Equal Allocation Scheduling for Data Intensive Applications

KWANGIL KO
THOMAS G. ROBERTAZZI, Senior Member, IEEE
Stony Brook University

A new analytical model for equal allocation of divisible computation and communication load is developed. Equal allocation of load is attractive in multiple processor systems when real time information on processor and link capacity that is necessary for optimal scheduling is not available. The model includes a detailed accounting of solution reporting time. Equal allocation scheduling is compared with sequential scheduling and a new type of multi-installment scheduling. Aerospace applications include the processing of satellite imagery, radar, and sensor networks.

Manuscript received September 16, 2003; revised September 16, 2004; released for publication February 26, 2004.

IEEE Log No. T-AES/40/2/831392.

Refereeing of this contribution was handled by M. G. Simoes.

This work was supported by the National Science Foundation through Grant CCR-99-12331.

Auhors' address: Dept. of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794, E-mail: (tom@ece.sunysb.edu).

I. INTRODUCTION

The combination of the cost decrease and performance improvement in both computers and data storage devices has led to new data intensive applications in the aerospace field. Examples include processing satellite imagery, radar, and sensor networks. It is becoming more common to conceive and implement systems processing on the order of a petabyte (i.e., 10^{15} bytes) of data a year. A useful tool for modeling and evaluating the performance of such data parallel applications is divisible load scheduling theory.

In a divisible load scheduling model, load is assumed to be completely partitionable (divisible) in terms of both computation and communication. Model parameters include processor and link speed(s), and computation and communication intensity. A specific model is also characterized by the parallel processor interconnection topology, scheduling policy, and load distribution assumptions. Divisible load scheduling analysis makes use of linear and continuous variable mathematics to produce a tractable model. Typically one seeks to solve a particular model for the optimal allocation of load, optimal speedup, and optimal solution time. The study of divisible load models began in 1988 with papers by Cheng and Robertazzi [11] and Agrawal and Jagadish [2]. There are now tutorials [17], surveys [5], and a monograph [6] on this subject.

An alternative to optimal scheduling for divisible loads, equal allocation of load to processors, is considered here. That is, we consider a policy where each of N processors receives $1/N$ of the load. There are two practical reasons for considering such a policy. One reason is that optimal policies require real time knowledge of available processor and link capacities. If a system is not instrumented to provide such information, which is not a trivial undertaking, then equal allocation scheduling may be a reasonable policy. This is particularly true of clusters of homogeneous (i.e., identical) computers, an environment that often arises in practice. A second reason to study equal allocation scheduling is to determine how much finish time or speedup is degraded compared with optimal scheduling policies.

A typical aerospace application of equal allocation scheduling is processing a stream of still images from a satellite. A cluster of computers on the ground may be used to scan each of many images for significant features. Naturally, to the granularity level of a single image, the load is divisible and may be assigned either in equal division style or optimally among the computers. Optimal allocation of load is advantageous when the cluster is heterogeneous; there is significant communication delay in transporting the load to processors and/or asymmetry in the load distribution, when a finite batch of jobs is processed and when, most importantly, available processor/link effort information is accessible in real time. Equal allocation scheduling is advantageous when these conditions are reversed, particularly when the cluster is homogeneous and real time information on available processor/link effort is not accessible.

The target architecture models developed here are for a heterogeneous L-level K-ary tree topology. Using a tree topology is quite generic as any arbitrary interconnection topology can be spanned by a tree. Thus spanning distribution trees can be used to distribute or receive load in meshes, hypercubes, torii, and other popular interconnection topologies (this is not to say that detailed studies of specific architectures are not of interest [9, 12]). Also if $L = 1$ and all link speeds are equal, the tree reduces to a bus architecture. While the processors and links considered here are heterogeneous, a symmetrical tree is considered as a baseline. The results can be extended to nonsymmetrical trees of particular interest, though.

The paper presents the first published closed-form results for speedup for a multilevel tree network under equal allocation scheduling. This is compared with optimal single installment and a novel multiple installment scheduling. We find improvements in speedup under optimal scheduling versus equal allocation scheduling of as large as 70%.

In terms of related work, divisible load models involving single installment load distribution for trees were first considered in 1988 by Agrawal and Jagadish using a linear programming approach [2] and in 1990 by Cheng and Robertazzi [10] using an algebraic approach. Agrawal and Jagadish presented numerical (though not analytical) results for equal allocation scheduling, which they referred to as "naive" scheduling. Load distribution sequencing in trees is discussed by Kim, Jee, and Lee in [14] and by Bharadwaj, Ghose, and Mani in [8]. Load distribution in trees is also studied by Barlas in [4]. Note that Cheng and Robertazzi considered equal allocation scheduling for linear daisy chains in [11]. The use of multiple installments of load distribution in tree networks was first examined by Ghose, Mani, and Bharadwaj in 1995 [7] and by Casanova and Yang [19] in 2003. Asymptotic results for large trees using the single installment policy by Ghose and Mani appeared in 1994 [13] and were also published by Bataineh and Robertazzi in 1997 [3]. Asymptotic multi-installment results appear in [7]. Finally, the concept of an equivalent processor, used here was introduced in Robertazzi [16]. A proof that optimal load allocation can be found by forcing all processors to stop computing at the same instant is presented in [18]. The superiority of processors returning solutions in the same order that work is received was demonstrated in 1988 by Agrawal and Jagadish [2] and, using a different method, by Adler, Gong, and Rosenberg in 2003 [1].

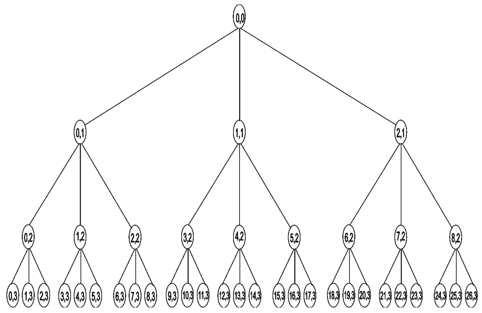


Fig. 1. 3-level 3-ary tree network.

This paper is organized as follows. The system model is presented in Section II. Equal allocation scheduling, sequential optimal scheduling, and multi-installment scheduling are modeled in Sections III, IV, and V, respectively. Numerical results appear in Section VI. The conclusion is in Section VII.

II. SYSTEM MODEL OF L-LEVEL K-ARY TREE NETWORK

A heterogeneous L-level K-ary tree network of communicating processors is considered. For an example, a 3-level, 3-ary tree network is shown in Fig. 1. It is a 3-level tree as the first (root) level is level 0. Each processor is labeled in terms of indexes from left to right and level to level. Here $p_{i,j}$ is the ith processor at the jth level. Processor 0 at level 0 is assumed to be the originating (root) processor which sequentially distributes the fractions of the entire load to K processors. All processors in the Lth level are terminal nodes and other processors each have K children processors. Nodes which have children processors distribute load sequentially to their children.

It is assumed that communication speeds are high enough relative to computation speeds that eliminating subtrees does not result in a speedup improvement [6].

There exists K^j processors at the jth level for $j = 0,1,2,\ldots,L$. Thus, the model of an L-level K-ary tree network consists of $\sum_{j=0}^{L} K^j$ processors. Without loss of generality, it is assumed that the load is instantaneously available at processor 0 at time 0. Each processor is interfaced with the network via a front-end communication processor for communication off-loading. That is, the processors can communicate and compute at the same time.

It is important for $p_{i,j}$ to know the index of its parent processor since $p_{i,j}$ receives load fractions from its parent processor. Naturally, the parent processor of $p_{i,j}$ is located at the $(j-1)$th level just above the jth level. The integer part of i/K indicates the order (index) of parent processor as all processors at the $(j-1)$th level have K children processors. Thus, the parent processor of any processor, $p_{i,j}$ for $j = 1,2,\ldots,L$ and $i = 0,1,\ldots,K^j - 1$ is $p_{\text{int}(i/K),j-1}$. Let

$P_P(p_{i,j})$ be the parent processor of $p_{i,j}$

$$P_P(p_{i,j}) = p_{\text{int}(i/K),j-1}. \tag{1}$$

Here, int(·) is the rounding down to the nearest integer. Further the grandparent processor of $p_{i,j}$ for any $j = 1,2,\ldots,L$ and $i = 0,1,\ldots,K^j - 1$ is $P_P[P_P(p_{i,j})]$

$$\begin{aligned} P_P[P_P(p_{i,j})] &= P_P(p_{\text{int}(i/K),j-1}) \\ &= P_{\text{int}(\text{int}(i/K)/K),j-2} \\ &= P_{\text{int}(i/K^2),j-2}. \end{aligned} \tag{2}$$

Generally, the ancestor processor of $p_{i,j}$ at the lth level ($l < j$) is $(j - l)$ levels above the jth level. In a manner similar to (2), the ancestor processor of $p_{i,j}$ at the lth level defined as $A^l_P(P_{i,j})$ is expressed as follows:

$$A^l_P(P_{i,j}) = p_{\text{int}(i/K^{j-l}),l}. \tag{3}$$

This expression allows one to identify the ancestor processors of any processor $p_{i,j}$, and perceive which processor distributes load fractions to it.

Alternately, a processor $p_{i,j}$ for any $j = 0,1,\ldots,L-1$ and $i = 0,1,\ldots,K^j - 1$ has K children processors labeled $p_{iK+m,j+1}$ for $m = 0,1,\ldots,K-1$.

The following notation are used throughout this work.

$p_{i,j}$: The ith processor at the jth level.

$\alpha_{i,j}$: The fraction of the entire processing load that is assigned to the ith processor at the jth level.

$w_{i,j}$: A constant inversely proportional to the computation speed of the ith processor at the jth level (see Fig 1).

$z_{i,j}$: A constant inversely proportional to the channel speed of the ith link at the jth level.

T_{cp}: Computing intensity constant. The entire load is processed in $w_{i,j}T_{\text{cp}}$ seconds by the ith processor at the jth level.

T_{cm}: Communication intensity constant. The entire load can be transmitted in $z_{i,j}T_{\text{cm}}$ seconds over the ith link at the jth level.

$T^{\text{sol}}_{\text{cm}}$: Solution reporting communication intensity constant. The entire solution report can be transmitted in $z_{i,j}T^{\text{sol}}_{\text{cm}}$ seconds over the ith link at the jth level.

Note that T_{cp} and T_{cm} are properties of the load. As computation and communication intensities, respectively, they affect the relative duration of computing and communication in the scheduling process.

III. EQUAL ALLOCATION SCHEDULING

A multilevel tree is considered where load is distributed from the root to the children in a store and forward mode of operation and "solutions" are transmitted back to the root.

Each processor transmits load fractions to its children processors in sequence. That is, each processor transmits all the load that its left child (and its children) will require, then it does the same for the next (to the right) child and so on. Each processor, that is not a terminal node, repeats this load distribution policy. Thus, although load originates at the root, as load distribution proceeds, multiple nodes in the tree will be concurrently distributing load. In equal division load scheduling, each processor keeps the same fraction of the total load for processing. An L-level K-ary tree network has $\sum_{j=0}^{L} K^j$ processors. Let ε be the fraction assigned to any processor. Consequently the fraction of normalized load for each processor is obtained from the inverse of the total number of processors

$$\varepsilon = \frac{1}{\sum_{j=0}^{L} K^j}. \tag{4}$$

The root processor at level 0 keeps ε, a fraction of the total processing load, for itself to compute and divides and distributes the remaining load to its children processors at the next level. The processors at this level perform the same operation with the load they receive. This process continues until the processors located at the terminal nodes of the tree are assigned their share of the processing load.

Our goal is to find expressions for the solution (finish) time and speedup for the system described under equal division scheduling. Towards this end, the following subsection shows how to calculate a communication delay for each processor. Each processor starts to process its load fraction as soon as it receives its load share and its descendants' load shares completely.

A. Communication Delay for Processor $p_{i,j}$ to Receive its Load Fraction from Root Processor

Communication delay is divided into three parts; one is the time delay incurred by the parent processor, the second is the time delay incurred by the previous brother processors at the same level (which are children of the parent node), and the third is the time taken for $p_{i,j}$ to receive its load fraction and load fractions for descendant processors. The time at which the processor $p_{i,j}$ finishes receiving its load fraction is defined as $C_d(p_{i,j})$. Assume that the parent processor distributes load fractions to its children processor starting from the left to the right

$$C_d(p_{i,j}) = t_r(p_{i,j}) + t_i(p_{i,j}) + t_p(p_{i,j}). \tag{5}$$

Here, in a different order, $t_r(p_{i,j})$, $t_i(p_{i,j})$, and $t_p(p_{i,j})$ are the times taken to receive load fractions over the link to $p_{i,j}$, the time delay incurred by the prior brothers processors, and the time delay incurred by the parent processor of $p_{i,j}$, respectively.

Fig. 2 illustrates equal division scheduling. In the diagram communication time appears above each horizontal time axis and computation time appears below each horizontal time axis. In Fig. 2, the third row is for $p_{i,j}$. Referring to the figure for $p_{i,j}$, prior to the start of computation for this processor it receives load from its parent, immediately after the start of its computation it distributes load to its descendants. The first receiving brother processor is located on the second axis. Note that from the figure, solutions are reported back up the tree to the root in the same order that load is distributed in. This assumption is also made for the two optimal techniques appearing later in the paper.

The difference between the instant that the first brother receiving processor begins to receive and the instant that $p_{i,j}$ begins to receive is $t_i(p_{i,j})$.

When $p_{i,j}$ finishes receiving load fractions, that time instant indicates the communication delay $C_d(p_{i,j})$ for $p_{i,j}$. This is the same as $t_p(p_{iK+m,j+1})$, the time delay incurred by the processor ($p_{i,j}$) which is the parent of its children processors ($p_{iK+m,j+1}$) for $m = 0, 1, \ldots, K-1$.

The root processor can process its load fraction while distributing the remaining load to its children processors. Thus there is no communication delay time for the root processor.

Next we develop expressions for the three components of $C_d(p_{i,j})$.

1) *Receiving Time Delay,* $t_r(p_{i,j})$: Each processor at the same level has the same number of children and grandchildren processors. Processors at any level except the Lth level have K children processors, K^2 grandchildren processors, and so on. This is summed to the Lth level. The number of descendent processors for $p_{i,j}$ is defined as $N_D(p_{i,j})$

$$N_D(p_{i,j}) = \sum_{m'=j+1}^{L} K^{m'-j} \tag{6}$$

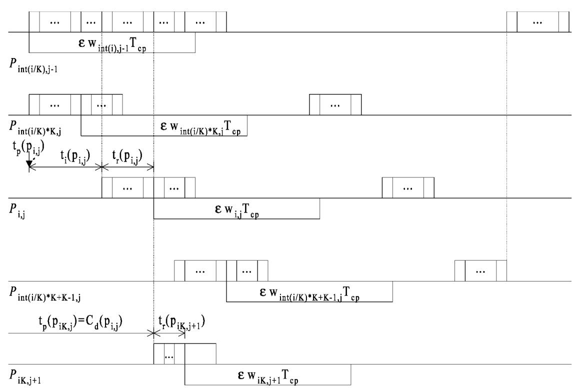
$$= \sum_{m=1}^{L-j} K^m \tag{7}$$

$$= \frac{K - K^{L-j+1}}{1-K}. \tag{8}$$

The processor $p_{i,j}$ receives $[1 + N_D(p_{i,j})]$ fractions for itself and for its descendent processors from its parent processor and then sends $N_D(p_{i,j})$ fractions to its descendent processors. Because $p_{i,j}$ receives $[1 + N_D(p_{i,j})]$ fractions from its parent processor, $t_r(p_{i,j})$ is expressed as follows:

$$t_r(p_{i,j}) = \varepsilon[1 + N_D(p_{i,j})] z_{i,j} T_{\text{cm}} \tag{9}$$

$$= \varepsilon \left[1 + \sum_{m=1}^{L-j} K^m \right] z_{i,j} T_{\text{cm}} \tag{10}$$



$\varepsilon w_{\text{int}(i),j-1} T_{cp}$
$P_{\text{int}(i/K),j-1}$
$P_{\text{int}(i/K)*K,j}$
$\varepsilon w_{\text{int}(i/K)*K,j} T_{cp}$
$t_p(p_{i,j})$
$t_i(p_{i,j})$
$t_r(p_{i,j})$
$P_{i,j}$
$\varepsilon w_{i,j} T_{cp}$
$P_{\text{int}(i/K)*K+K-1,j}$
$\varepsilon w_{\text{int}(i/K)*K+K-1,j} T_{cp}$
$t_p(p_{iK,j}) = C_d(p_{i,j})$
$t_r(p_{iK,j+1})$
$P_{iK,j+1}$
$\varepsilon w_{iK,j+1} T_{cp}$

Fig. 2. Timing diagram of equal division scheduling.

$$= \varepsilon \sum_{m=0}^{L-j} K^m \cdot z_{i,j} T_{\text{cm}} \tag{11}$$

$$= \frac{1 - K^{L-j+1}}{1-K} \varepsilon z_{i,j} T_{\text{cm}}. \tag{12}$$

Here, ε is the size of a fraction. The receiving time delay from the parent processor of $p_{i,j}$ depends on the level i, and only $z_{i,j}$, the inverse link speed connected to $p_{i,j}$.

2) *Time Delay of the Prior Brothers Processors Located at the Same Level,* t_i: Now, if $p_{i,j}$ does not receive first on its level from its parent processor, $p_{i,j}$ should wait while its prior receiving brother processors at the same level receive load fractions from their parent processor. The remainder after dividing i by K decides the receiving order (position) of $p_{i,j}$. The processor $p_{i,j}$ is the first receiving order processor at the jth level when $\text{mod}(i/K)$ is zero. Here $\text{mod}(i/K)$ is the remainder after dividing i by K. Thus, the time delay due to the prior receiving processors can be expressed as follows:

$$t_i(p_{i,j}) = \sum_{n=0}^{\text{mod}(i/K)-1} t_r(p_{\text{inx}(n,i),j}). \tag{13}$$

Here $\text{inx}(n,i)$ is used to find the nth receiving processor's index of brother processors of $p_{i,j}$. The indexes of processors at the jth level are sequentially written starting from the first receiving child processor of $p_{0,j-1}$. From (1), the index of $P_P(p_{i,j})$, is $\text{int}(i/K)$. Thus, the index of the children processors of $P_P(p_{i,j})$ starts from $\text{int}(i/K) \cdot K$. Furthermore, the nth receiving processor index among brother processors of $p_{i,j}$ is obtained adding n and $\text{int}(i/K) \cdot K$. For $n = 0, 1, \ldots, \text{mod}(i/K) - 1$,

$$\text{inx}(n,i) = n + \text{int}(i/K) \cdot K. \tag{14}$$

Applying (12) to (13), the time delay by the prior brother processors is obtained as follows:

$$t_i(p_{i,j}) = \varepsilon \sum_{n=0}^{\text{mod}(i/K)-1} \left[\sum_{m=0}^{L-j} K^m \cdot z_{\text{inx}(n,i),j} T_{\text{cm}} \right] \tag{15}$$

$$= \varepsilon \sum_{m=0}^{L-j} K^m \sum_{n=0}^{\mathrm{mod}(i/K)-1} z_{\mathrm{inx}(n,i),j} T_{\mathrm{cm}} \tag{16}$$

$$= \varepsilon \frac{1-K^{L-j+1}}{1-K} \left[\sum_{n=0}^{\mathrm{mod}(i/K)-1} z_{\mathrm{inx}(n,i),j} T_{\mathrm{cm}} \right]. \tag{17}$$

The processors at the same level have the same number of load fractions as in (8) since they have the same number of descendant processors. The bracket in the above equation is the sum of the load distribution delays to the prior brother processors over their links.

3) *Time delay by the Parent Processor of $p_{i,j}$, t_p*: Each child processor has a time delay caused by waiting for its parent processor. This time delay $t_p(p_{i,j})$ equals the total communication delay of the parent processor of $p_{i,j}$

$$t_p(p_{i,j}) = C_d(A_P^{j-1}(p_{i,j})). \tag{18}$$

Here, $A_P^{j-1}(p_{i,j})$ is the parent processor of $p_{i,j}$. Using (5) and (18), the following recursive equations can be obtained

$$t_p(p_{i,j}) = C_d(A_P^{j-1}(p_{i,j})) \tag{19a}$$

$$C_d(A_P^{j-1}(p_{i,j})) = t_p(A_P^{j-1}(p_{i,j})) + t_i(A_P^{j-1}(p_{i,j})) + t_r(A_P^{j-1}(p_{i,j})) \tag{19b}$$

$$t_p(A_P^{j-1}(p_{i,j})) = C_d(A_P^{j-2}(p_{i,j})) \tag{19c}$$

$$C_d(A_P^{j-2}(p_{i,j})) = t_p(A_P^{j-2}(p_{i,j})) + t_i(A_P^{j-2}(p_{i,j})) + t_r(A_P^{j-2}(p_{i,j})) \tag{19d}$$

$$\vdots$$

$$t_p(A_P^{2}(p_{i,j})) = C_d(A_P^{1}(p_{i,j})) \tag{19e}$$

$$C_d(A_P^{1}(p_{i,j})) = t_p(A_P^{1}(p_{i,j})) + t_i(A_P^{1}(p_{i,j})) + t_r(A_P^{1}(p_{i,j})) \tag{19f}$$

$$t_p(A_P^{1}(p_{i,j})) = C_d(A_P^{0}(p_{i,j})). \tag{19g}$$

Summing both sides of the above recursive equations, $t_p(p_{i,j})$ can be rewritten as follows:

$$t_p(p_{i,j}) = C_d(A_P^0(p_{i,j})) + \sum_{l=1}^{j-1} t_r(A_P^l(p_{i,j})) + \sum_{l=1}^{j-1} t_i(A_P^l(p_{i,j})) \tag{20}$$

$$= \sum_{l=1}^{j-1} t_r(A_P^l(p_{i,j})) + \sum_{l=1}^{j-1} t_i(A_P^l(p_{i,j})). \tag{21}$$

Note that the root processor has no communication delay. Thus $C_d(A_P^0(p_{i,j}))$ is zero.

Substituting (21) into (5), communication delay for $p_{i,j}$, $C_d(p_{i,j})$, is expressed as follows:

$$C_d(p_{i,j}) = t_r(p_{i,j}) + t_i(p_{i,j}) \tag{22}$$

$$+ \sum_{l=1}^{j-1} t_r(A_P^l(p_{i,j})) + \sum_{l=1}^{j-1} t_i(A_P^l(p_{i,j})). \tag{23}$$

Note that $A_P^j(p_{i,j}) = p_{i,j}$.

$$C_d(p_{i,j}) = \sum_{l=1}^{j} [t_r(A_P^l(p_{i,j})) + t_i(A_P^l(p_{i,j}))]. \tag{24}$$

Equation (3) is applied to (12),

$$t_r(A_P^l(p_{i,j})) = t_r(p_{\mathrm{int}(i/K^{j-l}),l}) = \varepsilon \frac{1-K^{L-l+1}}{1-K} z_{\mathrm{int}(i/K^{j-l}),l} T_{\mathrm{cm}} \tag{25}$$

and (3) is applied to (17):

$$t_i(A_P^l(p_{i,j})) = t_i(p_{\mathrm{int}(i/K^{j-l}),l}) = \varepsilon \frac{1-K^{L-l+1}}{1-K} \sum_{n=0}^{\mathrm{mod}[\mathrm{int}(i/K^{j-l})/K]-1} z_{\mathrm{inx}[n,\mathrm{int}(i/K^{j-l})],l} T_{\mathrm{cm}}. \tag{26}$$

In the above equation, if $n = \mathrm{mod}[\mathrm{int}(i/K^{j-l})/K]$, then $\mathrm{inx}[n, \mathrm{int}(i/K^{j-l})] = \mathrm{int}(i/K^{j-l})$. From (14):

$$\mathrm{inx}\left\{ \mathrm{mod}\left[\frac{\mathrm{int}\left(\frac{i}{K^{j-l}}\right)}{K}\right], \mathrm{int}\left(\frac{i}{K^{j-l}}\right) \right\} = \mathrm{mod}\left[\frac{\mathrm{int}\left(\frac{i}{K^{j-l}}\right)}{K}\right] + \mathrm{int}\left(\frac{\mathrm{int}\left(\frac{i}{K^{j-l}}\right)}{K}\right) \cdot K \tag{27}$$

$$= \mathrm{int}\left(\frac{i}{K^{j-l}}\right). \tag{28}$$

Thus, the summation of $t_r(A_P^l(p_{i,j}))$ and $t_i(A_P^l(p_{i,j}))$ is mentioned

$$t_r(A_P^l(p_{i,j})) + t_i(A_P^l(p_{i,j})) = \varepsilon \frac{1-K^{L-l+1}}{1-K} \sum_{n=0}^{\mathrm{mod}[\mathrm{int}(i/K^{j-l})/K]} z_{\mathrm{inx}[n,\mathrm{int}(i/K^{j-l})],l} T_{\mathrm{cm}}. \tag{29}$$

Now (29) is substituted into (24)

$$C_d(p_{i,j}) = \sum_{l=1}^{j} \left[\varepsilon \frac{1-K^{L-l+1}}{1-K} \sum_{n=0}^{\mathrm{mod}[\mathrm{int}(i/K^{j-l})/K]} z_{\mathrm{inx}[n,\mathrm{int}(i/K^{j-l})],l} T_{\mathrm{cm}} \right]. \tag{30}$$

In the special case of homogeneous link speeds ($z_{i,j} = z$)

$$C_d(p_{i,j}) = \sum_{l=1}^{j}\left[\varepsilon\frac{1-K^{L-l+1}}{1-K}\sum_{n=0}^{\mathrm{mod}[\mathrm{int}(i/K^{j-l})/K]} zT_{\mathrm{cm}}\right] \tag{31}$$

$$= \sum_{l=1}^{j}\left[\varepsilon\frac{1-K^{L-l+1}}{1-K}\left[1+\mathrm{mod}\left[\frac{\mathrm{int}\left(\frac{i}{K^{j-l}}\right)}{K}\right]\right]\cdot zT_{\mathrm{cm}}\right]. \tag{32}$$

B. Closed Form of Finish Time and Speedup

For equal allocation scheduling, even neglecting for the moment solution reporting time, since load is simply equally divided among processors different processors may finish computing at different times. This is in contrast to optimal scheduling (again without considering solution reporting time) where one constrains all of the processors to cease computing at the same time instant in order to achieve an optimal solution. With the inclusion of solution reporting time optimal schedules are more coordinated and more tightly "packed," which partly explains their lower values of finish (solution) time compared with equal allocation scheduling.

The last receiving processor is $p_{K^L-1,L}$ in the L-level K-ary tree network. As soon as this node finishes processing its load fraction, the node reports its solution. The scheduling process is finished when the solution of $p_{K^L-1,L}$ is delivered to the originating processor.

First the time delay for $p_{K^L-1,L}$ to report solution is considered. It takes $\varepsilon\cdot z_{K^L-1,L}T_{\mathrm{cm}}^{\mathrm{sol}}$ to transmit the solution of $p_{K^L-1,L}$ from this node at level L to $p_{K^{L-1}-1,L-1}$, its parent processor at level $L-1$. The parent processor of $p_{K^L-1,L}$ collects the solutions of K children processors and transmits $(1+K)$ solutions including its own solution to the ancestor processor at level $L-1$. This procedure keeps until the originating processor receives all solutions. Let S_d be the time delay for $p_{K^L-1,L}$ to report its solution to the originating processor

$$\frac{S_d}{\varepsilon T_{\mathrm{cm}}^{\mathrm{sol}}} = 1\cdot z_{K^L-1,L} + (1+K)z_{K^{L-1}-1,L-1} + (1+K+K^2)z_{K^{L-2}-1,L-2} + \cdots + (1+K+K^2+\cdots+K^{L-1})z_{K-1,1}.$$

The above equation can be condensed as follows:

$$S_d = \varepsilon\sum_{m=0}^{L-1}\sum_{n=0}^{m}K^n\cdot z_{K^{L-m}-1,L-m}T_{\mathrm{cm}}^{\mathrm{sol}} \tag{33}$$

$$= \varepsilon\sum_{m=0}^{L-1}\frac{1-K^{m+1}}{1-K}\cdot z_{K^{L-m}-1,L-m}T_{\mathrm{cm}}^{\mathrm{sol}}. \tag{34}$$

For homogeneous network speeds:

$$S_d = \frac{\varepsilon zT_{\mathrm{cm}}^{\mathrm{sol}}}{1-K}\cdot\left[L-K\frac{1-K^L}{1-K}\right] \tag{35}$$

$$= \frac{\varepsilon zT_{\mathrm{cm}}^{\mathrm{sol}}}{K-1}\cdot\left[\frac{K^{L+1}-K}{K-1}-L\right]. \tag{36}$$

The finish (solution) time for equal allocation scheduling is thus obtained as follows:

$$T_f^{\mathrm{EAS}}(L,K) = C_d(p_{L,K^L}) + \varepsilon w_{K^L-1,L}T_{\mathrm{cp}} + S_d. \tag{37}$$

The first term is the communication delay for $p_{K^L-1,L}$, the second term is the computation time for $p_{K^L-1,L}$, and the third term is the reporting time.

Note that the speedup is

$$S^{\mathrm{EAS}}(L,K) = \frac{wT_{\mathrm{cp}}}{T_f^{\mathrm{EAS}}(L,K)}. \tag{38}$$

Speedup is the ratio of solution time on one processor to solution time on N processors. It is thus a measure of parallel processing advantage.

IV. SEQUENTIAL OPTIMAL SCHEDULING

In sequential optimal scheduling, each processor that is not a terminal node distributes load to each child (once) in turn from left to right. The single transmission of load to a child includes all loads that child's descendants will need. Thus the "sequencing" is similar to equal allocation scheduling except that the size of load fractions will now be determined optimally. In sequential optimal scheduling, solution reporting times are staggered in a subtree of an L-level K-ary tree network. Children processors finish reporting their solution while a parent processor is processing.

In this section, the closed forms of the finish time and speedup for an L-level K-ary tree network is considered. The scheduling discussed here includes solution reporting time, unlike previous work. It is assumed that the solution reporting order (of processors) is the same as the order of load distribution. The procedure to obtain the finish time for an L-level K-ary tree network can be expanded to a general tree network.

The technique used here, established in [10], is to calculate the multilevel tree finish time (and speedup) by finding an equivalent processor that exactly represents the multilevel tree operating characteristics. This is done by finding equivalent processors for each single level subtree (one subroot with K children) starting from the bottom of the tree and proceeding recursively upwards. At level j the processor that replaces processor i and its descendants has equivalent inverse processing speed $w_{i,j}^{\mathrm{eq}}$. When the recursive process finishes, one has the equivalent inverse speed of the root processor $w^{\text{eq}}_{0,0}$ which is the same as the overall multilevel tree. Note that [10] does not consider solution reporting time, as the model in the work presented here does.

To find $w^{\text{eq}}_{i,j}$ at the jth level for the ith processor, X_m is defined as follows:

$$X_m = \frac{w^{\text{eq}}_{iK+m+1,j+1}T_{\text{cp}} + z_{iK+m+1,j+1}T_{\text{cm}}}{w^{\text{eq}}_{iK+m,j+1}T_{\text{cp}} + z_{iK+m,j+1}T^{\text{sol}}_{\text{cm}}}. \qquad (39)$$

Then $\alpha_{i,j}$, the fraction of load assigned to the ith processor at the jth level, is obtained as follows:

$$\alpha_{i,j} = \frac{\theta_{i,j}\cdot\left(\sum_{n=0}^{K-1}\left(\prod_{m=n}^{K-1}X_m\right)\frac{z_{iK+n,j+1}T_{\text{cm}}}{w_{i,j}T_{\text{cp}}} + \frac{w^{\text{eq}}_{(i+1)K-1,j+1}T_{\text{cp}}}{w_{i,j}T_{\text{cp}}} + \frac{z_{(i+1)K-1,j+1}T^{\text{sol}}_{\text{cm}}}{w_{i,j}T_{\text{cp}}}\right)}{\sum_{n=0}^{K-1}\left(\prod_{m=n}^{K-1}X_m\right)\left(1+\frac{z_{iK+n,j+1}T_{\text{cm}}}{w_{i,j}T_{\text{cp}}}\right) + \frac{w^{\text{eq}}_{(i+1)K-1,j+1}T_{\text{cp}}}{w_{i,j}T_{\text{cp}}} + \frac{z_{(i+1)K-1,j+1}T^{\text{sol}}_{\text{cm}}}{w_{i,j}T_{\text{cp}}}}. \qquad (40)$$

Now (39) is substituted into the above equation. The equivalent processor speed can be obtained as follows:

$$w^{\text{eq}}_{i,j} = \alpha_{i,j}\cdot w_{i,j} \qquad (41)$$

$$= w_{i,j}A/B. \qquad (42)$$

Here $0 \le j \le K-1$.

$$A = \sum_{n=0}^{K-1}\left(\prod_{m=n}^{K-1}\frac{w^{\text{eq}}_{iK+m+1,j+1}T_{\text{cp}} + z_{iK+m+1,j+1}T_{\text{cm}}}{w^{\text{eq}}_{iK+m,j+1}T_{\text{cp}} + z_{iK+m,j+1}T^{\text{sol}}_{\text{cm}}}\right)\frac{z_{iK+n,j+1}T_{\text{cm}}}{w_{i,j}T_{\text{cp}}} + \frac{w^{\text{eq}}_{(i+1)K-1,j+1}T_{\text{cp}}}{w_{i,j}T_{\text{cp}}} + \frac{z_{(i+1)K-1,j+1}T^{\text{sol}}_{\text{cm}}}{w_{i,j}T_{\text{cp}}} \qquad (43)$$

$$B = \sum_{n=0}^{K-1}\left(\prod_{m=n}^{K-1}\frac{w^{\text{eq}}_{iK+m+1,j+1}T_{\text{cp}} + z_{iK+m+1,j+1}T_{\text{cm}}}{w^{\text{eq}}_{iK+m,j+1}T_{\text{cp}} + z_{iK+m,j+1}T^{\text{sol}}_{\text{cm}}}\right)\left(1+\frac{zT_{\text{cm}}}{w_{i,j}T_{\text{cp}}}\right) + \frac{w^{\text{eq}}_{(i+1)K-1,j+1}T_{\text{cp}}}{w_{i,j}T_{\text{cp}}} + \frac{zT^{\text{sol}}_{\text{cm}}}{w_{i,j}T_{\text{cp}}}. \qquad (44)$$

In the above equation, $w^{\text{eq}}_{i,j}$ is expressed using its original processor speed, $w_{i,j}$ and the equivalent children processor speeds, $w^{\text{eq}}_{iK+m,j+1}$ for $m = 0,1,\ldots,K-1$. Then the finish time is

$$T^{\text{SOS}}_f(L,K) = w^{\text{eq}}_{0,0}T_{\text{cp}}. \qquad (45)$$

Also, the speedup is

$$S^{\text{SOS}}(L,K) = \frac{w}{w^{\text{eq}}_{0,0}}. \qquad (46)$$

A complete derivation of this result appears in [15].

V. MULTI-INSTALLMENT OPTIMAL SCHEDULING

In the equal allocation and sequential distribution of the previous sections, a child processor receives load fractions at the same time for itself and for processors at the next level. This causes the processors at each level to have long idle time. In this section, a processor at the jth level doesn't distribute all load at once to each descendent processors but instead distributes load in turns (installments) to its descendent processors. The first version of multi-installment optimal scheduling was developed originally by Bharadwaj, Ghose, and Mani [6, 7] as a way to reduce solution time by modifying the load distribution policy. In Bharadwaj, et al., partial load is delivered in several installments (rounds) to each processor to minimize idle time.

A somewhat different approach is taken here for the first time, distributing load in complete integral units to each individual processor but in "installments" to the processors in the tree as a whole. That is, each node including the root distributes load to each of (only) K processors in turn during each set of installments. During each succeeding installment load is distributed in integral units for another K processors. The process repeats until all of the tree's processors have received load. The amount of load to allocate to each processor is determined optimally in the context of this scheduling policy.

This scheduling strategy is best illustrated by way of example. Referring to Fig. 1 let, again, i be the children number and j be the level number for processor $p_{i,j}$. In a 3-level 3-ary tree network, for instance, the root processor $p_{0,0}$ distributes fractions to children processor in the sequence of $p_{0,1}$, $p_{1,1}$, $p_{2,1}$. As soon as each processor at the first level receives its load fraction, it begins to process. Again $p_{0,0}$ distributes load fractions to $p_{0,1}$, $p_{1,1}$, and $p_{2,1}$ in sequence. As $p_{0,1}$, $p_{1,1}$, and $p_{2,1}$ already received their load fractions, these processors can redistribute load to their children processors. That is as soon as they receive fractions, $p_{0,1}$, $p_{1,1}$, and $p_{2,1}$ distribute load fractions to $p_{0,2}$, $p_{3,2}$, and $p_{6,2}$, respectively. After that, additional load fractions are distributed to $p_{0,1}$, $p_{1,1}$, and $p_{2,1}$. As $p_{0,2}$, $p_{3,2}$, and $p_{6,2}$ already received their load fractions, this time, $p_{0,1}$, $p_{1,1}$, and $p_{2,1}$ distribute load fractions to $p_{1,2}$, $p_{4,2}$, and $p_{7,2}$, respectively.

The receiving order at the second level is $p_{0,2}$, $p_{3,2}$, $p_{6,2}$, then $p_{1,2}$, $p_{4,2}$, $p_{7,2}$, then $p_{2,2}$, $p_{5,2}$, and $p_{8,2}$. After each processor at the second level receives its load fraction, it begins to distribute the load fractions received from its parent processor. This procedure continues until the terminal processors receive their fraction.

As in the previous optimal strategy, solution reporting order is the same as the order in which load is distributed.

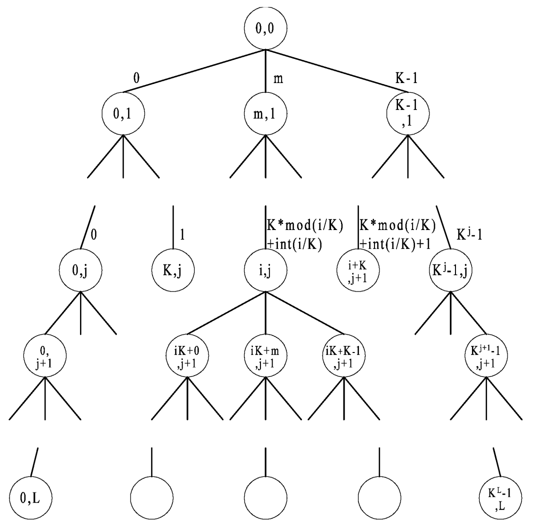


Fig. 3. L-level K-ary tree network.

This strategy shuffles the index i in $p_{i,j}$. In Fig. 3, the number beside the link indicates the distribution sequence at the same level. Now, the actual sequence of load distribution at the jth level of our type of multi-installment scheduling as described above can be calculated with a "processor identification number," the index i in $p_{i,j}$

$$p_{i,j} = p'_{K\cdot\mathrm{mod}(i/K)+\mathrm{int}(i/K),j} \tag{47}$$

or

$$p'_{m,j} = p_{K\cdot\mathrm{mod}(m/K)+\mathrm{int}(m/K),j}. \tag{48}$$

Thus $p_{i,j}$ is the $(K\cdot\mathrm{mod}(i/K)+\mathrm{int}(i/K))$th receiving processor at the jth level. Let $p'_{m,j}$ be the mth receiving processor at the jth level. Furthermore, $\alpha'_{m,j}$, $w'_{m,j}$, and $z'_{m,j}$ are relative to $p'_{m,j}$. The prime variable is written in terms of the actual sequence of load distribution to account for the load distribution shuffling of processor identification.

The goal in the following is to find expressions for the finish time and speedup of this optimal multi-installment load distribution policy for the described multilevel tree network. To accomplish this, one sets up linear timing equations, as is usually done in the literature [6], for this scheduling policy. To achieve a solution with optimal finish time all of the processors should stop computing at the same instant (intuitively otherwise load could be transferred between processors to improve the solution [6, 18]). One uses this fact to algebraically solve for the optimal fraction of load to assign to each processor. Note that only the final result is presented here, a complete derivation appears in [15].

Let:

$$X_{m,j} = \frac{\alpha'_{m,j}}{\alpha'_{m+1,j}} = \frac{w'_{m+1,j}T_{\mathrm{cp}} + z'_{m+1,j}T_{\mathrm{cm}}}{w'_{m,j}T_{\mathrm{cp}} + z'_{m,j}T^{\mathrm{sol}}_{\mathrm{cm}}}. \tag{49}$$

Then $\alpha'_{n,j}$ for $0 \le n \le K^j - 1$ can be rewritten in terms of $\alpha'_{0,j}$ as

$$\alpha'_{n,j} = \prod_{m=0}^{n-1} X^{-1}_{m,j}\alpha'_{0,j}. \tag{50}$$

Also, let

$$Y_j = \frac{\alpha'_{0,j}}{\alpha'_{0,j+1}} \tag{51}$$

$$= \left[z'_{0,j}T_{\mathrm{cm}} + \sum_{n=0}^{K^{j+1}-1}\left(\prod_{m=0}^{n-1} X^{-1}_{m,j+1}\right) z'_{n,j+1}T_{\mathrm{cm}} + \prod_{m=0}^{K^{j+1}-2} X^{-1}_{m,j+1} \cdot (w'_{K^{j+1}-1,j+1}T_{\mathrm{cp}} + z'_{K^{j+1}-1,j+1}T^{\mathrm{sol}}_{\mathrm{cm}} + z'_{K^j-1,j}T^{\mathrm{sol}}_{\mathrm{cm}})\right] \div \left[w'_{0,j}T_{\mathrm{cp}} - \sum_{n=1}^{K^j-1}\left(\prod_{m=0}^{n-1} X^{-1}_{m,j}\right) z'_{n,j}T_{\mathrm{cm}}\right]. \tag{52}$$

Now the processing time of the root processor, and hence system finish time, can be found as follows.

Let

$$Y_0 = \frac{\sum_{n=0}^{K-1}\left(\prod_{m=0}^{n-1} X^{-1}_{m,1}\right) z'_{n,1}T_{\mathrm{cm}} + \left(\prod_{m=0}^{K-2} X^{-1}_{m,1}\right)(w'_{K-1,1}T_{\mathrm{cp}} + z'_{K-1,1}T^{\mathrm{sol}}_{\mathrm{cm}})}{w_{0,0}T_{\mathrm{cp}}}. \tag{53}$$

From the above equations, $\alpha'_{0,l}$ for $l = 1,2,\ldots,L-1$ can be expressed in terms of $\alpha_{0,0}$

$$\alpha'_{0,l} = \prod_{j=0}^{l-1} Y^{-1}_j \cdot \alpha_{0,0}. \tag{54}$$

Furthermore $\alpha'_{n,l}$, the optimal load fraction for the nth processor at the lth level, can be expressed in terms of $\alpha'_{0,l}$ by substituting (54) into (50)

$$\alpha'_{n,l} = \left(\prod_{m=0}^{n-1} X^{-1}_{m,l}\right) \cdot \left(\prod_{j=0}^{l-1} Y^{-1}_j\right) \cdot \alpha_{0,0}. \tag{55}$$

The normalization equation is

$$\alpha'_{0,0} + \sum_{l=1}^{L}\sum_{n=0}^{K^l-1} \alpha'_{n,l} = 1. \tag{56}$$

Equation (55) is substituted into the above equation. Then α_0 can be obtained as follows:

$$\alpha'_{0,0} = \left[1 + \sum_{l=1}^{L}\sum_{n=0}^{K^l-1}\left(\prod_{m=0}^{n-1} X^{-1}_{m,l}\right) \cdot \left(\prod_{j=0}^{l-1} Y^{-1}_j\right)\right]^{-1}. \tag{57}$$

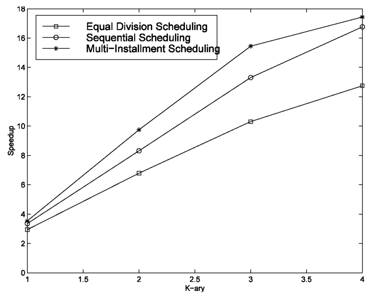


Fig. 4. Speedup versus K and L. $L = 3$, $w_i = 1$, $z_i = 0.05$, $T_{\mathrm{cp}} = 1$, $T_{\mathrm{cm}} = 1$, $T_{\mathrm{cm}}^{\mathrm{sol}} = 0.2$.

TABLE I
Speedup Improvement in Percentage. Equal Allocation Scheduling Versus Sequential Optimal Scheduling

	$K = 1$	$K = 2$	$K = 3$	$K = 4$
$L = 1$	3.0000	4.8657	6.6068	8.2321
$L = 2$	7.7911	13.2612	18.8595	23.6171
$L = 3$	14.0182	22.4653	29.0953	31.5415
$L = 4$	21.3361	29.3861	32.7270	26.7681

Note: $w_i = 1$, $z_i = 0.05$, $T_{\mathrm{cp}} = 1$, $T_{\mathrm{cm}} = 1$, $T_{\mathrm{cm}}^{\mathrm{sol}} = 0.2$.

Since the root processor processes load during the entire scheduling process, the finish (solution) time is

$$T_f^{\mathrm{MOS}}(L,K) = \alpha'_{0,0} w T_{\mathrm{cp}} \tag{58}$$

$$= \frac{wT_{\mathrm{cp}}}{1 + \sum_{l=1}^{L}\sum_{n=0}^{K^l-1}\left(\prod_{m=0}^{n-1} X_{m,l}^{-1}\right)\cdot\left(\prod_{j=0}^{l-1} Y_j^{-1}\right)}. \tag{59}$$

Also the speedup is

$$S^{\mathrm{MOS}} = \frac{1}{\alpha_{0,0}}. \tag{60}$$

VI. NUMERICAL RESULTS

As mentioned, speedup, for a computational problem, is the ratio of solution time on one processor to solution time on N processors. It is thus a measure of parallel processing advantage.

Representative values of speedup versus the K and L for equal allocation scheduling, sequential optimal scheduling and multi-installment optimal scheduling appears in Fig. 4. Note that if, for instance, $L = 1$ and $K = 3$, there is one root and three children.

Sequential scheduling and the multi-installment scheduling are compared with equal allocation scheduling in Tables I and II. The speedup improvement in the tables is obtained as follows. Let the speedup improvement measures be

$$I_S\big|_{\text{Table I}} = \frac{S^{\mathrm{SOS}}(L,K) - S^{\mathrm{EAS}}(L,K)}{S^{\mathrm{EAS}}(L,K)} \times 100\ [\%] \tag{61}$$

TABLE II
Speedup Improvement in Percentage. Equal Allocation Scheduling Versus Multi-Installment Optimal Scheduling

	$K = 1$	$K = 2$	$K = 3$	$K = 4$
$L = 1$	3.0000	4.8657	6.6068	8.2321
$L = 2$	9.5714	18.6093	26.8686	32.4850
$L = 3$	20.2847	43.5657	49.8413	36.7902
$L = 4$	34.6968	70.2669	42.7284	20.8299

Note: $w_i = 1$, $z_i = 0.05$, $T_{\mathrm{cp}} = 1$, $T_{\mathrm{cm}} = 1$, $T_{\mathrm{cm}}^{\mathrm{sol}} = 0.2$.

$$I_S\big|_{\text{Table II}} = \frac{S^{\mathrm{MOS}}(L,K) - S^{\mathrm{EAS}}(L,K)}{S^{\mathrm{EAS}}(L,K)} \times 100\ [\%]. \tag{62}$$

Here, $S^{\mathrm{EDS}}(L,K)$, $S^{\mathrm{SOS}}(L,K)$, and $S^{\mathrm{MOS}}(L,K)$ are the speedups for equal allocation scheduling, sequential scheduling, and multi-installment scheduling, respectively. The speedups are defined as follows

$$S^{\mathrm{EAS}}(L,K) = \frac{wT_{\mathrm{cp}}}{T_f^{\mathrm{EAS}}(L,K)} \tag{63}$$

$$S^{\mathrm{SOS}}(L,K) = \frac{wT_{\mathrm{cp}}}{T_f^{\mathrm{SOS}}(L,K)} \tag{64}$$

$$S^{\mathrm{MOS}}(L,K) = \frac{wT_{\mathrm{cp}}}{T_f^{\mathrm{MOS}}(L,K)}. \tag{65}$$

Five or six digits accuracy is shown in the tables, not because real scheduling is that precise, but to aid in result replication.

Comparing the multi-installment scheduling with the sequential scheduling, the multi-installment strategy has the higher speedup. It is expected that speedup for all three strategies will saturate for large K or L. All three strategies are ultimately limited by communication delays across levels of the tree and by the assumed sequential distribution to each child in each subtree.

In the tables it can be seen that speedup improvements of optimal scheduling over equal allocation scheduling of from 3% to 70% were found for the tree topology. As L and K are increased the speedup improvement first increases then may decrease for certain parameter combinations.

It is interesting to ask over what range of parameter values is the speedup improvement most pronounced. For a homogeneous network, if a job is computation intensive one would expect $1/N$ equal allocation scheduling to be optimal. If a job is communication intensive, the use of a single processor may well be optimal. It is that range where computation intensity is on the order of communication intensity that one can expect optimal scheduling to be most efficacious.

VII. CONCLUSION

This is the first published analysis of equal allocation scheduling for multilevel tree networks. Equal allocation scheduling is a pragmatic choice for situations where processor and link effort are not known in real time and where networks are homogeneous. Results here, which agree with the experience of the senior author, is that the degradation in performance compared with optimal scheduling is often less than a factor of two, which would be adequate and cost effective for certain situations. We believe that the relative performance ordering noted here of the three scheduling policies will carry over to other topologies though the numerical amount of improvement will, of course, differ. Useful future work would include determining performance degradation bounds, both across possible parameter values for a particular topology and across different topologies.

REFERENCES

[1] Adler, M., Gong, Y., and Rosenberg, A. L. (2003)
Optimal sharing of bags of tasks in heterogeneous clusters.
In *Proceedings of SPAA'03*, 2003.

[2] Agrawal, R., and Jagadish, H. V. (1988)
Partitioning techniques for large-grained parallelism.
IEEE Transactions on Computers, **37** (1988), 1627–1634.

[3] Bataineh, S., and Robertazzi, T. G. (1997)
Performance limits for processor networks with divisible jobs.
IEEE Transactions on Aerospace and Electronic Systems, **33** (1997), 1189–1198.

[4] Barlas, G. D. (1998)
Collection-aware optimum sequencing of operations and closed-from solutions for the distribution of divisible load on arbitrary processor trees.
IEEE Transactions on Parallel and Distributed Systems, **9** (1998), 929–941.

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

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

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

[8] Bharadwaj, V., Ghose, D., and Mani, V. (1994)
Optimal sequencing and arrangement in distributed single-level tree networks with communication delays.
IEEE Transactions on Parallel and Distributed Systems, **5** (1994), 968–976.

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

[10] Cheng, Y-C., and Robertazzi, T. G. (1990)
Distributed computation for a tree network with communication delays.
IEEE Transactions on Aerospace and Electronic Systems, **26** (1990), 511–516.

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

[12] Drozdowksi, M., and Glazek, W. (1999)
Scheduling a divisible load in a three-dimensional mesh of processors.
Parallel Computing, **25** (1999), 381–404.

[13] Ghose, D., and Mani, V. (1994)
Distributed computation with communication delays: Asymptotic performance analysis.
Journal of Parallel and Distributed Computing, **23** (1994), 293–305.

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

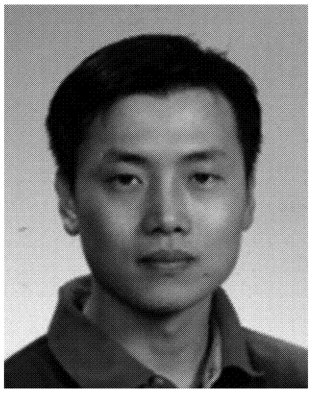
[15] Ko, K., and Robertazzi, T. G. (2003)
Naive versus optimal scheduling for data intensive applications.
Technical Report 808, Stony Brook University College of Engineering and Applied Science, 2003.

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

[17] Robertazzi, T. G. (2003)
Ten reasons to use divisible load theory.
Computer, **36** (2003), 63–68.

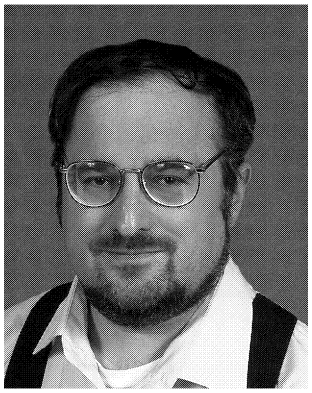
[18] Sohn, J., and Robertazzi, T. G. (1996)
Optimal load sharing for a divisible job on a bus network.
IEEE Transactions on Aerospace and Electronic Systems, **32** (1996), 34–40.

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



Kwangil Ko received the M.S. and Ph.D. degrees in 1996 and 2000 from Stony Brook University, Stony Brook, NY.

He is currently with Samsung Electronics, Suwon, Korea, where he is working as a traffic engineer on radio access networks. His research interests include the performance measurement of networks, scheduling algorithms, flow control, QoS, and resource management.



Thomas G. Robertazzi (S'75—M'77—SM'91) 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 Dept. of Electrical and Computer Engineering at Stony Brook University, Stony Brook, NY. In supervising a very active research area, he has published extensively in the areas of parallel processor and grid scheduling, ad hoc radio networks, telecommunications network planning, ATM switching, queueing, and Petri networks.

Dr. Robertazzi authored, coauthored, or edited four books in the areas of performance evaluation, scheduling, and network planning.