Scheduling Nonlinear Computational Loads

JUI TSUN HUNG
THOMAS G. ROBERTAZZI, Fellow, IEEE
Stony Brook University

A scheduling model for a tree network is studied where the computation time for each node is nonlinear in the size of the assigned load. Optimal load allocation and speedup for simultaneous load distribution for a quadratic nonlinearity are obtained using simple equations. An iterative solution for sequential load distribution is presented for a nonlinearity of arbitraty power. Superlinear speedup is possible when computational complexity is nonlinear in the size of assigned loads. Aerospace applications include spectrum computation, radar and sensor data processing, and satellite image processing.

Manuscript received March 13, 2007; revised September 10, 2007; released for publication October 11, 2007.

IEEE Log No. T-AES/44/3/929762.

Refereeing of this contribution was handled by P. K. Willett.

Authors' address: J. T. Hung, Dept. of Computer Science, Stony Brook University, Stony Brook, NY 11794; T. G. Robertazzi, Cosine Laboratory, Dept. of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794, E-mail: (tom@ece.sunysb.edu).

I. INTRODUCTION

It is well known that many algorithms have a computational complexity that is nonlinear in the problem size. These algorithms are widely used in aerospace applications for such purposes as spectrum computation, radar and sensor data processing, and satellite image processing. Divisible loads can occur in these target aerospace applications. Divisible load scheduling techniques are employed in this paper because of their tractability in order to make analytical progress. A divisible load is an input load that can be arbitrarily partitioned and assigned to distributed processors to gain the benefits of parallel processing. No precedence relationships between atomic units of the entire load are assumed.

We note, and discuss below, that algorithms of nonlinear complexity that assume the divisibility of the input data are generally different from algorithms of linear complexity, in terms of a need for significant postprocessing. That is, generally the results of nonlinear subproblems solved among individual processors need to be integrated (postprocessed) to obtain an overall solution. For a fundamental example, a large list to be sorted can be partitioned and distributed among processors (or nodes) of a tree network. After being processed at each node, the fractional loads become sorted sublists that need to be merged (postprocessed) to a final sorted list. We make an empirical observation that to some extent the need to do significant postprocessing arises for those algorithms with a nonlinear nature because of dependencies among the data of such nonlinear problem.

Single level tree networks are largely considered in this paper as a single level tree (star network) forms a fundamental interconnection topology. Multilevel tree networks can be used as a spanning distribution tree embedded in other interconnection topologies as well as being an interconnection topology of interest in itself.

Two representative types of solutions to the scheduling problem of tree networks are considered here. First, speedup and optimal load allocation for simultaneous load distribution (i.e., the root can transmit load to its children simultaneously) are found for a single level tree. For simplicity, a computing time function with a quadratic computational complexity of the size of input load is considered here. Secondly, an iterative solution for sequential load distribution for a single level tree where the computing time function is a power of χ is developed. The order of optimal load allocation at a root node or among parent nodes is assumed to conform to the sequence of communication speeds of parent-child links from highest to lowest. Optimal load allocation of nonlinear loads in multilevel tree networks is also briefly discussed in this paper.

It should be noted that sequential and simultaneous load distribution provide a wide variety of modeling possibilities. Sequential load distribution has been well studied as a scheduling model with linear complexity where a root can communicate with only one child at a time. The improved performance and scalability of simultaneous distribution [2, 3] over sequential distribution motivates future server architectures where one server can distribute load on multiple outgoing links concurrently. This fits in well with the needs of grids such as the military Global Information Grid or the ATLAS physics experiments at CERN (Center European for Nuclear Research) where expensive wide area links need to be kept at high utilizations.

While most of the works on divisible load theory are of linear models, an exception has been developed by Drozdowski and Wolniewicz [4], who demonstrated superlinear speedup by defining processing time as a piecewise linear (and thus nonlinear) function of the size of input load for modeling the memory hierarchy of a computer. Drozdowski and Wolniewicz's results were obtained through mathematical programming, but analytic results are presented in this paper.

A final note is that this study is somewhat limited in scope compared with the wealth of findings available for linear models. For instance, it is assumed that we do not consider return commnications, that communications is substantially faster than computation, and processor ordering for sequential distribution is fixed. However, this is an early study and these issues are substantial topics in themselves.

A. Divisible Load Theory Review

Divisible loads are data parallel loads that are perfectly partitionable among links and processors. Such loads arise in the parallel and data intensive processing with massive amounts of data in grid computing, signal processing, image processing, and aerospace data processing. Since 1988 works by a number of researchers [1–24] have developed algebraic means of determining optimal fractions of a load distributed to processors via corresponding links under a given interconnection topology and a given scheduling policy. Here optimality is defined in terms of speedup and execution time. The theory to date largely involves loads of linear computational complexity. In other words, computational or communication time is proportional to the size of fractional loads distributed to processors via corresponding links. Divisible load modeling should be of interest as it models both computation and network communication in a completely integrated manner. Moreover, it is tractable with its linearity assumption. Optimal divisible load scheduling has been developed for various interconnection topologies [14], such as linear daisy chains [6], buses [8], trees [7, 15, 27], hypercubes [9], and two- and three-dimensional meshes [16, 17]. A number of scheduling policies have been investigated including multi-installments [18], and multi-round scheduling [11, 28], simultaneous distribution [2, 13] and simultaneous start [12]. Also studied are detailed parameterizations and solution time optimization [21], and combinatorial schedule optimization [19]. Generalizations have included models with limited memory [30], and multiple loads [29]. Divisible loads may be divisible in fact or as an approximation as in the case of a large number of relatively small independent tasks [10, 26]. Combinatorics relating to divisible load scheduling is examined in [31]. Introductions to divisible load scheduling theory appear in [1], [5], [20].

The next section describes models and notation. The properties of the computing function are described in Section III. The performance in scheduling a heterogeneous single level tree using store and forward switching, simultaneous distribution, and staggered start protocols is derived in Section IV. The computing function is considered a quadratic function of the size of assigned fractional load. In Section V the performance in scheduling a single level tree using sequential distribution and staggered start is explored. The computing function is a function of power χ of the size of an assigned load. Section VI briefly discusses optimal load distribution for multilevel tree networks. The conclusion and lessons learned are stated in Section VII.

II. MODELS AND NOTATION

In this paper we only consider staggered start. Under staggered start a node cannot process any partial assigned load in advance unless it has already received the entire assigned load. In contrast to staggered start, simultaneous start allows a node to process the assigned load as soon as an atomic piece of data arrives [12] (this is not discussed for reasons of space). As to distribution policies in a single level tree, we consider both simultaneous distribution (Section IV) and sequential distribution (Section V). Simultaneous distribution was first proposed by Piriyakumar and Murthy [13] as a mechanism whereby a parent node in a tree network transmits fractional loads concurrently over multiple links. In contrast, sequential distribution is a different mechanism under which a parent node distributes fractional loads to its children one at a time until all fractional loads are delivered.

A. Model and Notations for A Single Level Tree

A heterogeneous single level tree using staggered start is illustrated in Fig. 1. Each node in this figure

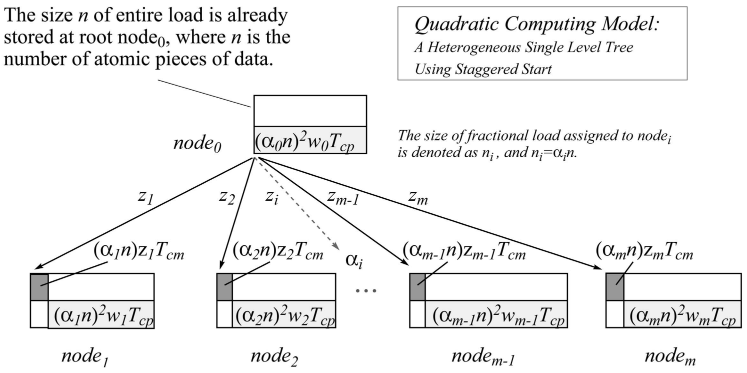

The size n of entire load is already stored at root $node_0$, where n is the number of atomic pieces of data.
Quadratic Computing Model:
A Heterogeneous Single Level Tree Using Staggered Start
$node_0$
$(\alpha_0 n)^2 w_0 T_{cp}$
The size of fractional load assigned to $node_i$ is denoted as n_i, and $n_i=\alpha_i n$.
z_1
z_2
z_i
z_{m-1}
z_m
$(\alpha_1 n)z_1T_{cm}$
$(\alpha_2 n)z_2T_{cm}$
α_i
$(\alpha_{m-1} n)z_{m-1}T_{cm}$
$(\alpha_m n)z_mT_{cm}$
$(\alpha_1 n)^2 w_1 T_{cp}$
$(\alpha_2 n)^2 w_2 T_{cp}$
...
$(\alpha_{m-1} n)^2 w_{m-1} T_{cp}$
$(\alpha_m n)^2 w_m T_{cp}$
$node_1$
$node_2$
$node_{m-1}$
$node_m$

Fig. 1. Single level tree using staggered start. Worst case temporal running cost of an algorithm at $node_i$ is assumed to be $\Theta(n_i^2)$.

is represented by a miniature timing diagram with a distinct computing speed. This single level tree, rooted at $node_0$, can be collapsed into an equivalent $node_{\langle 0\rangle}$ with an equivalent inverse computing speed $\omega_{\langle 0\rangle}$ that describes the computing capability of the entire tree. Collapsing a single tree into an equivalent node is important in scheduling theory when evaluating the performance in a scheduling model specified for a multilevel tree. The concept of processor equivalence was introduced by Robertazzi in 1993 [1, 24].

The following represent notations and symbols for tree networks.

m The number of children in a single level tree

n The total number of records (or indivisible pieces, atomic pieces) forming an entire load at the root node. As the size of an entire load in a tree, it can be denoted $n_{\langle 0\rangle}$ as well

α_0 The load fraction assigned to the root processor

α_i The load fraction assigned to the ith link-processor pair in a single level tree (where $i = 1,2,\ldots,m$)

$\alpha_{\langle i\rangle}$ The load fraction assigned to the ith link-subtree pair in a multilevel tree (where $i = 1,2,\ldots,m$)

$n_i = \alpha_i n$ The number of records processed at $node_i$ (where $i = 0,1,2,\ldots,m$)

$n_{\langle i\rangle} = \alpha_{\langle i\rangle} n$ The number of records processed at equivalent $node_{\langle i\rangle}$ (where $i = 1,2,\ldots,m$), which is a collapsed subtree rooted at $node_i$ in a multilevel tree

w_i The inverse computing speed at the ith processor (where $i = 0,1,2,\ldots,m$)

$w_{\langle i\rangle}$ The equivalent inverse computing speed at equivalent $node_{\langle i\rangle}$ (where $i = 1,2,\ldots,m$) for a collapsed subtree with root at $node_i$

$w_{\langle 0\rangle}$ The equivalent inverse computing speed at equivalent $node_{\langle 0\rangle}$ for an entire tree with root at $node_0$

z_i The inverse communication speed on the ith link (where $i = 0,1,2,\ldots,m$).

T_{cp} Computing intensity constant. The entire load can be processed on the ith processor in time w_iT_{cp}.

T_{cm} Communication intensity constant. The entire load can be delivered over the ith link in time z_iT_{cp}

T_f The finish time. Time at which every processor completes computation.

DEFINITION 1 $\gamma_{\langle 0\rangle}$, the ratio of the inverse computing speed at equivalent $node_{\langle 0\rangle}$ to that at root $node_0$.

$$\gamma_{\langle 0\rangle} = \frac{w_{\langle 0\rangle}}{w_0}. \tag{1}$$

DEFINITION 2 Speedup, the ratio of the computing speed at the equivalent node to that at the root node. In other words, speedup is the inverse of $\gamma_{\langle 0\rangle}$.

$$\text{Speedup} = \frac{1}{\gamma_{\langle 0\rangle}} = \frac{\omega_0}{\omega_{\langle 0\rangle}}. \tag{2}$$

III. PROPERTIES OF COMPUTING FUNCTIONS

To analyze a scheduling model applied to a tree network in terms of recursive equations describing features of computation and communication time, we propose an instance, a Gantt chart-like timing diagram, as shown in Fig. 2. The instance illustrates a scheduling process in a tree network. The tree network can be either a single level tree or a subtree in a multilevel tree and both employ simultaneous distribution and staggered start here. Subscript notation $\langle i\rangle$ denotes an equivalent node collapsed from a subtree rooted at $node_i$ in a multilevel tree. In a single level tree, the subscript notation $\langle i\rangle$ can be converted to subscript notation i, indicating a physical $node_i$. An "equivalent" node, an established concept [6, 24], has identical operating characteristics to the subnetwork it replaces. Here we take a specific policy that every node completes its computing (or

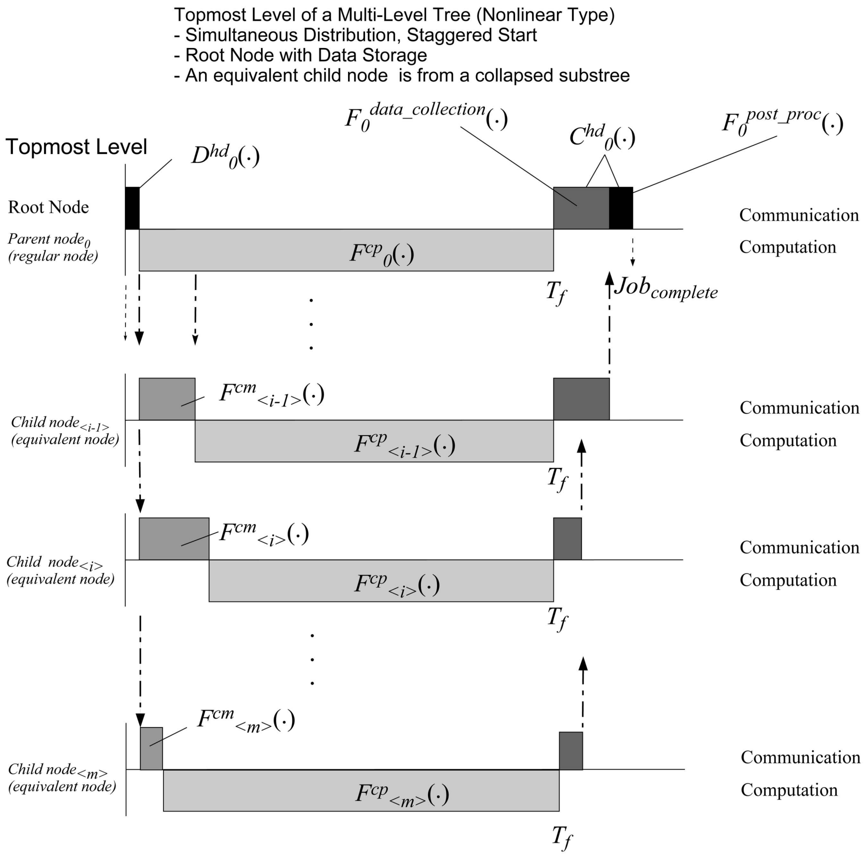


Fig. 2. Timing diagram shows scheduling model for topmost level in multilevel tree with simultaneous distribution and staggered start.

conquering a certain problem) at the same time, then data collecting and postprocessing ensue at the root node to obtain a final outcome. In Fig. 2 $F^{cp}_{\langle 0\rangle}(\cdot)$ is composed of $F_0^{cp}(\cdot)$, $D_0^{hd}(\cdot)$, and $C_0^{hd}(\cdot)$, and the final result is obtained at time $Job_{complete}$.

The following describes the notation in Fig. 2.

1) $F_i^{cp}(\cdot)$, the computing time function at $node_i$. The arguments for this function are the size of the input load, the range of certain parameters for an algorithm, and the like. Here only the argument, the size of input load, are emphasized.

2) $F^{cp}_{\langle i\rangle}(\cdot)$, the computing time function at an equivalent $node_{\langle i\rangle}$.

3) $F^{cm}_{\langle i\rangle}(\cdot)$, the communication time function at link i. Link i is the link by which either $node_i$ or equivalent $node_{\langle i\rangle}$ connects to its parent node, $node_{\langle i\rangle}$. $F^{cm}_{\langle i\rangle}(\cdot)$ can be denoted as $F_i^{cm}(\cdot)$.

4) $F_0^{cp}(\cdot)$, the computing time function at root $node_0$.

5) $D_0^{hd}(\cdot)$, the temporal cost function of partitioning data at root $node_0$.

6) $C_0^{hd}(\cdot)$, the temporal cost function of data collecting and postprocessing at root $node_0$ for obtaining the final outcome.

A computing time function is defined as a product of an algorithm running time (or running steps, an alternative) and the inverse of CPU speed of a node where an input load is processed. The run time of an algorithm is sometimes defined as the number of steps [25] in the literature. This is an appropriate description for running time because the performance in executing an algorithm should be based on a standard, which is independent of the computing powers among distinct machines. Hence, we use running steps instead of running time while an algorithm is run at a node.

The optimal performance in scheduling for a tree network is machine dependent. Here it is assumed that all input loads are processed (to some extent) concurrently. As mentioned earlier, a computing function while an algorithm is executed at $node_i$ is defined as

$$F_i^{cp}(\cdot) = F_i^{cp\cdot algm}(\cdot) \times F_i^{inv\cdot CPU\cdot sp}(\cdot) \tag{3}$$

where

1) $F_i^{cp}(\cdot)$ is the computing function at $node_i$ (unit second).

2) $F_i^{cp\cdot algm}(\cdot)$ is the function of running steps of an algorithm at $node_i$ (unit step).

3) $F_i^{inv\cdot CPU\cdot sp}(\cdot)$ is the function of the inverse of CPU speed at $node_i$ (unit seconds/per step).

As $F_i^{inv\cdot CPU\cdot sp}(\cdot)$ is the inverse of CPU speed, it can be expressed as a conventional notation w_i. On the other hand, we assume that the argument of $F_i^{cp\cdot algm}(\cdot)$ only refers to n_i. Here $F_i^{cp\cdot algm}(\cdot)$ can be either a linear or a nonlinear function of the size n_i of an input load. Furthermore, $F_i^{cp\cdot algm}(\cdot)$ can be induced into a product of a function of the size of an input load and a computing intensity constant T_{cp}. Hence (3) becomes

$$F_i^{cp}(\cdot) = F_i^{cp\cdot algm}(\cdot)w_i \rightarrow F_i^{cp\cdot algm}(n_i)w_iT_{cp}. \qquad (4)$$

By contrast, the communication time function $F_i^{cm}(\cdot)$ can be derived as follows.

$$\begin{aligned} F_i^{cm}(\cdot) &= F_i^{cm\cdot algm}(\cdot) \times F_i^{inv\cdot link\cdot sp}(\cdot) \\ &= F_i^{cm\cdot algm}(\cdot)z_i \rightarrow F_i^{cm\cdot algm}(n_i)z_iT_{cm}. \qquad (5) \end{aligned}$$

Provided that the communication time is linearly proportional to the size of fractional load n_i, then $F_i^{cm\cdot algm}(n_i) = kn_i + c$, where k, c are constants. For simplicity, we assume $F_i^{cm\cdot algm}(n_i) = n_i = \alpha_i n$ and then the communication time function becomes

$$F_i^{cm}(\cdot) = F_i^{cm\cdot algm}(n_i)z_iT_{cm} = n_iz_iT_{cm} = (\alpha_i n)z_iT_{cm} \qquad (6)$$

where

1) $F_i^{cm}(\cdot)$ is the communication time function at link i (unit second).

2) $F_i^{cm\cdot algm}(\cdot)$ is the function of running steps of an algorithm transmitting a fractional load (unit step) via link i. For simplicity, $F_i^{cm\cdot algm}(\cdot)$ can be reduced to a product of a function of distributing the size n_i of a fractional load via link i and the time constant T_{cm}.

3) $F_i^{inv\cdot link\cdot sp}(\cdot)$ is the inverse of link speed function at link i (unit seconds/per step). It can be represented by z_i.

It is necessary to distinguish between a hardware partition and a software partition. A hardware partition means that a load is partitioned and distributed to multiple processors. A software partition means that a load is partitioned at a single machine according to the algorithms used. Details of these two types of partitions are described as follows.

A. Hardware Partition

The core of parallel computing is partitioning a load into fractions, then distributing these fractions to distinct nodes, and finally processing these fractional input loads in parallel. This mechanism is implemented in a hardware partition as defined. The hardware partition considerably decreases the finish time of a data-intensive processing job. In other words, speedup for the job can significantly increase. Unlike the hardware partition, a software partition involves recursively partitioning a fractional load into smaller sizes on a single machine while an algorithm is required to be able to recursively process these smaller divisions of data in a process. A more detailed description of software partition is discussed in the next subsection.

Referring to Fig. 2, fundamental recursive equations for calculating the size of fractional loads assigned to distinct nodes (i.e., the root node and its equivalent child nodes) at the topmost level in a multilevel tree are obtained as

$$F_{\langle i-1\rangle}^{cm}(\cdot) + F_{\langle i-1\rangle}^{cp}(\cdot) = F_{\langle i\rangle}^{cm}(\cdot) + F_{\langle i\rangle}^{cp}(\cdot), \quad i = 2,3,\ldots,m \qquad (7)$$

$$F_0^{cp}(\cdot) = F_{\langle 1\rangle}^{cm}(\cdot) + F_{\langle 1\rangle}^{cp}(\cdot) \qquad (8)$$

$$\alpha_0 + \alpha_{\langle 1\rangle} + \alpha_{\langle 2\rangle} + \cdots + \alpha_{\langle m\rangle} = 1 \quad \text{the normalization equation.} \qquad (9)$$

The computing function at equivalent $node_{\langle 0\rangle}$, collapsed from the entire tree network, can be expressed as

$$F_{\langle 0\rangle}^{cp}(\cdot) = F_0^{cp}(\cdot) + D_0^{hd}(\cdot) + C_0^{hd}(\cdot) \qquad (10)$$

as mentioned earlier. Furthermore, as considering the effectiveness of parallel computing, constraints should be imposed on the hardware partition by the following conditions

$$F_{\langle 0\rangle}^{cp}(\cdot) \ll F_0^{cp}(\cdot) \qquad (11)$$

$$F_{\langle i\rangle}^{cp}(\cdot) \ll F_i^{cp}(\cdot). \qquad (12)$$

That is, the computing time function value at an equivalent node is significantly less than the computing time function value at the root node it replaces.

If the algorithm running at the equivalent $node_{\langle 0\rangle}$ is assumed to be equivalent to the algorithms used at all physical nodes, (10) becomes

$$\begin{aligned} F_{\langle 0\rangle}^{cp}(\cdot) &= F_{\langle 0\rangle}^{cp\cdot algm}(\cdot)w_{\langle 0\rangle} \\ &= F_{\langle 0\rangle}^{cp\cdot algm}(n_{\langle 0\rangle})w_{\langle 0\rangle}T_{cp} \\ &= F_{\langle 0\rangle}^{cp\cdot algm}(n)w_{\langle 0\rangle}T_{cp} \\ &= F_0^{cp\cdot algm}(n_0)w_0T_{cp} + D_0^{hd}(\cdot) + C_0^{hd}(\cdot). \qquad (13) \end{aligned}$$

Here the *algm* superscript indicates a specific algorithm. As shown in Fig. 2, we specify n (a sufficiently large number), the number of an entire load, and n_i, the size of load assigned to $node_i$ (where $i = 0,1,2,\ldots,m$).

Consequently, the hardware partition possesses certain divide-and-conquer properties as follows.

Divide: The number of divide steps is constant and of a linear function of $m + 1$, supposed that there are only $m + 1$ nodes in the tree. This leads $D_0^{hd}(n)$ to the order of $\Theta(1)$, a computational complexity of order 1.

Conquer: There are $m + 1$ subproblems in a processing task and each node is assigned a subproblem with a fractional load.

Combine: The combined procedure depends on the specific algorithm. For instance, the combined procedure of a sorting problem depends on the extent to which the records are already somewhat sorted. Provided that the outcome from each node is already sorted, $C_0^{hd}(n)$ becomes a function of order $\Theta(n)$, which is a computational complexity of order n.

According to the divide-and-conquer properties, (13) can be further simplified as

$$\begin{aligned} F_{\langle 0\rangle}^{cp}(\cdot) &= F_{\langle 0\rangle}^{cp\cdot algm}(n) w_{\langle 0\rangle} T_{cp} \\ &= F_0^{cp\cdot algm}(n_0) w_0 T_{cp} + D_0^{hd}(n) + C_0^{hd}(n) \\ &= F_0^{cp\cdot algm}(n_0) w_0 T_{cp} + \Theta(1) + \Theta(n). \end{aligned} \tag{14}$$

B. Software Partition

Unlike the hardware partition, a software partition is defined as a mechanism under which a fractional load is processed by a divide-and-conquer algorithm at a single machine (a node), rather than at an equivalent node collapsed from multiple processors. Considering that a fractional load of size n_i is processed at a physical $node_i$, if the overall running time on the load of size n_i can be expressed with the running cost (or running steps) on smaller (partitioned) portions of the load, the algorithm makes recursive calls to itself and the running cost can be represented by a recurrence equation [25]. As in the literature, the recurrence equation of running cost $T(n_i)$ for the divide-and-conquer algorithm at $node_i$ can be expressed as

$$T(n_i) = \begin{cases} \Theta(1) & \text{if} \quad n_i \le c, \\ aT(\frac{n_i}{b}) + D(n_i) + C(n_i) & \text{otherwise} \end{cases}. \tag{15}$$

In (15), if the load of size n_i is small enough (say $n_i \le c$ for some constant) and there is no need for further partitioning, a straightforward solution of the divide-and-conquer algorithm would take a constant time $\Theta(1)$. On the other hand, if the load of size n_i is large enough and needs to be partitioned into a subproblems, each of which is $1/b$ the size of the original load, and assuming that dividing the problem into subproblems takes $D(n_i)$ time and combining these subsolutions for a final outcome takes $C(n_i)$ time, it eventually takes a running time cost of $aT(n_i/b) + D(n_i) + C(n_i)$ for the divide-and-conquer algorithm. As a consequence, $T(n_i)$ can represent the time function of $F_i^{cp\cdot algm}(n_i)$

$$F_i^{cp\cdot algm}(n_i) = T(n_i) = aT\left(\frac{n_i}{b}\right) + D(n_i) + C(n_i). \tag{16}$$

Note here we use a, which is different from b, to be more general.

In contrast to the hardware divide-and-conquer properties, software divide-and-conquer properties are expressed as follow.

Divide: The process of divide steps takes only constant time because the data processing problem partitioned into b computational subproblems results $D(n_i)$ on the order of $\Theta(1)$.

Conquer: Generally, a subproblems with the size n_i/b are solved recursively.

Combine: If the combine procedure at $node_i$ has n_i records, the combining cost is denoted as $C(n_i)$. If an algorithm is a sorting algorithm, the cost of its combing process is of the order of computing complexity of $\Theta(n_i)$.

According to the above discussion, the running cost of a sorting problem is expressed as

$$F_i^{cp\cdot algm}(n_i) = T(n_i) = aT\left(\frac{n_i}{b}\right) + \Theta(1) + \Theta(n_i) \tag{17}$$

$T(n_i)$ can be of the order of growth $n_i \log n_i$, n_i^2, n_i^3, 2^{n_i}, or $n_i!$, and so on.

C. Applications

Two categories of linear and nonlinear applications are illustrated as follows.

1) *Linear Applications*: Provided that the running cost is a linear function of the number of records (the size of an input load), then $F_{\langle i\rangle}^{cp\cdot algm}(n_i)$ and $F_i^{cp\cdot algm}(n_i)$ possess the computing complexity of order $\Theta(n_i)$. According to the linearity property, the outcomes of indivisible pieces of load are independent of each other. This leads to a usually negligible postprocessing cost for linear problems of $C_0^{hd}(\cdot)$ of zero. However in exceptional cases significant $C_0^{hd}(\cdot)$ could be included. Because $F_{\langle i\rangle}^{cp\cdot algm}(n_i)$ and $F_i^{cp\cdot algm}(n_i)$ are of the order of $\Theta(n_i)$, we further assume that both of them are functions of n_i. Ignoring the scale factor and constant, $F_{\langle i\rangle}^{cp\cdot algm}(n_i)$ and $F_i^{cp\cdot algm}(n_i)$ become n_i. As a consequence, (14) is further derived

$$F_{\langle 0\rangle}^{cp}(\cdot) = F_{\langle 0\rangle}^{cp\cdot algm}(n) w_{\langle 0\rangle} T_{cp} = n w_{\langle 0\rangle} T_{cp} \tag{18}$$

$$\begin{aligned} &= F_0^{cp\cdot algm}(n_0) w_0 T_{cp} + D_0^{hd}(n) + C_0^{hd}(n) \\ &= n_0 w_0 T_{cp} + \Theta(1) + 0 \\ &= \alpha_0 n w_0 T_{cp} + \Theta(1). \end{aligned} \tag{19}$$

Referring to (18) and (19), one obtains

$$\begin{aligned} n w_{\langle 0\rangle} T_{cp} &= \alpha_0 n w_0 T_{cp} + \Theta(1) \\ w_{\langle 0\rangle} T_{cp} &= \alpha_0 w_0 T_{cp} + \frac{\Theta(1)}{n}. \end{aligned} \tag{20}$$

If the number of records is sufficiently large such that $\Theta(1)/n$ approaches to zero, (20) becomes

$$w_{\langle 0\rangle} T_{cp} = \alpha_0 w_0 T_{cp}. \tag{21}$$

2) *Nonlinear Applications*: As an example, provided that $F_i^{cp\cdot algm}(n_i)$ is of order $\Theta(n_i^2)$, and it can be further simplified to n_i^2, without a loss of generality, (14) becomes

$$
\begin{aligned}
F^{cp}_{\langle 0\rangle}(\cdot) &= F^{cp\cdot algm}_{\langle 0\rangle}(n^2)w_{\langle 0\rangle}T_{cp} = n^2 w_{\langle 0\rangle}T_{cp} && (22)\\
&= F_0^{cp\cdot algm}(n_0)^2 w_0 T_{cp} + D_0^{hd}(n) + C_0^{hd}(n)\\
&= n_0^2 w_0 T_{cp} + \Theta(1) + \Theta(n)\\
&= (\alpha_0 n)^2 w_0 T_{cp} + \Theta(1) + \Theta(n). && (23)
\end{aligned}
$$

The equivalent computing function $F^{cp}_{\langle 0\rangle}(\cdot)$ at $node_{\langle 0\rangle}$ becomes a quadratic equation of the load size $\alpha_0 n$ as shown in (23). According to (22) and (23), one obtains

$$
\begin{aligned}
n^2 w_{\langle 0\rangle}T_{cp} &= (\alpha_0 n)^2 w_0 T_{cp} + \Theta(1) + \Theta(n)\\
w_{\langle 0\rangle}T_{cp} &= \alpha_0^2 w_0 T_{cp} + \frac{\Theta(1)}{n^2} + \frac{\Theta(n)}{n^2}.
\end{aligned} \qquad (24)
$$

If the number n of records is sufficiently large such that $\Theta(1)/n^2$ and $\Theta(n)/n^2$ approach zero, (24) would be reduced to

$$w_{\langle 0\rangle}T_{cp} = \alpha_0^2 w_0 T_{cp}. \qquad (25)$$

IV. SPEEDUP PERFORMANCE OF A SINGLE LEVEL TREE USING SIMULTANEOUS DISTRIBUTION

In this section we consider a heterogeneous single level tree in which processors use simultaneous load distribution and the staggered start protocol to process the load fractions assigned. Using the staggered start protocol a processor must receive its load completely before it begins to process the load. The root node can distribute load to its children while processing some fraction of the load. In this sense the root may be considered to have a front-end subprocessor for communications off-loading.

A. Speedup Derivation for A Single Level Tree with Running Time $\Theta(n_i^2)$

The structure of a single level tree network with $m+1$ processors and m links is illustrated in Fig. 1. All children processors are connected to the root processor via direct communication links. Assumed to be the only one where the divisible load arrives, the root processor in a single level tree partitions the load into $m+1$ fractions and subsequently distributes fractions $\alpha_1, \alpha_2, \ldots$, and α_m to children processors concurrently, while fraction α_0 of its own is processed under computation. Given that the entire load received is of *n records* (or *n atomic pieces*), the fractional load at the root $node_0$ is denoted n_0 (where $n_0 = \alpha_0 n$) and the other fractional load at child $node_i$ is represented n_i (where $n_i = \alpha_i n$, $i = 1,2,\ldots,m$).

As an example in this section we assume that the worst case running cost of an algorithm is $\Theta(n_i^2)$ ($i = 0,1,2,\ldots,m$) and the computation time function at a node becomes a quadratic equation in the load size n_i. However, the communication time function on a link is still assumed linear in load size transmitted via the link.

In order to minimize the processing finish time, all of the utilized processors in the network must finish computing at the same time [1]. Intuitively, otherwise the load could be transferred from busy processors to idle processors to improve the solution (see the Appendix for a proof). The process of load distribution can be represented by Gantt chart-like timing diagrams as illustrated in Fig. 3. It is assumed that at the root node the entire load is available for distribution at time $t = 0$.

To calculate the speedup of a tree network, four types of equations are employed in this section, which are the recursive, normalization, speedup, and constraint equations.

1) *Recursive Equations*: As mentioned, it is known that for an optimal solution in terms of makespan for linear problems all processors should stop at the same time [1]. The same is true for a nonlinear problem such as in this section (see the proof in Appendix). Thus according to the timing diagram Fig. 3, the fundamental recursive equations of the system can be formulated as follows

$$(\alpha_0 n)^2 w_0 T_{cp} = (\alpha_i n) z_i T_{cm} + (\alpha_i n)^2 w_i T_{cp}, \qquad i = 1,2,\ldots,m. \qquad (26)$$

In addition, the normalization equation for a single level tree is

$$\alpha_0 + \alpha_1 + \alpha_2 + \cdots + \alpha_m = 1. \qquad (27)$$

This yields $m+1$ equations with $m+1$ unknowns. Manipulating the recursive equations and normalization equation can yield the solution for the fractions of load distribution. Now (26) can be converted to

$$\alpha_i^2 + \frac{z_i T_{cm}}{n w_i T_{cp}}\alpha_i - \frac{w_0 T_{cp}}{w_i T_{cp}}\alpha_0^2 = 0. \qquad (28)$$

Let

$$\xi_i = \frac{w_0 T_{cp}}{w_i T_{cp}} = \frac{w_0}{w_i}, \qquad i = 1,2,\ldots,m \qquad (29)$$

and

$$\varsigma_i = \frac{z_i T_{cm}}{n w_i T_{cp}} = \frac{\sigma_i}{n} \qquad \text{where} \quad \sigma_i = \frac{z_i T_{cm}}{w_i T_{cp}}, \qquad i = 1,2,\ldots,m \qquad (30)$$

then recursive equation (28) becomes

$$\alpha_i^2 + \varsigma_i \alpha_i - \xi_i \alpha_0^2 = 0. \qquad (31)$$

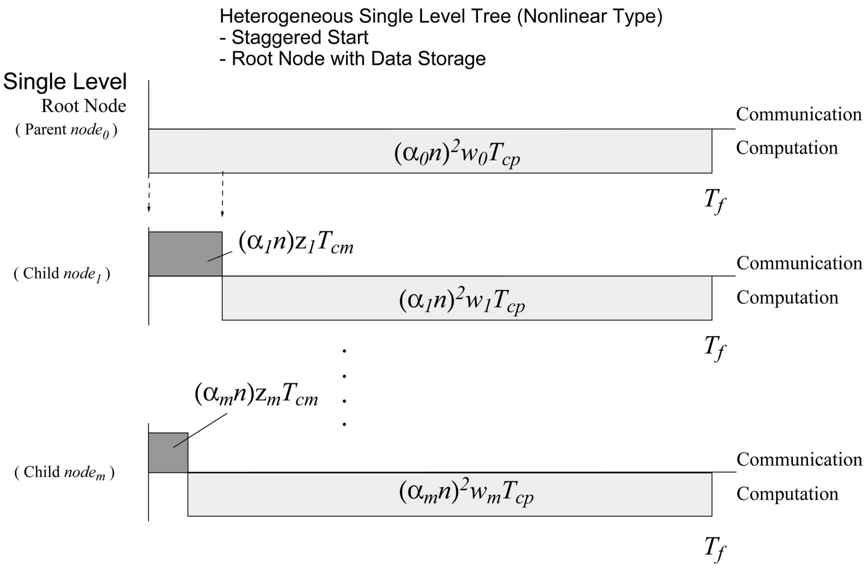

Heterogeneous Single Level Tree (Nonlinear Type)
- Staggered Start
- Root Node with Data Storage
Single Level
Root Node
(Parent $node_0$)
Communication
Computation
$(\alpha_0 n)^2 w_0 T_{cp}$
T_f
$(\alpha_1 n) z_1 T_{cm}$
(Child $node_1$)
Communication
Computation
$(\alpha_1 n)^2 w_1 T_{cp}$
T_f
$(\alpha_m n) z_m T_{cm}$
(Child $node_m$)
Communication
Computation
$(\alpha_m n)^2 w_m T_{cp}$
T_f

Fig. 3. Timing diagram of single level tree with simultaneous distribution, staggered start.

Applying the quadratic formula to (31), one obtains

$$\alpha_i = \frac{-\varsigma_i \pm \sqrt{\varsigma_i^2 + 4\xi_i\alpha_0^2}}{2\times 1}. \tag{32}$$

Since the value of α_i is the load fraction at $node_i$, it does not make any physical sense if $\alpha_i < 0$. Hence, $\alpha_i \geq 0$ and the solution of α_i becomes

$$\alpha_i = \frac{-\varsigma_i + \sqrt{\varsigma_i^2 + 4\xi_i\alpha_0^2}}{2}, \qquad i = 1,2,\ldots,m. \tag{33}$$

2) *Normalization Equation*: Employing (33), normalization equation (27) becomes

$$\alpha_0 + \sum_{i=1}^{m} \frac{-\varsigma_i + \sqrt{\varsigma_i^2 + 4\xi_i\alpha_0^2}}{2} = 1. \tag{34}$$

To obtain the value of variable α_0, (34) is solved by the quadratic formula. Here the value of variable α_0 is specified as C_0 (a specific value), and then the load fractions for children nodes in (33) can be represented as follows:

$$\alpha_i = \frac{-\varsigma_i + \sqrt{\varsigma_i^2 + 4\xi_i C_0^2}}{2}. \tag{35}$$

3) Speedup Equation: Now if a single level tree rooted at $node_0$ is collapsed into an equivalent node $node_{\langle 0\rangle}$, and the total load size is n, the computational time can be expressed as $(n)^2 w_{\langle 0\rangle} T_{cp}$ ($w_{\langle 0\rangle}$ is the inverse computing speed of the equivalent $node_{\langle 0\rangle}$). According to the Gantt chart-like timing diagrams Fig. 3, the computational time of the equivalent node (or the tree network) is equal to the computational time at the root in the tree network. That is, the finish time T_f becomes

$$T_f = (n)^2 w_{\langle 0\rangle} T_{cp} = (\alpha_0 \times n)^2 w_0 T_{cp} = (C_0 \times n)^2 w_0 T_{cp}. \tag{36}$$

Moreover,

$$w_{\langle 0\rangle} T_{cp} = \alpha_0^2 w_0 T_{cp} = C_0^2 w_0 T_{cp}. \tag{37}$$

According to Definition 1 in Section II (i.e., $\gamma_{\langle 0\rangle} = w_{\langle 0\rangle}/w_0$), the value of $\gamma_{\langle 0\rangle}$ can be obtained from (37) as

$$\gamma_{\langle 0\rangle} = C_0^2 = \alpha_0^2. \tag{38}$$

In this section speedup is the ratio of job solution time at one processor to job solution time at a tree network with $m+1$ processors (see Definition 2 in Section II). As a result,

$$\text{Speedup} = \frac{1}{\gamma_{\langle 0\rangle}} = \frac{1}{C_0^2} = \left(\frac{1}{\alpha_0}\right)^2. \tag{39}$$

4) *Conditions*:

a) The value of σ_i: The definition of σ_i in (30) is the ratio of communication time to computation time at $node_i$. Under a simultaneous distribution protocol, the communication speed on $link_i$ is assumed to be significantly faster than the computing speed at $node_i$, a node receiving the fractional load via $link_i$. This will guarantee that the physical characteristics of tree networks are well fitted for our analysis model. On the other hand, if the communication time at some node is too slow relative to its corresponding computation time, not all nodes are needed for an optimal solution [1]. Assuming σ_i is significantly smaller than 1 (communication time is assumed considerably less than computing time) and n is large enough for data-intensive problem, ς_i in (30) would become infinitesimal.

b) The range of ξ_i: For isometric (balanced) rather than drastically unbalanced computing power for parallel computing, the computing speed of each node in a tree network is specified as less than or equal to the computing speed of the child's parent by a factor of m, and greater than or equal to that of the parent by a factor of $1/m$. That is,

$$\frac{1}{m}\cdot\frac{1}{w_0} \le \frac{1}{w_i} \le m\cdot\frac{1}{w_0}, \qquad i = 1,2,\ldots,m. \tag{40}$$

Hence, the condition of a balanced computing tree network is given as follows.

$$\frac{1}{m} \le \xi_i = \frac{w_0}{w_i} \le m, \qquad i = 1,2,\ldots,m. \tag{41}$$

The range of ξ_i is not a required condition, but here it makes a tree model better fitted to the developed mathematical analysis if it follows the above condition.

c) The speedup of the tree network: In (34) given that $\varsigma_i = 0$ (assuming communication time is significantly smaller than computing time and the total number n of records for data-intensive problems is considerably large) and $\xi_i = 1$ (the root processor has the same processing speed as the children processors), the value of variable α_0 becomes

$$\alpha_0 = \frac{1}{m+1}. \tag{42}$$

This results in the speedup of the tree model (39) as

$$\text{Speedup} = (m+1)^2. \tag{43}$$

Speedup is a measure of the achievable parallel processing advantage. Note the speedup here is greater than a linear speedup. This outcome is different from linear models where speedup growth is linear or less than linear. For instance, a homogeneous single level tree with m child nodes may have a speedup of $m+1$, which is linear to the number of nodes within this tree network. The superlinear speedup is a consequence of the nonlinear computing time assumption and was noted by Drozdowksi and Wolniewicz [4].

V. SPEEDUP OF A SINGLE LEVEL TREE WITH SEQUENTIAL DISTRIBUTION AND STAGGERED START

Sequential load distribution is employed in this section in a heterogeneous single level tree using staggered start. Sequential load distribution is used as the model in most of the divisible load scheduling literature. Even though a closed-form solution for optimal load allocation and speedup is not possible, an iterative solution is developed.

A. Speedup Derivation for A Single Level Tree with Running Time $\Theta(n_i^\chi)$

The structure of a single level tree network with root, $m+1$ processors, and m links is illustrated in Fig. 1. In this section we assume that the worst case running cost of an algorthm is $\Theta(n_i^\chi)$ ($i = 0,1,2,\ldots,m$), then the computation time function at a node becomes a power χ function ($\chi \ge 2$) in load size n_i. Still, the communication time function on a link is a linear function in its assigned load size.

In order to minimize the processing finish time, all of the utilized processors in the network must finish computing at the same time [1]. The process of load distribution can be represented by Gantt chart-like timing diagrams, as illustrated in Fig. 4. It is assumed that all of the load is available at the root node at time $t = 0$.

Four types of equations are again needed to determine the speedup. They are the recursive, normalization, constraints, and speedup equations.

1) *Recursive Equations and Normalization Equation*: According to the timing diagram Fig. 4, the fundamental recursive equations of the system can be formulated as follows:

$$(\alpha_i n)^\chi w_i T_{cp} = (\alpha_{i+1} n)^\chi w_{i+1} T_{cp} + (\alpha_{i+1} n) z_{i+1} T_{cm}, \qquad i = 0,1,2,\ldots,m-1. \tag{44}$$

The normalization equation is

$$\alpha_0 + \alpha_1 + \alpha_2 + \cdots + \alpha_m = 1. \tag{45}$$

This yields $m+1$ equations with $m+1$ unknowns. Manipulating the recursive equations and normalization equation can yield the solution for the fractions of load distribution. Now from (44),

$$\alpha_i^\chi = \frac{w_{i+1} T_{cp}}{w_i T_{cp}} \alpha_{i+1}^\chi + \frac{z_{i+1} T_{cm}}{n^{\chi-1} w_i T_{cp}} \alpha_{i+1}, \qquad i = 0,1,2,\ldots,m-1. \tag{46}$$

Let

$$\xi_{i+1} = \frac{w_{i+1} T_{cp}}{w_i T_{cp}} = \frac{w_{i+1}}{w_i}, \qquad i = 1,2,\ldots,m \tag{47}$$

and

$$\varsigma_i = \frac{z_i T_{cm}}{n^{\chi-1} w_i T_{cp}} = \frac{\sigma_i}{n^{\chi-1}} \qquad \text{where} \quad \sigma_i = \frac{z_i T_{cm}}{w_i T_{cp}}, \qquad i = 1,2,\ldots,m. \tag{48}$$

This results (46) in

$$(\alpha_i)^\chi = \xi_{i+1}(\alpha_{i+1})^\chi + \xi_{i+1}\varsigma_{i+1}\alpha_{i+1}, \qquad i = 0,1,2,\ldots,m-1. \tag{49}$$

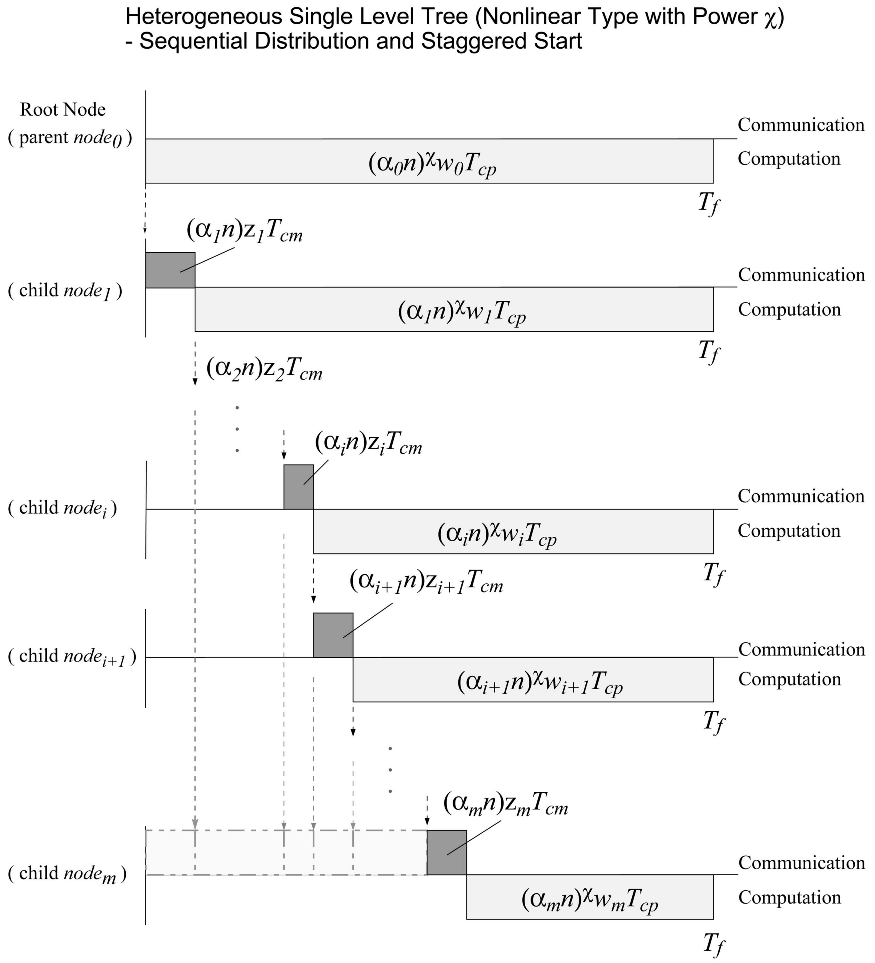


Fig. 4. Timing diagram of heterogeneous single level tree using sequential distribution and staggered start.

2) *Conditions*: The features of σ_i, ς_i, and the range of ξ_i are the same as in Section IV.

The matrix equation consisting of recursive equations and the normalization equation is represented as follows

$$\begin{bmatrix} \alpha_0^\chi \\ \alpha_1^\chi \\ \alpha_2^\chi \\ \alpha_3^\chi \\ \vdots \\ \alpha_{m-1}^\chi \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & \xi_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \xi_2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \xi_3 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & \xi_m \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \begin{bmatrix} \alpha_0^\chi \\ \alpha_1^\chi \\ \alpha_2^\chi \\ \alpha_3^\chi \\ \vdots \\ \alpha_{m-1}^\chi \\ \alpha_m^\chi \end{bmatrix}$$

$$+ \begin{bmatrix} 0 & \xi_1\varsigma_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \xi_2\varsigma_2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \xi_3\varsigma_3 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & \xi_m\varsigma_m \\ 1 & 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \vdots \\ \alpha_{m-1} \\ \alpha_m \end{bmatrix}.$$

These unknowns, $\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_m$, can be solved by standard iterative techniques. That is, one substitutes an initial guess of the α (and α^χ) vector into the right-hand side of the matrix equation, to create the (left-hand side) new estimate of the α^χ vector which is then substituted into the right side, and on and on, until convergence occurs. Such iterative solution is a well-known applied mathematics technique for implicit equation solution and it is well known that it has robust convergence properties.

3) *Alternative Recursive Equations and Normalization Equation*: According to the timing diagram Fig. 4, the fundamental recursive equations of the system can be formulated as follows:

$$(\alpha_0 n)^\chi w_0 T_{cp} = (\alpha_i n)^\chi w_i T_{cp} + \sum_{h=1}^{i} (\alpha_h n) z_h T_{cm}, \quad i = 1,2,\ldots,m. \tag{50}$$

The normalization equation is

$$\alpha_0 + \alpha_1 + \alpha_2 + \cdots + \alpha_m = 1. \tag{51}$$

This yields $m+1$ equations with $m+1$ unknowns.

Equation (50) becomes

$$(\alpha_i)^\chi w_i + \sum_{h=1}^{i} \alpha_h \varsigma_h w_h = (\alpha_0)^\chi w_0, \qquad i = 1,2,\ldots,m \tag{52}$$

where

$$\varsigma_h = \frac{z_h T_{cm}}{n^{\chi-1} w_h T_{cp}} = \frac{\sigma_h}{n^{\chi-1}}. \tag{53}$$

The matrix equation consists of recursive equations and normalization equation, represented as follows

$$\begin{bmatrix} 1 \\ \alpha_0^\chi w_0 \\ \alpha_0^\chi w_0 \\ \alpha_0^\chi w_0 \\ \vdots \\ \alpha_0^\chi w_0 \\ \alpha_0^\chi w_0 \end{bmatrix} = \begin{bmatrix} 0 \\ \alpha_1^\chi w_1 \\ \alpha_2^\chi w_2 \\ \alpha_3^\chi w_3 \\ \vdots \\ \alpha_{m-1}^\chi w_{m-1} \\ \alpha_m^\chi w_m \end{bmatrix} + \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 & 1 \\ 0 & \varsigma_1 w_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \varsigma_1 w_1 & \varsigma_2 w_2 & 0 & \cdots & 0 & 0 \\ 0 & \varsigma_1 w_1 & \varsigma_2 w_2 & \varsigma_3 w_3 & \cdots & 0 & 0 \\ 0 & \varsigma_1 w_1 & \varsigma_2 w_2 & \varsigma_3 w_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \varsigma_1 w_1 & \varsigma_2 w_2 & \varsigma_3 w_3 & \cdots & \varsigma_{m-1} w_{m-1} & \varsigma_m w_m \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \vdots \\ \alpha_{m-1} \\ \alpha_m \end{bmatrix}.$$

These unknowns, $\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_m$, can, again, be solved iteratively.

4) *Speedup Equation*: Now, if a single level tree rooted at $node_0$ is collapsed into an equivalent node $node_{\langle 0 \rangle}$, and the total load size is n, the computational time can be expressed as $(n)^\chi w_{\langle 0 \rangle} T_{cp}$ ($w_{\langle 0 \rangle}$ is the inverse computing speed of equivalent $node_{\langle 0 \rangle}$). According to the Gantt chart-like timing diagrams, Fig. 4, the computational time of the equivalent node (or the tree network) is equal to the computational time at the root in the tree network. Consequently, the finish time T_f becomes

$$T_f = (n)^\chi w_{\langle 0 \rangle} T_{cp} = (\alpha_0 \times n)^\chi w_0 T_{cp}. \tag{54}$$

Hence,

$$w_{\langle 0 \rangle} T_{cp} = \alpha_0^\chi w_0 T_{cp}. \tag{55}$$

According to Definition 1 in Section II (i.e., $\gamma_{\langle 0 \rangle} = w_{\langle 0 \rangle}/w_0$) and (55), the value of $\gamma_{\langle 0 \rangle}$ becomes

$$\gamma_{\langle 0 \rangle} = \alpha_0^\chi. \tag{56}$$

Thus, the expression for superlinear speedup is

$$\text{Speedup} = \frac{1}{\gamma_{\langle 0 \rangle}} = \left(\frac{1}{\alpha_0}\right)^\chi. \tag{57}$$

VI. EXTENSION TO MULTILEVEL TREE NETWORKS

Using available methods in the literature [1, 3, 23], optimal load allocation can be determined for multilevel tree networks where load originates at the root node. This is true for both simultaneous and sequential load distribution. The basic idea is one solves for equivalent processing speed of one single level subtree at a time, working from the bottom of the tree upwards. As single level trees within the multilevel tree are considered, they are replaced by equivalent processors [1, 24] until the entire tree is replaced by a single equivalent processor. After this one can solve for the optimal load allocations by considering subtrees of equivalent processors from top to bottom of the tree. Tree networks are important,from an applied point of view, as the nodes in any general network topology can be interconnected using a (spanning) tree overlay network.

VII. CONCLUSION AND LESSONS LEARNED

The following are the findings that have resulted from this study.

1) It is possible to solve for optimal load allocations and speedup for models with nonlinear power law computational complexity, either through relatively simple equations or iteratively. A proof has been provided of the condition for optimal load distribution of nonlinear loads.

2) Nonlinear problems have a need for postprocessing, because of the dependency of the input data when processed by a nonlinear algorithm.

3) We analytically corroborate the results of Drozdowksi and Wolniewicz [4] that superlinear speedup can result for nonlinear divisible load processing.

4) It should be pointed out that higher order nonlinear equations can suffer from numerical error (due to finite computer word size) problems and so some care is warranted.

5) We note that the findings of this study are somewhat limited compared with the wealth of information available for linear models. This is due to the early nature of this study and the simplifying assumptions made in it (see Introduction).

6) A proof of the simultaneous distribution method's optimality by contradiction appears in the Appendix. It seems it would be true for sequential distribution (by intuition) as well. However, because of the apparent complexity of the sequential distribution proof, it is not provided here.

We have sought to demonstrate the possibility of optimal scheduling for a number of representative scheduling policies on tree interconnection networks under power law nonlinearties in the space available. Of course for specific applications other scheduling policies, nonlinear functional forms and topologies may be of interest. Because of the superlinear speedup, parallel processing of loads with nonlinear computational complexity is a promising technique to maximize computational efficiency on multiple processor systems.

APPENDIX

The following theorem [1] is proved here.

THEOREM *If all of the nodes of the nonlinear computing model receiving non-zero load fractions stop computing at the same time, the processing time (makespan) is minimum for the simultaneous distribution strategy.*

A simultaneous distribution (See Fig. 3) in single level trees with $m+1$ nodes ($node_0, node_1, \ldots, node_m$), and m links ($l_1, \ldots, l_m$) is taken into account. Before the proof, some definitions need to be illustrated first [1].

1) *Load Distribution*: $\boldsymbol{\alpha}$ is an ordered $m+1$ tuple

$$\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_m) \tag{58}$$

where α_i is the load fraction assigned to $node_i$. Further, the normalization equation is

$$\sum_{i=0}^{m} \alpha_i = 1 \qquad \text{where} \quad 0 \le \alpha_i \le 1, \qquad i = 0, 1, \ldots, m. \tag{59}$$

The set of all feasible load distributions is denoted by L.

2) *Finish Time*: The finish time of $node_i$ is denoted by $T_i(\boldsymbol{\alpha})$, for a given load distribution $\boldsymbol{\alpha} \in L$.

3) *Processing Time*: For a given $\boldsymbol{\alpha} \in L$, this is defined as

$$T(\boldsymbol{\alpha}) = \max\{T_0(\boldsymbol{\alpha}), T_1(\boldsymbol{\alpha}), \ldots, T_m(\boldsymbol{\alpha})\}. \tag{60}$$

In other words, $T(\boldsymbol{\alpha})$ is the time at which the entire load is processed.

4) *Minimum Processing Time*: This is defined as

$$T^* = \min_{\alpha \in L} T(\boldsymbol{\alpha}). \tag{61}$$

5) *Optimal Load Distribution*: This is defined as the load distribution $\boldsymbol{\alpha}^* \in L$ such that the processing time is a minimum. That is,

$$\boldsymbol{\alpha}^* = \arg\min_{\alpha \in L} T(\boldsymbol{\alpha}). \tag{62}$$

Only the simultaneous distribution (See Fig. 3) is proved by the contradiction method here.

We assume that a nonlinear computing function at a node in a single level tree, such as the tree shown in Fig. 3, is of power χ, where $\chi \ge 1$. This condition is used for the proof of simultaneous distribution and illustrated as follows.

PROOF Let $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_m) \in L$ be the initial load distribution such that all the nodes stop computing at the same time. Provided that the processing time is not a minimum, there must exist an $\boldsymbol{\alpha}^* = (\alpha_0^*, \alpha_1^*, \alpha_2^*, \ldots, \alpha_m^*) \in L$ such that $\boldsymbol{\alpha}^*$ satisfiies

$$\boldsymbol{\alpha}^* = \arg\min_{\alpha \in L} T(\boldsymbol{\alpha}). \tag{63}$$

This leads to

$$T_i(\boldsymbol{\alpha}^*) < T_i(\boldsymbol{\alpha}) \qquad \text{where} \quad i = 0, 1, 2, \ldots, m. \tag{64}$$

1) At $node_0$: Because the finish time at the root $node_0$ is $(\alpha_0 n)^\chi w_0 T_{cp}$, (64) becomes

$$(\alpha_0^* n)^\chi w_0 T_{cp} < (\alpha_0 n)^\chi w_0 T_{cp}. \tag{65}$$

Without loss of generality, let χ be an integer, where $\chi \ge 1$. Now, (65) is converted to $((\alpha_0^*)^\chi - (\alpha_0)^\chi) n^\chi w_0 T_{cp} < 0$, such that one may obtain

$$(\alpha_0^* - \alpha_0)\{(\alpha_0^*)^{\chi-1} + (\alpha_0^*)^{\chi-2}\alpha_0 + \cdots + (\alpha_0^*)^1(\alpha_0)^{\chi-2} + (\alpha_0)^{\chi-1}\} n^\chi w_0 T_{cp} < 0. \tag{66}$$

Because α_i^*, α_i, n, w_0, and T_{cp} are all positive, this leads to

$$\{(\alpha_0^*)^{\chi-1} + (\alpha_0^*)^{\chi-2}\alpha_0 + \cdots + (\alpha_0^*)^1(\alpha_0)^{\chi-2} + (\alpha_0)^{\chi-1}\} n^\chi w_0 T_{cp} > 0. \tag{67}$$

Hence, one obtains $(\alpha_0^* - \alpha_0) < 0$, such that

$$\alpha_0^* < \alpha_0. \tag{68}$$

2) At $node_i$: According to (26) where the power is replaced with χ, the finish time of the child node $node_i$ takes $(\alpha_i n)^\chi w_i T_{cp} + (\alpha_i n) z_i T_{cm}$. Regarding (64) while $i = 1, 2, \ldots, m$, this yields

$$(\alpha_i^* n)^\chi w_i T_{cp} + (\alpha_i^* n) z_i T_{cm} < (\alpha_i n)^\chi w_i T_{cp} + (\alpha_i n) z_i T_{cm}, \qquad i = 1, 2, \ldots, m. \tag{69}$$

This can be transformed into

$$((\alpha_i^*)^\chi - (\alpha_i)^\chi) n^\chi w_i T_{cp} + (\alpha_i^* - \alpha_i) n z_i T_{cm} < 0 \tag{70}$$

then

$$(\alpha_i^* - \alpha_i)\{[(\alpha_i^*)^{\chi-1} + (\alpha_i^*)^{\chi-2}\alpha_i + \cdots + (\alpha_i^*)^1(\alpha_i)^{\chi-2} + (\alpha_i)^{\chi-1}] n^\chi w_i T_{cp} + n z_i T_{cm}\} < 0. \tag{71}$$

Because α_i^*, α_i, n, w_i, z_i, T_{cp}, and T_{cm} are all positive, this leads to

$$[(\alpha_i^*)^{\chi-1} + (\alpha_i^*)^{\chi-2}\alpha_i + \cdots + (\alpha_i^*)^1(\alpha_i)^{\chi-2} + (\alpha_i)^{\chi-1}]n^{\chi} w_i T_{cp} + nz_i T_{cm} > 0. \tag{72}$$

Hence, one obtains $(\alpha_i^* - \alpha_i) < 0$, such that

$$\alpha_i^* < \alpha_i \qquad \text{where} \quad i = 1,2,\ldots,m. \tag{73}$$

According to (68) and (73), it turns out that

$$\sum_{j=0}^{m} \alpha_j^* < \sum_{j=0}^{m} \alpha_j. \tag{74}$$

This comes out a contradiction since both $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^* \in L$ and their components should sum to one.

REFERENCES

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

[2] Hung, J. T., Kim, H. J., and Robertazzi, T. G.
Scalable scheduling in parallel processors.
In *Proceedings of the 2002 Conference on Information Sciences and Systems*, Princeton University, Princeton, NJ, 2002.

[3] Hung, J. T., and Robertazzi, T. G.
Scalable scheduling for clusters and grids using cut through switching.
International Journal of Computers and their Applications, **26** (2004), 147–156.

[4] Drozdowksi, M., and Wolniewicz, P.
Out-of-core divisible load processing.
IEEE Tranactions on Parallel and Distributed Systems, **14** (2003), 1048–1056.

[5] Bharadwaj, V., Ghose, D., and Robertazzi, T. G.
Divisible load theory: A new paradigm for load scheduling in distributed systems.
Cluster Computing, **6** (2003), 7–18.

[6] Cheng, Y. C., and Robertazzi, T. G.
Distributed computation with communication delays.
IEEE Transactions on Aerospace and Electronic Systems, **24** (1988), 700–712.

[7] Barlas, G. D.
Collection aware optimum sequencing of operations and closed form solutions for the distribution of divisible load on arbitrary processor trees.
IEEE Transactions on Parallel and Distributed Systems, **9** (1998), 429–441.

[8] Bataineh, S., and Robertazzi, T. G.
Bus oriented load sharing for a network of sensor driven processors.
IEEE Transactions on Systems, Man and Cybernetics, **21** (1991), 1202–1205.

[9] Blazewicz, J., and Drozdowski, M.
Scheduling divisible jobs on hypercubes.
Parallel Computing, **21** (1995), 1945–1956.

[10] Beaumont, O., Carter, L., Ferrante, J., Legrand, A., and Robert, Y.
Bandwidth-centric allocation of independent tasks on heterogeneous platforms.
In *Proceedings of the International Parallel and Distributed Processing Symposium* (IPDPS'02), Fort Lauderdale, FL, 2002.

[11] Yang, Y., and Casanova, H.
UMR: A multi-round algorithm for scheduling divisible workloads.
In *Proceedings of the International Parallel and Distributed Processing Symposium* (IPDPS'03), Nice, France, 2003.

[12] Kim, H. J.
A novel load distribution algorithm for divisible loads.
Cluster Computing, **6** (2003), 41–46.

[13] Piriyakumar, D. A. L., and Murthy, C. S. R.
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, **28** (1998), 245–251.

[14] Li, K.
Parallel processing of divisible loads on partitionable static interconnection networks.
Cluster Computing, **6** (2003), 47–56.

[15] Kim, H. J., Jee, G.-I., and Lee, J. G.
Optimal load distribution for tree network processors.
IEEE Transactions on Aerospace and Electronic Systems, **32** (1996), 607–612.

[16] Blazewicz, J., Drozdowski, M., Guinand, F., and Trystram, D.
Scheduling a divisible task in a 2-dimensional mesh.
Discrete Applied Mathematics, (1999), 35.

[17] Glazek, W.
A multistage load distribution strategy for three dimensional meshes.
Cluster Computing, **6** (2003), 31–40.

[18] Bharadwaj, V., Ghose, D., and Mani, V.
Multi-installment load distribution in tree networks with delay.
IEEE Transactions on Aerospace and Electronic Systems, **31** (1995), 555–567.

[19] Dutot, P.-F.
Divisible load on heterogeneous linear array.
In *Proceedings of the International Parallel and Distributed Processing Symposium* (IPDPS'03), Nice, France, 2003.

[20] Robertazzi, T. G.
Ten reasons to use divisible load theory.
Computer, **31** (2003), 63–68.

[21] Adler, M., Gong, Y., and Rosenberg, A. L.
Optimal sharing of bags of tasks in heterogeneous clusters.
Presented at the Symposium on Parallelism in Algorithms and Architectures (SPAA'03), San Diego, CA, 2003.

[22] Hung, J. T., and Robertazzi, T. G.
Scheduling nonlinear computational loads: Analysis and proof.
Stony Brook University College of Engineering and Applied Science, CEAS Technical Report 823, 2006.

[23] Hung, J. T., and Robertazzi, T. G.
Divisible load cut through switching in sequential tree networks.
IEEE Transactions on Aerospace and Electronic Systems, **40** (2004), 968–982.

[24] Robertazzi, T. G.
Processor equivalence for a linear daisy chain of load sharing processors.
IEEE Transactions on Aerospace and Electronic Systems, **29** (1993), 1216–1221.

[25] Cormen, T. H., Leiserson, C. E., and Rivest, R. L.
Introduction to Algorithms.
New York: McGraw-Hill, 1998.

[26] Bharadwaj, V., and Viswanadham, N.
Suboptimal solutions using integer approximation techniques for scheduling divisible loads on distributed bus networks.
IEEE Transactions on System, Man, and Cybernetics—Part A: Systems and Humans, **30** (2000), 680–691.

[27] Beaumont, O., Casanova, H., Legrand, A., Robert, Y., and Yang, Y.
Scheduling divisible loads on star and tree networks: Results and open problems.
IEEE Transactions on Parallel Distributed Systems, **16** (2005), 207–218.

[28] Beaumont, O., Legrand, A., and Robert, Y.
Scheduling divisible workloads on heterogeneous platforms.
Parallel Computing, **29** (2003), 1121–1132.

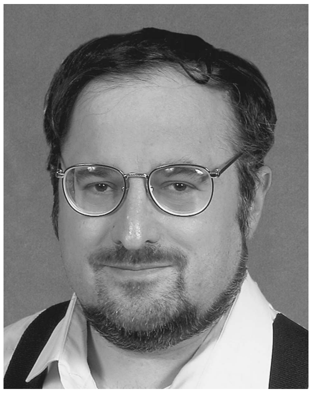
[29] Drozdowski, M., Lawenda, M., and Guinand, F.
Scheduling multiple divisible loads.
The International Journal of High Performance Computing Applications, **20** (2006), 19–30.

[30] Drozdowski, M., and Wolniewicz, P.
Optimum divisible load scheduling on heterogeneous stars with limited memory.
European Journal of Operational Research, **172** (2006), 545–559.

[31] Drozdowski, M., and Lawenda, M.
The combinatorics in divisible load scheduling.
Foundations of Computing and Decision Sciences, **30** (2005), 297–308.

Jui Tsun Hung received the B.S. and M.S. degrees in mechanical engineering from National Sun Yat-Sen University and Chuang Yuan Christian University, Taiwan, in 1986 and 1991. He also received the M.S. and Ph.D. degrees in electrical and computer engineering from the State University of New York at Stony Brook, NY, in 2001 and 2003, respectively.

He was a lecturer in the Wu Feng Institute of Technology and Commerce, Taiwan. He also joined Golden Circuit Electronics Corporation as an engineer for managing the fabrication of printed circuit board, Wintek Corporation as an engineer for designing liquid crystal display drivers, and Memes Technology as a senior engineer in IC design for digital audio broadcast systems and 802.11a/b/g radios. Since 2006, he has been a visiting scholar in Computer Science, Stony Brook University, NY. His current research interests are in radio technology, wireless communications, computer networks, and scheduling. Recently he works in software defined radio (gnu radio) for signals from meteors and for cognitive radio networks, handover strategies for mobile multiaccess ambient networks, and TCP retransmission dynamics analysis.



Thomas G. Robertazzi (S'75—M'77—SM'91—F'06) received the Ph.D. from Princeton University, Princeton, NJ, in 1981 and the B.E.E. from the Cooper Union, New York, NY, in 1977.

He is presently a professor in the Department of Electrical and Computer Engineering at Stony Brook University, Stony Brook, NY. In supervising a very active research group, he has published extensively in the areas of parallel processing and grid scheduling, ad hoc radio networks, telecommunications network planning, ATM switching, queueing and Petri networks. He has also authored, coauthored or edited five books in the areas of networking, performance evaluation, scheduling and network planning. For eleven years Professor Robertazzi has been the Faculty Director of the Stony Brook Living Learning Center in Science and Engineering.