Distributed Computation for a Hypercube Network of Sensor-Driven Processors with Communication Delays Including Setup Time

D. Antony Louis Piriyakumar and C. Siva Ram Murthy

***Abstract*—In this paper, the problem of optimal distribution of measurement data to be processed in minimal time on a hypercube network of sensor driven processors is considered. An analytical model is developed for solving the problem efficiently. Unlike the previous models, this model considers**

1) **explicitly the setup time which constrains exploiting all the available processors;**
2) **simultaneous use of links to expedite the communication;**
3) **partial solution combining time to encompass wider class of related problems.**

By deriving a *lower bound* on the amount of data to be received by a processor for efficient distribution, a new technique called *fractal hypercube* is introduced here to get the optimal solution with fewer processors. An optimal iterative method for hypercubes and a near-optimal recursive method with a refinement are presented for the same with the analysis. The effect of varying the originating processor and the choice of fractal hypercube are discussed with an effective technique called *processor isomorphism*. This study reveals that always the fractal hypercubes outperform the other two methods, the optimal iterative method for hypercubes and the near-optimal method.

I. Introduction

In the recent years, there has been considerable interest focused by researchers on distributed intelligent sensor networks [1]–[10], [13], [14], [16]–[19] due to the requirements of military surveillance systems and other civilian applications. An intelligent sensor network is a collection of interconnected processors with sensing, computational, and communication ability. A major issue in distributed intelligent sensor networks is how to solve a computational problem in minimal amount of time. The motivation is a situation where one processor (sensor), called the originating processor, receives a burst of processing load (measurement data), distributes the load optimally to the other sensor driven processors to obtain the benefits of distributed processing, and receives the partial solutions from the recipient processors (children processors). Since the load distribution has a direct impact on processing the measurement data in minimal amount of time (optimal solution), a naive approach may result in excessive interprocessor communication and ineffective utilization of the processors which might nullify any gain from distributed processing. In short, the problem here is to determine the best distribution of processing load among the sensor driven processors.

Recently, a bus-oriented load sharing for a linear network of sensor driven processors has been considered to solve the above problem [1]. But reporting back the partial solutions to the originating processor was not considered and hence the optimal time is achieved when all processors stop at the same time. However, this was included and dealt in [4]. Yet the intricate problem of setup time involved in interprocessor communication was not given the due importance and hence not considered. Moreover, this is restricted only to a system of linear array of processors. This restriction also was removed in [5] by considering a general tree network of processors. Nevertheless, the setup time and the effective use of all the available communication links simultaneously were not considered.

Manuscript received August 20, 1995; revised June 27, 1996 and May 1, 1997. This work was supported by the Department of Science and Technology, Government of India, New Delhi.

The authors are with the Department of Computer Science and Engineering, Indian Institute of Technology, Madras 600 036, India (e-mail: murthy@iitm.ernet.in).

Publisher Item Identifier S 1083-4427(98)01586-0.

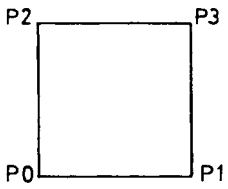


Fig. 1. Two-dimensional hypercube.

In this paper, we have considered a hypercube architecture (an example of a two-dimensional (2-D) hypercube (dimension, $n = 2$) is shown in Fig. 1). The hypercube architecture has four major characteristics [12] namely

1) high degree of connectivity $(= n)$;
2) less diameter $(= n)$;
3) simple routing mechanism;
4) isomorphic processors, apart from its commercial availability.

This paper is organized as follows. The need to consider the setup time separately, the effectiveness of using simultaneously all the available links, the inclusion of partial solution combining time, and the efficient use of fewer processors especially for hypercubes as *fractal hypercubes* are discussed in Section II with the condition for lower bound on the amount of load to be received for distribution. An analytical model [4], [5] for hypercubes and fractal hypercubes developed by considering the above factors to get the optimal solution is presented in Section III. A recursive method to get a near-optimal solution and a refinement to the recursive method to get (the near-optimal) solution closer to the optimal solution are given in Section IV. A new technique called *processor isomorphism* which reduces enormously the modeling required to get the optimal solution when the load is received by any processor in the distributed intelligent sensor network is discussed in Section V. The performance analysis of various methods for obtaining the optimal solution is presented in Section VI. The results are summarized in Section VII.

II. Enhanced Model for Optimal Load Distribution

A. Inclusion of Setup Time

In hypercube systems, when a processor wants to send some data to a neighboring processor (where a direct link exists between them), it incurs an overhead called setup time. This overhead is due to setting up of DMA channels and buffering delays [12] involved in the interprocessor communication of the data (in Intel iPSC hypercubes, it is 1.7 ms [12]). In most of the available literature [1], [4], [5], the setup time was neither considered separately nor included with the communication time properly. When the measurement data is distributed among the processors, it is probable that the time to communicate a fraction of data can be in the order of setup time. This is possible because to minimize the time as much as possible, the load is distributed among maximum number of processors. When the communication time is in the order of the setup time, first we have to check whether distributing and gathering back the solution will incur lesser time than processing in a single processor. For example,

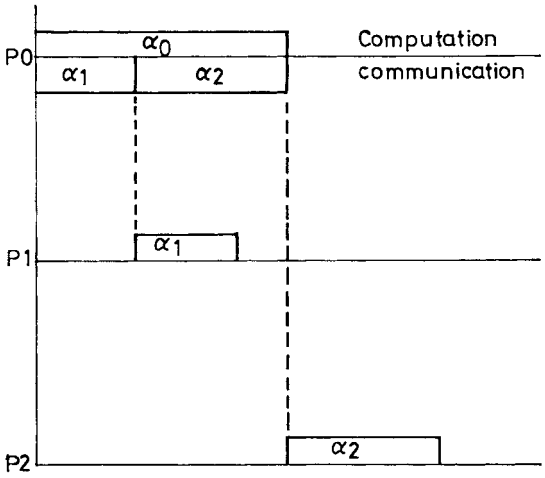


Fig. 2. Sequential distribution.

consider a one-dimensional (1-D) hypercube. Assume the processing speeds (p bytes/s) of the processors are the same, the link capacity is c bytes/s, the setup time is τ s, the partial solution length is s bytes and the partial solution combining time is β s. Also assume that the load l bytes is applied at the originating processor. Let α_i be the fraction of load assigned to processor i. Then to get the optimal solution considering 1-D case, the following conditions must be satisfied:

$$\alpha_0 l/p = \alpha_1(l/p + l/c) + 2\tau + s/c \quad (1)$$

$$\alpha_0 + \alpha_1 = 1 \quad (2)$$

$$\alpha_0 l/p + \beta < l/p. \quad (3)$$

Equation (1) is for optimality as when the originating processor ends its computation, the partial solution from the other processor should reach at the same time. Equation (2) just implies that the load distribution is proper. The third condition (3) states more importantly, the necessity of the distributed optimal solution to be less than the sequential solution.

B. Simultaneous Use of Links

In the previous work [4], [5], even though the computation and communication were carried out simultaneously at each processor in the general tree network [5], from the parent processor the data is sent to the children processors only sequentially. Many of the commercially available hypercubes have a number of links (channels per processor) on which each processor can simultaneously send or receive data [12]. For example, JPL Mark-III has eight channels per processor, Intel iPSC has seven channels per processor, Ametek System/14 has eight channels per processor, and Ametek System/14 beta has eight channels per processor (node). Hence the links have to be used more effectively while distributing the measurement data. Consider a 2-D hypercube shown in Fig. 1. Fig. 2 depicts the scenario when processor $P0$ receives the load and distributes to its children (neighboring) processors $P1$ and $P2$ sequentially. Fig. 3 portrays vividly the simultaneous use of the links. The equations corresponding to them are discussed in Section III. It is evident from Fig. 3 that this method (the simultaneous use of links) results in a better solution than the previous method as the starting time of processing in the children processors is earlier than in the previous sequential method.

C. Inclusion of Solution Combining Time

Adding one more factor, the partial solution combining time, to the question of "what to compute and what to communicate," the core problem of distributed fusion system design [4], [18], encompasses still wider class of problems. However, there are situations in which the partial solution can be combined and then communicated to the parent processor. Our enhanced model can be reduced to the previous models [4], [5] in case such is not possible by setting the value of the solution combining time to zero.

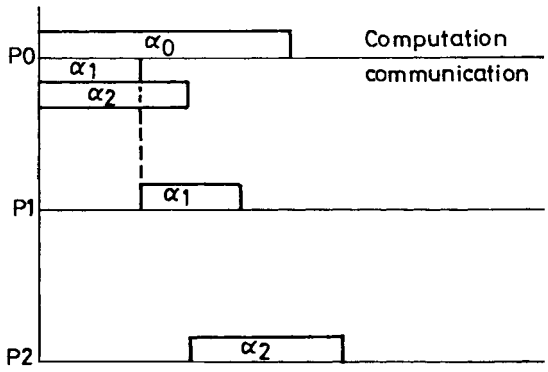


Fig. 3. Simultaneous distribution.

D. Fractal Hypercubes

Once a load is applied to the originating processor, then trying to employ all the processors in the hypercube to get the optimal solution considering wide range of lengths of measurement data, may lead to nonoptimal solutions. Hence, it is mandatory to find the minimum amount of data that has to be received by a processor for further distribution of the data to minimize the processing time.

1) Lower Bound on the Amount of Data for Efficient Distribution: Taking the 1-D hypercube network of processors with system parameters viz. p, c, and τ, and the problem specific parameters viz. l, s, and β into consideration, (1)–(3) are now used. Substitute the value of $\alpha_1 = 1 - \alpha_0$ in (1) and solve for α_0. Then, substituting α_0 in (3) and solving for l we get

$$l > p(2\tau + s/c + \beta(2 + p/c)). \quad (4)$$

So, given the system parameters, p, τ, and c and the problem specific parameters, l, β, and s, the load l given by (4) (condition for lower bound) is the lower bound on the amount of data that has to be received so that it can be distributed for processing efficiently.

This is not only true for 1-D case but generally if a processor P_i receives α_i amount of data, if α_i is not greater than the lower bound specified in (4), the processor P_i itself can process the data without attempting to distribute it as this will be more efficient.

2) Need for Fractal Hypercubes: Let us consider a 2-D hypercube case. Let P_0 be the originating processor receiving the load l. Let the load l be greater than the lower bound and is divided between the processors P_0 and P_1 as α_0 and α_1, respectively, satisfying the conditions. Since, two more processors are there and if α_0 and α_1 amount of load l could be further divided, one must do it optimally. It is equally likely that either α_0 or α_1 alone may satisfy the lower bound condition. In this case, either one should have stopped with 1-D hypercube itself (carved from the given 2-D hypercube) or one would have over exploited the other two processors also by distributing the load to them irrespective of the lower bound condition. In either case, one cannot get the optimal solution.

In this context, we define a network of sensor driven processors which is the same as the hypercube of dimension n excepting for the number of processors (m) present is less, i.e., $(m < 2^n)$ as a *fractal hypercube* as the dimension of this fractal hypercube is a fraction $(\log_2 m)$. Fig. 4 gives an example of $\log_2 5$ hypercube carved out

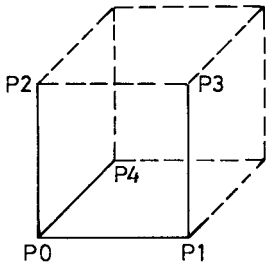


Fig. 4. Log_2five-dimensional hypercube.

from a three-dimensional (3-D) hypercube. It should be noted that there are only five processors now, yet the interconnection is the same as that of the 3-D hypercube for these processors present. The fractal hypercubes are now used in our model when a load is applied, to first find whether fewer number of processors are sufficient to get the optimal solution.

III. Analytical Model for Hypercube

A. n-Dimensional Hypercube

The major issue here is that a processor can be reached from any other processor in several ways. However, to fix only one path, the easiest way is to consider the originating processor as the root and all other processors as nodes at each level of the spanning tree [11] corresponding to hypercube architecture where the levels denote the minimum number of hops required to reach the processors from the root or vice versa. The processors which do not have any child processor are christened as leaf processors. Yet another important point to be considered is that when any two paths lead to the same processor, prefer the path which is connected to lesser number of processors so that the communications can be faster as links are now used simultaneously. Moreover, start constructing the spanning tree level by level considering minimal hops between the originating processor and any other processor, and choosing the path having lesser number of processors. The spanning tree corresponding to hypercube architecture constructed as explained above, will have only n levels. Equations for an n-dimensional hypercube can now be setup as follows.

For any processor i which does not correspond to a leaf processor in the spanning tree of the hypercube architecture

$$\alpha_i l/p + j\beta = \alpha_h' l/c + \alpha_h l/p + 2\tau + s/c + r_h\beta \tag{5}$$

$$\alpha_i l/p + j\beta = \alpha_k(l/c + l/p) + 2\tau + s/c \tag{6}$$

where j takes value from 0 to $r_i - 1$ (r_i, number of children processors of processor i), processors k and h are children processors of processor i, (5) corresponds to processor h being a nonleaf processor, and (6) corresponds to processor k being a leaf processor. The value of j for a child processor k or h, is chosen to be possibly higher whenever the length of the path from the processor i through processor k or h, is more. However, when the length of the paths of different children processors is the same, the value of j is chosen for each one of them randomly and it will not affect the optimal solution as the corresponding processors are isomorphic processors as explained in Section V. As each equation corresponds to an edge in the spanning tree with 2^n nodes, there will be $2^n - 1$ equations of this type as a tree with x nodes will have $x - 1$ edges [11]. There are 2^n, α_i variables involved each corresponding to the respective processor i. There are say m interior nodes in the spanning tree excepting the root. There will be m, α_i' variables involved each corresponds to the respective processor i which are not leaf processors. Corresponding to these we have m equations of the form

$$\alpha_i' = \alpha_i + \sum \alpha_k + \sum \alpha_h' \tag{7}$$

where the first summation involves the sum of fractions of load assigned to the children processors (k) of processor i which are leaf processors and the second summation involves the sum of fractions of load communicated to the children processors (h) of processor i which are nonleaf processors. There will be one more equation of the this type for the originating processor (the processor that receives the burst of measurement data) as

$$\alpha_0 + \sum \alpha_k + \alpha_h' = 1. \tag{8}$$

So, there are as many variables as the number of equations to form a system of linear equations. Solving these equations by varying the dimension either incrementally or like binary search, we can get the optimal solution. The binary search method will be better when n is large and the incremental method will be better when n is small.

For example, in a 2-D case (Fig. 1) the spanning tree has the processor $P0$ as the root, $P1$ and $P2$ (leaf processor) as its children at level 1 and $P3$ (leaf processor) as a child of $P1$ at level 2. With the usual notation including α_i', the amount of load communicated to processor i which in turn can be distributed further, we present the equations as follows:

$$\alpha_0 l/p = \alpha_2(l/p + l/c) + 2\tau + s/c \tag{9}$$

$$\alpha_0 l/p + \beta = \alpha_1' l/c + \alpha_1 l/p + 2\tau + s/c + \beta \tag{10}$$

$$\alpha_1 l/p = \alpha_3(l/p + l/c) + 2\tau + s/c \tag{11}$$

$$\alpha_0 + \alpha_2 + \alpha_1' = 1 \tag{12}$$

$$\alpha_1' = \alpha_1 + \alpha_3. \tag{13}$$

These five equations have five unknowns $(\alpha_i, i = 0 \ldots 3, \alpha_i')$ only. So it can be solved using any method which solves a system of linear equations viz. Gaussian elimination method [15]. The optimal solution (total completion time for the measurement data) is $\alpha_0 l/p + n\beta$, where n is the dimension of the hypercube.

B. Fractal Hypercube

In the same way as explained in the previous subsection, with lesser-dimensional hypercube we could get better solution, we have extended to lesser fraction-dimensional hypercubes, i.e., instead of varying dimension wise, we vary processor wise also within the same dimension. The same analysis as in the hypercubes is also valid in the fractal hypercubes. The problems related to multiple paths and minimum hops are resolved in the same way as hypercubes. We give the analysis for $\log_2 5$-dimensional hypercube and explain how for higher fraction-dimensional hypercubes it can be extended. Fig. 4 gives an example of $\log_2 5$-dimensional hypercube carved out from a 3-D hypercube

$$\alpha_0 l/p = \alpha_4(l/p + l/c) + 2\tau + s/c \tag{14}$$

$$\alpha_0 l/p + \beta = \alpha_2(l/p + l/c) + 2\tau + s/c \tag{15}$$

$$\alpha_0 l/p + 2\beta = \alpha_1' l/c + \alpha_1 l/p + 2\tau + s/c + \beta \tag{16}$$

$$\alpha_1 l/p = \alpha_3(l/p + l/c) + 2\tau + s/c \tag{17}$$

$$\alpha_0 + \alpha_2 + \alpha_4 + \alpha_1' = 1 \tag{18}$$

$$\alpha_1' = \alpha_1 + \alpha_3. \tag{19}$$

As the number of processors alone is lesser than that of the hypercubes, i.e., lesser number of nodes in the spanning tree of the fractal hypercube than that of the hypercube, the remaining analysis is the same. There will be as many number of equations as the

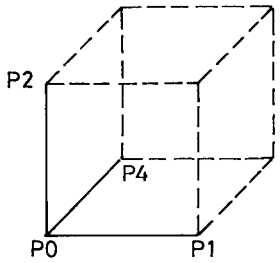


Fig. 5. Log_2four-dimensional fractal hypercube.

Fig. 6. Log_2five-dimensional hypercube.

number of variables forming the system of linear equations whose solution is optimal as far as that fractal hypercube is concerned. Given the load, finding the optimal solution and the optimal fractal dimension is computed in the same way either by incremental method or like binary search with the difference that instead of varying the dimension, here the fractal dimension is varied. Here it should be noted that this fractal dimension q $(q = \log_2 p)$ depends on p which is the number of processors present in the fractal hypercube. There are only $2^n - 3$ [neglecting zero-dimensional (0-D)], 1-D, and n-dimensional hypercubes) fractal-dimensional hypercubes carved out from the hypercube with dimension n. Hence, the processing will also be finite even when fractional dimensions are dealt with. It should be also noted that $(n - 1)$ fraction-dimensional hypercube carved out from the n-dimensional hypercube, is a fractal hypercube as the interconnection of this fractal hypercube will not be same as $(n - 1)$-dimensional hypercube. For example, a 2.0-fraction-dimensional hypercube carved out from 3-D hypercube is different from 2-D hypercube as shown in Figs. 5 and 1, respectively. In most of the cases, m-fraction-dimensional hypercube $(m > \log_2 n1)$, carved out from $n1$-dimensional hypercube will not be same as the m-fraction-dimensional hypercube carved out from $n2$-dimensional hypercube $(n2 > n1)$ as shown in Figs. 4 and 6 as the spanning trees for these hypercubes are different. So, depending upon the dimension of the hypercube from which it is carved out, the analysis of a fractal hypercube varies. However, the method we have presented, will be applicable to all these cases without exception and care should be taken to differentiate between the same fractal hypercubes carved out from hypercubes of different dimensions.

One more point relating to fractal hypercubes is that given a fraction dimension there can be many fractal hypercubes of this fraction dimension carved out from a given n-dimensional hypercube. For example, $\log_2 5$ fraction-dimensional hypercubes carved out from 3-D hypercube, are given in Fig. 7(a)–(f). It is amazing to see that all these fraction-dimensional hypercubes are same due to processor isomorphism (explained in Section V) and their analysis is also same. Hence there is no need to take them into consideration while finding the optimal solution for the given load. It should be noted that fractal hypercubes always give better optimal solutions than the (optimal)

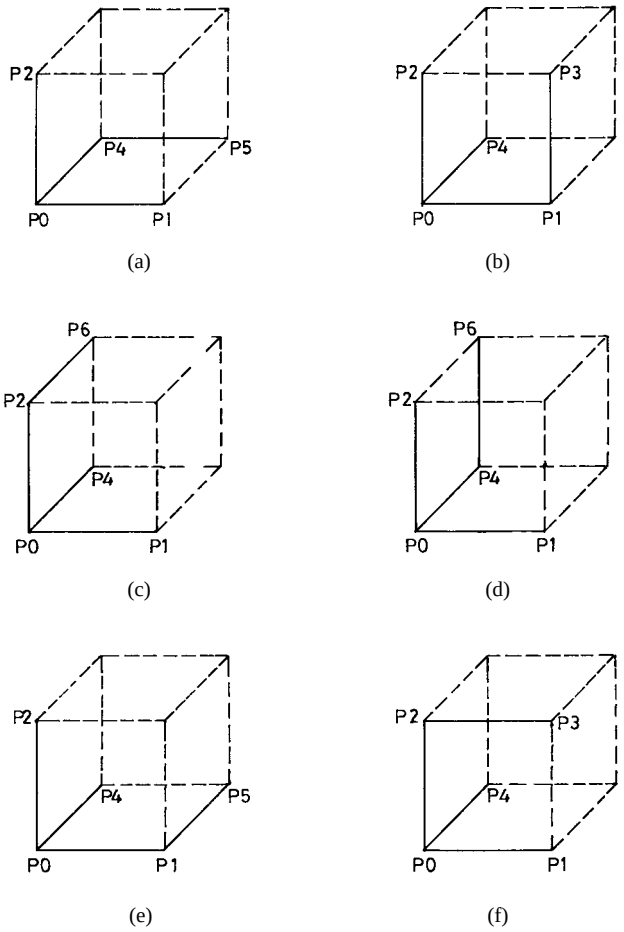


Fig. 7. Isomorphic $\log_2$five-dimensional hypercube.

solutions given by hypercubes. For example, when $\tau = 5$, $p = 1$, $l = 200$, $c = 100$, $s = 10$, and $\beta = 1$, the optimal dimension is three and the optimal solution is 43.840 whereas in case of fractal hypercube, the optimal fraction dimension is $\log_2 12$ and the optimal solution is 27.71. An extensive study on this is presented in Section VI.

IV. A Simple Near-Optimal Recursive Method for Hypercubes

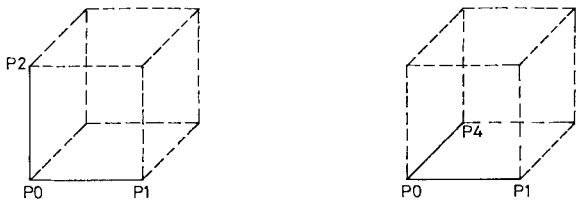
A. Basic Theory of the Method

The main problem with the previous method is that one has to derive equations separately for each individual case especially for same fractal hypercubes carved out from hypercubes of different dimensions. Moreover, when the dimension of the hypercube becomes large, the complexity of deriving the equations also increases very much. The methods to solve this linear system of equations may also pose the problem viz. ill-conditioned as the matrix becomes very sparse [15]. Hence we have developed a method which is recursive in nature and gives near-optimal solutions which can be improved by the very same recursive method both for hypercubes and fractal hypercubes.

The main idea behind this method is that any n-dimensional hypercube can be considered as two $(n-1)$-dimensional hypercubes interconnected pairwise with 2^{n-1} links. Thus, it can be viewed as a 1-D hypercube with two processors replaced by two $(n - 1)$-dimensional hypercubes taking into consideration one link out of 2^{n-1} links. So, when a load is applied at a processor, each fraction of the load is divided recursively in the $(n-1)$-dimensional

(a) (b)

Fig. 8. (a) $\mathrm{Log}_2 4$-fractal hypercube by fractal analytical method and (b) $\mathrm{Log}_2 4$-fractal hypercube by near-optimal recursive method.



(a) (b)

Fig. 9. (a) $\mathrm{Log}_2 3$-fractal hypercube by fractal analytical method and (b) $\mathrm{Log}_2 3$-fractal hypercube by near-optimal recursive method.

hypercubes. This process is repeated until the current dimension of the fraction of the load applied becomes one or no more division will improve the solution. Then the solutions are gathered back from each $(n-1)$-dimensional hypercubes. As we are considering only a 1-D hypercube, when the load $(\alpha_i' l)$ is applied to processor i, it divides it into $\alpha_i l$ to be computed by itself and $(\alpha_i' - \alpha_i)l$ is sent to the other processor (in the 1-D hypercube). To get the optimal solution in this 1-D case, the condition is

$$\alpha_i l/p = (\alpha_i' - \alpha_i)(l/p + l/c) + 2\tau + s/c. \tag{20}$$

As all except α_i is known, it can be computed. Now such a division of load will be better than the 0-D hypercube when,

$$\alpha_i < \alpha_i' - (p/l)\beta. \tag{21}$$

If the condition in (21) is satisfied then the given load is divided. Otherwise, it is solely processed by the processor i.

B. Salient Features of the Method

We now explain with an example why in some cases the above method does not lead to optimal solutions but only near-optimal solutions. For example, take the $\log_2 4$ dimension fractal hypercube carved out from a 3-D hypercube by optimal method and recursive method. Former method follows the spanning tree with minimal number of hops to reach at each level in choosing the processors in that fractal hypercube as shown in Fig. 8(a). Latter method follows dimension by dimension expansion in choosing the processors as shown in Fig. 8(b). Here it is obvious to note that the optimal method exploits the use of all the available links simultaneously unlike the recursive method which leads to only near-optimal solutions. However, in some cases the fractal hypercubes carved out from a given dimension are the same both in the optimal method as well as in the recursive method (by processor isomorphism). For example, $\log_2$ 3-D fractal hypercube carved out from 3-D hypercube in both the methods are shown in Fig. 9(a) and 9(b), respectively.

The important point to notice is that when the dimension of the fractal hypercube is an integer, fractal hypercube carved out from a given hypercube by the recursive method is the same as the integer-dimensional hypercube. For example, $\log_2$four–dimension fractal hypercube carved out from 3-D hypercube by the recursive method is the same as the 2-D hypercube as shown in Fig. 8(b). Hence, this method maneuvers both the hypercubes and fractal hypercubes in an efficient way.

C. Refinement to the Method

The recursive method gives near-optimal solutions in most of the cases as explained earlier. The question is how to improve the recursive method to bridge the gap between the solution presented by the method and the optimal solution. The optimal solution is achieved in case of 1-D hypercube when (1)–(3) are satisfied. If α_0 is varied, the solution will also be different from the optimal solution. Now for n-dimensional hypercube, store the value of α_0 in the first division. This is the amount of fraction (α_0) with the load $(\alpha_0 l)$ kept for processing by itself and the $(\alpha - \alpha_0)l$ sent to the other half of the hypercube for processing. Set epsilon, ε, to a desired small fraction (say, 0.01). Add with α_0 and repeat the same computation (instead of the first divide, use this value) for the optimal dimension computed by the recursive method. Similarly subtract ε from α_0 and repeat the same computation for the optimal dimension computed by the recursive method. Set the new optimal solution as the minimum of the already available optimal solution, solution got by adding ε to α_0, and solution got by subtracting ε from α_0. If this new solution is the solution got by adding ε to α_0 then repeat adding until the solution is not better. Do the same for subtraction also. The choice of ε also plays a vital role. Less value for ε leads to more time to converge and relatively large value of ε impedes bridging the gap between it (near-optimal solution) and the optimal solution.

V. The Effect of the Position of Originating Processor

It is well known that the longer the communication delay, the longer the total processing time. Hence, the originating processor has an important role to play. If the originating processor in the hypercube network of processors is different, does the method need any change to be made? In case of a linear network, the total processing time is minimized when the originating processor is at the center [1]. Now, to find the effect of the position of the originating processor, have we to try with all the processors to arrive at a conclusion? We present a new technique, called processor isomorphism, which groups the processors so that the analysis can be done for groups and not for individual processors. This technique can be applied to any type of network of processors. First, we define any two processors $P1$ and $P2$ to be isomorphic iff

1) all their physical characteristics viz. processing speed, memory capacity, etc. are the same;
2) if there exists r number of k distant neighbor processors for $P1$, only the same r number of k distant neighbor processors should be there for $P2$.

Moreover, pairwise those neighbor processors of $P1$ and $P2$ should be isomorphic.

Take for example a linear network of processors shown in Fig. 10. The isomorphic groups are $(P0, P4)$, $(P1, P3)$, and $P2$. $P0$ is not isomorphic to $P1$ because $P0$ has a neighbor at a distance of 4, which $P1$ does not have. $P2$ is not isomorphic to $P1$ because $P2$ has two neighbors at a distance of 2 whereas $P1$ has only one neighbor at a distance of 2. Even though $P1$ has two neighbors at a distance of 1 as $P2$, pairwise $P0$, a neighbor of $P1$ at a distance of 1 is not isomorphic as seen before to $P1$, a neighbor of $P2$ at a distance of 1. Interestingly, in a homogeneous hypercube network of processors of any dimension (n), all processors are isomorphic to each other forming only one group. This enormously reduces the need to derive

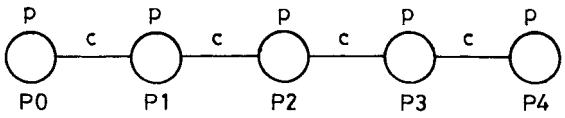


Fig. 10. Linear array of five homogeneous processors.

equations considering each processor as the originating processor and solving them to get optimal solutions.

One more use of processor isomorphism is to group fractal hypercubes that are carved out from the hypercube, as there can be many fractal hypercubes of the same fractal dimension. This will be helpful for us to choose the fractal hypercube which gives the optimal solution. We consider two fractal hypercubes to be isomorphic as far as both are carved out from the same homogeneous hypercube, if both of them have same number of isomorphic groups and in each isomorphic group the number of elements are also same. For example, a set of fractal hypercubes of dimension $\log_2 5$ is given in Fig. 7(a)–(f). The isomorphic groups of Fig. 7(b) are $\{P0\}$, $\{P1, P2, P4\}$, and $\{P3\}$. The isomorphic groups of Fig. 7(d) are $\{P0\}$, $\{P1, P2, P4\}$, and $\{P6\}$. Here, all the fractal hypercubes in this set are isomorphic. In fact, it is obvious to see that all the fractal hypercubes of a given fractal dimension carved out from the hypercube are all isomorphic. So, we can choose any one of the fractal hypercubes of a given fractional dimension carved out from a hypercube and this leads to optimal solution as far as that fractal hypercube is concerned.

VI. Performance Analysis of the Methods

We have programmed all the three methods in C language under the UNIX environment on Sun 3/50 workstations. We have varied the setup times and computed the optimal solutions by the three methods for a specified range of data. It may be noted from the graphs in Figs. 11–13 corresponding to each method that the setup time constraints the use of all processors especially in hypercubes to obtain the optimal solution. The above three graphs are drawn with the values of the parameters set as $l = 100$, $p = 1$, $c = 100$, $s = 10$, $\beta = 1$ apart from varying the setup times as shown. For example, when $\tau = 5$, $l = 100$, $p = 1$, $c = 100$, $s = 10$, and $\beta = 1$ in the proper units, the optimal solution 37.397 is achieved at 2-D hypercube, whereas the same parameters excepting $\tau = 0$, permit all the 16 processors to be used to give the optimal time of 10.903 units. This clearly depicts that setup time should be considered to get both realistic and optimal solution. However, when τ is very small the difference is only in the optimal solution and not in restricting the fullest use of processors.

The crux of this paper is that the fractal hypercube always outperforms the other two methods, as is evident from the graph in Fig. 14. This graph is drawn with the values of the parameters set as in the previous graphs in addition with $\tau = 2$. For example, when $\tau = 2$, $l = 100$, $p = 1$, $c = 100$, $s = 10$, and $\beta = 1$, the hypercube gives the optimal solution of 22.011 at the optimal dimension 3, the near-optimal division method gives 21.392 with 15 processors, and the fractal hypercube method gives 18.916 at the optimal dimension $\log_2 12$. Another important inference contributed here is that the near-optimal solution method gives better solution than the hypercube method and as close as possible to the fractal hypercube which is obvious from the graph in Fig. 14. To cite an example, when $\tau = 5$, $l = 100$, $p = 1$, $c = 100$, $s = 10$, and $\beta = 1$, the hypercube method gives the optimal solution 37.397 at the optimal dimension 2, the near-optimal solution method gives

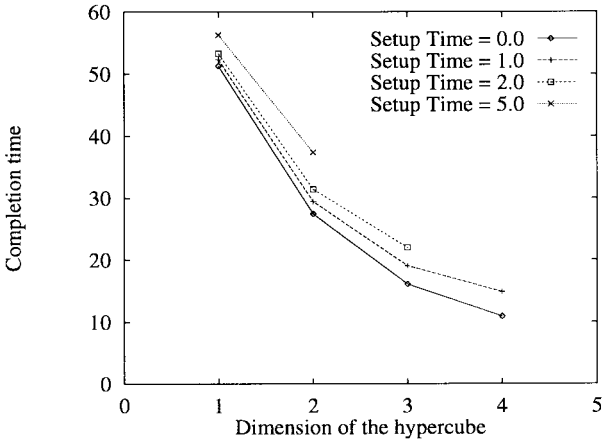


Fig. 11. Effect of setup time on completion time in hypercubes.

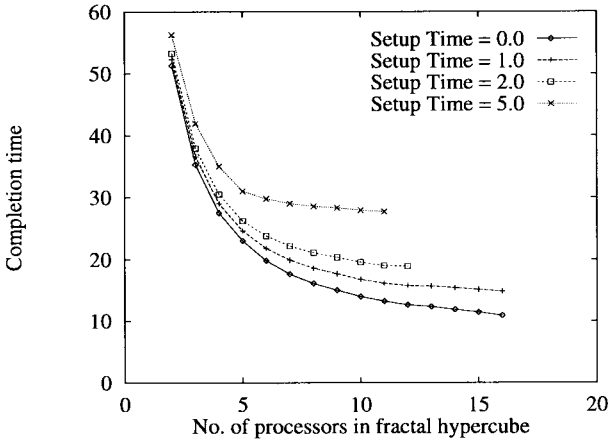


Fig. 12. Effect of setup time on completion time in fractal hypercubes.

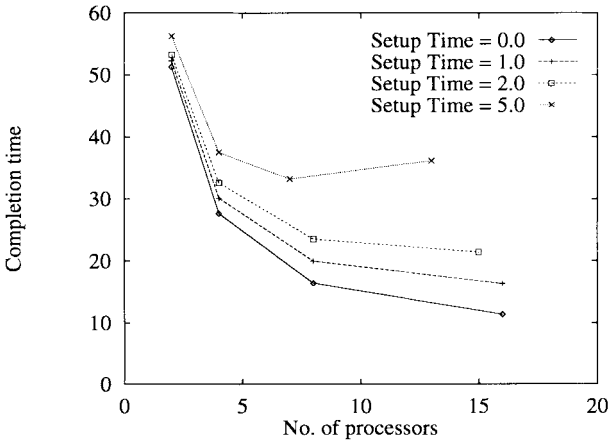


Fig. 13. Effect of setup time on completion time in near-optimal method.

33.212 using seven processors, and the fractal hypercube method gives 27.716 at the optimal dimension $\log_2 11$. Especially, when τ is small, the gap between the optimal solution obtained by the fractal method and the near-optimal division method is very small. When

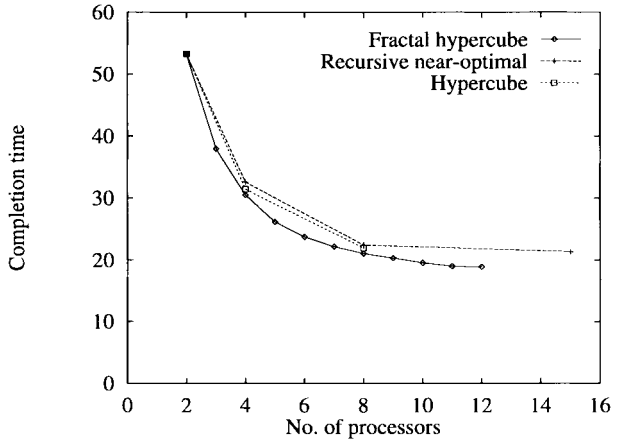


Fig. 14. Comparison of the three methods.

$\tau = 0.1$, $l = 100$, $p = 1$, $c = 100$, $s = 10$, and $\beta = 1$, the optimal solution produced by the fractal method is 11.296 with dimension $\log_2 16$ (a fractal hypercube of dimension $\log_2 2^n$, carved from the hypercube of dimension n is the same as the hypercube) and the near-optimal method gives 11.936 with 16 processors. The main emphasis is that given a hypercube, we need not use all the processors to get the optimal solution.

VII. Conclusion

In this paper, we considered the problem of optimally distributing the measurement data for processing in minimal time on a hypercube network of sensor driven processors. The setup time in the interprocessor communication is considered explicitly with the analysis of the effect specifying that it constrains the fullest use of all the processors. The improvement in the solution due to simultaneous use of links is also incorporated here. The partial solution combining time has been introduced to encompass wider class of problems in the solution time explicitly. A new technique, called fractal hypercube has been introduced and its impact on optimal solution has been discussed. An optimal iterative method with equations derived separately for each of the processors and a near-optimal recursive method to solve the same with a refinement to bridge the gap between the near-optimal solution and the optimal solution, have been suggested. From the analysis, we conclude that the fractal hypercubes always outperform the hypercubes. The role of setup time in these circumstances is also studied which demand a serious concern whether to send or to compute. Using the above methods, given the load (measurement data), the optimal solution and the optimal dimension for the optimal distribution of the measurement data are found efficiently.

Acknowledgment

The authors are grateful to B. Sreekant and the anonymous referees for providing comments and suggestions that greatly helped in improving this paper.

References

[1] B. Bataineh and T. G. Robertazzi, "Bus-oriented load sharing for a network of sensor driven processors," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, pp. 1202–1205, Sept./Oct. 1991.

[2] ——, "Ultimate performance limits for networks of load sharing processors," in *Proc. Conf. Information Science Systems*, 1992, pp. 794–799.

[3] Z. Chair and P. K. Varshiney, "Optimum data fusion in multiple sensor detection systems," *IEEE Trans. Aerosp. Electron. Syst.*, vol. AES-22, pp. 98–101, 1986.

[4] Y. C. Cheng and T. G. Robertazzi, "Distributed computation with communication delay," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 24, no. 6, pp. 700–712, 1988.

[5] ——, "Distributed computation with a tree network with communication delays," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 26, no. 3, pp. 511–516, 1990.

[6] B. V. Dasarathy, "Paradigms for information processing in multisensor environments," in *Proc. SPIE, Sensor Fusion III*, 1990, vol. 1306, pp. 69–80.

[7] B. V. Dasarathy, "Decision fusion," *IEEE Tutorial*, ISBN-0-8186-4452-4, 1994.

[8] S. S. Iyengar, R. L. Kashyap, and R. N. Madan, "Distributed sensor networks—Introduction to the special section," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, pp. 1027–1031, Sept./Oct. 1991.

[9] H. J. Kim, G. I. Jee, and J. G. Lee, "Optimal load distribution for tree network processors," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 32, no. 2, pp. 607–612, 1996.

[10] V. Mani and D. Ghose, "Distributed computation in linear networks: Closed-form solutions," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 30, no. 2, pp. 471–483, 1994.

[11] D. Narasingh, *Graph Theory with Applications to Engineering and Computer Science*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[12] D. A. Reed and R. M. Fujimoto, *Multicomputer Networks Message-Based Parallel Processing*. Cambridge, MA: MIT Press, 1987.

[13] A. R. Reibman and L. W. Notle, "Optimal detection and performance of distributed sensor systems," *IEEE Trans. Aerosp. Electron. Syst.*, vol. AES-23, no. 1, pp. 24–30, 1987.

[14] ——, "Design and performance comparison of distributed detection systems," *IEEE Trans. Aerosp. Electron. Syst.*, vol. AES-23, no. 6, pp. 789–797, 1987.

[15] J. R. Rice, *Numerical Methods, Software, and Analysis*. New York: McGraw-Hill, 1985.

[16] R. R. Tenney and N. R. Sandell, Jr., "Detection with distributed sensors," *IEEE Trans. Aerosp. Electron. Syst.*, vol. AES-17, pp. 501–510, 1981.

[17] S. C. A. Thomopoulos, R. Viswanathan, and D. C. Bougoulias, "Optimal decision fusion in multiple sensor systems," *IEEE Trans. Aerosp. Electron. Syst.*, vol. AES-23, pp. 644–653, 1987.

[18] R. M. Tong, E. Tse, and R. P. Wishner, "Distributed hypothesis formation in sensor fusion system," in *Proc. 20th IEEE Conf. Decision Control*, 1981, pp. 1421–1424.

[19] R. Wesson, F. Hayes-Roth, J. W. Buge, C. Stast, and C. A. Sunshine, "Network structure for distributed situations assessment," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-11, no. 1, pp. 5–23, 1981.