Distributed Computation with Communication Delay

YUAN-CHIEH CHENG, Member, IEEE
THOMAS G. ROBERTAZZI, Member, IEEE
SUNY at Stony Brook

A linear network of communicating processors is examined. The objective is to solve a computational problem in a minimal amount of time. The processors in the networks may be equipped either with front end processors for communication off-loading or without front end processors. The cases of equal division of processing load and optimal division of processing load are discussed for both the network with front-end processors and the network without front end processors. An example of the inclusion of solution time, the time taken for processors to report the solution back to the problem originator, is also presented.

Manuscript received October 26, 1987; revised March 4, 1988.

IEEE Log No. 24688.

This work was supported in part by the SDIO/IST and managed by the United States Office of Naval Research under Grant N00014-85-K0610 and in part by the NSF under Grant No. NCR8703689.

Authors' address: Department of Electrical Engineering, College of Engineering and Applied Sciences, SUNY at Stony Brook, Stony Brook, NY 11794-2350.

0018-9251/88/1100-0700 $1.00 © 1988 IEEE.

I. INTRODUCTION

In recent years there has been an increasing interest in distributed intelligent sensor networks [1–12]. This interest has been generated by the requirements of military surveillance systems and other civilian applications.

An intelligent sensor network is a collection of units with sensing, computational, and communication ability. Thus one has a network of communicating processors with sensing capability. The processors' function is to cooperatively monitor the environment and be able to ascertain its salient features. These intelligent sensors may be dedicated deployable units or may be resident on moving platforms.

One major issue in distributed intelligent sensor networks is the tradeoff between communication and computation. In a hostile environment where low probability of intercept considerations are important, one may want to minimize communication among intelligent sensors. But on the other hand, when time is an important consideration, one may want to minimize the total time taken in order to get the solution and one may want to spread observations to as many intelligent sensors as possible.

Several authors have recognized the importance of the compute versus communicate issue for distributed intelligent networks. As it is put in [10]:

> The question of 'what to compute' and 'what to communicate' are at the core of the distributed fusion system design problem. They are, unfortunately, very complex questions and with our present understanding it is easier to make lists of subsidiary questions than to answer the primary ones. Consequently, very little has appeared in the literature and what has tended to be descriptive rather than prescriptive. Thus we find many proposals for distributed systems, but little design methodology.

There has been a great deal of work on only communication and on only computation, but not that much on both communication and computation.

In this paper, a linear network of communicating processors is considered. The objective is to solve a computational problem in a minimal amount of time. The motivation is a situation where one processor receives a burst of measurement data (the processing load) and distributes the processing load to other sensors to obtain the benefits of parallel processing. Different processors may have different processing speeds and each link between processors may have a different channel capacity. The problem is that of determining the best distribution of processing load among the sensors.

This paper is organized as follows. In section II, the case of equal division of processing load in a distributed intelligent sensor network is discussed. In section III, the case of optimal division of processing load is considered where front end processors are available to subsume the task of communication. In section IV, the case without front end processors is examined. Finally in section V, the time required to compile solutions is considered.

II. EQUAL DIVISION OF PROCESSING LOAD

Consider a linear network of identical communicating processors. Every processor can communicate with only its right and left immediate neighbors. All links have identical channel capacities. The processors may be equipped either with or without front end processors. Both cases are discussed in the following.

A. Linear Network Without Front-End Processor

Consider first the case that processors are equipped without front end processors, so that the processors can not compute and communicate at the same time. Suppose that the processor at the left end of the chain receives a burst of measurement data, which is its processing load. There are two cases of interest. In the first the entire measurement data is required by each processor. In the second the measurement data is divided equally among the processors.

If the entire measurement data is to be transmitted to the other $N-1$ processors, the total process time T_t is

$$T_t = (N-1)T_{cm} + \frac{T_{cp}}{N} \tag{1}$$

where T_{cm} is the time to transmit the entire measurement data over the channel, and T_{cp} is the time it takes for one processor to process the entire processing load. By differentiating both sides of (1), the number of processors at which T_t is minimized is found:

$$N = \sqrt{\frac{T_{cp}}{T_{cm}}} = \sqrt{\frac{1}{\rho}} \tag{2}$$

where ρ is defined as T_{cm}/T_{cp}, the ratio of communication time over computation time. Note that too few processors leads to insufficient parallelism and too many processors leads to excessive communication delays.

If the processor should decide to divide the processing load equally into N smaller parts and share this with the other $N-1$ processors, it will transmit $N-1$ parts to its right immediate neighbor and compute the part left over. Upon receiving the $N-1$ parts, the neighbor then transmits the $N-2$ parts to its immediate neighbor and computes the part left over. The process repeats itself and stops at the Nth processor. The timing diagram for $N=5$ of the process is shown in Fig. 1. For each processor its graph shows communication time above the horizontal axis and computation time below it.

Since T_{cp}/N is always overlapped with T_{cm}/N except at the last processor, the total process time T_t is

$$T_t = \frac{T_{cm}}{N}(N-1+N-2+\cdots+1) + \frac{T_{cp}}{N}$$

$$= \frac{N-1}{2}T_{cm} + \frac{T_{cp}}{N}. \tag{3}$$

By differentiating both sides of (3), T_t can be minimized when

$$N = \sqrt{\frac{2T_{cp}}{T_{cm}}} = \sqrt{\frac{2}{\rho}}. \tag{4}$$

In the above discussion, the time it takes for every other processor to report its solution back to the starting processor is assumed to be very short and can be ignored. In some application this may not be true. The inclusion of this time in the case of parallel processing is examined next.

The same linear network as mentioned before is considered. When the last processor that received the measurement data finishes its computation, it starts sending back the solution to its left immediate neighbor. The transmission takes time T_S. Upon receiving the solution, the neighbor transmits the solution and its own solution to its left immediate neighbor. This transmission takes $2T_S$. The process repeats itself until all the solutions

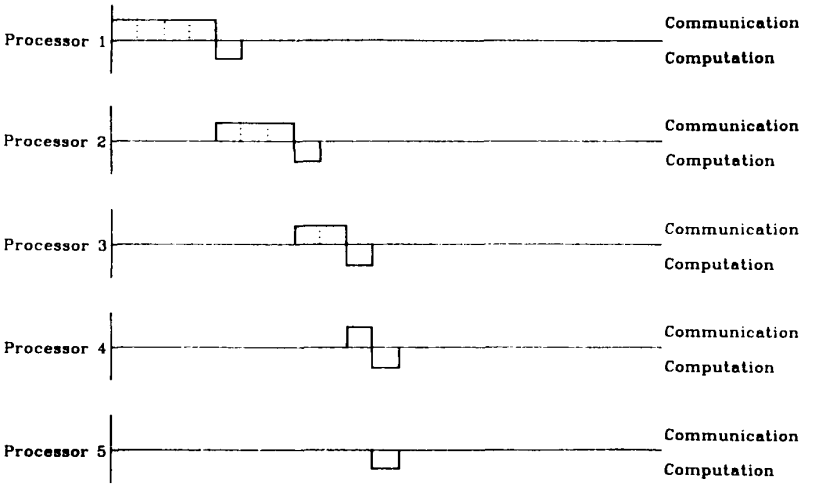


Fig. 1.

are sent back to the starting processor. The timing diagram for $N = 5$ is shown in Fig. 2. The time to transmit solutions is shown at the right of the diagram above the axis. The total time T_t takes for the whole process is

$$T_t = \frac{T_{cm}}{N}(N-1+N-2+\cdots+1) + \frac{T_{cp}}{N} + T_S(1+2+\cdots+N-1) \quad (5)$$

$$= \frac{N-1}{2}T_{cm} + \frac{T_{cp}}{N} + T_S\frac{(N-1)N}{2}.$$

To minimize T_t, (5) can be differentiated at both sides and becomes

$$N^3T_S + N^2T_{cm} - 2T_{cp} = 0. \quad (6)$$

Equation (6) may be solved numerically for N to minimize T_t.

B. Linear Network With Front End Processor

For the linear network of processors that are equipped with front end processors, the processors can communicate and compute at the same time. The total processing time taken for both of the previous cases is actually the same as for the linear network without front end processors. Solution time may or may not be ignored. The timing diagram for $N = 5$ of a linear network with front end processors and the inclusion of solution time is shown in Fig. 3.

From the timing diagrams (Fig. 2 and Fig. 3), the total processing time is the total communication time plus the computation time of the last processor for both the network with front end processors and the network without front end processors. Their T_t are thereby the same.

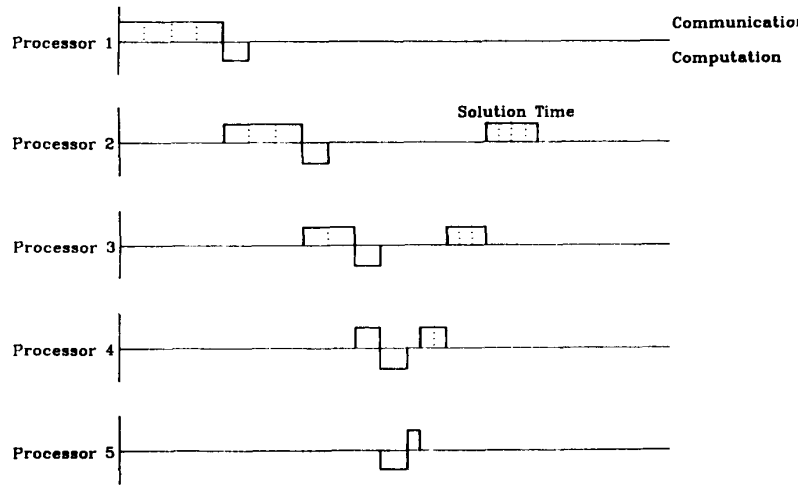


Fig. 2.

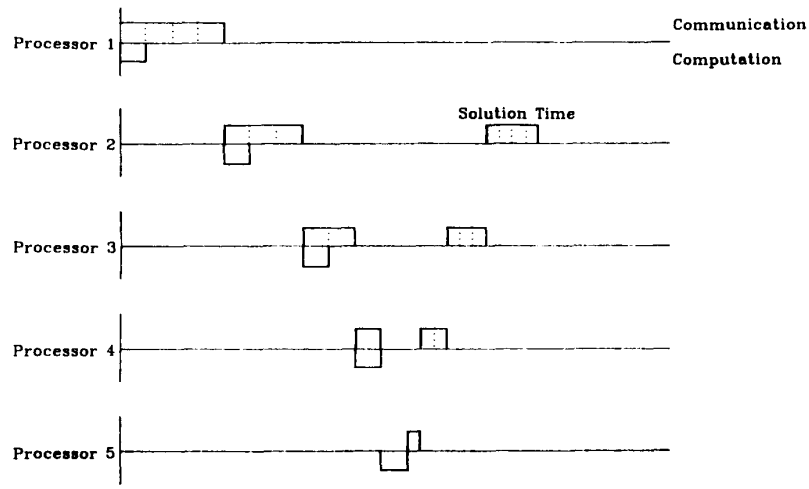


Fig. 3.

III. OPTIMAL DIVISION OF PROCESSING LOAD. LINEAR NETWORK WITH FRONT END PROCESSOR

In section II, it was shown that the linear network of processors equipped with front end processors has the same minimal total processing time as the network without front end processors. In this section and the next section, the optimal division of processing load division is examined and the performance of this approach for both the network with front-end processors and the network without front end processors is compared.

We consider a linear network with front end processors where each of the processors in the network may have different computing capabilities and the channel capacity between processors may be different.

A. Origination at Network Boundary

Suppose that the processor at left end of the chain receives a burst of measurement data and is to share the data with the other $N-1$ processors. The starting (or the first) processor then divides the processing load into N smaller parts optimally instead of equally as in section II. It keeps the fraction $\hat{\alpha}_1$ of the processing load for itself and transmits the remaining measurement data to its right immediate neighbor. Upon receiving the measurement data, the neighbor (the second processor) keeps the fraction $\hat{\alpha}_2$ of what it has received to process and transmits the remaining to its right immediate neighbor (the third processor). For the ith processor, it keeps the fraction $\hat{\alpha}_i$ of what it has just received and transmits the remaining to the $i+1$st processor. The process repeats itself until the Nth processor is reached. The timing diagram of the entire process is shown in Fig. 4.

In order to obtain maximum parallelism and a minimum time solution all the processors must stop computing at the same time. This can be proved by induction (see Appendix I). This can be seen intuitively as one should keep all processors utilized up until the last moment for maximum efficiency. The starting processor should compute its fraction of the processing load during the entire processing period, so that the total processing time T_t equals the processing time of the starting processor. From Fig. 4 it can also be seen that the processing time $\alpha_i w_i T_{cp}$ of the ith processor equals the transmission time, $(1-\alpha_1-\alpha_2-\cdots-\alpha_i)z_i T_{cm}$, from the ith processor to the $i+1$st processor plus the processing time, $\alpha_{i+1} w_{i+1} T_{cp}$, of its right immediate neighbor (the $i+1$st processor), where α_is are the actual fraction of processing load of the ith processor and can be expressed as a function of $\hat{\alpha}_i$s:

$$\alpha_1 = \hat{\alpha}_1 \tag{7}$$

$$\alpha_i = \left(\prod_{j=1}^{i-1}(1-\hat{\alpha}_j)\right)\hat{\alpha}_i, \qquad i = 2, 3, \ldots, N-1. \tag{8}$$

The total computing time of the ith processor equals

$$\alpha_i w_i T_{cp} = (1-\alpha_1-\alpha_2-\cdots-\alpha_i)z_i T_{cm} + \alpha_{i+1} w_{i+1} T_{cp}. \tag{9}$$

Substituting (8) into (9) yields

$$\hat{\alpha}_i w_i T_{cp} = (1-\hat{\alpha}_i)z_i T_{cm} + (1-\hat{\alpha}_i)\hat{\alpha}_{i+1} T_{cp}, \qquad i = 1, 2, 3, \ldots, N-2 \tag{10}$$

and

$$\hat{\alpha}_{N-1} w_{N-1} T_{cp} = (1-\hat{\alpha}_{N-1})z_{N-1} T_{cm} + (1-\hat{\alpha}_{N-1}) T_{cp} w_N \tag{11}$$

where w_is are inversely proportional to the speed of the ith processor and z_is are inversely proportional to the channel speed between the ith and the $i+1$st processor. From (10) and through some simple algebra, $\hat{\alpha}_i$ may be expressed as

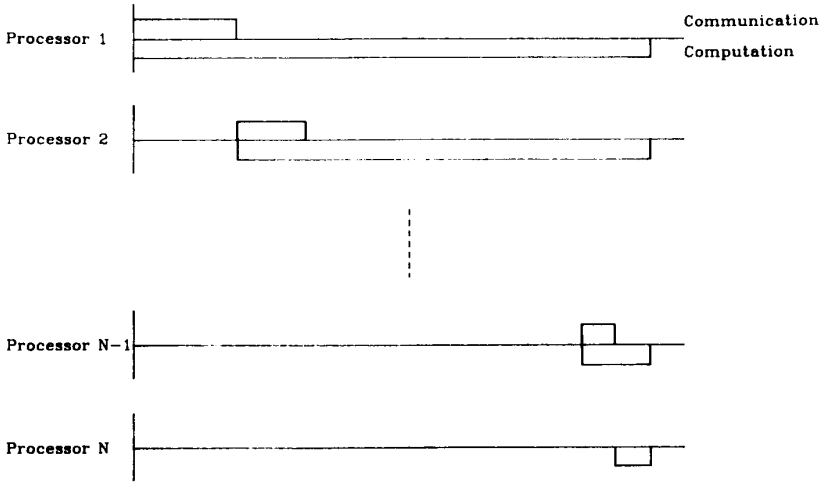


Fig. 4.

$$\hat{\alpha}_i = \frac{z_i\rho + \hat{\alpha}_{i+1}w_{i+1}}{w_i + z_i\rho + \hat{\alpha}_{i+1}w_{i+1}} \tag{12}$$

where ρ, as defined before, equals T_{cm}/T_{cp}, and $\hat{\alpha}_i$ is a function of $\hat{\alpha}_{i+1}$. From (11) $\hat{\alpha}_{N-1}$ can be solved as

$$\hat{\alpha}_{N-1} = \frac{z_{N-1}\rho + w_N}{w_{N-1} + z_{N-1}\rho + w_N}. \tag{13}$$

Thi $\hat{\alpha}_i$s can be solved recursively through (12) and (13). The total processing time is

$$T_t = \alpha_1 w_1 T_{cp}. \tag{14}$$

In Fig. 5 the minimum total processing time is plotted against the number of the processors in the linear network with $w_i = 1$, $T_{cp} = 1$, $\rho = 0.5$, and the five performance curves are obtained with $z_i = 0.1$, 0.2, 1, 5, and 10, respectively. As shown in the figure, the longer the communication delay, the longer the total processing time. Note that minimum total processing time levels off after a certain number of processors.

B. Origination from Network Interior

Suppose that a processor in the middle of the linear network, instead of the one at the end, receives a burst of measurement data and is to share the data with N_l other processors to its left and N_r processors to its right in the network. Since every processor in the network has only one front end processor, the starting processor can transmit in only one direction at a time. As in the unidirectional transmission case above, the starting processor first divides the processing load into smaller parts, then it transmits the fraction β_l of the total processing load to its left immediate neighbor and the fraction β_r of the total processing load to its right immediate neighbor, and keeps the remaining fraction $1 - \beta_l - \beta_r$ for itself to compute at the same time. Upon receiving the data, the left neighbor (the left first processor) transmits the fraction $1 - \hat{\alpha}_{l1}$ of what it has received to its left immediate neighbor and keeps the remaining load for itself to compute. The whole process at the left side repeats itself until the N_lth processor is reached. The right neighbor (the right first processor) performs the same operation as the left first processor, and the process at the right side stops when the N_rth processor is reached. The timing diagram of the entire process is shown in Fig. 6.

As in the unidirectional case, in order to achieve maximum parallelism and a minimum time solution all the processors must stop computing at the same time, and the starting processor should compute its fraction of the processing load during the entire processing period. The total processing time is equal to the processing time of the starting processor. From the timing diagram, it can be seen that the processing time $(1 - \beta_l - \beta_r)w_0T_{cp}$ of the starting processor equals the transmission time $\beta_l z_{l0}T_{cm}$ of the fraction β_l of the measurement data transmitted to its left immediate neighbor plus the processing time $\beta_l\hat{\alpha}_{l1}w_{l1}T_{cp}$ of the left first processor. The processing time $\beta_l\hat{\alpha}_{l1}w_{l1}T_{cp}$ of the left first processor also equals the transmission time $\beta_r z_{r0}T_{cm}$ from the starting station to its right immediate neighbor plus the processing time $\beta_r\hat{\alpha}_{r1}w_{r1}T_{cp}$ of the right first processor. Both equations are shown below:

$$(1 - \beta_l - \beta_r)w_0T_{cp} = \beta_l z_{l0}T_{cm} + \beta_l\hat{\alpha}_{l1}w_{l1}T_{cp} \tag{15}$$

$$\beta_l\hat{\alpha}_{l1}w_{l1}T_{cp} = \beta_r z_{r0}T_{cm} + \beta_r\hat{\alpha}_{r1}w_{r1}T_{cp} \tag{16}$$

where $\hat{\alpha}_{li}$ and $\hat{\alpha}_{ri}$ are the fraction of what the left and right ith processors have received for themselves to compute, respectively, w_{li}, w_{ri} are inversely proportional to the computing speed of the left and right ith processors, respectively, and z_{li}, z_{ri} are inversely proportional to the channel speed between the ith and the $i+1$st processor of both the left and right sides of the

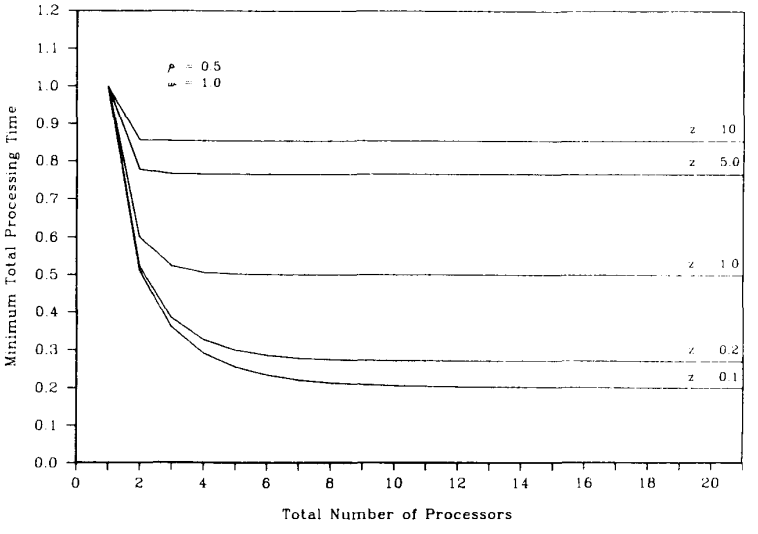


Fig. 5. Origination at network boundary with front end processor.

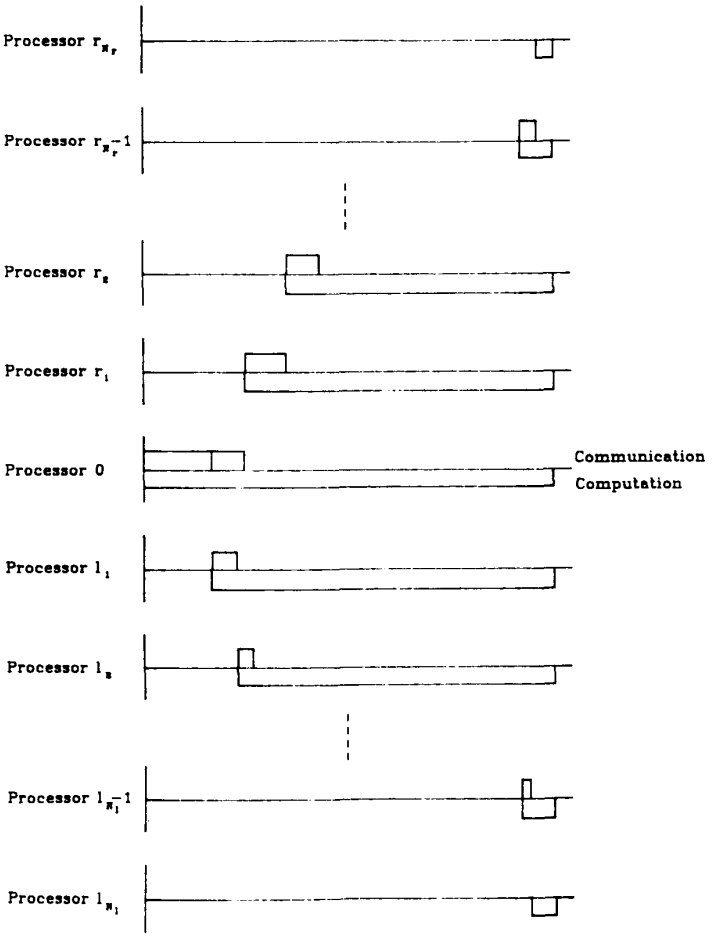


Fig. 6.

starting processor. The 0th processor is the starting processor.

From (15) and (16), both β_l and β_r can be expressed as

$$\beta_l = \frac{w_0(z_{r0}\rho + \hat{\alpha}_{r1}w_{r1})}{(w_0 + z_{l0}\rho + \hat{\alpha}_{l1}w_{l1})(z_{r0}\rho + \hat{\alpha}_{r1}w_{r1}) + \hat{\alpha}_{l1}w_{l1}w_0} \tag{17}$$

and

$$\beta_r = \frac{\hat{\alpha}_{l1}w_{l1}w_0}{(w_0 + z_{l0}\rho + \hat{\alpha}_{l1}w_{l1})(z_{r0}\rho + \hat{\alpha}_{r1}w_{r1}) + \hat{\alpha}_{l1}w_{l1}w_0} \tag{18}$$

From Fig. 6 it can also be seen that both the left side and right side of the starting processor are identical to the unidirectional case discussed earlier in this section with the left first processor and the right first processor as their starting processor, respectively. Thus $\hat{\alpha}_{li}$ and $\hat{\alpha}_{ri}$ are readily obtained from (12) and (13) recursively with corresponding w_{li}, z_{li} and w_{ri}, z_{ri}. Thus from (7) and (8) the actual fraction α_{li} and α_{ri} of the total processing load of the left and right ith processor can be calculated and from (17) and (18) β_l and β_r can be solved. The total processing time is then

$$T_t = (1 - \beta_l - \beta_r)w_0T_{cp}. \tag{19}$$

In Fig. 7 the minimum total processing time is plotted against the position of the processors in a linear network of 21 processors with $w_i = 1$, $T_{cp} = 1$, $\rho = 0.5$, and the five performance curves are obtrained with $z_i = 0.1$, 0.2, 1, 5, and 10, respectively. As shown in the figure, the longer the communication delay the longer the total processing time, and the total processing time is minimized when the starting processor is at the center of the linear network, though this is a shallow minima.

IV. OPTIMAL DIVISION OF PROCESSING LOAD. LINEAR NETWORK WITHOUT FRONT END PROCESSOR

A linear network without front end processors is now considered. Each of the processors in the network, as those of the network in Section III, may have different

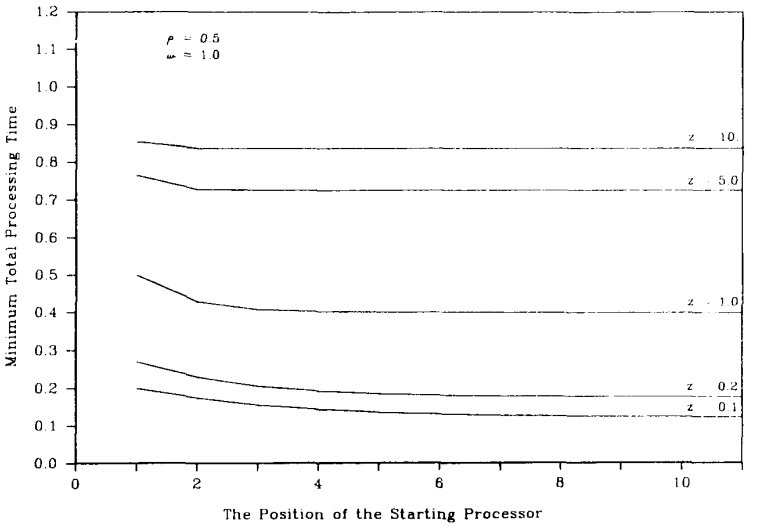


Fig. 7. Origination from network interior with front end processor.

computing capabilities and the channel capacity between processors may be different.

A. Origination at Network Boundary

Suppose that the processor at the left end of the chain receives a burst of measurement data and is to share the data with the other $N-1$ processors. The starting (or the first) processor then divides the processing load into N smaller parts optimally. It first transits the fraction $1-\hat{\alpha}_1$ of the measurement data to its right immediate neighbor and then computes the fraction $\hat{\alpha}_1$ of the processing load itself. Upon receiving the measurement data, the neighbor (the second processor) transmits the fraction $1-\hat{\alpha}_2$ of what it has received to its right immediate neighbor (the third processor) and then keeps the remaining $\hat{\alpha}_2$ for itself to compute. For the ith processor, it transmits the fraction $1-\hat{\alpha}_i$ of what it has just received to the $i+1$st processor and keeps the remaining $\hat{\alpha}_i$ for itself to compute. The process repeats itself until the Nth processor is reached. The timing diagram of the entire process is shown in Fig. 8.

In order to obtain maximum parallelism and a minimum time solution all the processors must stop computing at the same time. This can also be proved by induction. The starting processor should remain operating, i.e., transmitting first to the second processor and then computing its fraction of the processing load, during the entire processing period. The total processing time, T_t, thus equals the transmission time plus the computing time of the starting processor. From Fig. 8 it can be seen that the computing time of the ith processor, $\alpha_i w_i T_{cp}$, equals the transmission time $(1-\alpha_1-\alpha_2-\cdots-\alpha_i)z_{i+1}T_{cm}$ from the $i+1$st to the $i+2$nd processor plus the computing time $\alpha_{i+1}T_{cp}$ of the $i+1$st processor, where α_is are the actual fraction of processing load of the ith processor and can be expressed as a function of $\hat{\alpha}_i$s ((7) and (8)). The total computing time of the ith

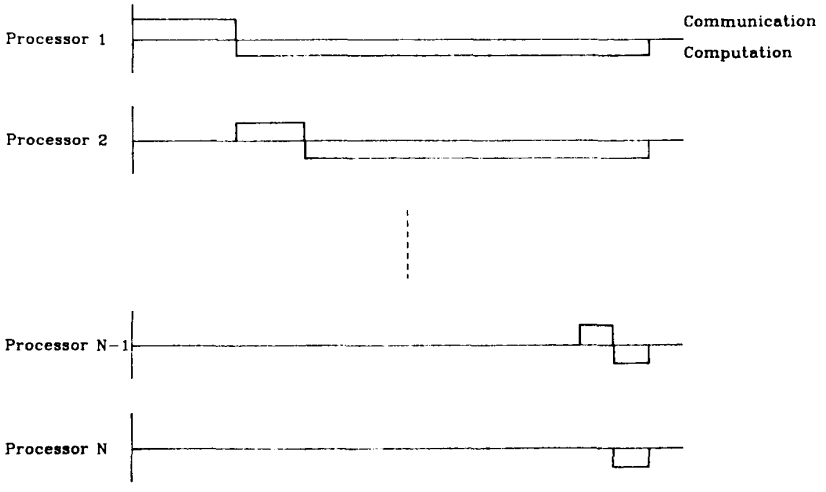


Fig. 8.

processor equals

$$\alpha_i w_i T_{cp} = (1 - \alpha_1 - \alpha_2 - \cdots - \alpha_i) z_{i+1} T_{cm} + \alpha_{i+1} w_i T_{cp}, \quad i = 1, 2, \ldots, N-2. \quad (20)$$

Substituting (8) yields

$$\hat{\alpha}_i w_i T_{cp} = (1 - \hat{\alpha}_i)(1 - \hat{\alpha}_{i+1}) z_{i+1} T_{cm} + (1 - \hat{\alpha}_i)\hat{\alpha}_{i+1} w_{i+1} T_{cp}, \quad i = 1, 2, \ldots, N-2 \quad (21)$$

and

$$\hat{\alpha}_{N-1} w_{N-1} T_{cp} = (1 - \hat{\alpha}_{N-1}) w_N T_{cp} \quad (22)$$

where w_is are inversely proportional to the speed of the ith processor and z_is are inversely proportional to the channel speed between the ith and the $i+1$st processor. From (21) and through some simple algebra, $\hat{\alpha}_i$ may be expressed as

$$\hat{\alpha}_i = \frac{(1 - \hat{\alpha}_{i+1}) z_i \rho + \hat{\alpha}_{i+1} w_{i+1}}{(w_i + (1 - \hat{\alpha}_{i+1}) z_i \rho + \hat{\alpha}_{i+1} w_{i+1})}, \quad i = 1, 2, \ldots, N-2 \quad (23)$$

where ρ equals T_{cm}/T_{cp} and $\hat{\alpha}_i$ is a function of $\hat{\alpha}_{i+1}$. From (22) $\hat{\alpha}_{N-1}$ can be solved as

$$\hat{\alpha}_{N-1} = \frac{w_N}{w_{N-1} + w_N}. \quad (24)$$

So the $\hat{\alpha}_i$s can be solved recursively through (23) and (24). The total processing time is

$$T_t = (1 - \alpha_1) z_1 T_{cm} + \alpha_1 w_1 T_{cp}. \quad (25)$$

In Fig. 9 the minimum total processing time is plotted against the number of the processors in the linear network with $w_i = 1$, $T_{cp} = 1$, $\rho = 0.5$, and the five performance curves are obtained with $z_i = 0.1$, 0.2, 1, 5, and 10, respectively. As shown in the figure, the longer the communication delay, the longer the total processing time. The minimum total processing time levels off after a certain number of processors. For the same parameters the total processing time is longer than the network with front end processors.

B. Origination from Network Interior

Suppose that a processor in the middle of the linear network, instead of the one at the end, receives a burst of measurement data and is to share the data with N_l other processors to its left and N_r processors to its right in the network. It may transmit the measurement data to both sides of the network. The starting processor can transmit in only one direction at a time. As in the unidirectional transmission case above, the starting processor first divides the processing load into smaller parts, then it first transmits the fraction β_l of the total processing load to its left immediate neighbor and the fraction β_r of the total processing load to its right immediate neighbor, and keeps the remaining $1 - \beta_l - \beta_r$ fraction for itself to compute. Upon receiving the data, the left neighbor (the left first processor) transmits $1 - \hat{\alpha}_{l1}$ fraction of what it has received to its left immediate neighbor and keeps the remaining $\hat{\alpha}_{l1}$ for itself to compute. The whole process at the left side repeats itself until the N_lth processor is reached. The right neighbor (the right first processor) performs the same operation as the left first processor, and the process at the right side stops when the N_rth processor is reached. The timing diagram of the entire process is shown in Fig. 10.

As in the unidirectional case, in order to achieve maximum parallelism and a minimum time solution all the processors must stop computing at the same time. This can also be proved by induction. The starting processor should remain operating i.e., transmitting first

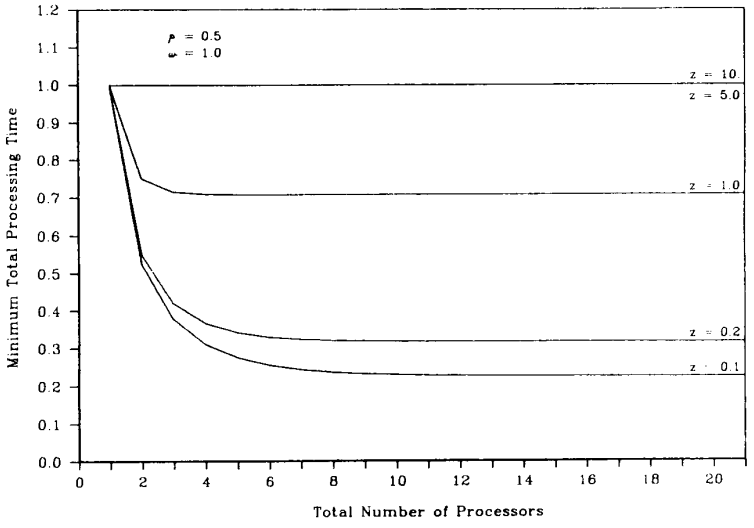


Fig. 9. Origination at network boundary with no front end processor.

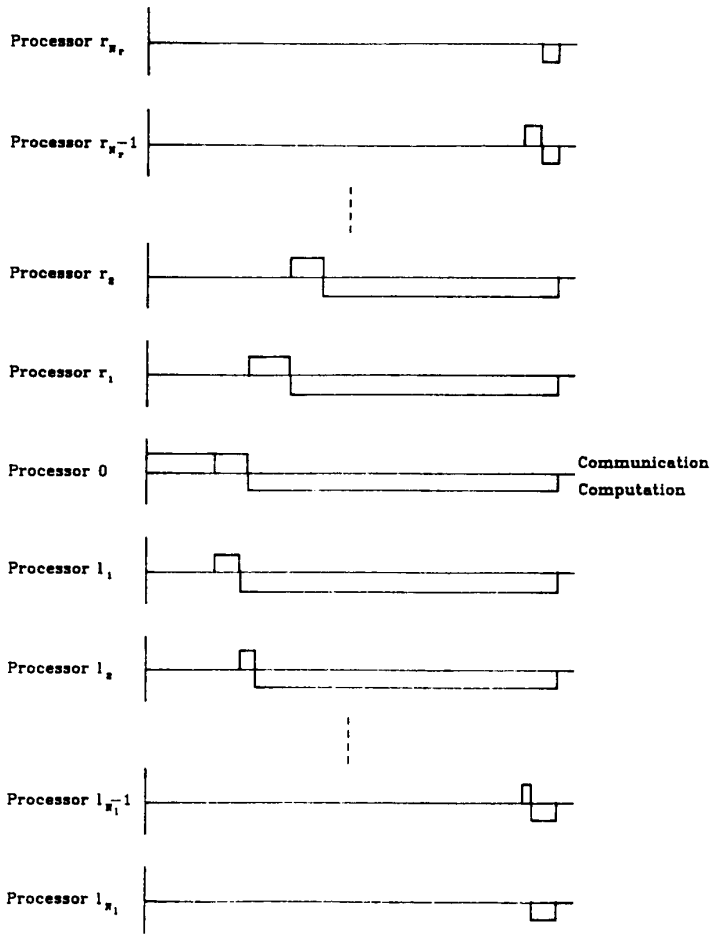


Fig. 10.

to the left processor, then to the right processor and then computing its fraction of the processing load, during the entire processing period. The total processing time T_t equals to the transmission time plus the computing time of the starting processor. From the timing diagram, it can be seen that the computing time $(1-\beta_l-\beta_r)w_0T_{cp}$ of the starting processor equals the transmission time $\beta_r(1-\hat{\alpha}_{r1})z_{r1}T_{cm}$ of the right first processor plus the computing time $\beta_r\hat{\alpha}_{r1}w_{r1}T_{cp}$ of the same processor. The transmission time, from the starting processor to the right first processor, $\beta_r z_{r0}T_{cm}$ plus the computing time $(1-\beta_l-\beta_r)w_0T_{cp}$ of the starting processor equals the transmission time $\beta_l(1-\hat{\alpha}_{l1})z_{l1}T_{cm}$ of the left first processor plus the computing time $\beta_l\hat{\alpha}_{l1}w_{l1}T_{cp}$ of the same processor. Both equations are shown below:

$$(1-\beta_l-\beta_r)w_0T_{cp} = \beta_r(1-\hat{\alpha}_{r1})z_{r1}T_{cm} + \beta_r\hat{\alpha}_{r1}w_{r1}T_{cp} \tag{26}$$

$$\beta_r z_{r0}T_{cm} + (1-\beta_l-\beta_r)w_0T_{cp} = \beta_l(1-\hat{\alpha}_{l1})z_{l1}T_{cm} + \beta_l\hat{\alpha}_{l1}w_{l1}T_{cp} \tag{27}$$

where $\hat{\alpha}_{li}$ and $\hat{\alpha}_{ri}$ are the fraction of what the left and right ith processors have received for themselves to compute, respectively, w_{li}, w_{ri} are inversely proportional to the computing speed of the left and right ith processors, respectively, and z_{li}, z_{ri} are inversely proportional to the channel speed between the ith and the $i+1$st processor on both the left and right sides of the starting processor. The 0th processor is the starting processor.

From (26) and (27), both β_l and β_r can be expressed as

$$\begin{aligned}\beta_r = {} & (w_0(1-\hat{\alpha}_{l1})z_{l0}\rho + \hat{\alpha}_{l1}w_{l1}w_0)/[((1-\hat{\alpha}_{r1})z_{r0}\rho \\ & + \hat{\alpha}_{r1}w_{r1}+w_0)((1-\hat{\alpha}_{l1})z_{l0}\rho \\ & + \hat{\alpha}_{l1}w_{l1}+w_0) + (z_{r0}\rho - w_0)w_0]\end{aligned} \tag{28}$$

and

$$\beta_l = \frac{\beta_r(z_{r0}\rho - w_0) + w_0}{(1-\hat{\alpha}_{l1})z_{l0}\rho + \hat{\alpha}_{l1}w_{l1} + w_0} \tag{29}$$

From Fig. 10 it can also be seen that both the left side and right side of the starting processor are identical to the unidirectional case discussed earlier in this section with the left first processor and the right first processor as

their starting processor, respectively. Thus $\hat{\alpha}_{li}$ and $\hat{\alpha}_{ri}$ are readily obtained from (23) and (24) recursively with corresponding w_{li}, z_{li}, and w_{ri}, z_{ri}. From (7) and (8) the actual fraction α_{li} and α_{ri} of the total processing load of the left and right ith processor can be calculated, and from (28) and (29) β_l and β_r can be solved. The total processing time is then

$$T_t = (1 - \beta_l - \beta_r) w_0 T_{cp} + \beta_l z_{l0} T_{cm} + \beta_r z_{r0} T_{cm}. \quad (30)$$

In Fig. 11 the minimum total processing time is plotted against the position of the processors in the linear network of 21 processors with $w_i = 1$, $T_{cp} = 1$, $\rho = 0.5$, and the five performance curves are obtained with $z_i = 0.1$, 0.2, 1, 5, and 10, respectively. As shown in the figure, the longer the communication delay, the longer the total processing time, and the toal processing time is minimized when the starting processor is at the center of the linear network, though, as with the network with front end processors, this is a shallow minima. By comparing Fig. 7 and Fig. 11, it can be seen that the network with front end processors performs better than the one without.

V. OPTIMAL DIVISION OF PROCESSING LOAD. INCLUSION OF SOLUTION TIME

In both Sections III and IV, the time taken for every other processor to report its solution back to the starting processor is assumed to be very short and can be ignored. As in the equal division of processing load case, in some applications this may not be true. There are many possible ways for the other processors to report their solutions back to the starting processor. As an example, the inclusion of this solution time in a linear network with front end processors is discussed here.

We consider the same linear network with front end processors as in Section III where all the processors in the network may have differing computing capabilities and the channel capacities between the processors may be different. The processor at the left end receives a burst of measurement data and is to share the measurement data with the other processors in the network. The entire process is similar to the one in Section III except for the inclusion of solution time. When the Nth processor finishes its computation, it starts sending back the solution to its left immediate neighbor. The transmission takes time T_S. Upon receiving the solution, the neigbor transmits the solution and its own solutions to its right immediate neighbor. This transmission takes $2T_S$. The solution sending process repeats itself until all the solutions are sent back to the starting processor. The timing diagram of the entire process is shown in Fig. 12.

In order to obtain maximum parallelism and a minimum time solution, the starting processor should compute its fraction of the processing load during the entire processing period, and the computing time $\alpha_i w_i T_{cp}$ of the ith processor should overlap exactly with the transmission time $(1 - \alpha_1 - \alpha_2 - \cdots - \alpha_i) z_i T_{cm}$ from the ith to the $i+1$st processor plus the cumulative solution transmission time $(N-i) z_{i-1} T_S + z_i T_S$ from the previous processors plus the computation time of the ith processor. With α_i defined as the actual fraction of the total processing load, the equations are shown below.

For the first processor,

$$\alpha_1 w_1 T_{cp} = (1 - \alpha_1) z_1 T_{cm} + \alpha_2 w_2 T_{cp} + z_1 T_S \quad (31)$$

and for the ith processor,

$$\alpha_i w_i T_{cp} = (1 - \alpha_1 - \alpha_2 - \cdots - \alpha_i) z_i T_{cm} + \alpha_{i+1} w_{i+1} T_{cp} + z_i T_S + (N-i) z_{i-1} T_S, \quad i = 2, 3, \ldots, N \quad (32)$$

and

$$\alpha_N = 1 - \alpha_1 - \cdots - \alpha_{N-1} \quad (33)$$

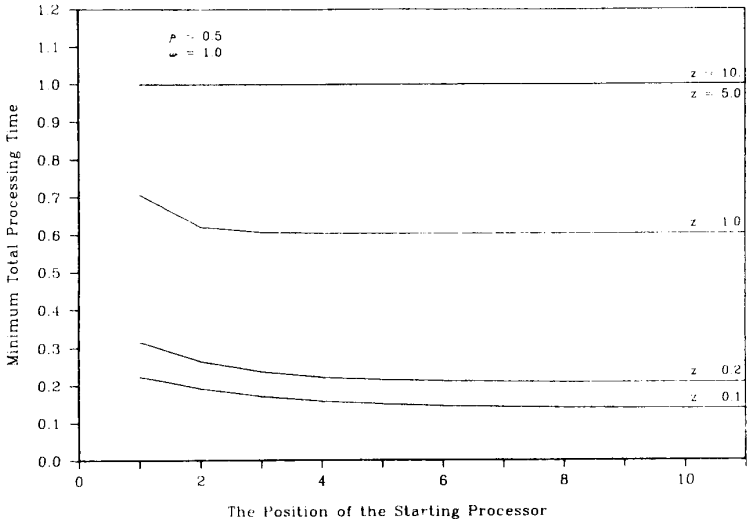


Fig. 11. Origination at network interior with no front end processor.

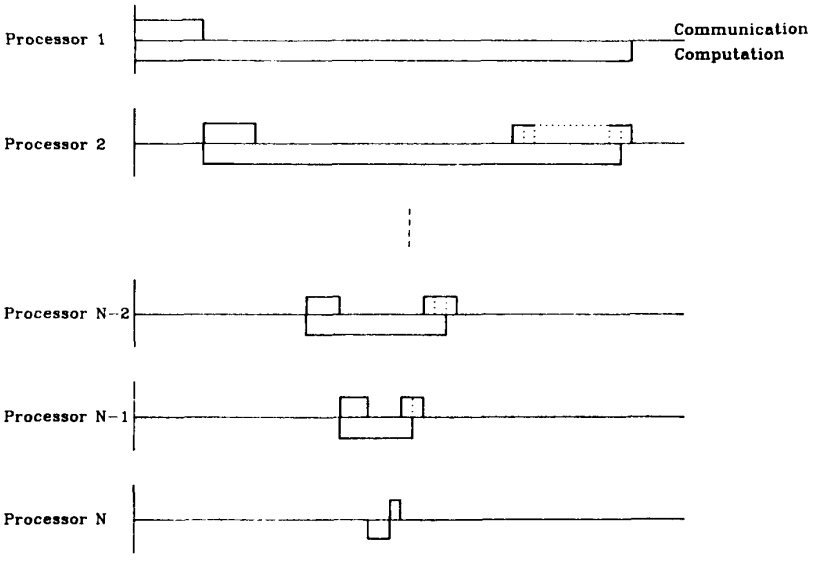


Fig. 12.

where w_is are inversely proportional to the speed of the ith processor and z_is are inversely proportional to the channel speed between the ith and the $i+1$st processor.

There are totally N linear equations and N unknowns, so the α_i's that minimize the total processing time can be determined, and the total processing time is

$$T_t = \alpha_1 w_1 T_{cp}. \tag{34}$$

There are other ways of reporting the solution back to the starting processor. For instance, the solution time may be of fixed length when transmitted from each processor, or solution time may be proportional to the length of the data processed by a particular processor. Similar sets of linear equations can be formed, and α_is and T_t can thus be solved as well.

VI. CONCLUSION

In this paper a linear network of intelligent processors is examined. The interaction between communication and computation is the most interesting aspect of this work. It is of practical interest and the optimal division of processing load can be determined relatively simply. It is also interesting that minimum time solutions do not require large number of processors in the network.

APPENDIX I

It is demonstrated that for a minimum time solution all processors must stop at the same time. For simplicity the values of w_is, which are inversely proportional to the speed of the ith processor, and z_is, which are inversely proportional to the channel speed between the ith and the $i+1$st processor, are assumed to be one. This is done without loss of generality. The problem is assumed to originate from the left boundary and all the processors are equipped with a front end processor for communication off-loading.

PROOF. Consider the two rightmost, the $N-1$st and the Nth, processors. The $N-1$st processor keeps α fraction of data that it has received and transmit the remaining $1-\alpha$ fraction to the Nth processor. There are two possibilities, either the $N-1$st or the Nth processor will stop computing first. Both cases are discussed below.

Case I

The Nth processor stops first

$$T_t = \alpha T_{cp} \tag{35}$$

$$\alpha T_{cp} \geq (1-\alpha)(T_{cm} + T_{cp}) \tag{36}$$

where T_t is the total processing time for these two processors, T_{cm} is the time to transmit the entire measurement data over the channel, T_{cp} is the time it takes for one processor to process the entire processing load, αT_{cp} is the computing time of the $N-1$st processor, and $(1-\alpha)(T_{cm}+T_{cp})$ is the sum of the transmission time from the $N-1$st to the Nth processor and the computing time of the Nth processor.

From (36) we have

$$\alpha \geq \frac{T_{cm} + T_{cp}}{T_{cm} + 2T_{cp}} \tag{37}$$

and from (35)

$$\begin{aligned} \min(T_t) &= \min(\alpha) T_{cp} \\ &= \left(\frac{T_{cm} + T_{cp}}{T_{cm} + 2T_{cp}}\right) T_{cp} \\ &= (1 - \min(\alpha))(T_{cm} + T_{cp}) \end{aligned} \tag{38}$$

where $\min(x)$ is the minimum of x. It can be seen from

(38) that it is preferable to stop the processors simultaneously rather than have the Nth stop first.

Case II

The $N-1$st processor stops first

$$T_t = (1-\alpha)(T_{cm}+T_{cp}) \tag{39}$$

$$\alpha T_{cp} \leq (1-\alpha)(T_{cm}+T_{cp}). \tag{40}$$

From (40) we have

$$\alpha \leq \frac{T_{cm} + T_{cp}}{T_{cm} + 2T_{cp}} \tag{41}$$

and from (39)

$$\begin{aligned} \min(T_t) &= (1-\max(\alpha))(T_{cm}+T_{cp}) \\ &= \left(1 - \frac{T_{cm} + T_{cp}}{T_{cm} + 2T_{cp}}\right)(T_{cm}+T_{cp}) \\ &= \max(\alpha)T_{cp} \end{aligned} \tag{42}$$

where $\max(x)$ is the maximum of x. From (42) it is apparent that it is preferable to stop the processors simultaneously rather than stop the $N-1$st processor first. Together with Case I this shows that the minimal time solution results when both processors stop simultaneously.

One can extend the above result to the $N-2$nd processor by replacing the $N-1$st and Nth processors by an equivalent processor with processing time αT_{cp} (35). In this way the result can be generalized to an entire network of processors. For other linear networks mentioned in the paper, the proof is similar.

APPENDIX II

When front end processors are not included for communication off-loading there are certain parameter values for which distributing the computational load to other processors does not result in a time savings. To see this, consider two adjacent processors. The time taken by the first processor is

$$(1-\alpha_1)zT_{cm} + \alpha_1 w_1 T_{cp}. \tag{43}$$

The time taken by the second processor is

$$(1-\alpha_1)zT_{cm} + (1-\alpha_1)w_2 T_{cp}. \tag{44}$$

Consider these two equations as functions of α_1. The second equation has a negative slope. The first equation has a positive slope if $w_1 T_{cp} > zT_{cm}$ and a negative slope if $w_1 T_{cp} < zT_{cm}$. In the former case there is a value of α_1, $0 < \alpha_1 < 1$, which minimizes the computational time. In the latter case computational time is minimized when α_1 equals one. This corresponds to no distribution of the computational load.

The condition for distribution of the computational load, $w_1 T_{cp} > zT_{cm}$, intuitively means that the communication time over the link between the processors must be less than the computation time of the first processor in order to achieve a net savings through distribution of the processing load.

REFERENCES

[1] Chong, C.Y., Mori, S., and Wishner, R.P. (1982)
Distributed estimation in distributed sensor networks.
Presented at the American Control Conference, Arlington, Va., 1982.

[2] Chong, C.Y., Tse, E., and Mori, S. (1983)
Distributed estimation in networks.
Presented at the American Control Conference, San Francisco, Calif., 1983.

[3] Delaney, J.R., Lacoss, R.T., and Green, P.E. (1983)
Distributed estimation in the MIT/LL DSN testbed.
Presented at the American Control Conference, San Francisco, Calif., 1983.

[4] Hughes, R.P., and Tenny, R.R. (1980)
Multiobject tracking algorithm for a distributed sensor network.
In *Proceedings of the 3rd MIT/ONR Workshop on Distributed Information and Decision Systems*, Sept. 1980.

[5] Lacoss, R., and Walton, R. (1978)
Strawman design for a DSN to detect and track low flying aircraft.
In *Proceedings of the Distributed Sensor Nets Conference*, Carnegie-Mellon University, Pittsburgh, Pa., Dec. 1978, pp. 41–52.

[6] Lacoss, R. (1982)
Overview of the distributed sensor networks program at Lincoln Laboratory.
In *Proceedings of the Distributed Sensor Networks Workshop*, M.I.T. Lincoln Lab, Jan. 1982.

[7] Megna, V.A. (1981)
Dispersed sensor processing mesh project.
In *Proceeding 303*, Advisory Group for Aeronautical Research and Development Conference on Tactical Airborne Distributed Computing and Networks, Roros, Norway, 1981.

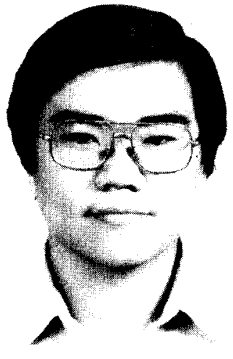
[8] Morgen, S.D., and Paulicks, D. (1983)
Distributed sensing for acoustic surveillance.
Presented at Melcon '83, Mediterranean Electrotechnical Conference, Athens, Greece 1983.

[9] Tenny, R.R., and Sendell, N.R., Jr. (1981)
Detection with distributed sensors.
IEEE Transactions on Aerospace and Electronic Systems, AES-17, 4 (July 1981), 501–510.

[10] Tong, R.M., Tse, E., and Wishner, R.P. (1981)
Distributed hypothesis formation in sensor fusion systems.
In *Proceedings of the 20th IEEE Conference on Decision and Control*, 1981, 1421–1424.

[11] Verriest, E., Friedlander, B., and Morf, M. (1979)
Distributed processing in estimation and detection.
Presented at the IEEE 1979 Conference on Decision and Control, Fort Lauderdale, Fla., 1979.

[12] Wesson, R., Hayes-Roth, F., Buge, J.W., Stast, C., and Sunshine, C.A. (1981)
Network structure for distributed situations assessment.
IEEE Transactions on Systems, Man and Cybernetics, SMC-11, 1 (Jan. 1981), 5–23.



Yuan-Chieh Cheng was born in Taichung, Taiwan, in 1962. He received the B.E. degree in electronic engineering from Tamkang University, Tamsui, Taiwan, in 1984, and the M.S. degree in electrical engineering from SUNY at Stony Brook, New York, in 1986. He is currently a Ph.D. candidate in electrical engineering at Stony Brook.

Yuan-Chieh Cheng's research interests include stochastic and queueing processes, performance evaluation of communication and computer systems and parallel architecture. He is a member of the Institute of Electrical and Electronics Engineers.

Thomas G. Robertazzi (S'75—M'81) received the Ph.D. from Princeton University, N.J., in 1981 and the B.E.E. from Cooper Union, N.Y., in 1977.

Dr. Robertazzi is presently an Assistant Professor of electrical engineering at SUNY at Stony Brook. He was with Manhattan College during 1982–1983. His research interests involve the performance evaluation of computer and communication systems. He is a member of the Institute of Electrical and Electronics Engineers, the Association for Computing Machinery, and the Operations Research Society of America.