ELSEVIER Parallel Computing 21 (1995) 1945–1956

PARALLEL COMPUTING

Scheduling divisible jobs on hypercubes[†]

J. Błażewicz *, M. Drozdowski

Institute of Computing Science, Poznań University of Technology ul.Piotrowo 3a, 60-965 Poznań, Poland

Received 3 July 1994; Revised 6 June 1995

Abstract

In this work a problem of finding an optimal distribution of a divisible computational job among a set of processors is considered. In the model of parallel computer systems two important factors must be taken into account: speeds of processors and speeds of communications links. With regard to this, we propose a deterministic approach finding an optimal distribution of the job's load on a hypercube of processors. The method used allows also the determination of performance bounds on the hypercube architecture.

Keywords: Parallel processing; Deterministic scheduling; Performance analysis; Hypercube

1. Introduction

Parallel computer systems receive a lot of attention for many years. There are many models of such systems. The progress in technology created a new situation in this field. Computers based on the parallel architecture are not only successfully implemented, but are becoming an everyday tool. Such systems are no longer designed to run only one dedicated application, but must be capable of executing many concurrent jobs competing for system's resources [13,14]. Thus, the problem of representation and modelling of parallel applications is becoming more and more significant from the point of view of the operating system designer. There are many aspects of this problem, and many ways of dealing with them. There is, for example, a concept of multiprocessor tasks (also known as co-scheduling) [3,4,8,12,19], where each task may require more than one processor at the same time. There are also other approaches (see e.g. [2,9,17,18] and [6,7] for the problem taxonomy).

* Corresponding author. Email: blazewic@pozn1v.tup.edu.pl

This research has been partially supported by KBN grant 3P40600106.

In this work we propose a method of finding an optimal distribution (in the sense of the minimum execution time of a job) of a divisible job among a set of processors connected by communication links and forming a hypercube. This method is also useful in estimating the ultimate performance bounds of the hypercube architecture. Unlike in many other works based on a probabilistic approach [11,15] the performance assessment here is based on a more practical deterministic model. The methodology we apply is similar to the one introduced in a series of works [1,2,9,10], were various interconnection networks have been analysed. In [9] the problem of optimal distributing a computational problem on a chain of processors has been addressed. The processing elements could have communication co-processors or not, their speeds were assumed to be different. In [10] this model has been applied to solve the problem of a job distributing on a tree network of processors, and in [1] on the processors interconnected through a bus type medium. Finally, in [2] the performance limits are given for infinite chain and tree networks of processors. The same methodology has been applied to analyse two-dimensional mesh of processors [5].

Let us set up our problem more formally. We consider a computer system consisting of a set of identical processing elements (PE's): processors with local memories connected by a network of communication links. The architecture of the interconnection network is assumed to be a hypercube [16], i.e. each processor is a node of a multidimensional cube. For the hypercube of dimension d there are 2^d processors in the system. Each of the processors has direct links to d neighbours. A useful method of naming the processors is to use label consisting of a binary string d- position long. Hence, the label of a processor is a binary number from the interval $[0,2^d - 1]$. Note, that each of the processor's neighbours has a label differing on exactly one position. The processors' speeds and the speed of the transmission are limited.

At time 0 a job arrives at the processor numbered 0. It is assumed that the job (the load) can be arbitrarily divided and processed independently (in parallel) on many processors. It can be the case of processing large database files, signal processing [9], modelling etc. Some part α_0 of the total load is processed by processor 0, the rest of the load $(1 - \alpha_0)$ is transmitted in equal parts to the d neighbours of processor 0 for processing. The immediate neighbours of processor 0 take some part α_1 of the total load and re-transmit the rest to the still idle neighbours. This process is continued until the last idle processor in the hypercube is reached. We assume that the processing time of the job on a standard processor is T_{cp}, while one processor with a different speed it is wT_{cp}. Thus, w is proportional to the reciprocal of the processor's speed. On the other hand, the transmission time of the whole job's data is T_{cm} for a standard data link, while for a link with different capabilities it is zT_{cm}. Hence, z is reciprocal of the link bandwidth. Although the links are identical and the processors are identical, we will still use w and z, which will be useful in the following sections. We assume two things about the processing element: it must receive all its load before transmitting the proper part to the neighbours, and it is capable of simultaneous transmitting and computing (i.e. it has some kind of DMA, which is true for most of contemporary computers and many processors, e.g. transputers)

In the following section, the way to divide the load on the hypercube so that the completion time of the computation is minimised will be given. Then, formulae describing performance of the hypercube network of processors will be presented. In Section 3, the results of modelling performance of the system according to the formulae from Section 2 will be described. The last section presents conclusions.

2. A model of the computation process

In this section we present a formal analysis of the computational process. The expected results are the answer to the question how to divide the load between the processors, formulae expressing the equivalent speed of the whole network, speedup and utilisation of processors. Firstly, we will analyse the process of data distribution, later on we will present the formulae.

The processing and distributing the data is depicted in the form of a Gantt chart in Fig. 1. When processor *0* receives a burst of data to process, it takes α_0 of it for local processing; $(1-\alpha_0)$ of the load is transmitted to d neighbours. The computation on processor *0* last $\alpha_0 w T_{cp}$. Since processor *0* has no 1 in its address, its neighbours have exactly one 1 in their addresses. The part $(1-\alpha_0)$ transmitted from processor *0* is fairly divided among all d neighbours. Thus, the transmission time is $(1-\alpha_0)zT_{\mathrm{cm}}/d$. When transmission is finished, each of the processors with only one 1 in the address takes α_1 of the whole load for local processing from the part it receives from processor *0*. The rest is transmitted, in equal shares, to its idle neighbours. Processors with one 1 in the address have $d-1$ idle neighbours with exactly two 1's in the address. Hence, this transmission time is $(1-\alpha_0-d\alpha_1)zT_{cm}/d/(d-1)$. Note, that processors with one 1 in the address can be reached from the originator of the load via only one link, while processors with two 1's can be accessed via two links. The process of data dissemination is repeated until the last processing element with address 11 .. 1 is reached via d links. Let us call by *layer i* a set of all processors reached in the same number i of hops, starting from layer *0* consisting of the originator only (processor *0*). The last layer d consists of a single processor. Note, that data transmission does not cause contention in use of any communication link because each link is used only once and each communication path is one link long. It can be observed, that the computation stage must finish on all exploited processors at the same moment of time. This can be explained in this way, that if some processor(s) stop earlier, we can move some part of the load to these processors and reduce the completion time of the job. Let us denote by $\hat{\alpha}_i$ a part of the received load that is intercepted for local processing in a processor of layer i. The $\hat{\alpha}_i$ will be used to facilitate the process of finding proper solutions.

The following lemma describes useful topological properties of a hypercube.

Lemma. In each layer i of d-dimensional hyper cube there are $\binom{d}{i}$ processing elements each of which can be accessed through i communications links and is capable of transmitting to d-i still idle processors.

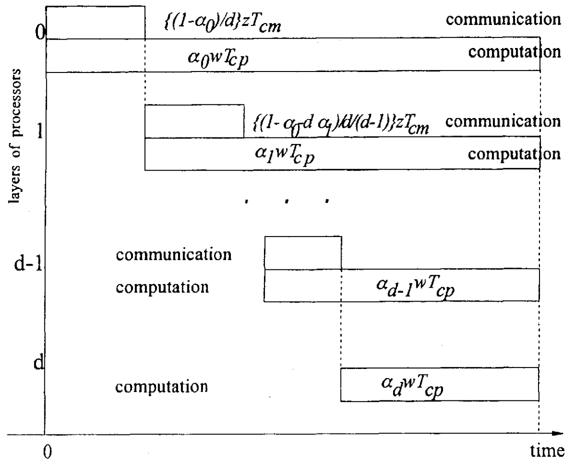

Fig. 1. Process of computation and data transfer.

Proof. First, let us note that the method of originator's connection to the outside of the hypercube is beyond our consideration.

We can observe that it is a general rule in the hypercube architecture that any link joins two PE's differing in only one position of their addresses. We can assume that in the process of data distribution we get from a busy processor to idle processors via links reflecting 0 in the current processor address. Thus, processor 0 is connected with d neighbours, each of which represents one of d dimensions of the hypercube. Since the originator has the address $0_1 0_2 \dots 0_d$, its neighbours have addresses with exactly one 1 (i.e. addresses: $1_1 0_2 \dots 0_d$, $0_1 1_2 \dots 0_d$,...$0_1 0_2 \dots 1_d$). Then, PE's with exactly one 1 in the address have idle neighbours with exactly two 1's in the address, and so on. Let us consider layer i. Each of its processors can have an address with i 1's and can be reached from processors with $i-1$ 1's. Since the address is d position long and i of its positions are 1, then there are $\binom{d}{i}$ processors in layer i. Next, each of processors in layer i transmits some part of the load to $d-i$ processors (the number of 0's in the current address) with $i+1$ 1's in the address. On the other hand, address of the current processor with i 1's can be accessed from processors with $i-1$ 1's. There are i such processors that differ in exactly one position (number of 1's). The induction over i finishes the proof. □

Now, we are ready to present the qualitative model of computation. Let us consider layer i and denote by Vol_i the amount of data received by a processors in layer i. Hence, $Vol_0 = 1$. The processor in layer i works the same amount of time as it takes to transmit data to $d-i$ processors in layer $i+1$ and to compute in layer $i+1$ (cf. Fig. 1). What is more, processors in layer $i+1$ receive data to process from $i+1$ links. Thus,

$$\hat{\alpha}_i Vol_i w T_{cp} = \frac{(1-\hat{\alpha}_i) Vol_i \big((i+1) w_{i+1} T_{cp} + z T_{cm}\big)}{d-i} \quad \text{for } i=0,\ldots d-1, \tag{1}$$

where w_{i+1} is equal to a reciprocal of the equivalent speed for all processors in layers $i+d,\ldots,d$ which receive some part of the load from the considered processor in layer i. Hence, $\hat{\alpha}_i$ is equal to

$$\hat{\alpha}_i = \frac{1}{1 + \dfrac{(d-i) w T_{cp}}{(i+1) w_{i+1} T_{cp} + z T_{cm}}} \quad \text{for } i=0,\ldots,d-1 \tag{2}$$

Layers $i,\ldots d$ process Vol_i of the load in time $\hat{\alpha}_i Vol_i w T_{cp}$. Taking into account the above-introduced reciprocal of the equivalent speed W_i, the above load could be processed in layers $i, i+1,\ldots,d$ in time $w_i\ Vol_i T_{cp}$. Thus,

$$w_i = \frac{\hat{\alpha}_i Vol_i w T_{cp}}{Vol_i T_{cp}} = \hat{\alpha}_i w.$$

For $i=0$ Eq. (1) has the following form

$$\hat{\alpha}_0 w T_{cp} = \frac{(1-\hat{\alpha}_0)(w_1 T_{cp} + z T_{cm})}{d},$$

hence,

$$\alpha_0 = \hat{\alpha}_0 = \frac{1}{1 + \dfrac{d w T_{cp}}{w_1 T_{cp} + z T_{cm}}} \tag{3}$$

Eqs. (2) and (3) form a set of expressions which can be solved for $\hat{\alpha}_i$, recursively starting from $i=d-1$, for which we know that $\hat{\alpha}_d = 1$, $w_d = w$, until $i=1$. Then, the portions α_i of the whole load can be calculated. Thus, the originator processes locally α_0 of the whole load, sends to each of it's d neighbours a share of load equal to $(1-\alpha_0)/d$. Each processor in layer i $(i=1,\ldots,d)$ receives through each of i links a share of load equal to

$$\frac{(1-\hat{\alpha}_{i-1}) Vol_{i-1}}{(d-i+1)}, \text{ thus totally } Vol_i = \frac{(1-\hat{\alpha}_{i-1}) i Vol_{i-1}}{(d-i+1)},$$

from which

$$\alpha_i = \frac{\hat{\alpha}_i(1-\hat{\alpha}_{i-1})iVol_{i-1}}{(d-i+1)}$$

is intercepted for local processing.

We can calculate an equivalent reciprocal of the speed for the whole hypercube of processors:

$$w^{eq} = \frac{\alpha_0 w T_{cp}}{T_{cp}} = \alpha_0 w$$

Then, we can find speedup (S - measured as a ratio of the sequential computation time, i.e. on the sole originator, to the working time of the originator embedded in the hypercube) and average utilisation of processors (U):

$$S = \frac{wT_{cp}}{\alpha_0 w T_{cp}} = \frac{1}{\alpha_0} = 1 + \frac{dwT_{cp}}{w_1 T_{cp} + zT_{cm}} \qquad U = \frac{S}{2^d} = \frac{1}{2^d \alpha_0},$$

where w_1 is calculated according to the above recursive procedure.

From the above formulae we can derive several qualitative conclusions. The speedup depends on the dimension d of the hypercube but depends also on the w_1, which would have to decrease at least linearly with d in order to preserve linear speedup. The average utilisation of the processors has 2^d in the denominator, then again, to preserve linear speedup and utilisation close to 1, α_0 must decrease very fast.

We conclude this section with an example. Consider a hypercube with $d = 2$, $z = w = T_{cp} = T_{cm} = 1$. We know that $\hat{\alpha}_d = 1$, $w_d = w = 1$. From Eq. (2) we get $\hat{\alpha}_1 = \frac{3}{4} = w_1$, and from (3) $\hat{\alpha}_0 = \alpha_0 = \frac{7}{15}$. By definition $Vol_0 = 1$, hence we calculate

$$Vol_1 = \frac{(1-\hat{\alpha}_0)Vol_0}{(d-1+1)} = \tfrac{4}{15}, \quad \alpha_1 = \hat{\alpha}_1 Vol_1 = \tfrac{3}{15},$$

and

$$Vol_2 = 2\frac{(1-\hat{\alpha}_1)Vol_1}{(d-2+1)} = \tfrac{2}{15}, \quad \alpha_2 = \hat{\alpha}_2 Vol_2 = \tfrac{2}{15}.$$

From the above we obtain $w^{eq} = \frac{7}{15}$, $S = \frac{15}{7}$, $U = \frac{15}{28}$.

In the next section we present results of modelling the above expressions for different values of network parameters (or performance evaluation of the hypercube network, in other words).

3. Modelling

In this section we present results of modelling the set of equations introduced in Section 2. If not stated otherwise, in the following graphs $z = 1$, $w = 1$, $T_{cp} = T_{cm} = 1$. We will present dependence of α_i, total processing time $(w_0^{eq} T_{cp})$, S, U on the parameters describing the hypercube in the above model.

Firstly, we will illustrate the distribution of the load in the hypercube. Figs. 2 and 3 present dependence of the load share on the dimension of the hypercube (d). Fig. 2 shows what part of the total load is distributed in a processor of a given layer, while Fig. 3 shows what part of the total load is processed in a given layer of processors. The load of processors across the layers decreases very fast. We can conclude from this that processors in deeper layers compute less. The total load of a layer is a function which maximum moves from the originator, for small dimensions, to the internal layers, for bigger dimensions. Thus, we can say that for bigger dimensions central layers are computing more, while the front layers are communicating more.

As a second topic, we have analysed the dependence of the total processing time of the job on the speed of processors (Fig. 4), communication medium (Fig. 5) and the size of computing job (Fig. 6). The execution time of the job decreases with the increase of the dimension of the hypercube. However, for faster processors ($w = 0.1$) this reduction is relatively smaller than for slow processors ($w = 10$) where execution time can be reduced by two orders of magnitude. Conclusion is that gain from parallel processing on slow processors is higher than on fast processors. In Fig. 5 we can see that fast communication network ($z = 0.1$) is more reasonable than the slow one ($z = 10$). In this picture we have also included a

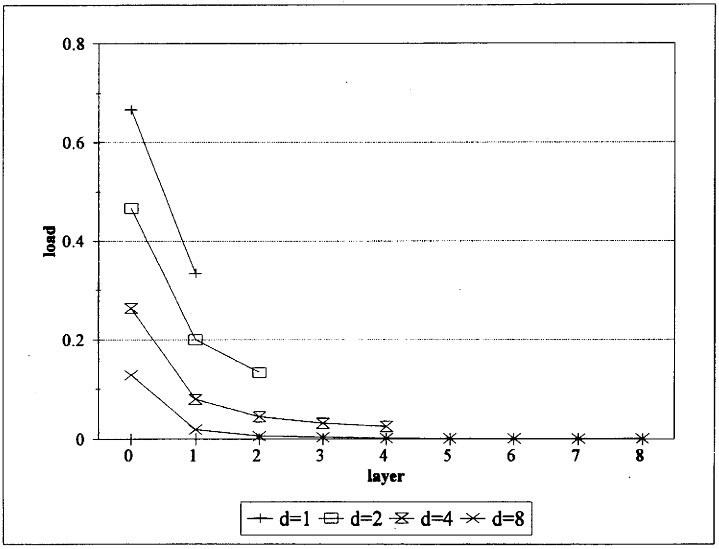

Fig. 2. Load of processors vs. layers and dimension d.

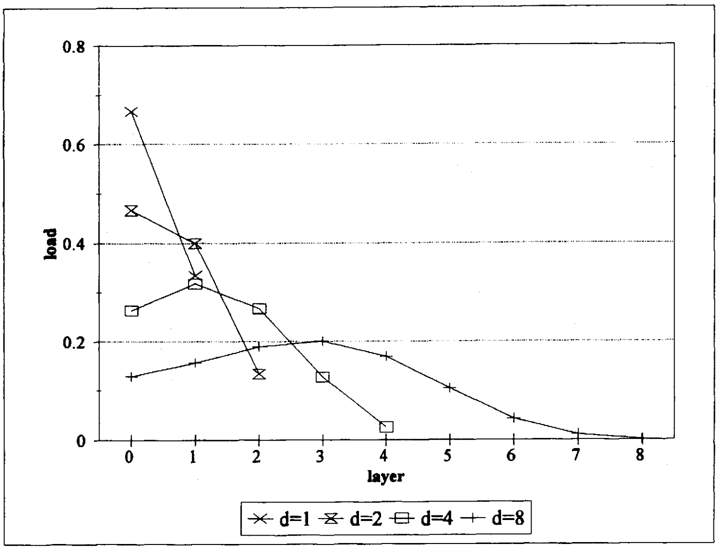

Fig. 3. Load of layers for different dimensions *d*.

curve for $z = 0$ which is the case of the ideal network introducing no transportation delays. We can see in Fig. 5 that the reduction of the execution time between the curve for $z = 0.1$ and for $z = 0$ is not very high. Although this reduction is not big in the absolute terms of time, for $z = 0$ the execution time decreases proportio-

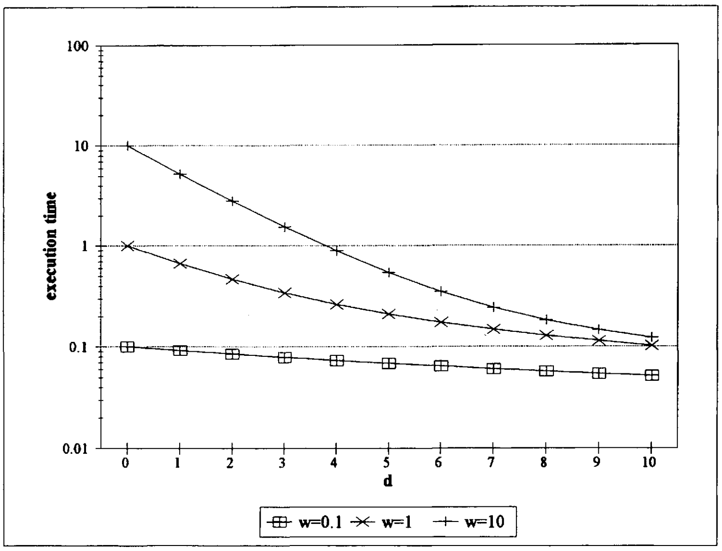

Fig. 4. Execution time vs. processor speed and dimension.

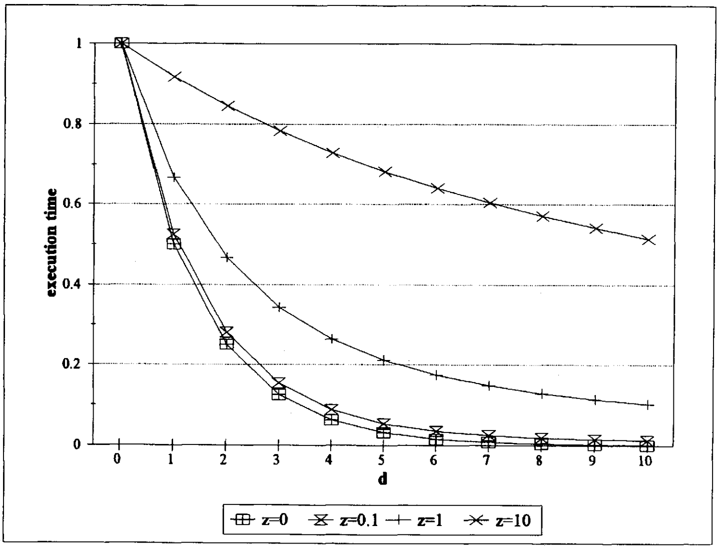

Fig. 5. Execution time vs. communication speed and dimension.

nately to the number of processors (cf. Fig. 7 and Fig. 8). Finally, Fig. 6 demonstrates relative processing time as a function of T_{cp} and d. The relative execution time is equal to the quotient of the actual processing time and processing time on one processor. We can see that the gain from parallel processing is bigger for big tasks ($T_{cp} = 10$).

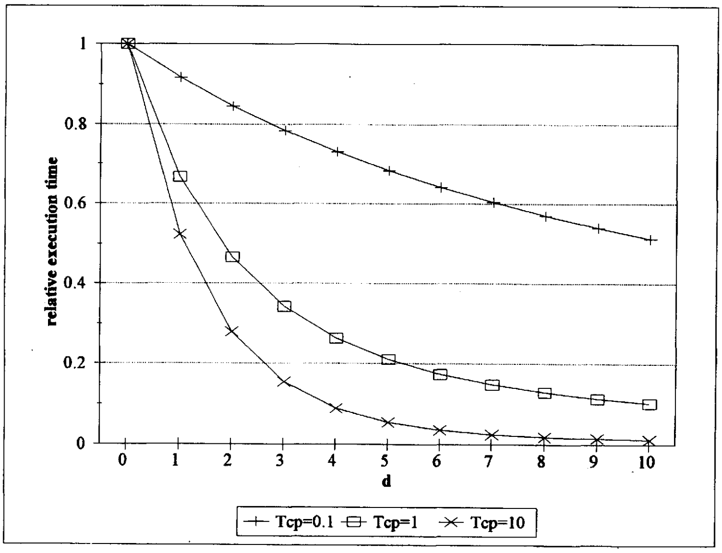

Fig. 6. Execution time vs. size of the computation task (T_{cp}) and dimension.

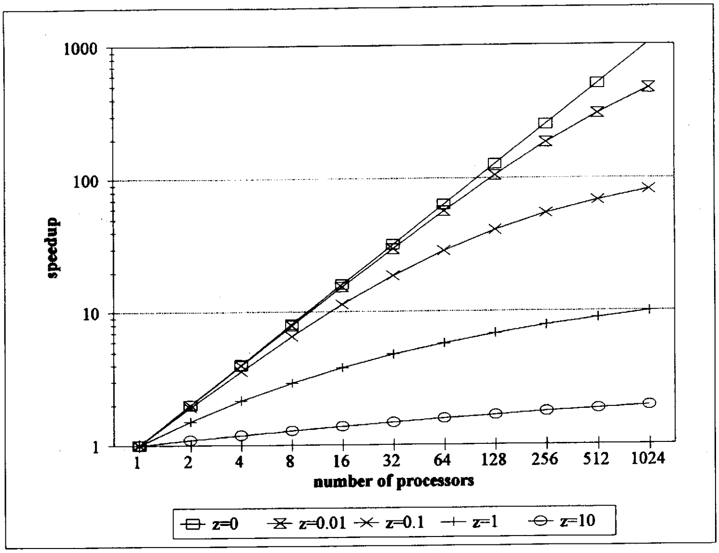

Fig. 7. Speedup for different z vs. number of processors.

The last parameters we consider are speedup (Fig. 7) and utilisation of processors (Fig. 8). In both cases curves are presented for different values of z, among which a curve of $z=0$ is present. The network with $z=0$ represents an ideal network. As it can be seen, the linear speedup can be achieved when the communication medium is perfect. In more realistic cases ($z>0$) the speedup

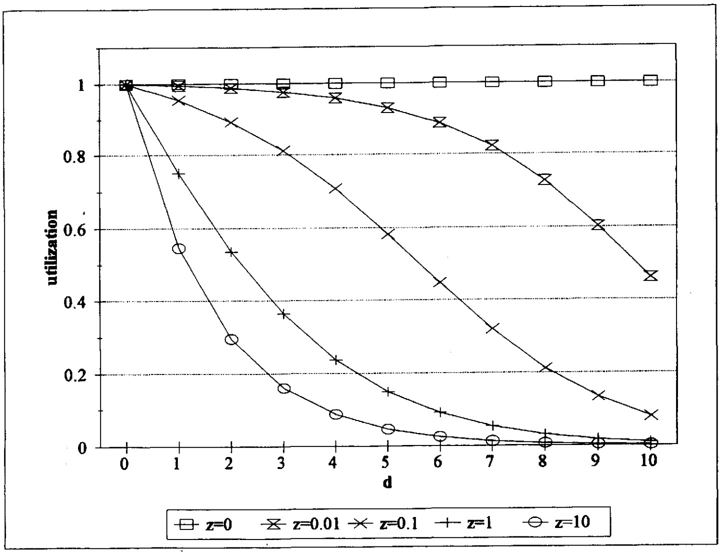

Fig. 8. Utilisation for different z vs. dimension.

curve levels off very fast, especially for slow networks ($z = 10$). The utilisation of processors decreases with the size of the network. Only for the perfect communication network, can utilisation equal to 1 be achieved.

4. Conclusions

In this work we have presented an analytical model for parallel computations. The model presented here is strictly deterministic. We have assumed that a computational job can be arbitrarily divided and processed in parallel. Next, the cost of computations and transmissions have been assumed to be linearly dependent on the size of the task. The architecture of the network is a hypercube of processors. Various parameters can be derived from the model presented in this paper e.g. execution time of the job, speedup and utilisation of the network. The intuitive expectations are fully supported by the model. The faster the processors and the communication links are, the more efficient the hypercube is. On the other hand, the gain from parallel processing is more significant on slower (cheaper) processors than on fast processors. What is more, big computational task gain more from parallel processing than small tasks. Finally, we have shown that the linear speedup in the hypercube can be obtained only in a perfect network introducing no communication delays.

References

[1] S. Bataineh and T.G. Robertazzi, Bus-oriented load sharing for network of sensor driven processors, *IEEE Trans. on Systems Man, and Cybernetics* 21 (1991) 1202–1205.
[2] S. Bataineh and T.G. Robertazzi, Ultimate performance limits for networks of load sharing processors, State University of New York at Stony Brook, CEAS Tech. Rept. 623, 1992.
[3] J. Błażewicz, M. Drabowski and J. Weglarz, Scheduling multiprocessor tasks to minimize schedule length, *IEEE Trans. Comput.* 35 (1986) 389–393.
[4] J. Błażewicz, P. Dell'Olmo, M. Drozdowski and M.G. Sperana, Scheduling multiprocessor tasks on three dedicated processors, *IPL* 41 (1992) 275–280.
[5] J. Błażewicz and M. Drozdowski, Performace limits of two-dimensional network of load-sharing processors, *Foundations of Computing and Decision Sciences*, to appear.
[6] J. Błażewicz, K. Ecker, G. Schmidt and J. Weglarz, *Scheduling in Computer and Manufacturing Systems* (Springer, Berlin New-York, 1993).
[7] T. Casavant and J. Kuhl, A taxonomy of scheduling in general-purpose distributed computing systems, *IEEE Trans. on Software Eng.* 14 (1988) 141–154.
[8] G.I. Chen and T.H. Lai, Preemptive scheduling of independent jobs on a hypercube, *IPL* 28 (1988) 201–206.
[9] Y.C. Cheng and T.G. Robertazzi, Distributed computation with communication delays, *IEEE Trans. Aerospace and Electronic Syst.* 24 (Nov. 1988) 700–712.
[10] Y.C. Cheng and T.G. Robertazzi, Distributed computation for a tree network with communication delays, *IEEE Trans. Aerospace and Electronic Syst.* 26 (1990) 511–516.
[11] S. Dandamundi, Performance analysis of a class of hierarchical hypercube multicomputer networks, *Performace Evaluation* 13 (191) 159–179.