Scheduling Divisible Workloads from Multiple Sources in Linear Daisy Chain Networks

Xinxin Liu, Han Zhao, and Xiaolin Li
Scalable Software Systems Laboratory
Department of Computer Science
Oklahoma State University
Stillwater, OK 74078, USA
Email: {xinxin, haz, xiaolin}@cs.okstate.edu

***Abstract*— This paper considers scheduling divisible workloads from multiple sources in linear networks of processors. We propose a two phase scheduling strategy (TPSS) to minimize the overall processing time of these workloads by taking advantage of the processor equivalence technique. A case study with two sources of workloads is presented to illustrate the general approach for multiple sources of workloads. In the first phase, using processor equivalence, we derive recursive equations to obtain near-optimal workload distribution for all processors and the minimum processing time of the overall workloads. In the second phase, we propose an efficient algorithm to obtain near-optimal load distribution among processors represented by the equivalent processor. Experimental evaluation through simulations demonstrate performance improvement using our schemes compared to the equal partition scheme.**

Keywords: Linear Daisy Chain, Divisible Load Theory, Processor Equivalence

I. Introduction

During the past decade, divisible load scheduling and distribution techniques have been widely studied forming the divisible load theory (DLT). The load is defined to be "divisible" for computation and communication if it can be partitioned arbitrarily among processors. DLT offers a tractable and realistic approach to scheduling and allows integrated modeling of computation and communication in parallel and distributed computing systems [1]. DLT has been successfully applied to a wide spectrum of applications and systems, including parallel and distributed image processing [2], [3], sensor networks [4], computational grids [5], and bioinformatics [6], just to name a few. Intensive research efforts have been made to study scheduling problems with various constrains on different interconnection topologies [7], [8], [9]. However, most DLT researches focused on the single source load distribution because the complexity of optimal scheduling increases exponentially with the increase of load sources. The concept of processor equivalence was first proposed in [10]. This inspires us to adopt this model to tackle multi-source workload scheduling problems. In this paper, we present a heuristic to solve the multi-source divisible load scheduling in a linear daisy chain network of homogeneous processors. The proposed approach can be readily extended to solve the more general scheduling problem with multiple sources in heterogeneous networks. Our approach can be summarized as two phases. First, it collapses the processors from the left source to the right source into one equivalent processor to derive the optimal load distribution with other processors. Second, an effective algorithm is proposed to solve the local distribution within the collapsed processors.

Our contributions in this work are two folds: (1) We propose to use the processor equivalence technique to study the multi-source divisible load scheduling problem; (2) We propose an efficient algorithm to assign near-optimal workloads among processors in the collapsed equivalent processor cluster. Further, the experimental evaluation is presented to demonstrate the efficiency of the our strategy.

A. Related Work

Single Source Load Distribution. With single workload, previous research efforts have successfully solved the optimal scheduling problem and obtained closed-form solutions on different network topologies, including trees [11], hypercubes [12] and meshes [8]. Furthermore, practical problems have been studied: scheduling on networks with communication startup costs [13], with buffer constraints [14], and bioinformatics [6]. For very large workload, processing time can be further reduced using multiple installment load distribution [15]. A novel

multi-round algorithm called UMR was developed considering latencies and heterogeneous platforms in [16].

Multiple sources Load Distribution. Compared to studies for the case of single-source load distribution, the multi-source load distribution problem has not received sufficient attention until recent years. Ko and Robertazzi combined Markovian queueing theory and divisible load scheduling theory on multi-source load distribution in 2002 [17]. One year later, multi-source grid scheduling with capacity constraints was studied in [18].

This paper investigates the multi-source load scheduling problem using the processor equivalence technique. This work provides a new vision to address multi-source load scheduling problems.

The rest of the paper is organized as follows. Problem description and notations are in Section II. The specific solution is illustrated in Section III, first applied to the five processors case then extended to n processors. In Section IV simulation results are presented. Section V concludes the paper.

II. Problem description

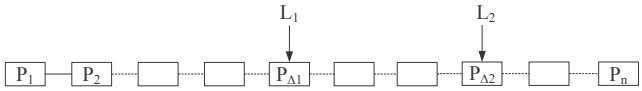

Fig. 1. Two load sources in a N processors network

We consider a linear daisy chain of n processors, as shown in Fig. 1. The n processors are referred to as P_1, P_2,...,P_n. Two divisible jobs originate from any two of these processors. The sizes of the workloads are denoted as L_1 and L_2. All the processors are assumed to have front-end, which means communication and computation can be performed simultaneously. Further, we assume that the network environment is homogeneous, that is, all the processors have the same computation capacity, and the link speeds between any two processors are same. our objective is to develop scheduling strategies on how to partition and schedule the workloads amongst these processors to minimize the overall processing time.

We describe the load scheduling process as follows. When divisible loads L_1 and L_2 arrived at any two processors, they are divided into n load fractions, and distributed among all processors. Upon receiving their respective load fractions, processors start processing their own workload and relay all the other workload fractions to the next neighbor processor at the same time.According to the proved optimal criterion in DLT, all the processors involved in the computation of the load must stop computing at the same time instant [2].

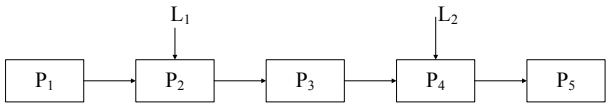

Fig. 2. A linear daisy chain with five processors and two load originators

Conceptually, to minimize the overall processing time, all processors should stop computing at the same time; otherwise, remaining workload could be transferred from busy processors to the idle ones. We will follow this optimal criterion in this paper.

The following notations and definitions are used throughout the paper:

- P_i: the i-th processor, where i is from 1 to N.
- P_{Δ_1}: the processor that originate work load L_1
- P_{Δ_2}: the processor that originate work load L_2
- P_{eq}: the equivalent processor of all the processors between processors Δ_1 and Δ_2, where the workload originated.
- L_i: the i-th workload. Particularly, i equals to 1 or 2 in our analysis, but it can be extended to m later on.
- a_i: the mount of workload that is assigned to the i-th processor.
- ω_i: a constant that is inversely proportional to the processing speed of processor i in the chain.
- z_i: a constant that is inversely proportional to the speed of the i-th link.
- T_i: finish time for the i-th processor.
- T_{finish}: the finish time for the whole system, actually it will be equal to T_i.
- T_{cp}: normalized solution time when ω_i=1.
- T_{cm}: normalized transmission time when z_i=1.
- ρ : the ratio of communication speed zT_{cm} over computation ωT_{cp}.
- ΣL_i = 1

III. Load Scheduling Strategies

We present our two phase scheduling strategy (TPSS) in this section. To ease the presentation, we first describe a simplified case with only five processors and two load originators in the system. We then further extend it into a general n processors system with two load originators. The TPSS strategy can also be extended to the n processors and m workloads situation using the equivalent processor technique.

A. Two Load Sources in a Five Processor System

First we consider a special case to describe our basic scheduling strategies, assuming that we have five proces-

sors in a linear network and the two jobs are originated at the second and the fourth processor respectively, i.e., $\Delta_1 = 2$ and $\Delta_2 = 4$, as shown in Fig. 2.

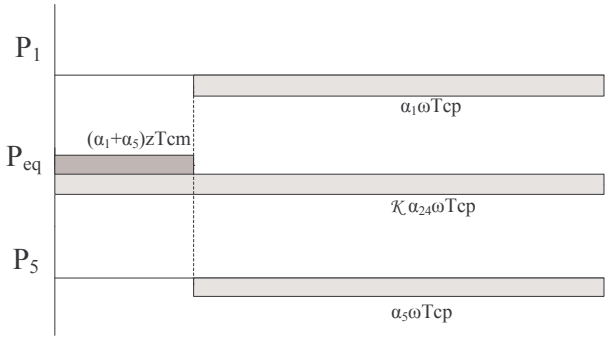

Fig. 3. Five processors timing diagram

*1) **Global Stage**:* To solve this scheduling problem, we divide it into two phases: first we collapse the three processors P_2, P_3, P_4 into one "big" equivalent processor P_{eq}. The concept of collapsing two or more processors into an equivalent processor was first proposed in [10] with the equivalent processing capability defined by

$$\omega_{eq} = -\frac{zT_{cm}}{2T_{cp}} + \frac{1}{2} * \sqrt{(\frac{zT_{cm}}{T_{cp}})^2 + 4\omega z * \frac{T_{cm}}{T_{cp}}} \quad (1)$$

Here we denote $\omega_{eq} = \kappa * \omega$ to simplify the presentation. After collapsing, the timing diagram for the remaining three processors is shown in Fig. 3. To have more processors to share processing the workloads, it is reasonable to first transfer the loads to the "outer" processors which are on the left and right sides of the equivalent processor so that the marginal processors can start computing loads distributed to them as soon as the workload has arrived in these processors.

Based on the timing diagram, we have the following equations for the finish time of each processor to get the optimal fragment size of the load distribution:

$$T_1 = \alpha_1 * \omega T_{cp} + (\alpha_1 + \alpha_5) * zT_{cm} \quad (2)$$

$$T_{24} = \alpha_{24} * \kappa * \omega T_{cp} \quad (3)$$

$$T_5 = \alpha_5 * \omega T_{cp} + (\alpha_1 + \alpha_5) * zT_{cm} \quad (4)$$

$$\alpha_1 + \alpha_{24} + \alpha_5 = L_1 + L_2 = 1 \quad (5)$$

According to the timing diagram, we have,

$$T_1 = T_{24} = T_5 = T_{finish} \quad (6)$$

Thus, from T_1 = T_5, we get α_1 = α_5. The α_{24} can then be derived,

$$\alpha_{24} = \frac{\omega T_{cp} + zT_{cm}}{\kappa \omega T_{cp}} * \alpha_1 + \alpha_1 * zT_{cm} \quad (7)$$

Now we get the optimal workload fractions distributed to each processor as:

$$\alpha_1 = \frac{\kappa \omega T_{cp}}{(2\kappa + 1)\omega T_{cp} + 2zT_{cm}} \quad (8)$$

$$\alpha_5 = \frac{\kappa \omega T_{cp}}{(2\kappa + 1)\omega T_{cp} + 2zT_{cm}} \quad (9)$$

$$\alpha_{24} = \frac{2zT_{cm} + \omega T_{cp}}{(2\kappa + 1)\omega T_{cp} + 2zT_{cm}} \quad (10)$$

Thus, the overall finish time is obtained,

$$T_{finish} = \frac{2zT_{cm} + \omega T_{cp}}{(2\kappa + 1)\omega T_{cp} + 2zT_{cm}} * \kappa \omega T_{cp} \quad (11)$$

*2) **Local Stage**:* After obtaining the global load distribution, we then investigate how to distribute the load fractions among internal processors in the equivalent processor. The remaining load can be transferred among these "middle" processors; but, it is hard to determine the direction of this kind of communication. As in Fig. 4, the workload can be passed from the leftmost processor to the rightmost one, or from right to the left, or even the two ends will transfer both towards some processor in the middle. All the possible directions can be determined using the factor of workload amount at each end.

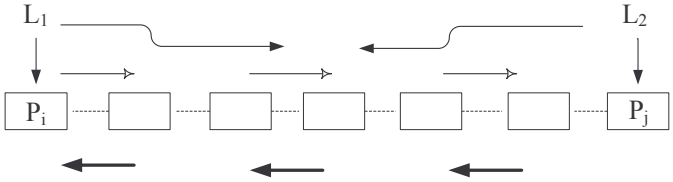

Fig. 4. Possible directions of communication

To address this issue, we propose a local distribution algorithm to transfer loads step by step. First by comparing the remaining loads at left and right end processors, we can determine the direction of load transfer. Now we get the transferring timing diagram for processor P_2– the left end processor of the whole equivalent processor, with global time formulas we can now calculate the optimal load fraction distributed to processor P_2. Next step we can compare the remaining part of left injected loads with loads injected from right to get the optimal loads distributed to the next processor. The steps can be repeated to solve the problem of these processors which are collapsed into one "big" processor in *Global Stage*.

In the five processor network, in the global stage, α_1 and α_5 have been obtained. Now we continue our scheduling by assuming that $L_1 - \alpha_1 \geq L_2 - \alpha_2$, let σ_2 be the workload fraction that P_2 receives from the

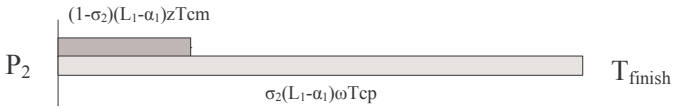

Fig. 5. Timing diagram for P_2

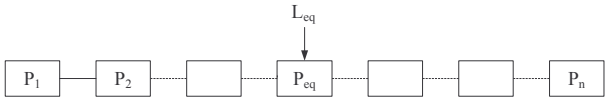

Fig. 6. N processors equivalent case

remaining workload. Fig. 5 shows the timing diagram for P_2.

From the timing diagram, we have,

$$T_{finish} = \sigma_2(L_1 - \alpha_1)\omega T_{cp} \tag{12}$$

therefore,

$$\alpha_2 = \sigma_2(L_1 - \alpha_1) = \frac{T_{finish}}{\omega T_{cp}} \tag{13}$$

The comparison and transfer steps are repeated until all the remaining loads are communicated and processed.

B. Two Load Sources in a N Processor System

1) ***Global Stage****:* By collapsing those processors from P_{Δ_1} to P_{Δ_2} in Fig. 1, we get one equivalent processor P_{eq} with an equivalent computational capability ($\kappa\omega T_{cp}$) as shown in Fig. 6. The resulting timing diagram with the equivalent processor is shown in Fig. 7.

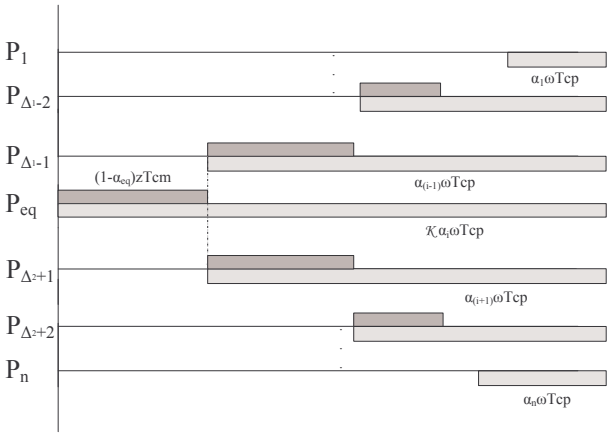

Fig. 7. Timing diagram for a N processor network with an equivalent processor

T_i $(i = 1, 2, 3...n)$ denotes the finish time of each processor and T_{eq} denotes the finish time of the equivalent processor. From the timing diagram in Fig. 7, we obtain the following equations:

$$T_{eq} = \alpha_{eq}\kappa\omega T_{cp} \tag{14}$$

$$T_{\Delta_1-1} = \alpha_{\Delta_1-1}\omega T_{cp} + (\sum_{i=1}^{\Delta_1-1}\alpha_i + \sum_{i=\Delta_2+1}^{n}\alpha_i) * zT_{cm} \tag{15}$$

$$T_{\Delta_2+1} = \alpha_{\Delta_2+1}\omega T_{cp} + (\sum_{i=1}^{\Delta_1-1}\alpha_i + \sum_{i=\Delta_2+1}^{n}\alpha_i) * zT_{cm} \tag{16}$$

$$T_{\Delta_1-2} = \alpha_{\Delta_1-2}\omega T_{cp} + (\sum_{i=1}^{\Delta_1-1}\alpha_i + \sum_{i=\Delta_2+1}^{n}\alpha_i + \sum_{i=1}^{\Delta_1-2}\alpha_i) * zT_{cm} \tag{17}$$

$$T_{\Delta_2+2} = \alpha_{\Delta_2+2}\omega T_{cp} + (\sum_{i=1}^{\Delta_1-1}\alpha_i + \sum_{i=\Delta_2+1}^{n}\alpha_i + \sum_{i=\Delta_2+2}^{n}\alpha_i) * zT_{cm} \tag{18}$$

$$T_{\Delta_1-3} = \alpha_{\Delta_1-3}\omega T_{cp} + (\sum_{i=1}^{\Delta_1-1}\alpha_i + \sum_{i=\Delta_2+1}^{n}\alpha_i + \sum_{i=1}^{\Delta_1-2}\alpha_i + \sum_{i=1}^{\Delta_1-3}\alpha_i) * zT_{cm} \tag{19}$$

$$T_{\Delta_2+3} = \alpha_{\Delta_2+3}\omega T_{cp} + (\sum_{i=1}^{i-1}\alpha_i + \sum_{i=\Delta_2+1}^{n}\alpha_i + \sum_{i=\Delta_2+2}^{n}\alpha_i + \sum_{i=\Delta_2+3}^{n}\alpha_i) * zT_{cm} \tag{20}$$

...

...

$$T_1 = \alpha_1\omega T_{cp} + (\sum_{i=1}^{\Delta_1-1}\alpha_i + \sum_{i=\Delta_2+1}^{n}\alpha_i + ... + \sum_{i=1}^{1}\alpha_i) * zT_{cm} \tag{21}$$

$$T_n = \alpha_n\omega T_{cp} + (\sum_{i=1}^{\Delta_1-1}\alpha_i + \sum_{i=\Delta_2+1}^{n}\alpha_i + ... + \sum_{i=n}^{n}\alpha_i) * zT_{cm} \tag{22}$$

According to the divisible load theory, the optimality principle is that all processors must stop processing at the same time instant. Hence, the following condition holds:

$$T_1 = T_2 = ... = T_n \tag{23}$$

In addition, we have a normalization condition,

$$\sum_{i=1}^{n} \alpha_i = 1 \tag{24}$$

There are 2*($n-\Delta_2+\Delta_1$) equations and 2*($n-\Delta_2+\Delta_1$) unknowns so these equations could be solved.

From T_1 = T_2, we can get

$$\alpha_2 = \frac{\omega T_{cp} + zT_{cm}}{\omega T_{cp}} * \alpha_1 \tag{25}$$

Recall that $\rho = \frac{zT_{cm}}{\omega T_{cp}}$. The equation above can be written as

$$\alpha_2 = (1+\rho) * \alpha_1 \tag{26}$$

From T_3 = T_2, we obtain,

$$\alpha_3 = (1+\rho) * \alpha_2 + \alpha_1 zT_{cm} \tag{27}$$

Finally, we obtain a uniform equation for processors from P_1 to P_{Δ_1-1},

$$\alpha_i = (1+\rho) * \alpha_{i-1} + \sum_{j=1}^{i-2} \alpha_j zT_{cm} \tag{28}$$

Recursively substituting α_i with α_1, we can get the workload fractions from P_1 to P_{Δ_1-1} represented by α_1.

On the other side, starting from P_n back to P_{Δ_2-1}, similarly, we obtain,

$$\alpha_i = (1+\rho) * \alpha_{i+1} + \sum_{j=i+2}^{n} \alpha_j zT_{cm} \tag{29}$$

Especially for P_{n-1} and P_n

$$\alpha_{n-1} = (1+\rho) * \alpha_n \tag{30}$$

$$\alpha_n = \alpha_n \tag{31}$$

Using substitution, these $(n-\Delta_2)$ load fractions from P_{Δ_2+1} to P_n can be represented by α_n.

According to Fig. 7, we have the following equality,

$$\alpha_{eq} * \kappa\omega T_{cp} = \alpha_{\Delta_1-1} * \omega T_{cp} + (\sum_{i=1}^{\Delta_1-1} \alpha_i + \sum_{i=\Delta_2+1}^{n} \alpha_i) * zT_{cm} \tag{32}$$

We also have another equality,

$$\alpha_{eq} * \kappa\omega T_{cp} = \alpha_{\Delta_2+1} * \omega T_{cp} + (\sum_{i=1}^{\Delta_1-1} \alpha_i + \sum_{i=\Delta_2+1}^{n} \alpha_i) * zT_{cm} \tag{33}$$

From (32) and (33), we obtain

$$\alpha_{\Delta_1-1} = \alpha_{\Delta_2+1} \tag{34}$$

This equation along with (28) and (29) allows us to use α_1 to represent all the workload fractions.

TABLE I

TPSS ALGORITHM: LOCAL STAGE

Initialization

S_1: Set includes α_1 to α_{Δ_1-1}
S_2: Set includes α_{Δ_2+1} to α_n
$Index1$: Index number from 1 to $\Delta_1 - 1$
$Index2$: Index number from $\Delta_2 + 1$ to n
S : Set includes all the optimal load from α_1 to α_n
$Direction$: could be $Right$ or $Left$
$temp$: temporary value for current index

Local Distribution

Do {
/* Comparing remaining load*/
If ((L_1-$\sum_{i=1}^{\Delta_1-1}\alpha_i$)≥($L_2$-$\sum_{i=1}^{\Delta_2+1}\alpha_i$))
{
$Direction$ = $Left$;
$temp$ = max($i \in Index1$)+1;
/*Find optimal load for current step*/
/* See computing procedure */
Compute α_{temp};
$\alpha_{temp} \longrightarrow S_1$;
}
Else {
/*Compute from right side*/
$Direction$ = $Right$;
$temp$ = min($i \in Index2$)-1
/* See computing procedure */
Compute α_{temp};
$\alpha_{temp} \longrightarrow S_2$;
}
} While($S_1 \bigcup S_2 \neq S$)

Further, with the normalization condition that the total workload is 1, and all load fractions α_i $(i = 1, 2, 3...n)$ are represented by α_1, we can readily obtain the load distribution solution.

2) ***Local Stage****:* As in the five processor case, the idea in the local stage is to compare the workload to the left side and to the right side at each step. The detailed algorithm in pseudo-code is presented in Table I.

To compute the current workload distribution, the near-optimal finish time calculated in global stage is used, the computing sub-procedure is presented in Table II.

The TPSS algorithm in the local stage is basically iterative. At each step, the algorithm uses the load fractions calculated in previous steps and compute the finish time of the current processor, which consists of both communication and computation time of previous workload. The algorithm terminates when all the middle processors have been assigned workload fractions by the algorithm. To ease the understanding, numerical examples are presented in Section IV.

TABLE II

COMPUTING PROCEDURE

Initializaion

σ_i: optimal fraction of remaining load distributed to $i-th$ processor

t: index variable

BEGIN

If ($Direction == Left$)

{

$t = temp - \Delta_1$;

If($t == 0$){

$T_{finish} = \sum_{i=1}^{\Delta_1-1} \alpha_i z T_{cm} + \sigma_{\Delta_1}(L_1 - \sum_{i=1}^{\Delta_1-1} \alpha_i)\omega T_{cp}$

}

Else /*t≥1*/ {

$T_{finish} = \sum_{i=1}^{\Delta_1-1} \alpha_i T_{cm} + [(1-\sigma_{\Delta_1}) + (1-\sigma_{\Delta_1} - \sigma_{\Delta_1+1}) + ...+(1-\sigma_{\Delta_1}-\sigma_{\Delta_1+1}-...-\sigma_{\Delta_1+t-1})](L_1 - \sum_{i=1}^{\Delta_1-1} \alpha_i) z T_{cm} + \sigma_{\Delta_1+t}(L_1 - \sum_{i=1}^{\Delta_1-1} \alpha_i)\omega T_{cp}$;

$\alpha_{temp} = \alpha_{\Delta_1+t}(L_1 - \sum_{i=1}^{\Delta_1-1} \alpha_i)$;

}

Else { /*direction is right*/

$t = \Delta_2 - temp$

If(t==0)

$T_{finish} = \sum_{i=\Delta_2+1}^{n} \alpha_i z T_{cm} + \sigma_{\Delta_2}(L_2 - \sum_{i=\Delta_2+1}^{n} \alpha_i)\omega T_{cp}$

Else /*t≥1*{

$T_{finish} = \sum_{i=\Delta_2+1}^{n} \alpha_k z T_{cm} + [(1-\sigma_{\Delta_2}) + (1-\sigma_{\Delta_2} - \sigma_{\Delta_2-1}) + ...+(1-\sigma_{\Delta_2}-\sigma_{\Delta_2-1}-...- \sigma_{\Delta_2-t+1})](L_2 - \sum_{i=\Delta_2+1}^{n} \alpha_i) z T_{cm} + \sigma_{\Delta_2-t}(L_2 - \sum_{i=\Delta_2+1}^{n} \alpha_i)\omega T_{cp}$

$\alpha_{temp} = \sigma_{\Delta_2-t}(L_2 - \sum_{i=\Delta_2+1}^{n} \alpha_i)$

} }

END

C. Extend to multiple sources problem

The proposed TPSS scheduling algorithm can be easily extended to the n processors, m sources situation. Suppose we have m workloads to be processed among n processors. We can collapse two processors at a time, and then apply TPSS incrementally. In this way the problem of multiple workload scheduling on n processors can be solved.

IV. SIMULATION RESULTS

In this section, we first show that the load scheduling results can be influenced significantly by different factors. By highlighting all the important issues, we establish three sets of experimental data, and use the TPSS algorithm to obtain the load scheduling solution. Second we compare our approach with the equal partitioning scheme (EPS), which partitions workload equally among all processors. The comparison result demonstrate that the TPSS algorithm outperforms the EPS algorithm as expected.

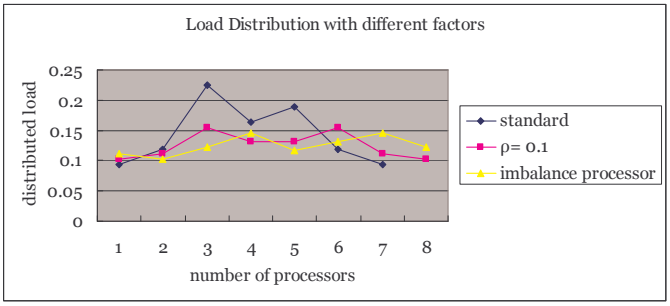

Fig. 8. Numerical Experiment 1

A. Affecting Factors for Load Distribution

The problem of partitioning multiple load sources between n processors can be greatly influenced by various factors, such as the workload difference between load sources, ratio of computation over communication, and workload originators in the system. All these factors make the scheduling problem more complicated than a single workload scheduling. We use three examples to show how different factors can affect the workload distribution. In the first example we consider a linear daisy chain with 7 processors, denoted as "Standard". Assume $L_1 = 0.6$, $\Delta_1 = 3$, $L_2 = 0.4$, $\Delta_2 = 5$, and $\rho = 0.25$. We then use the TPSS scheme to obtain the load distribution. In the second case we change the number of processors to 8, and set $\rho = 0.1$ while keeping all the other factors the same. In the last case, we adjust the system setup as follows: $n = 8$, $L_1 = L_2 = 0.5$, $\Delta_1 = 4$, $\Delta_2 = 7$,and $\rho = 0.1$. The simulation results for these three cases are presented in Fig. 8.

From Fig. 8, we observe that when computational speed is much greater than the communication speed, processors that hold the original workloads tend to not distribute the work to other processors, which results in more unbalanced workload distribution than in other cases. Another observation is that workload distribution is also affected by specific load originating positions.

B. Finish Time Improvement Using TPSS

We further conduct preliminary simulation evaluations to compare TPSS algorithm with the equal partitioning scheme, EPS. The left source load L_1 is 0.6 to the processor P_3 while the right source load L_2 is 0.4 to the processor P_5. The finish time results are shown in Fig. 9 as a function of number of processors using the EPS and the TPSS algorithm. The finish time trend curves are also plotted for $\rho = 0.1$ and $\rho = 0.05$ respectively.

Compared to the EPS, the TPSS algorithm efficiently improves the overall finish time. For example, in the

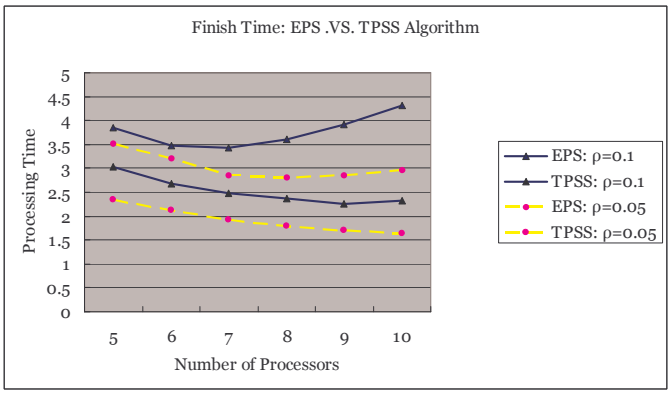

Fig. 9. Numerical Experiment 2

case when $\rho = 0.05$, the TPSS gains 44.86% finish time reduction compared to EPS. Also note that when the computation communication ratio ρ becomes larger, the system shows better scalability due to relatively reduced communication overheads.

V. Conclusion and Future Work

We presented a two phase scheduling strategy (TPSS) to schedule multi-source workloads in linear daisy chain networks. We adopt a processor equivalence technique to simplify the derivation of workload scheduling. TPSS works in two phases: in the global stage, a recursive solution can be used to derive the load assignments outside of the equivalent processor; in the local stage, inside the equivalent processor, the load assignment is further distributed among processors represented by the equivalent processor. Various factors that influence the load distribution are shown in the numerical experiment. Preliminary simulation results further demonstrate that TPSS outperforms a simple equation partition scheme. The proposed two phase strategies can be also applied to solve multi-source load scheduling problems in other network topologies.

References

[1] T. G. Robertazzi, "Ten reasons to use divisible load theory," *Computer*, 2003.

[2] V. Bharadwaj, D. Ghose, and T. G. Robertazzi, "Divisible load theory: A new paradigm for load scheduling in distributed systems," *Cluster Computing*, vol. 6, no. 1, pp. 7–17, 2003.

[3] J. Yao, J. Guo, L. Bhuyan, and Z. Xu, "Scheduling real-time multimedia tasks in network processors," *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, vol. 3, 2004.

[4] M. Moges and T. Robertazzi, "Wireless sensor networks: scheduling for measurement and data reporting," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 42, 2006.

[5] P. Thysebaert, F. D. Turck, B. Dhoedt, and P. Demeester, "Using divisible load theory to dimension optical transport networks for computational grids," in *Optical Fiber Communication Conference, 2005. Technical Digest. OFC/NFOEC*, vol. 2, 2005.

[6] W. H. Min and B. Veeravalli, "Aligning biological sequences on distributed bus networks: a divisible load scheduling approach," *IEEE Transactions on Information Technology in Biomedicine*, vol. 9, no. 4, pp. 489–501, 2005.

[7] H. Kim, G.-I. Jee, and J. Lee, "Optimal load distribution for tree network processors," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 32, no. 2, pp. 607–612, 1996.

[8] M. Drozdowski and W. Glazek, "Scheduling divisible loads in a three-dimensional mesh of processors," *Parallel Computing*, vol. 25, no. 4, pp. 381–404, 1999.

[9] M. Mogas and T. G. Robertazzi, "Optimal divisible load scheduling and markov chain models," 2003.

[10] T. G. Robertazzi, "Processor equivalence for daisy chain load sharing processors," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 29, no. 4, pp. 1216–1221, 1993.

[11] V. Bharadwaj, D. Ghose, and V. Mani, "Optimal sequencing and arrangement in distributed single-level tree networks with communication delays," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 9, pp. 968–976, 1994.

[12] J. Blazewicz and M. Drozdowski, "Scheduling divisible jobs on hypercubes," *Parallel Computing*, vol. 21, pp. 1945–1956, 1995.

[13] ——, "Distributed processing of divisible jobs with communication startup costs," *Discrete Applied Mathematics*, vol. 76, no. 1-3, pp. 21–41, 1997.

[14] X. Li, V. Bharadwaj, and C. Ko, "Divisible load scheduling on single-level tree networks with buffers constraints," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 36, no. 4, pp. 1298–1308, 2000.

[15] V. Bharadwaj, D. Ghose, and V. Mani, "Multi-installment load distribution in tree networks with delays," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 31, no. 2, pp. 555–567, 1995.

[16] Y. Yang and H. Casanova, "Umr: a multi-round algorithm for scheduling divisible workloads," in *Parallel and Distributed Processing Symposium*, 2003.

[17] T. G. Robertazzi and K., "Scheduling in an environment of multiple job submissions," Mar. 2002 2002.

[18] H. M. Wong, D. Yu, B. Veeravalli, and T. G. Robertazzi, "Data intensive grid scheduling: Multiple sources with capacity constraints," Nov. 2003 2003.