Efficient Parallel Video Processing Through Concurrent Communication on a Multi-port Star Network

Taeyoung Lim, *Student Member, IEEE* and Thomas G. Robertazzi, *Fellow, IEEE*

***Abstract*— Communication delay in a processor network is very critical to the throughput for parallel video processing. We propose a simultaneous distribution and collection method (SD) from the root processor to children processors via a multi-port switch network. For the proposed mechanism, we analyze the video encoding time and derive a closed-form solution for a star interconnection network topology. The results show that the total encoding time is significantly faster than the previous method, Parallel Interlaced (PI). In addition, we achieve scalability in terms of the number of processors, which means that as the number of processors increases over the optimal number of processors of PI, one continues to achieve much better performance.**

***Index Terms*— Parallel video scheduling, divisible load theory, concurrent communication, star network.**

I. INTRODUCTION

In parallel video processing, various scheduling algorithms were presented such as PI (parallel interlaced) and PR (parallel recursive) which can assign video frames to multiple processors. For these two algorithms the authors [3,4] found both the maximum processing throughput and I/O utilization, and the optimal number of processors for each of algorithms under a bus architecture, using divisible load analysis [1,2]. However the algorithms have inherent limitations of sequential I/O communication, due to the use of a bus based architecture, on communication in terms of throughput and the optimal number of processors.

In this paper we propose an efficient scheduling mechanism, SD (Simultaneous Distribution), for parallel video processing which distributes raw video loads and collects encoded video results concurrently among the root (control) processor and each child worker processor on a multi-port star topology. Note that simultaneous distribution was proposed by Piriyakumar and Murthy [9] and analyzed by Hung and Robertazzi [10]. We consider two cases: one is that load is assigned to the root processor, the other is it is not assigned. For the two cases, we obtain closed-form solutions for the total video processing time, and then compare them with the performance under the optimal number of processors which is proposed in previous scheduling algorithms, such as PI and PR. Both of the two cases show much better performance in video processing, more than 6 times as much for the parameters we use as those under the optimal number of processors of previous methods, such as PI and PR. In terms of the number of scalable processors, our proposed method, SD, reaches up to 30, more than the optimal number of processors (12) of PI or PR [4].

We know that when the number of processors is small, the factors that affect the total processing time are the method to distribute and collect load as well as the root processor participation in computation and processing speed. As the number of processors increases, all of the SD methods show still better performance than PI and PR methods, because all of SD methods have good scalability. However when the number of processors is 30, the performance improvement of SD-COMP method (SD with computation) is relatively small, just 1.3 times, against the SD-NO method (SD with no computation). When we compare it with the sequential distribution method, PI, the improvement is 6 to 8 times. It means that when the number of processors increases enough to process the whole load, the most critical part is the way to distribute and collect load rather than whether the root processor receives and computes load.

Also of practical interest is that we propose a multi-port star topology among the root (control) processor and children worker processors. This means that the control processor has multiple ports to each of the children processors for I/O communication. One of the reasons to select the multi-port star topology is that there is only communication between the root processor and each of children processors without communication among children processors. The other aspect is that the star topology is cost effective model for parallel video processing and relatively easy to implement compared with other complex architectures, such as 2D meshes, or Hypercube.

This paper is organized as follows. Section II describes a multi-port star network for concurrent communication, system modeling, and mathematical definitions. In section III, our scheduling methods are proposed and analyzed mathematically. In section IV, several comparisons with the previous works are presented, and section V is the conclusion.

Manuscript received Feb 24, 2006.

Taeyoung. Lim is with Dept. of Electrical & Computer Engineering, Stony Brook University, Stony Brook NY 11794 (e-mail: tlim@ece.sunysb.edu)

Thomas. G. Robertazzi., is with Dept. of Electrical & Computer Engineering, Stony Brook University, Stony Brook NY 11794 USA (phone: 631-632-8412/10; fax: 631-632-8494; e-mail: tom@ece.sunysb.edu*).*

II. System Modeling and Mathematical Definitions

A. System Modeling

In this paper, a star network topology is considered, which consists of the one root (control) processor with multiple ports and *m* children processors. The root (control) processor distributes raw video data (load) and collects the encoded video data (results) to/from each child processor concurrently via multiple ports. While the children processors encode the video, the root processor waits for the encoded video data from each child processor.

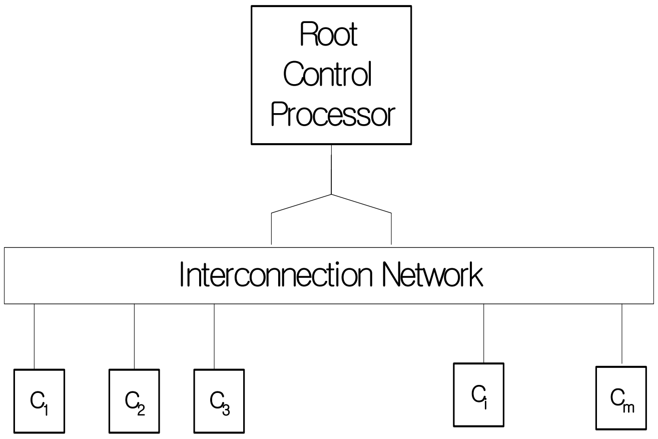

Fig.1. Block diagram for multi-ports star network

We have two scenarios for concurrent scheduling. The first scenario is for the root processor to only distribute and collect load without computation. This is because we try to compare its performance with one in the previous papers [3, 4]. Here we assume that our multi-port star network is homogeneous, which means all of the children processors are identical in terms of the processing speed. In addition, the communication speed between the root processor and each child processor is also identical.

The other case is for the root processor to do both communication, such as loads distribution and results collection, and computation (video encoding). Here we assume that all of the children processors are homogeneous in terms of processing speed and communication speed as in the previous scenario, but the root processor speed can be different from the children's speed. We analyze how much processing power the root processor needs to do both communication and computation and achieve good performance.

B. Mathematical definition

The variables we will use in the following are based on the papers [1, 2, 4].

α_i	the load fraction assigned to the ith link-processor pair
w_i	the inverse of the computing speed of the ith processor
z_i	the inverse of the link speed of the ith link
T_{cp}	computing intensity constant: the entire load is processed in w_iT_{cp} seconds by the ith processor
T_{cm}	communication intensity constant: the entire load can be transmitted in z_iT_{cm} seconds over the ith link
$T_{f,m}$	the finish time. Time at which the last processor of m children processors ceases computation.
$T_{f,0}$	the finish time. Time at which (only) the root processor ceases computation.

In Fig.2, the value of 'k' is defined as the ratio of the result (an encoded video) obtained from each child processor to the load sent (an original raw video). That is,

$$k = \frac{result\_received}{load\_sent}$$

We have the three cases as follows:

- k = 1, if the amount of load sent is same as that of result received.
- k < 1, if the amount of load sent is greater than that of result received. This case is typical in digital video processing.
- k > 1, if the amount of load sent is less than that of result received.

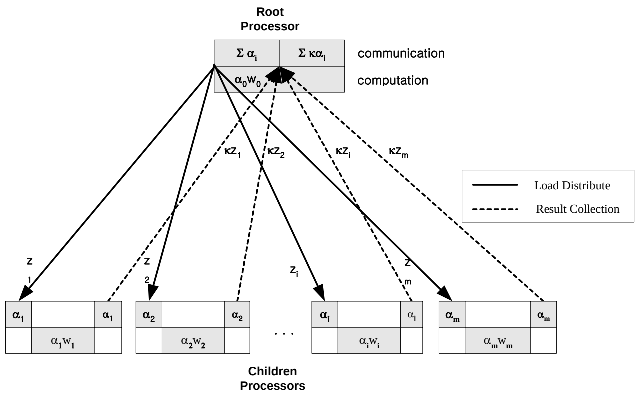

Fig.2. The fraction of load may or may not be assigned to the root processor. If it is assigned, the root processor not only distributes load and collects results to/from each child processor, but also joins computation itself. Otherwise, the root processor just distributes and collects load. Here 'k' is the ratio of result received to original load sent.

Then $\alpha_iw_iT_{cp}$ is the time to process the fraction α_i of the entire load on the ith processor. Note that the units of $\alpha_iw_iT_{cp}$ are [load] x [sec/load] x [dimensionless quantity] = seconds. Likewise, $\alpha_iz_iT_{cm}$ is the time to transmit the fraction α_i of the entire load over the ith link. Our goal is to propose more efficient scheduling methods and analyze the solution in parallel video processing through concurrent communication.

III. Concurrent Load Scheduling Method

A. The load is not assigned to the root processor (SD-NO)

We consider the case of a homogenous processor network, which means all children processors except the root processor are identical; the inverse processor speed is w_i = w and the inverse network speed is z_i = z. The root processor does no computation by itself, and just distribute load and collect results to/from the children processors. The timing diagram for concurrent scheduling is shown in Fig. 3.

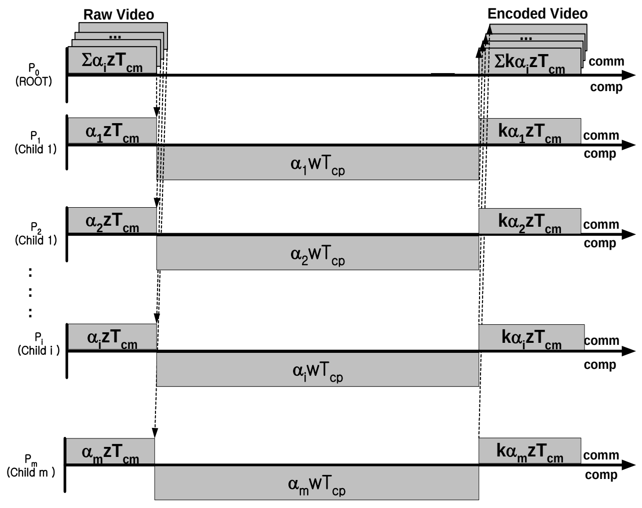

Fig.3 The timing diagram for concurrent load scheduling mechanism without load assigned to the root (control) processor. Here the root processor does not compute in itself, but just distributes and collects load (SD-NO).

From the timing diagram in Fig.3, the equations for SD-NO scheduling are obtained as follows:

$$\alpha_1 zT_{cm} + \alpha_1 wT_{cp} + \alpha_1 kzT_{cm} = \alpha_2 zT_{cm} + \alpha_2 wT_{cp} + \alpha_2 kzT_{cm} \quad (1)$$

$$\alpha_1 = \frac{(z+kz)T_{cm} + wT_{cp}}{(z+kz)T_{cm} + wT_{cp}}\alpha_2 = \alpha_2 \quad (2)$$

From (2), we deduce as follows:

$$\alpha_1 = \alpha_2 = \alpha_3 = \ \ldots = \alpha_m \quad (3)$$

The normalization equation is

$$\sum_{i=1}^{m} \alpha_i \ = 1 \quad (4)$$

From equation (4), we obtain

$$\alpha_1 \times m \ = 1, \quad \alpha_1 = \frac{1}{m} \quad (5)$$

$$\alpha_m \times m \ = 1, \quad \alpha_m = \frac{1}{m} \quad (6)$$

The total processing time, T(m), is achieved as

$$T(m) = \alpha_1 zT_{cm} + \alpha_1 wT_{cp} + \alpha_1 kzT_{cm}$$
$$= \alpha_1(1 + \rho + k\rho)\, wT_{cp} \quad (7)$$

where $\rho = \frac{zT_{cm}}{wT_{cp}}$

From (5), the above equation, the total processing time for the entire load can be rewritten as follows:

$$T(m) = \frac{(1+\rho+k\rho)}{m} wT_{cp} \quad (8)$$

Our finding is that the total processing time decreases linearly as the number of children processors increases.

B. The load is assigned to the root processor (SD-COMP)

In case load is assigned to the root processor itself, we assume the root processor has more processing power than that of the children processors, while all of the children are identical in terms of processing power and link speed. We define the inverse computing speed of children processors as $w_1 = w_2 = \ldots = w_m = w$, and the inverse link speed of children processors as $z_1 = z_2 = \ldots = z_m = z$. As for the root processor, the processor speed is greater than those of children processors, which means the inverse value of the root processor, w_0, is less than 'w'. The link speed of the root processor is identical to children's speed, so the inverse value, z_0, is equal to 'z', since they all are connected to the same network.

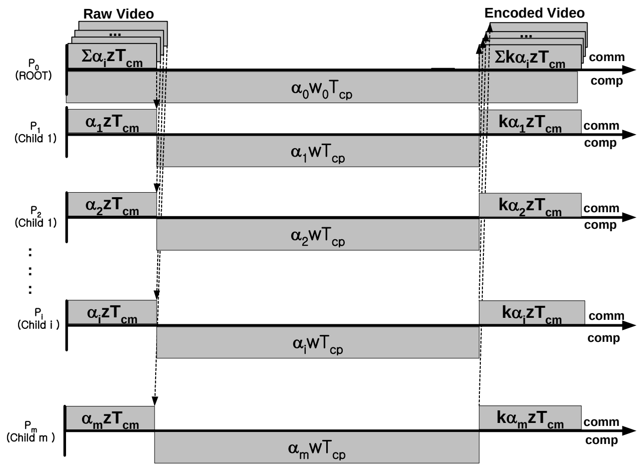

Fig.4 The timing diagram for concurrent load scheduling with load assigned to the root processor. The root (control) processor computes load assigned to itself as well as distributes and collects load (SD-COMP).

From the timing diagram in Fig.4, the equations for SD-COMP scheduling method, in which the root processor has load assigned to compute, are obtained as follows:

$$\alpha_0 w_0 T_{cp} = \alpha_1 z_1 T_{cm} + \alpha_1 w_1 T_{cp} + \alpha_1 kz_1 T_{cm} \quad (9)$$

$$\alpha_1 z_1 T_{cm} + \alpha_1 w_1 T_{cp} + \alpha_1 kz_1 T_{cm} = \alpha_2 z_2 T_{cm} + \alpha_2 w_2 T_{cp} + \alpha_2 kz_2 T_{cm} \quad (10)$$

…

$$\alpha_{m-1} z_{m-1} T_{cm} + \alpha_{m-1} w_{m-1} T_{cp} + \alpha_{m-1} kz_{m-1} T_{cm} = \alpha_m z_m T_{cm} + \alpha_m w_m T_{cp} + \alpha_m kz_m T_{cm} \quad (11)$$

The normalization equation is

$$\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 + \ \ldots + \alpha_m \ = 1 \quad (12)$$

From equation (9),

$$\alpha_o = \frac{[(z_1 + kz_1)T_{cm} + w_1 T_{cp}]}{w_0 T_{cp}}\alpha_1 = \frac{1}{k_1}\alpha_1 \quad (13)$$

where $k_1 = \frac{w_0 T_{cp}}{[(z_1 + kz_1)T_{cm} + w_1 T_{cp}]}$

From equation (11),

$$\alpha_i = \frac{[w_{i-1}T_{cp} + (z_{i-1} + kz_{i-1})T_{cm}]}{w_iT_{cp} + (z_i + kz_i)T_{cm}}\alpha_{i-1} = q_i\alpha_{i-1} \qquad (14)$$

where $q_i = \dfrac{w_{i-1}T_{cp} + (z_{i-1} + kz_{i-1})T_{cm}}{w_iT_{cp} + (z_i + kz_i)T_{cm}}$, for i = 2, 3,…, m

Equation (14) can be represented as

$$\alpha_i = q_i\alpha_{i-1} = (\prod_{l=2}^{i} q_l)\alpha_1 \qquad \text{i=2, 3, …, m} \qquad (15)$$

From (9), (11), the normalization equation (12) becomes

$$\frac{1}{k_1}\alpha_1 + \alpha_1 + \sum_{i=2}^{m}\alpha_i = 1 \qquad (16)$$

$$[\frac{1}{k_1} + 1 + \sum_{i=2}^{m}(\prod_{l=2}^{i} q_l)]\alpha_1 = 1 \qquad (17)$$

$$\alpha_1 = \frac{1}{[\frac{1}{k_1} + 1 + \sum_{i=2}^{m}(\prod_{l=2}^{i} q_l)]} \qquad (18)$$

From the timing diagram, Fig. 4, we can get the finish time with m+1 processors, $T_{f,m}$, as follows:

$$T_{f,m} = \alpha_0 w_0 T_{cp} = \frac{1}{k_1}\alpha_1 w_0 T_{cp} \qquad (19)$$

While the finish time with only one processor, $T_{f,o}$, is

$$T_{f,o} = \alpha_0 w_0 T_{cp} = 1 \bullet w_0 T_{cp} = w_0 T_{cp} \qquad (20)$$

The speed-up, which is the ratio of job solution time of one processor to that on m+1 processors, can be obtained like this:

$$\frac{T_{f,o}}{T_{f,m}} = k_1 \times \frac{1}{\alpha_1} = 1 + k_1[1 + \sum_{i=2}^{m}(\prod_{l=2}^{i} q_l)] \qquad (21)$$

Since $\prod_{l=2}^{i} q_l$ can be simplified as $\dfrac{w_1T_{cp} + (z_1 + kz_1)T_{cm}}{w_iT_{cp} + (z_i + kz_i)T_{cm}}$, the speed-up and the finish time, $T_{f,m}$, can be derived as follows:

$$\text{SPUP} = 1 + \frac{w_0T_{cp}}{w_1T_{cp} + (z_1 + kz_1)T_{cm}}[1 + \sum_{i=2}^{m}\frac{w_1T_{cp} + (z_1 + kz_1)T_{cm}}{w_iT_{cp} + (z_i + kz_i)T_{cm}}]$$

$$= 1 + w_0T_{cp}\sum_{i=1}^{m}\frac{1}{[w_iT_{cp} + (z_i + kz_i)T_{cm}]} \qquad (22)$$

$$T_{f,m} = \frac{1}{k_1}\alpha_1 w_0 T_{cp}$$

$$= w_0T_{cp} \Big/ [1 + k_1(\sum_{i=2}^{m}\frac{w_1T_{cp} + (z_1 + kz_1)T_{cm}}{w_iT_{cp} + (z_i + kz_i)T_{cm}})] \qquad (23)$$

For a special case, a homogeneous network, in which all of children processors are same, the finish time, $T_{f,m}$, is

$$T_{f,m} = \frac{w_0T_{cp}}{1 + k_1[1 + (m-1)]} = \frac{w_0T_{cp}}{1 + k_1 \times m}$$

$$= \frac{w_0T_{cp}}{1 + m \times \dfrac{(1+k)zT_{cm} + wT_{cp}}{w_0T_{cp}}}$$

$$= \frac{w_0T_{cp}}{1 + m \times [(1+k)\rho^* + 1]} \qquad (24)$$

where k_1 is from (13), and $\rho^* = \dfrac{zT_{cm}}{w_0T_{cp}}$.

From (21), speed up for a homogeneous network is obtained as follows:

$$\text{SPUP} = \frac{T_{f,o}}{T_{f,m}} = 1 + k_1[1 + \sum_{i=2}^{m} 1] = 1 + k_1 \times m \qquad (25)$$

where $q_i = \dfrac{wT_{cp} + (z + kz)T_{cm}}{wT_{cp} + (z + kz)T_{cm}} = 1$

We know that the value of speed-up is linearly related to the number of processors in a simultaneous distribution and collection method.

IV. Performance Analysis and Comparison

A. Speed up without computation on the root processor

In this section A, for SD-NO (Simultaneous Distribution with NO computation) scheduling method, we assume that the root processor is identical to each child processor in terms of processing speed. The root processor does not have load assigned to itself, but just distributes and collects load to/from children processors, We consider the same parameters as those of PI and PR in paper [4]. The inverse computing speed of the processor, w, is 1.0, and the inverse communication speed, z, is 0.2. Both T_{cp} and T_{cm} are 1.0. Three kinds of the ratio, k, is considered such as 0.2, 1.0, and 1.8.

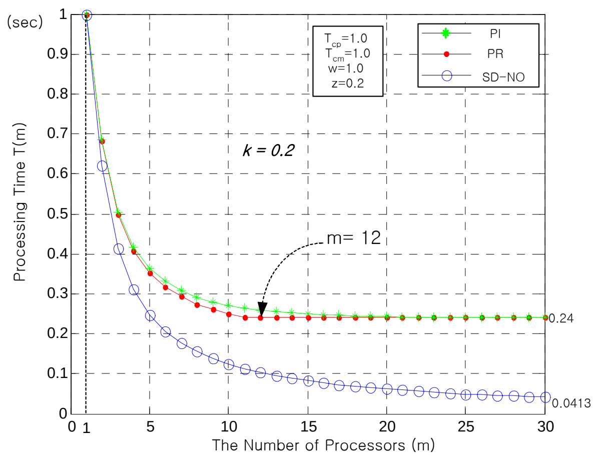

Fig.5. Total processing time versus the number of processors for SD-NO (Simultaneous Distribution with NO computation), PI and PR load scheduling methods. The load is not assigned to the root processor.

In Fig.5, our load scheduling mechanism, SD-NO shows a much better performance than the previous one, PI and PR. When the number of processors is 12, which is the optimal number of the processors in PI, the SD-NO method shows more than 2 times less processing time as PI and PR. Especially when we consider more processors added in the network, for example, 30, the difference is much larger, which is more than 8 times for PI. This means our mechanism, SD-NO, is more scalable and cost effective in terms of the processing power. When the number of processors increases from 12 to 30, the performance of the system increases almost 6 times, while the number of processors only increases 2.5 times.

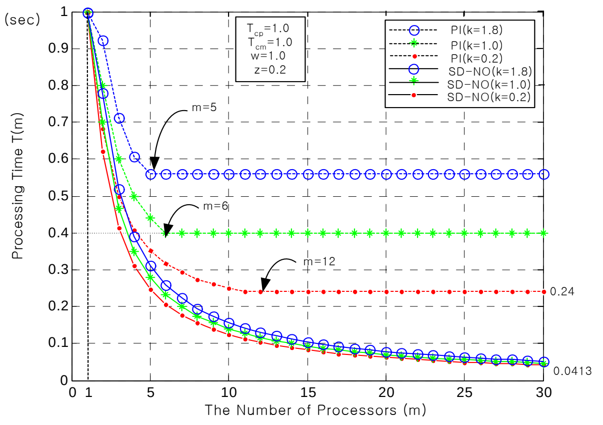

Fig. 6. Total processing time versus the number of processors for SD-NO and PI load scheduling methods on a homogeneous network. Here three values of the ratio, k, are considered, where k = 1.8, k = 1.0, k = 0.2.

In Fig. 6, we know that for all three cases of the ratio, k, where k <1, k=1, k>1, our mechanism shows a much better performance than that of PI. In terms of the optimal number of processors, SD-NO shows almost 2 times better performance than that of PI for three 'k' values. When we consider processor scalability, for a number of processors of 30, SD-NO achieves much better performance than that of PI. That is more than 10 times, 8 times, and 6 times for each of k>1, k = 1, k < 1.

B. *Speed up with computation on the root processor*

In this section B, for the SD-COMP (Simultaneous Distribution with Computation) scheduling method, we assume that the root processor is different from the children in terms of processing power and has load to compute itself. So the root processor not only distributes and collects load to/from children processors, but also computes load. The ratio of load received to load sent, k, is chosen as 0.2, since we suppose the case k is less than 1, as is usually the case for compressed results, like MPEG.

In Fig.7, we assume that the processing power of the root processor for SD-COMP is twice as much as that of each child processor. That is the inverse computing speed of the root processor, w_0, is half of that of each child, 'w'. We see that SD-COMP method is continuously faster for SD-NO method, and much faster, for example more than 6 times, for PI and PR method up to the number of processors, 10. In terms of

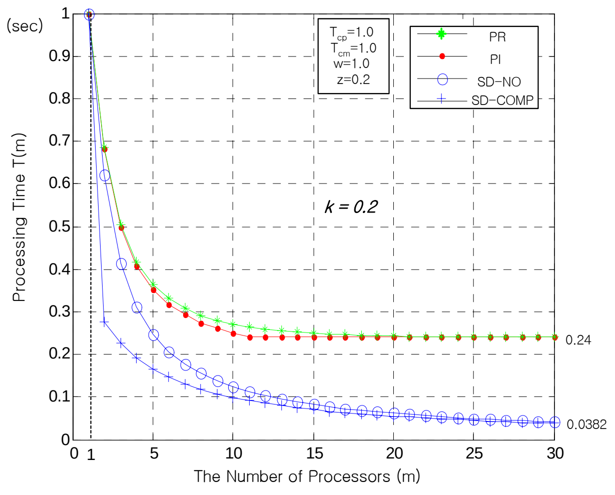

Fig.7. Total processing time versus the number of processors for SD-COMP, (SD with computation), SD-NO, PI, and PR load scheduling methods on a homogeneous network. As for SD-COMP, the computing speed of the root processor is twice as fast as that of each child processor and has load assigned.

processor scalability, SD-COMP has more improved result. When the number of processors increases from 12 to 30, the performance of SD-COMP goes up 2.24 times to 6 times as fast as respectively that of PI. However, SD-COMP and SD-NO method shows similar performance and good scalability.

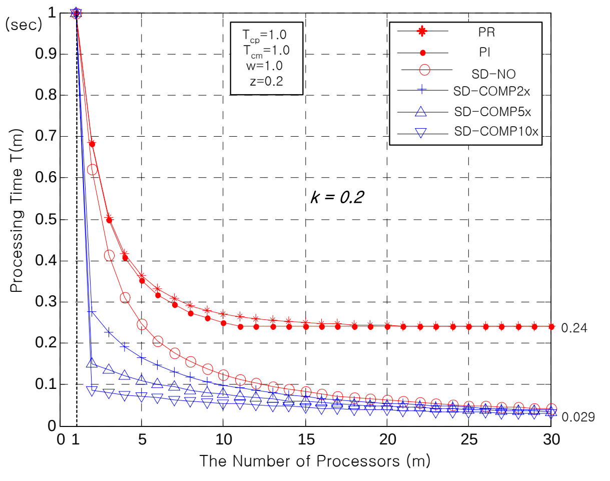

Fig.8. Total processing time versus the number of processors for SD-COMP 2x, SD-COMP5x, SD-COMP10x, SD-NO, PI, and PR load scheduling methods. As for each SD-COMP above, the computing speed of the root processor is twice, 5 times, and 10 times as fast as that of each child processor.

From Fig.8, we consider three cases of processing speed of the root processor for SD-COMP. Those are twice, 5 times, and 10 times as fast as that of each child processor. When the number of processors is small, for example 2 to 5, the performance of the SD-COMP method is much better than SD-NO, PI, and PR method, because the root processor of SD-COMP method participates in computation itself, involving around half to 20% of the whole load.

While the number of processors increases to 12, all of the SD methods show 2.4 times, 3 times, 4 times, 5 times improvement in the processing time irrespective of load assigned to the root processor. As the number of processors increases up to 30, all of the SD methods show still better performance than PI and PR methods, because all of SD methods have good scalability in the number of processors. However when the number of processors is 30, the performance improvement of SD-COMP is small, just 1.3 times, against the SD-NO method as compared to 6 to 8 times against PI and PR.

One point to note is that when the number of processors is small, it is the method to distribute load as well as the root processor speed that is important to total processing time. The other point is that when the number of processors increases enough, the most critical part is the method to distribute and collect load simultaneously or sequentially rather than whether load is assigned to the root processor.

V. Conclusion

We believe that this work is meaningful for showing not only a more efficient scheduling method for parallel video encoding, but also good scalability in the number of processors. Through simultaneous load distribution and collection, our proposed method, SD scheduling, achieves a minimum 3 times better performance under the optimal number of processors of the PI method, and 8 times better performance when the number of processors increases up to 30. In addition, in terms of a practical processor and network topology, we propose a mutil-port star network to achieve concurrent communication among the root processor and children processors. In the future, other appropriate network topologies could be considered.

References

[1] V. Bharawaj, D. Ghose and T.G. Robertazzi, “Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems,” in the special issue of *Cluster Computing* on Divisible Load Scheduling, spring 2003.

[2] Y.C. Cheng and T. G. Robertazzi, “Distributed Computation with Communication Delays,” *IEEE Transactions on Aerospace & Electronic Systems*, vol. 24, no. 6, Nov. 1988, pp. 700-712.

[3] D. T. Altilar, Y. P. Paker, “Optimal Scheduling Algorithms for Communication Constrained Parallel Processing”, LNCS2400, Euro-Par 2002, Germany

[4] M. Suresh, S. N. Omkar, H.J. Kim, “Parallel Video Processing using Divisible Load Scheduling Paradigm”, Korean Journal of Broadcast Engineering, vol. 10, no. 1, 2005.

[5] D. T. Altilar, Y.P. Paker, A. V. Sahiner “A parallel architecture for video processing”, LNCS 1225 Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking, April 1997

[6] G. D. Barlas, “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, vol. 9, no. 5, 1998.

[7] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, Y. Yang. “Scheduling Divisible Loads on Star and Tree Networks: Results and Open Problems,” IEEE Transactions on Parallel and Distributed Systems, vol. 16, no. 3, 2005.

[8] P. Li, B. Veeravalli, A. A. Kassim, “Design and Implementation of Parallel Video Encoding Strategies Using Divisible Load Analysis,” IEEE Transactions on Parallel and Distributed Systems, vol. 15, no. 9, 2005, pp 1098-1112.

[9] D.A.L. Piriyakumar, C.S.R. Murthy, "Distributed Computation for a Hypercube Network of Sensor-Driven Processors with Communication Delays Including Setup Time", IEEE Transactions on Systems, Man and Cybernetics-Part A: Systems and Humans, vol. 28, no. 2, March 1998.

[10] J.T. Hung, H.J. Kim, T.G. Robertazzi, "Scalable Scheduling in Parallel Processors," Proceedings of the 2002 Conference on Information Sciences and Systems, Princeton University, Princeton NJ, March 2002.