-1} L^{\gamma-1} A_3 / G_3 = 8.3 > 1 - \alpha_3 = 0.92$, this load distribution is feasible. The processing is $\alpha_0 L^\gamma A_0 = 0.3337 \times 100^2 \times 1 = 3337$. The processor P_0 computes $F(\mathcal{S}_0, \mathcal{S})$ processing steps for the data set $\mathcal{S}_0 \times \mathcal{S}$ and the processor P_i computes $F(\mathcal{S}_i, \mathcal{S}) + F(\mathcal{S}_{i+3}, \mathcal{S})$ processing steps for the data sets $\mathcal{S}_i \times \mathcal{S}$ and $\mathcal{S}_{i+3} \times \mathcal{S}$ where $1 \leq i \leq 3$.

In the first communication, P_0 transmits the data sets $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$ to P_1, P_2 and P_3 , respectively. While processors P_1, P_2, P_3 respectively computes the data set $\mathcal{S}_1 \times \mathcal{S}_1, \mathcal{S}_2 \times \mathcal{S}_2$ and $\mathcal{S}_3 \times \mathcal{S}_3$, they start the second communication. Since $\alpha_1^\gamma L A_1 / G_1 = 23.18 > 1 - \alpha_1 = 0.86$ and $\alpha_2^\gamma L A_2 / G_2 = 9.27 > 1 - \alpha_2 = 0.89$, P_1 and P_2 can receive the data sets $\mathcal{S}_{1,1} = \mathcal{S} \setminus \mathcal{S}_1$ and $\mathcal{S}_{2,1} = \mathcal{S} \setminus \mathcal{S}_2$, respectively. After P_1 and P_2 have computed the data set $\mathcal{S}_1 \times \mathcal{S}_1$ and $\mathcal{S}_2 \times \mathcal{S}_2$, they perform $F(\mathcal{S}_1, \mathcal{S}_{1,1}) = \alpha_1 \alpha_{1,1} L^2 = \alpha_1 (1 - \alpha_1) L^2$ processing steps and $F(\mathcal{S}_2, \mathcal{S}_{2,1}) = \alpha_2 \alpha_{2,1} L^2 =$

$\alpha_2(1 - \alpha_2)L^2$ processing steps, respectively. However, since $\alpha_3^2 LA_3/G_3 = 0.69 < 0.92 = 1 - \alpha_3$, P_3 only receives a data set $\mathcal{S}_{3,1} \subseteq \mathcal{S} \setminus \mathcal{S}_3$ with a size of $\alpha_{3,1}L = 68.91$. While P_3 computes $F(\mathcal{S}_3, \mathcal{S}_{3,1}) = \alpha_3\alpha_{3,1}L^2$ processing steps for the data set $\mathcal{S}_3 \times \mathcal{S}_{3,1}$, it receives the data set $\mathcal{S}_{3,2} = \mathcal{S} \setminus (\mathcal{S}_{3,1} \cup \mathcal{S}_3)$ with a size of $\alpha_{3,2}L = (1 - \alpha_3 - \alpha_{3,1})L = 22.79$. After P_3 has received the data set $\mathcal{S}_{3,2}$, it performs $F(\mathcal{S}_3, \mathcal{S}_{3,2}) = \alpha_3\alpha_{3,2}L^2$ processing steps for the data set $\mathcal{S}_3 \times \mathcal{S}_{3,2}$. Each child processor has received the entire data set at the first installment, that is, $\mathcal{S}_1 \cup \mathcal{S}_{1,1} = \mathcal{S}_2 \cup \mathcal{S}_{2,1} = \mathcal{S}_3 \cup \mathcal{S}_{3,1} \cup \mathcal{S}_{3,2} = \mathcal{S}$. Then, each child processor immediately computes the second installment after it has finished its first installment.

The following theorems explain the results.

Theorem 6. *For a single-level tree network with the root as the initial processor, we have*

$$T_{m+1,\rho}^{\mathbb{M}} = \alpha_0 L^\gamma A_0 = \frac{L^\gamma A_0}{1 + \rho \sum_{i=1}^m \frac{L^\gamma A_0}{\rho L^\gamma A_i + L G_i}}.$$

A homogeneous single-level tree network is considered, where $A_i = A$, $G_i = G$ and $\beta_i = \beta = \frac{A}{G}$ for all i . The speedup is given as

$$\text{Speedup}_{m+1}^{\mathbb{M}} = \frac{L^\gamma A}{T_{m+1,\rho}^{\mathbb{M}}} = 1 + \frac{\rho m L^{\gamma-1} \beta}{\rho L^{\gamma-1} \beta + 1}.$$

Its asymptotic speed-up is $\lim_{\rho \rightarrow \infty} S_{m+1}^{\mathbb{M}} = m + 1$.

Theorem 7. *For algorithms \mathbb{S} and \mathbb{M} without start-up costs, $\text{Speedup}_{m+1}^{\mathbb{M}} \geq \text{Speedup}_{m+1}^{\mathbb{S}}$ for $\rho \geq 1$ on homogeneous single-level tree networks.*

Proof: The inequality $1 + \frac{\rho m L^{\gamma-1} \beta}{\rho L^{\gamma-1} \beta + 1} \geq 1 + \frac{m L^{\gamma-1} \beta}{L^{\gamma-1} \beta + 1}$ holds for $\rho \geq 1$. The proof is complete. ■

A. Algorithms with Start-Up Costs

This section considers the computation start-up cost θ_{cp} and communication start-up cost θ_{cm} . In the affine cost model, the computation time and the communication time of a load L are $\theta_{cp} + LA$ and $\theta_{cm} + LG$, respectively, where G is the time to transmit a unit of load and A is the time to process a processing step. For analysing the influence of start-up costs, the number of communications needs to be known. The minimum number of communications can be obtained by the following lemma.

Lemma 8. *For a single-level tree network with the root as the initial processor, the minimum number of communications that are required to transfer the entire data set to P_i is*

$$n_i^* = \begin{cases} \frac{\ln\left(\frac{\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i - 1}{\alpha_i} + 1\right)}{\ln(\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i)} & \text{if } \alpha_i^{\gamma-1} L^{\gamma-1} \beta_i \neq 1 \\ 1/\alpha_i & \text{if } \alpha_i^{\gamma-1} L^{\gamma-1} \beta_i = 1 \end{cases}$$

TABLE IV: ALGORITHM FON (Finding Optimal Number)

ALGORITHM FON (Finding Optimal Number)

Output: the optimal number of multi-installment ρ^* .

```

1: if the multicomputer system is heterogeneous. then
2:    $i = \arg \min_i \{ \alpha_i^{\gamma-1} \beta_i \}$ .
3:    $l = 0$ .
4:   repeat
5:      $l = l + 1$ 
6:      $\alpha_0 = \frac{1}{1+(l+1) \sum_{i=1}^m \frac{L^\gamma A_0}{(l+1)L^\gamma A_i + L G_i}}$ 
7:      $\alpha_i = \frac{\alpha_0 L^\gamma A_0}{(l+1)L^\gamma A_i + L G_i}$ 
8:   until  $T_{m+1, \rho=l}^M \leq T_{m+1, \rho=l+1}^M$  or  $\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i + \alpha_i < 1$ .
9: else
10:   $\alpha \equiv \alpha_1$  where  $\alpha_1 = \alpha_2 = \dots = \alpha_m$ .
11:   $\rho_1 = \frac{L^\gamma \gamma - \sqrt{\beta^\gamma - 1}}{(m+1)L^{\gamma-1}\beta}$ .
12:   $\rho_2 = \frac{-1 + \sqrt{\frac{m L^{2\gamma-1} A \beta}{(m+2) \max\{\theta_{cp}, \theta_{cm}\}}}}{(m+1)L^{\gamma-1}\beta}$ .
13:   $\rho_3 = \frac{-1 + \sqrt{\frac{m L^{2\gamma-1} A \beta}{(m+1) \max\{\theta_{cp}, \theta_{cm}\}}}}{(m+1)L^{\gamma-1}\beta}$ .
14:   $LB = \lfloor \min \{ \rho_1, \rho_2 \} \rfloor$ .
15:   $UB = \lceil \max \{ \rho_1, \rho_3 \} \rceil$ .
16:   $l = \max \{ 1, LB \} - 1$ .
17:  repeat
18:     $l = l + 1$ 
19:     $\alpha_0 = \frac{1}{1+(l+1) \sum_{i=1}^m \frac{L^{\gamma-1} \beta}{(l+1)L^{\gamma-1} \beta + 1}}$ 
20:     $\alpha = \frac{\alpha_0 L^{\gamma-1} \beta}{(l+1)L^{\gamma-1} \beta + 1}$ 
21:  until  $T_{m+1, \rho=l}^M \leq T_{m+1, \rho=l+1}^M$  or  $\alpha^{\gamma-1} L^{\gamma-1} \beta + \alpha < 1$  or  $l = UB$ .
22:  $\rho^* = l$ .
```

Proof: The processor P_i receives a fraction of load with size $\alpha_i \left(\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i \right)^j L$ at $(j+1)^{\text{th}}$ communication. Since P_i needs to receive the entire data set, the following constraint must be satisfied

$$L \leq \alpha_i \sum_{j=0}^{n_i-1} \left(\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i \right)^j L.$$

If $\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i \neq 1$,

$$n_i \geq \frac{\ln \left(\frac{\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i - 1}{\alpha_i} + 1 \right)}{\ln \left(\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i \right)}.$$

If $\alpha_i^{\gamma-1} L^{\gamma-1} \beta_i = 1$, $n_i \geq 1/\alpha_i$. The minimum number of communications is the lower bound of the above inequalities. ■

TABLE V: ALGORITHM M (Multi-installment)

ALGORITHM M (Multi-installment)

Output: A load distribution.

- 1: call **Algorithm PE** with $\rho = 1$.
 - 2: call **Algorithm FON**.
 - 3: $\alpha_0 = \frac{1}{1 + \rho \sum_{i=1}^m \frac{L^\gamma A_0}{\rho L^\gamma A_i + L G_i}}$.
 - 4: $\alpha_i = \frac{\alpha_0 L^\gamma A_0}{\rho L^\gamma A_i + L G_i}$, where $1 \leq i \leq m$.
 - 5: $\alpha_{i,1} = \min \left\{ \alpha_i^\gamma L^{\gamma-1} \frac{A_i}{G_i}, 1 - \alpha_i \right\}$, where $1 \leq i \leq m$.
 - 6: **for all** $i = 0 \rightarrow m$ **do in parallel**
 - 7: **if** $i = 0$ **then**
 - 8: **the same as lines 7-13 of Algorithm S**
 - 9: **else**
 - 10: **the same as lines 15-21 of Algorithm S**
 - 11: compute $\sum_{j=2^{\gamma-1}}^{2^\gamma-3} F_i(j) + \sum_{k=1}^{\rho-1} \sum_{j=2^{\gamma-1}}^{2^\gamma-1} F_{km+i}(j)$ processing steps for the corresponding data set.
-

Considering the start-up costs, the parallel processing time of algorithm M can be obtained by the following theorem.

Theorem 9. *For a single-level tree network with the root as the initial processor, we have*

$$T_{m+1,\rho}^{\mathbb{M}} = \frac{L^\gamma A_0}{1 + \rho \sum_{i=1}^m \frac{L^\gamma A_0}{\rho L^\gamma A_i + L G_i}} + \theta_{cp} + \theta_{cm} + \max_{i=1}^m \{ \lceil n_i^* \rceil - 1 \} \max \{ \theta_{cp}, \theta_{cm} \}.$$

If a homogeneous single-level tree network is considered, where $A_i = A$, $G_i = G$ and $\beta_i = \beta = \frac{A}{G}$ for all i . The speedup is given as

$$\text{Speedup}_{m+1}^{\mathbb{M}} = \frac{L^\gamma A + \theta_{cp}}{T_{m+1,\rho}^{\mathbb{M}}}.$$

Proof: Assume that $i = \arg \min_i \{ \alpha_i^{\gamma-1} \beta_i \}$. We have $n_i^* \geq n_j^*$ for $j \in \{1, \dots, m\}$. The parallel processing time comprises a communication start-up cost in the first communication, $(n_i^* - 1) \max \{ \theta_{cp}, \theta_{cm} \}$ start-up costs for the receiving of the rest of the data set, and a computation start-up cost for the remaining processing steps. ■

When the start-up costs are considered, the performance may decline as the number of installments exceeds a certain value. The optimal number of installments needs to be found. However, the optimal number of installments is difficult to determine. Therefore, this work finds a range for searching an optimal number of installments. We use $U_{m+1,\rho}$ and $L_{m+1,\rho}$ in the analysis where

$$U_{m+1,\rho} = \frac{L^\gamma A_0}{1 + \rho \sum_{i=1}^m \frac{L^\gamma A_0}{\rho L^\gamma A_i + L G_i}} + \theta_{cp} + \theta_{cm} + \max_{i=1}^m \{ n_i^* \} \max \{ \theta_{cp}, \theta_{cm} \},$$

$$L_{m+1,\rho} = \frac{L^\gamma A_0}{1 + \rho \sum_{i=1}^m \frac{L^\gamma A_0}{\rho L^\gamma A_i + L G_i}} + \theta_{cp} + \theta_{cm} + \max_{i=1}^m \{ n_i^* - 1 \} \max \{ \theta_{cp}, \theta_{cm} \}.$$

In fact, $L_{m+1,\rho} \leq T_{m+1,\rho}^{\mathbb{M}} \leq U_{m+1,\rho}$. We have $U_{m+1,\rho} - L_{m+1,\rho} = \max\{\theta_{cp}, \theta_{cm}\}$ and $\frac{d}{d\rho} U_{m+1,\rho} = \frac{d}{d\rho} L_{m+1,\rho}$. Let

$$\rho_1 = \frac{L^{\gamma-1} \sqrt[\gamma]{\beta} - 1}{(m+1) L^{\gamma-1} \beta},$$

$$\rho_2 = \frac{-1 + \sqrt{\frac{m L^{2\gamma-1} A \beta}{(m+2) \max\{\theta_{cp}, \theta_{cm}\}}}}{(m+1) L^{\gamma-1} \beta}$$

and

$$\rho_3 = \frac{-1 + \sqrt{\frac{m L^{2\gamma-1} A \beta}{(m+1) \max\{\theta_{cp}, \theta_{cm}\}}}}{(m+1) L^{\gamma-1} \beta}.$$

The bound on the optimal number of installments is as follows.

Lemma 10. *For a homogeneous single-level tree network with the root as the initial processor, the optimal number of installments can be found in the following range*

$$\min\{\rho_1, \rho_2\} \leq \rho^* \leq \max\{\rho_1, \rho_3\},$$

where $A_i = A$, $G_i = G$, $\beta_i = \beta = \frac{A}{G}$, $\alpha_i = \alpha$ and $n_i^* = n^*$ for all $1 \leq i \leq m$.

Proof: Solving $\alpha^{\gamma-1} L^{\gamma-1} \beta = 1$ for ρ yields the root ρ_1 . According to equation (4), the fraction α_i is decreasing in ρ for $1 \leq i \leq m$. So, $\alpha^{\gamma-1} L^{\gamma-1} \beta$ is decreasing in ρ . In other words, $\alpha^{\gamma-1} L^{\gamma-1} \beta \geq 1$ at $\rho \leq \rho_1$ and $\alpha^{\gamma-1} L^{\gamma-1} \beta < 1$ at $\rho > \rho_1$.

By the Lemma 8, $n^* = 1/\alpha$ at $\rho = \rho_1$. Since $\alpha^{\gamma-1} L^{\gamma-1} \beta$ is decreasing in ρ , n^* is increasing in ρ . Because $n^* = 1/\alpha$ in $\rho = \rho_1$ and n^* is increasing in ρ , $n^* \leq 1/\alpha$ in $\rho \leq \rho_1$ and $n^* > 1/\alpha$ in $\rho > \rho_1$. We consider an upper bound of $U_{m+1,\rho}$ in the case of $\alpha^{\gamma-1} L^{\gamma-1} \beta \geq 1$. Since $n^* \leq 1/\alpha$ at $\rho \leq \rho_1$, the following inequality applies

$$\begin{aligned} U_{m+1,\rho} &\leq \frac{L^\gamma A}{1 + \rho \sum_{i=1}^m \frac{L^{\gamma-1} \beta}{\rho L^{\gamma-1} \beta + 1}} + \theta_{cp} + \theta_{cm} + \frac{1}{\alpha} \max\{\theta_{cp}, \theta_{cm}\} \\ &= RHS_1 \end{aligned} \quad (6)$$

at $\rho \leq \rho_1$. Moreover, we consider a lower bound $L_{m+1,\rho}$ in the case of $\alpha^{\gamma-1} L^{\gamma-1} \beta < 1$. Since $n^* > 1/\alpha$ at $\rho > \rho_1$, the following inequality holds

$$\begin{aligned} L_{m+1,\rho} &> \frac{L^\gamma A}{1 + \rho \sum_{i=1}^m \frac{L^{\gamma-1} \beta}{\rho L^{\gamma-1} \beta + 1}} + \theta_{cp} + \theta_{cm} + \left(\frac{1}{\alpha} - 1\right) \max\{\theta_{cp}, \theta_{cm}\} \\ &= RHS_2 \end{aligned} \quad (7)$$

at $\rho > \rho_1$.

By carefully inspecting the formulas $T_{m+1,\rho}^{\mathbb{M}}$, $U_{m+1,\rho}$, $L_{m+1,\rho}$, RHS_1 and RHS_2 , they have a same term

$$\frac{L^\gamma A}{1 + \rho \sum_{i=1}^m \frac{L^{\gamma-1} \beta}{\rho L^{\gamma-1} \beta + 1}} + \theta_{cp} + \theta_{cm}.$$

We call this term by the decreasing term. The decreasing term is decreasing in ρ . The remaining terms in $T_{m+1,\rho}^{\mathbb{M}}, U_{m+1,\rho}, L_{m+1,\rho}, RHS_1$ and RHS_2 are increasing in ρ . Therefore, the bound of optimal number of installments can be obtained by analyzing RHS_1 and RHS_2 .

Since $U_{m+1,\rho} - L_{m+1,\rho} = \max\{\theta_{cp}, \theta_{cm}\}$, the point with slope of $-\max\{\theta_{cp}, \theta_{cm}\}$ in RHS_1 needs to be found. Solving $\frac{d}{d\rho} RHS_1 = -\max\{\theta_{cp}, \theta_{cm}\}$ yields a single feasible root ρ_2 (the other is negative.) Solving $\frac{d}{d\rho} RHS_1 = 0$ yields the root ρ_3 . Solving $\frac{d}{d\rho} RHS_2 = 0$ also yields the root ρ_3 . Since $\frac{d}{d\rho} RHS_1$ is non-decreasing in ρ , $\rho_2 \leq \rho_3$. Therefore, three cases must be considered.

1. $\rho_2 \leq \rho_3 \leq \rho_1$.
2. $\rho_2 \leq \rho_1 \leq \rho_3$.
3. $\rho_1 \leq \rho_2 \leq \rho_3$.

In case 1, the slope of RHS_1 is at most $-\max\{\theta_{cp}, \theta_{cm}\}$ in $\rho \leq \rho_2$. Since $n^* \leq 1/\alpha$ in $\rho \leq \rho_1$ and the decreasing terms in $U_{m+1,\rho}, L_{m+1,\rho}$ and RHS_1 are the same, the slopes of $U_{m+1,\rho}$ and $L_{m+1,\rho}$ are less than the those of RHS_1 in $\rho \leq \rho_1$. The following inequality can be obtained $U_{m+1,\rho+1} \leq U_{m+1,\rho} - \max\{\theta_{cp}, \theta_{cm}\} = L_{m+1,\rho}$ in $\rho < \rho_2$. Therefore, $T_{m+1,\rho+1}^{\mathbb{M}} \leq U_{m+1,\rho+1} \leq L_{m+1,\rho} \leq T_{m+1,\rho}^{\mathbb{M}}$ in $\rho < \rho_2$ is obtained. Since the parallel processing time $T_{m+1,\rho}^{\mathbb{M}}$ is non-increasing in $\rho \leq \rho_2$, so the optimal number of installments ρ^* is greater than or equal to ρ_2 . Since RHS_2 is increasing in $\rho > \rho_1$ ($\because \rho_3 \leq \rho_1$) and $1/\alpha < \lceil n_i^* \rceil$ in $\rho > \rho_1$, $T_{m+1,\rho}^{\mathbb{M}}$ is also increasing in $\rho > \rho_1$ and the global minimum point of $T_{m+1,\rho}^{\mathbb{M}}$ is less than or equal to ρ_1 . Therefore, the optimal number of installments ρ^* is less than or equal to ρ_1 .

In case 2, as in case 1, the optimal number of installments ρ^* is greater than or equal to ρ_2 . Since RHS_2 is increasing in $\rho > \rho_3$ and $1/\alpha \leq \lceil n_i^* \rceil$ in $\rho > \rho_3 \geq \rho_1$, the global minimum point of $T_{m+1,\rho}^{\mathbb{M}}$ is less than or equal to ρ_3 . Thus, the optimal number of installments ρ^* is less than or equal to ρ_3 .

In case 3, the slope of RHS_1 is at most $-\max\{\theta_{cp}, \theta_{cm}\}$ in $\rho \leq \rho_1$ ($\because \rho_1 \leq \rho_2$). As in case 1, $T_{m+1,\rho}^{\mathbb{M}}$ is non-increasing in $\rho \leq \rho_1$. Thus, the optimal number of installments ρ^* is greater than or equal to ρ_1 . Since RHS_2 is increasing in $\rho > \rho_3$ and $1/\alpha \leq \lceil n_i^* \rceil$ in $\rho > \rho_3 \geq \rho_1$, the global minimum point of $T_{m+1,\rho}^{\mathbb{M}}$ is less than or equal to ρ_3 . Therefore, the optimal number of installments ρ^* is less than or equal to ρ_3 . ■

The algorithm FON is based on Lemma 2 and Lemma 10 to search an optimal number of installments. Another way can be used to avoid the search policy with little loss in performance. That is, the number of installments ρ can be directly set as $\max\{1, \lfloor \min\{\rho_1, \rho_2\} \rfloor\}$ or $\max\{1, \lceil \min\{\rho_1, \rho_2\} \rceil\}$. The following theorem shows that algorithm \mathbb{M} is not worse than algorithm \mathbb{S} if $\rho \in [1, \max\{1, \lfloor \min\{\rho_1, \rho_2\} \rfloor\}]$.

Theorem 11. For algorithms \mathbb{S} and \mathbb{M} with start-up costs, $T_{m+1,\rho}^{\mathbb{M}} \leq T_{m+1}^{\mathbb{S}}$ or $Speedup_{m+1}^{\mathbb{M}} \geq Speedup_{m+1}^{\mathbb{S}}$ for $\rho \in [1, \max\{1, \lfloor \min\{\rho_1, \rho_2\} \rfloor\}]$.

Proof: From Lemma 10, $T_{m+1,\rho}^{\mathbb{M}}$ is non-increasing in $\rho \in [1, \max\{1, \lfloor \min\{\rho_1, \rho_2\} \rfloor\}]$. Since $T_{m+1}^{\mathbb{S}} = T_{m+1,\rho=1}^{\mathbb{M}}$, the proof is complete. ■

Since n^* is increasing in the number of child processors, the performance may decline as the number of child processors exceeds a certain value. Therefore, this work finds a range for searching an optimal number of child

processors. Let

$$m_1 = \frac{\gamma^{-1}\sqrt{\beta}L}{\rho} - \frac{1}{\rho\beta L^{\gamma-1}} - 1,$$

$$m_2 = \frac{-1 + \sqrt{\frac{L^{2\gamma-1}A\rho\beta(L^{\gamma-1}\rho\beta+1)}{(\rho+1)\max\{\theta_{cp},\theta_{cm}\}}}}{L^{\gamma-1}\rho\beta} - 1$$

and

$$m_3 = \frac{-1 + \sqrt{\frac{L^{2\gamma-1}A\rho\beta(L^{\gamma-1}\rho\beta+1)}{\rho\max\{\theta_{cp},\theta_{cm}\}}}}{L^{\gamma-1}\rho\beta} - 1.$$

The bound on the optimal number of child processors is as follows.

Lemma 12. *For a homogeneous single-level tree network with the root as the initial processor, the optimal number of child processors can be found in the following range*

$$\min\{m_1, m_2\} \leq m^* \leq \max\{m_1, m_3\},$$

where $A_i = A$, $G_i = G$, $\beta_i = \beta = \frac{A}{G}$, $\alpha_i = \alpha$ and $n_i^* = n^*$ for all $1 \leq i \leq m$.

Proof: The proof is formally analogous to the proof of Lemma 10. ■

To avoid the search policy, the number of child processors m can be directly set as $\max\{1, \lfloor \min\{m_1, m_2\} \rfloor\}$ or $\max\{1, \lceil \min\{m_1, m_2\} \rceil\}$. According to Lemma 12, the following theorem shows that the performance of algorithm \mathbb{M} is non-decreasing in $m \in [1, \max\{1, \lfloor \min\{m_1, m_2\} \rfloor\}]$.

Theorem 13. *For algorithm \mathbb{M} with start-up costs, $T_{m+1,\rho}^{\mathbb{M}} \leq T_m^{\mathbb{M}}$ or $\text{Speedup}_{m+1}^{\mathbb{M}} \geq \text{Speedup}_m^{\mathbb{M}}$ for $m \in [1, \max\{1, \lfloor \min\{m_1, m_2\} \rfloor\}]$.*

VI. DISCUSSION OF RESULTS

This section compares the performance of the classical method with that of the proposed methods on a single level tree network. Figure 6 plots the speed-ups of algorithms \mathbb{C} and \mathbb{S} without start-up costs. The speed-ups of algorithm \mathbb{C} is decreasing in m and reveals the post-processing steps are increasing in m . Since $L^{\gamma-1}\beta$ dominate the other terms in the speed-up equation of algorithm \mathbb{S} , the algorithm \mathbb{S} approaches its maximal speed-up of $m+1$ when $L^{\gamma-1}\beta$ is very large. For example, the speed-up of algorithm \mathbb{S} reaches $1 + \frac{mL^{\gamma-1}\beta}{L^{\gamma-1}\beta+1} = 10.99$ at $m = 10$, which is very close to the asymptotic speed-up $m+1$.

Next, the numerical results concerning the speed-ups of algorithms \mathbb{S} and \mathbb{M} are presented. The parallel processing time of algorithm \mathbb{M} is composed of two terms $\frac{L^\gamma A_0}{1+\rho \sum_{i=1}^m \frac{L^\gamma A_0}{\rho L^\gamma A_i + L G_i}}$ and $\theta_{cp} + \theta_{cm} + \max_{i=1}^m \{ \lceil n_i^* \rceil - 1 \} \max\{\theta_{cp}, \theta_{cm}\}$. The first term is decreasing in m and the second term is increasing in m . We call the start-up costs are small if the increment of the second term is less than the decrease of first term, otherwise the start-up cost are large. Figure 7 shows the results for small start-up costs. Both algorithms are very close to the optimal solution $m+1$. The first terms of parallel processing time are 6291.4, 4203.5, 3156.1, 2526.6, 2106.4, 1806.1 and 1580.7 that are decreasing

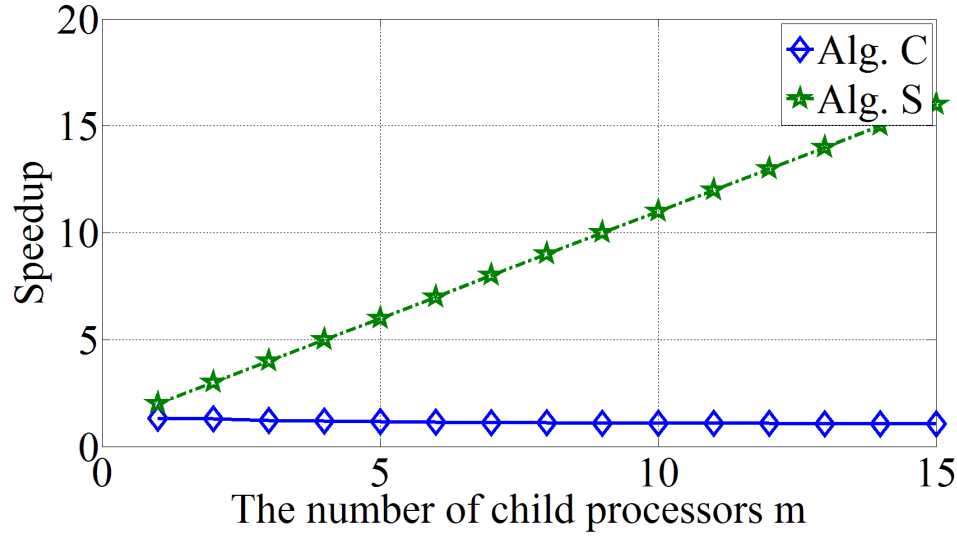


Fig. 6: Speed-up of algorithms \mathbb{C} and \mathbb{S} with $A = 10$, $G = 1$, $L = 500$ and $\gamma = 2$.

in m . The second terms of parallel processing time are 0.5, 0.5, 0.6, 0.7, 0.9, 1.2 and 2. The increment of the second term is less than the decrease of first term. Algorithm \mathbb{M} consistently outperforms algorithm \mathbb{S} when start-up costs are very small. For example, when the number of child processors is $m = 7$, the speed-up of algorithm \mathbb{M} is 7.8980, which is very close to the asymptotic value, whereas the speed-up of algorithm \mathbb{S} is 7.7284. Since both algorithms are very close to $m + 1$, the outperformance of algorithm \mathbb{M} is not considerable. Figure 8 shows the results for large start-up costs. For $m \geq 14$, the first terms of parallel processing time are 1677 and 1572 and the second terms of parallel processing time are 400 and 600. The increment of the second term is greater than the decrease of first term which results in a bad performance. The algorithm \mathbb{M} performs the worst when m is large. For example, the algorithm \mathbb{M} outperforms the algorithm \mathbb{S} when m is small ($m \leq 6$), whereas the algorithm \mathbb{S} performs best than the algorithm \mathbb{M} when m is large ($m > 6$). The start-up costs affected the algorithm \mathbb{M} more than the algorithm \mathbb{S} .

Figure 9 shows the results for algorithm \mathbb{M} with differently sized loads. As L increases, the speed-up of algorithm \mathbb{M} approaches $m + 1$ and is unaffected by start-up costs. According to the observation of Fig. 7 and Fig. 9, the algorithm \mathbb{M} consistently outperforms the algorithm \mathbb{S} when the load to be processed is very large or when the start-up costs are small. Moreover, the optimal number of child processors is in agreement with Lemma 12. For example, when $L = 1000$, the speed-up of algorithm \mathbb{M} is non-decreasing in $m \leq 28$, consistent with Lemma 12, which states that the optimal number of child processors can be found in a range $\min \{m_1 = 32.33, m_2 = 24.00\} \leq m^* \leq \max \{m_1 = 32.33, m_3 = 27.86\}$. The cases of $L = 500$ and $L = 750$ are also consistent with Lemma 12, which states that the optimal number of child processors can be found in ranges $\min \{m_1 = 15.66, m_2 = 11.49\} \leq m^* \leq \max \{m_1 = 15.66, m_3 = 13.43\}$ and $\min \{m_1 = 24.00, m_2 = 17.75\} \leq m^* \leq \max \{m_1 = 24.00, m_3 = 20.65\}$, respectively.

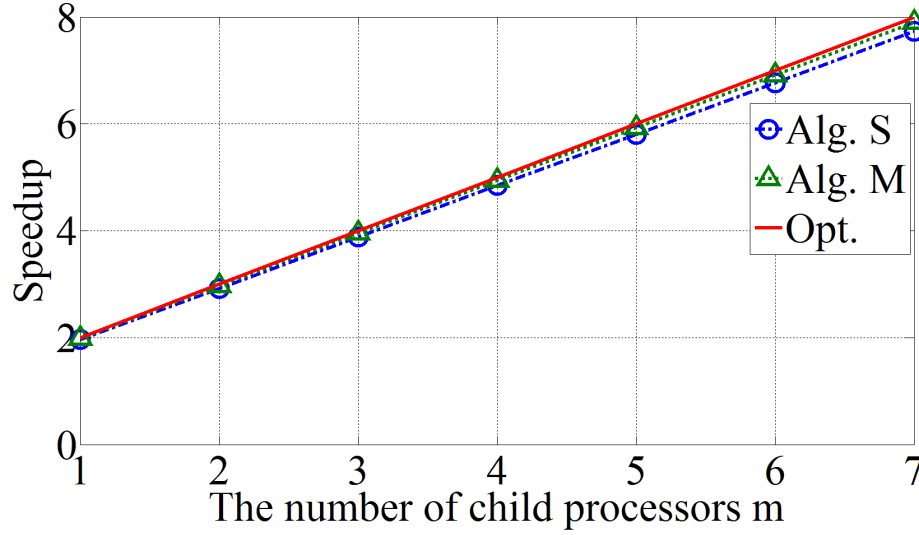


Fig. 7: Speed-up of algorithms \mathbb{S} and \mathbb{M} with $A = 0.05$, $G = 1$, $L = 500$, $\gamma = 2$, $\theta_{cp} = 0.1$, $\theta_{cm} = 0.1$ and $\rho = 3$.

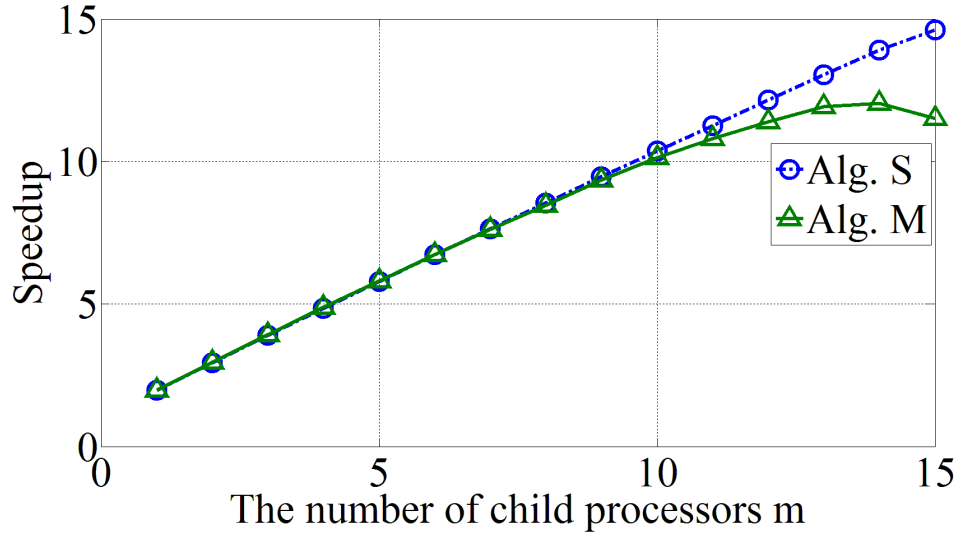


Fig. 8: Speed-up of algorithms \mathbb{S} and \mathbb{M} with $A = 0.1$, $G = 1$, $L = 500$, $\gamma = 2$, $\theta_{cp} = 20$, $\theta_{cm} = 20$ and $\rho = 3$.

Figure 10 presents the numerical results concerning the speed-up of algorithm \mathbb{M} with differently sized loads. Also, the speed-up of algorithm \mathbb{M} approaches $m+1$ and is unaffected by start-up costs if L is very large. We call L is very large if the increment of the second term of parallel processing time $\theta_{cp} + \theta_{cm} + \max_{i=1}^m \{ \lceil n_i^* \rceil - 1 \} \max \{ \theta_{cp}, \theta_{cm} \}$ is less than the decrease of first term of parallel processing time $\frac{L^\gamma A_0}{1 + \rho \sum_{i=1}^m \frac{L^\gamma A_0}{\rho L^\gamma A_i + L G_i}}$. The optimal number of installments is in agreement with Lemma 10. For example, when $L = 1000$, the speed-up of algorithm \mathbb{M} is non-decreasing in $\rho \leq 36$, consistent with Lemma 10, which states that the optimal number of installments can be found in a range $\min \{ \rho_1 = 62.50, \rho_2 = 5.87 \} \leq \rho^* \leq \max \{ \rho_1 = 62.50, \rho_3 = 6.05 \}$. The cases of $L = 500$ and

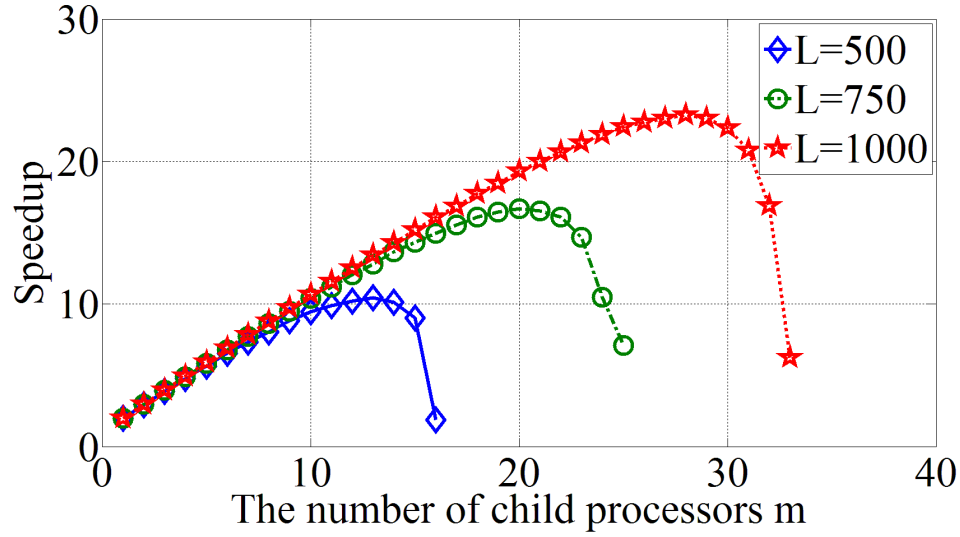


Fig. 9: Speed-up of algorithm \mathbb{M} with $A = 0.1$, $G = 1$, $\gamma = 2$, $\theta_{cp} = 40$, $\theta_{cm} = 40$ and $\rho = 3$.

$L = 750$ are also consistent with Lemma 10, which states that the optimal number of installments can be found in ranges $\min\{\rho_1 = 31.25, \rho_2 = 4.15\} \leq \rho^* \leq \max\{\rho_1 = 31.25, \rho_3 = 4.28\}$ and $\min\{\rho_1 = 46.87, \rho_2 = 5.08\} \leq \rho^* \leq \max\{\rho_1 = 46.87, \rho_3 = 5.24\}$, respectively.

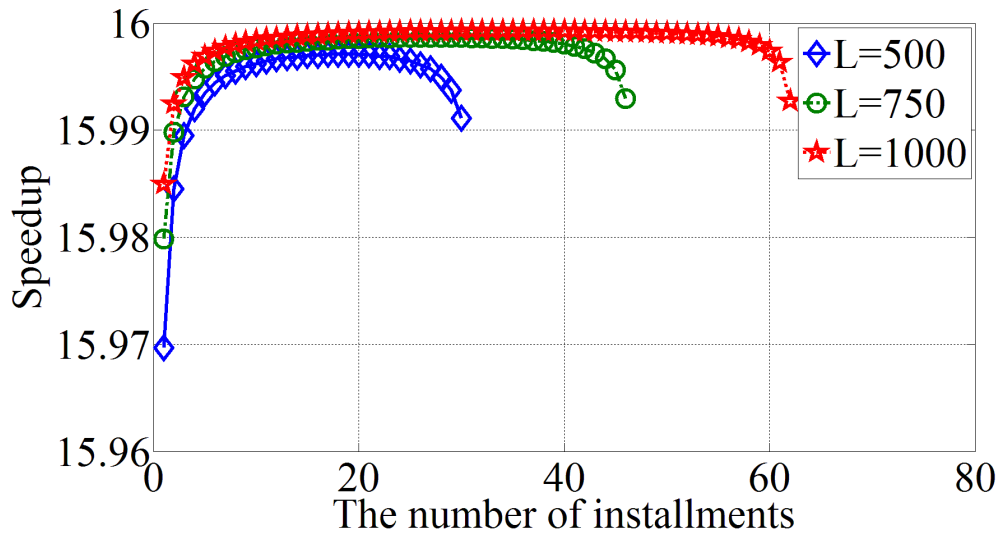


Fig. 10: Speed-up of algorithm \mathbb{M} with $A = 1$, $G = 1$, $\gamma = 2$, $\theta_{cp} = 0.1$, $\theta_{cm} = 0.1$ and $m = 15$.

VII. CONCLUSION

This work proposed two algorithms \mathbb{S} and \mathbb{M} to distribute a divisible non-linear load in a single-level tree network. The algorithm \mathbb{S} employs the single-installment processing approach. The algorithm \mathbb{M} applied the multi-installment processing approach to yield an improved non-linear load distribution on a single-level tree network. The closed-form solutions for parallel execution times and speed-ups, obtained by the proposed algorithms, are also derived. This work reveals that the performance of algorithm \mathbb{S} improves the classical algorithm. This work finds two ranges to search an optimal number of installments and an optimal number of child processors when the computation and communication start-up costs are considered.

ACKNOWLEDGMENT

The work was supported by Ministry of Science and Technology, R.O.C. under Projects MOST 102-2221-E-006-128 and MOST 106-2221-E-006-006.

REFERENCES

- [1] M. Adler, Y. Gong, and A. L. Rosenberg, "Optimal sharing of bags of tasks in heterogeneous clusters," in *SPAA*, vol. 3, 2003, pp. 1–10.
- [2] Y. Bai and R. C. Ward, "Parallel block tridiagonalization of real symmetric matrices," *Journal of Parallel and Distributed Computing*, vol. 68, no. 5, pp. 703–715, 2008.
- [3] S. Bataineh, T.-Y. Hsiung, and T. G. Robertazzi, "Closed form solutions for bus and tree networks of processors load sharing a divisible job," *IEEE Transactions on Computers*, vol. 43, no. 10, pp. 1184–1196, 1994.
- [4] S. M. Bataineh, "Toward an analytical solution to task allocation, processor assignment, and performance evaluation of network processors," *Journal of Parallel and Distributed Computing*, vol. 65, no. 1, pp. 29–47, 2005.
- [5] O. Beaumont, L. Marchal, V. Rehn, and Y. Robert, "Fifo scheduling of divisible loads with return messages under the one-port model," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006, pp. 14–pp.
- [6] V. Bharadwaj, D. Ghose, and V. Mani, "Multi-installment load distribution in tree networks with delays," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 31, no. 2, pp. 555–567, april 1995.
- [7] V. Bharadwaj and W. H. Min, "Scheduling divisible loads on heterogeneous linear daisy chain networks with arbitrary processor release times," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 3, pp. 273 – 288, march 2004.
- [8] V. Bharadwaj, T. G. Robertazzi, and D. Ghose, *Scheduling Divisible Loads in Parallel and Distributed Systems*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1996.
- [9] J. Błazewicz and M. Drozdowski, "Scheduling divisible jobs on hypercubes," *Parallel computing*, vol. 21, no. 12, pp. 1945–1956, 1995.
- [10] J. Błazewicz and M. Drozdowski, "The performance limits of a two-dimensional network of load-sharing processors," *Foundations of Computing and Decision Sciences*, vol. 21, no. 1, 1996.
- [11] J. Błazewicz, M. Drozdowski, F. Guinand, and D. Trystram, "Scheduling a divisible task in a two-dimensional toroidal mesh," *Discrete Applied Mathematics*, vol. 94, no. 13, pp. 35 – 50, 1999.
- [12] J. Błazewicz, M. Drozdowski, and M. Markiewicz, "Divisible task scheduling concept and verification," *Parallel Computing*, vol. 25, no. 1, pp. 87 – 98, 1999.
- [13] B. D. Carlson, E. D. Evans, and S. L. Wilson, "Search radar detection and track with the hough transform. iii. detection performance with binary integration," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 30, no. 1, pp. 116–125, Jan 1994.
- [14] Y.-K. Chang, J.-H. Wu, C.-Y. Chen, and C.-P. Chu, "Improved methods for divisible load distribution on k-dimensional meshes using multi-installment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 11, pp. 1618–1629, nov. 2007.
- [15] S. Charcranoon, T. Robertazzi, and S. Luryi, "Parallel processor configuration design with processing/transmission costs," *IEEE Transactions on Computers*, vol. 49, no. 9, pp. 987–991, sep 2000.

- [16] C.-Y. Chen and C.-P. Chu, "Improved methods for divisible load distribution on d-dimensional hypercube using multi-installment," *Journal of the Chinese Institute of Engineers*, vol. 31, no. 7, pp. 1199–1206, 2008.
- [17] C.-Y. Chen and C.-P. Chu, "A novel computational model for non-linear divisible loads on a linear network," *Computers, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.
- [18] C.-Y. Chen and C.-P. Chu, "Novel methods for divisible load distribution with start-up costs on a complete b-ary tree," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 26, no. 10, pp. 2836–2848, Oct 2015.
- [19] Y. Cheng and T. Robertazzi, "Distributed computation with communication delay [distributed intelligent sensor networks]," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 24, no. 6, pp. 700–712, nov 1988.
- [20] M. Drozdowski and P. Wolniewicz, "Out-of-core divisible load processing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 10, pp. 1048–1056, oct. 2003.
- [21] M. Drozdowski, "Selected problems of scheduling tasks in multiprocessor computer systems," 1997.
- [22] M. Drozdowski and W. Glazek, "Scheduling divisible loads in a three-dimensional mesh of processors," *Parallel Computing*, vol. 25, no. 4, pp. 381–404, 1999.
- [23] M. Drozdowski, M. Lawenda, and F. Guinand, "Scheduling multiple divisible loads," *International Journal of High Performance Computing Applications*, vol. 20, no. 1, pp. 19–30, 2006.
- [24] M. Drozdowski and P. Wolniewicz, "Performance limits of divisible load processing in systems with limited communication buffers," *Journal of Parallel and Distributed Computing*, vol. 64, no. 8, pp. 960–973, 2004.
- [25] M. Drozdowski and P. Wolniewicz, "Experiments with scheduling divisible tasks in clusters of workstations," in *Euro-Par 2000 Parallel Processing*. Springer, 2000, pp. 311–319.
- [26] M. Drozdowski and P. Wolniewicz, "Optimum divisible load scheduling on heterogeneous stars with limited memory," *European Journal of Operational Research*, vol. 172, no. 2, pp. 545–559, 2006.
- [27] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Commun. ACM*, vol. 15, no. 1, pp. 11–15, Jan. 1972.
- [28] P. Dutot, "Divisible load on heterogeneous linear array," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS03), Nice, France, 2003*.
- [29] M. Field, D. Stirling, Z. Pan, and F. Naghdy, "Learning trajectories for robot programming by demonstration using a coordinated mixture of factor analyzers," *IEEE Transactions on Cybernetics*, vol. 46, no. 3, pp. 706–717, March 2016.
- [30] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier, 2003.
- [31] D. Ghose and H. J. Kim, "Computing blas level-2 operations on workstation clusters using the divisible load paradigm," *Mathematical and Computer Modelling*, vol. 41, no. 1, pp. 49–70, 2005.
- [32] W. Glazek, "Distributed computation in a three-dimensional mesh with communication delays," in *Parallel and Distributed Processing, 1998. PDP '98. Proceedings of the Sixth Euromicro Workshop on*, jan 1998, pp. 38–42.
- [33] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface*, 1999, vol. 1.
- [34] N. Guil, J. Villalba, and E. L. Zapata, "A fast hough transform for segment detection," *IEEE Transactions on Image Processing*, vol. 4, no. 11, pp. 1541–1548, 1995.
- [35] J. Guo, J. Yao, and L. Bhuyan, "An efficient packet scheduling algorithm in network processors," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 2, march 2005, pp. 807–818 vol. 2.
- [36] J. T. Hung, H. J. Kim, and T. G. Robertazzi, "Scalable scheduling in parallel processors," in *Proceedings of the 2002 Conference on Information Sciences and Systems*, 2002.
- [37] J. T. Hung and T. Robertazzi, "Scheduling nonlinear computational loads," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 44, no. 3, pp. 1169–1182, july 2008.
- [38] J.-T. Hung and T. Robertazzi, "Distributed Scheduling of Nonlinear Computational Loads," in *Conference on Information Sciences and Systems*, 2003.
- [39] M. R. Inggs and A. D. Robinson, "Ship target recognition using low resolution radar and neural networks," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 35, no. 2, pp. 386–393, Apr 1999.
- [40] K. B. Khalifa, M. Boubaker, N. Chelbi, and M. H. Bedoui, "Learning vector quantization neural network implementation using parallel and serial arithmetic," *International Journal of Computer Sciences and Engineering Systems*, vol. 2, no. 4, pp. 251–256, 2008.

- [41] H.-J. Kim, "A novel optimal load distribution algorithm for divisible loads," *Cluster Computing*, vol. 6, no. 1, pp. 41–46, 2003.
- [42] K. Ko, *Scheduling Data Intensive Parallel Processing in Distributed and Networked Environments*. Stony Brook, NY, USA: State University of New York at Stony Brook, 2000.
- [43] K. Ko and T. G. Robertazzi, "Signature search time evaluation in flat file databases," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 44, no. 2, pp. 493–502, 2008.
- [44] Y. Kyong and T. G. Robertazzi, "Greedy signature processing with arbitrary location distributions: A divisible load framework," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 4, pp. 3027–3041, October 2012.
- [45] K. Li, "Managing divisible load on partitionable networks," 1998.
- [46] K. Li, "Improved methods for divisible load distribution on k-dimensional meshes using pipelined communications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 12, pp. 1250 – 1261, dec. 2003.
- [47] K. Li, "Parallel processing of divisible loads on partitionable static interconnection networks," *Cluster Computing*, vol. 6, no. 1, pp. 47–55, 2003.
- [48] K. Li, "Speed-up of parallel processing of divisible loads on k-dimensional meshes and tori," *The Computer Journal*, vol. 46, no. 6, pp. 625–631, 2003.
- [49] K. Li, "New divisible load distribution methods using pipelined communication techniques on tree and pyramid networks," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 47, no. 2, pp. 806 –819, april 2011.
- [50] P. Li, B. Veeravalli, and A. A. Kassim, "Design and implementation of parallel video encoding strategies using divisible load analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 9, pp. 1098–1112, 2005.
- [51] X. Li, B. Veeravalli, and C. C. Ko, "Divisible load scheduling on a hypercube cluster with finite-size buffers and granularity constraints," in *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, 2001, pp. 660 –667.
- [52] X. Li, B. Veeravalli, and C. C. Ko, "Distributed image processing on a network of workstations," *Intl J. Computers and Applications*, vol. 25, no. 2, pp. 1–10, 2003.
- [53] W. H. Min and B. Veeravalli, "Aligning biological sequences on distributed bus networks: a divisible load scheduling approach," *IEEE Transactions on Information Technology in Biomedicine*, vol. 9, no. 4, pp. 489–501, 2005.
- [54] H. Othman and T. Aboulnasr, "A separable low complexity 2d hmm with application to face recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 10, pp. 1229–1238, 2003.
- [55] D. A. L. Piriya Kumar and C. S. R. Murthy, "Distributed computation for a hypercube network of sensor-driven processors with communication delays including setup time," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 28, no. 2, pp. 245–251, 1998.
- [56] S. Suresh, H. Huang, and H. J. Kim, "Scheduling in compute cloud with multiple data banks using divisible load paradigm," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 51, no. 2, pp. 1288–1297, April 2015.
- [57] S. Suresh, H. Kim, C. Run, and T. Robertazzi, "Scheduling nonlinear divisible loads in a single level tree network," *The Journal of Supercomputing*, vol. 61, pp. 1068–1088, 2012.
- [58] S. Suresh, V. Mani, and S. Omkar, "The effect of start-up delays in scheduling divisible loads on bus networks: An alternate approach," *Computers & Mathematics with Applications*, vol. 46, no. 10, pp. 1545 – 1557, 2003.
- [59] S. Suresh, S. Omkar, and V. Mani, "Parallel implementation of back-propagation algorithm in networks of workstations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 1, pp. 24–34, 2005.
- [60] S. Suresh, C. Run, H. J. Kim, T. Robertazzi, and Y.-I. Kim, "Scheduling second-order computational load in master-slave paradigm," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 1, pp. 780 –793, jan. 2012.
- [61] S. Suresh, V. Mani, S. Omkar, and H. Kim, "Divisible load scheduling in distributed system with buffer constraints: Genetic algorithm and linear programming approach," *The International Journal of Parallel, Emergent and Distributed Systems*, vol. 21, no. 5, pp. 303–321, 2006.
- [62] A. Vakanski, I. Mantegh, A. Irish, and F. Janabi-Sharifi, "Trajectory learning for robot programming by demonstration using hidden markov model and dynamic time warping," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 4, pp. 1039–1052, Aug 2012.
- [63] B. Veeravalli, "Design and performance analysis of heuristic load-balancing strategies for processing divisible loads on ethernet clusters," *International Journal of Computers and Applications*, vol. 27, no. 2, pp. 97–107, 2005.

- [64] B. Veeravalli, X. Li, and C. C. Ko, “On the influence of start-up costs in scheduling divisible loads on bus networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 12, pp. 1288–1305, dec 2000.
- [65] S. Viswanathan, B. Veeravalli, and T. G. Robertazzi, “Resource-aware distributed scheduling strategies for large-scale computational cluster/grid systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 10, pp. 1450–1461, 2007.
- [66] K. Wang and T. G. Robertazzi, “Scheduling divisible loads with nonlinear communication time,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 51, no. 3, pp. 2479–2485, July 2015.
- [67] Y. Yang, K. van der Raadt, and H. Casanova, “Multiround algorithms for scheduling divisible loads,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 11, pp. 1092–1102, nov. 2005.
- [68] J. Yao and B. Veeravalli, “Design and performance analysis of divisible load scheduling strategies on arbitrary graphs,” *Cluster Computing*, vol. 7, no. 2, pp. 191–207, 2004.
- [69] Z. Zeng and B. Veeravalli, “Distributed scheduling strategy for divisible loads on arbitrarily configured distributed networks using load balancing via virtual routing,” *Journal of Parallel and Distributed Computing*, vol. 66, no. 11, pp. 1404–1418, 2006.
- [70] Q. Zhang, T. S. Yeo, H. S. Tan, and Y. Luo, “Imaging of a moving target with rotating parts based on the hough transform,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 46, no. 1, pp. 291–299, Jan 2008.