Serdica
Bulgariacae mathematicae publicationes

Сердика
Българско математическо списание

For further information on
Serdica Bulgaricae Mathematicae Publicationes
and its new series Serdica Mathematical Journal
visit the website of the journal http://www.math.bas.bg/~serdica
or contact: Editorial Office
Serdica Mathematical Journal
Institute of Mathematics and Informatics
Bulgarian Academy of Sciences
Telephone: (+359-2)9792818, FAX:(+359-2)971-36-49
e-mail: serdica@math.bas.bg

SERDICA — Bulgaricae
mathematicae publicationes
19 (1993) 66-70

ABOUT AN ALGORITHM FOR DOWNLOADING DATA INTO A HYPERCUBE MULTIPROCESSOR WITH A HOST PROCESSOR

DINKO GICHEV

Abstract. The problem of minimizing the time for downloading the data into a hypercube multiprocessor with a host processor is discussed. A new downloading algorithm which improves the Data Scattering Algorithm is proposed. The obtained result is better than that in [3].

1. Introduction. In this paper we shall consider the problem for downloading data into the processing elements of a distributed memory multiprocessor. This problem has already been discussed by many authors. Different loading strategies have been proposed so that the start-up delay time is reduced. Here we shall analyze one of these strategies, namely the new algorithm (Sequential Scattering Algorithm) which has been proposed in [3]. The algorithm given there is not correct (not all the data are downloaded). In [3] the authors have considered three other well-known algorithms for distributing the data - Sequential Loading, Ratioed Partitioning and Data Scattering. In the present paper we describe an algorithm which uses the same configuration as in [3]. It seems to work faster than the algorithms mentioned.

2. Statement of the Problem.

2.1. Configuration Design. A distributed memory multiprocessor is considered. It consists of a hypercube (i.e. a cube upto n dimensions, each of whose 2^n vertices is a processing element (PE)) and a host processor, directly linked with any of PEs. Ametek's system14 and iPSC are mentioned as typical examples. Each PE has its own memory, a copy of the operating system and an application program. All the processors work in an asynchronous way. The communication messages between the processors are of two different types:

(i) Communication messages between the host and a hypercube PE

(ii) Communication messages between the hypercube PEs.

The time for sending (or receiving) a message of the first type, consisting of b words is equal to

$$t = \beta_h + b\tau_h \tag{2.1}$$

where β_h is the start-up delay time for the host in seconds and τ_h is the element transfer time from the host to a PE or vice versa in seconds per word. Similarly, the time for sending (or receiving) a message of the second type, consisting of b words is equal to

$$t = \beta_n + b\tau_n \tag{2.2}$$

where β_n is the start-up delay time for a PE and τ_n is the element transfer time from a PE to another PE.

In [3] the authors have used the following values

$$\beta_h = \beta_n = 6.5ms \qquad \tau_h = \tau_n = 8.0\mu s. \tag{2.3}$$

2.2. Scheme of Work. At the first stage each of the PEs receives its own part of data and then executes the application program for it. At the second stage all the algorithms discussed in paper [3] send the desired result to the host by using the Dimensional Collapse Technique. It is clear that the differences between the four algorithms at the first stage are more significant for the our analysis. The algorithms, execpt for the RP one, distribute the initial array of data in equal parts on the PEs, which equals the time for executing the application program on any PE. So the most significant part in our research will take into account the consideration of the time required for distributing the data among the PEs by the host.

2.3. Basic Notations. We shall use the same notations as in [3], namely
N for the dimension of the initial array of data,
n for the dimension of the hypercube,
$k = 2^n$ for the number of PEs in the hypercube,
T_d for the time needed so as to transfer the data from the host to the PEs.
Let us assume for simplicity that N is divisible by k.

3. Short Description of the SL, RP and DS Algorithms.

3.1. Sequential Loading Algorithm (SL). The host processor sends its own portion of N/k elements to any of PEs in the hypercube when the SL Algorithm is used. The time required for the execution of this operation is given by

$$T_d = k[\beta_h + (N/k)\tau_h] = k\beta_h + N\tau_h. \tag{3.1}$$

3.2. Ratioed Partitioning Algorithm (RP). In RP each PE receives various portions of the initial data array and with the receipt of the last data element the

respective PE application program is started. The initial data array is divided among the PEs so that all the PEs complete their own application programs for one and the same time. When N is sufficiently large, so that all the PEs of the hypercube are used, the global time required for downloading the data is

$$T_d = k\beta_h + N\tau_h, \tag{3.2}$$

i.e. the same as in the previous case. But since RP is an improvement of SL, it has some advantages. We are interested in one of them. In these two algorithms the global time for downloading the data and performing the application program is equal to $T_d + T_l$, where T_l is the time, required for the last PE to perform the application program to its own data. In case of RP this time is much smaller (the last PE data are much smaller).

3.3 Data Scattering Algorithm (DS). When DS is used all the data are initially sent from the host to PE numbered 0 (i.e. the binary code label consists of n zeros). The time needed for this operation is

$$T_t = \beta_h + N\tau_h. \tag{3.3}$$

Next all the data are distributed in equal parts, each of N/k elements among the PEs by using the Data Scattering Algorithm (the reverse of the Dimensional Collapse) as shown below. Namely, (see e.g. [1], [3]) at the i-th step, $i = 1, 2, \ldots, n$, all the PEs whose binary labels are of form $0^{n-i+1}a$ (a is any (i-1)-bit binary number) split the data, received at the previous step, into two halves and then send one of these halves to their neighbours with binary labels $0^{n-i}1a$. The time required for downloading the data is given by

$$T_d = T_t + T_s = \beta_h + N\tau_h + n\beta_n + (N/k)(k-1)\tau_n \tag{3.4}$$

where T_t is as in (3.3), and T_s is the time for scattering the data over the hypercube.

4. Analysis of the Sequential Scattering Algorithm ([3]). In [3] the authors have pointed out that, with the use of the DS Algorithm, the host processor was idle once it had finished loading the data into the PE of binary label code 0^n. Hence, the time for downloading may be reduced by keeping it busy. Thus the Sequential Scattering Algorithm was suggested. The basic idea was to split the n-dimensional hypercube into two subcubes; the first one being x-dimensional and the second one – $(n-x)$-dimensional. Data Scattering was performed in the first subcube and Sequential Loading – in the second one.

But we have to point out the following fact. One can split an n-dimensional hypercube into 2^x $(n-x)$-dimensional subcubes (or vice versa, into 2^{n-x} x-dimensional subcubes), but he cannot do that into two subcubes, a x-dimensional subcube and a $(n-x)$-dimensional one. Now it is clear why the authors have used only $(N/2^n)(2^x + 2^{n-x})$ of all the N elements of the initial data array.

5. A new downloading algorithm. The following method seems to be reasonable.

We split the n-dimensional hypercube into two $(n-1)$-dimensional subcubes at the first step. Then the host sends one half of its elements (at this step $N/2$ elements are sent) to a PE of the first $(n-1)$-dimensional subcube. These operations take

$$T_1 = \beta_h + (N/2)\tau_h \tag{5.1}$$

time. Next the DS Algorithm is applied to the first $(n-1)$-dimensional subcube and the previous procedure is recursively applied to the second subcube.

More precisely, two different procedures are performed at the second step:

(i) The first step of the DS Algorithm is performed on the first $(n-1)$-dimensional subcube. $N/4$ elements are sent and

$$T_2' = \beta_n + (N/4)\tau_n \tag{5.2}$$

time is needed.

(ii) The previous procedure is applied to the second $(n-1)$-dimensional subcube. More precisely, we split this cube into two $(n-2)$-dimensional subcubes. Then the host sends one half of its elements ($N/4$ elements are sent at this step) to a PE of the first $(n-2)$-dimensional subcube. This operation takes

$$T_2'' = \beta_h + (N/4)\tau_h \tag{5.3}$$

time. If we use the same model as in [3], (i.e. if conditions (2.3) are fulfilled), then it is clear that $T_2' = T_2''$, thus, both operations at the second step of the algorithm can be done simultaneously.

The following procedures are performed at the third step:

(i) DS on the first $(n-1)$-dimensional subcube. $N/8$ elements are sent.

(ii) DS is started on the first $(n-2)$-dimensional subcube (which receives $N/4$ elements at the previous step). $N/8$ elements are sent.

(iii) Splitting the second $(n-2)$-dimensional subcube into two $(n-3)$-dimensional subcubes. The host sends $N/8$ elements to a PE of the first of them.

It is clear that all the operations at the third step can be performed parallely and that takes time equal to

$$T_3 = \beta + (N/8)\tau. \tag{5.4}$$

Here $\beta = \beta_h = \beta_n$ and $\tau = \tau_h = \tau_n$. The consequtive steps, up to the n-th one, are performed analog. Evidently, DS completes for one and the same time for all the subcubes in a given step. One of the last two 1-dimensional subcubes (i.e. PEs) has received its $N/2^n$ elements at the $n-th$ step. There are $N/2^n$ elements in the host too, and finally they are sent to the last PE at the $(n+1)-st$ step.

Then the global time required for downloading the data is given by

$$T_d = T_1 + T_2 + ... + T_n + T_{n+1} \tag{5.5}$$

where T_i is the time needed for the $i-th$ step of the algorithm. Note that $T_{n+1} = T_n$ because $N/2^n$ elements are sent again at the $(n+1)-st$ step. Thus

$$T_i = \beta + (N/2^i)\tau \tag{5.6}$$

for $i = 1, 2, \ldots, n$ and

$$T_{n+1} = T_n. \tag{5.7}$$

Hence

$$T_d = (n+1)\beta + N\tau. \tag{5.8}$$

6. Conclusions. From the above it follows that if conditions (2.3) are fulfilled, then the time (5.8) of this algorithm is better than time (3.4) of the DS Algorithm with the following value

$$T = N\tau(2^n - 1)/2^n. \tag{6.1}$$

That is why in our algorithm the host processor is used all the time. We have to point out that algorithms such as SL and RP, which do not use the capacities of a hypercube interprocessor network for downloading the data are not very suitable in such a configuration (Star plus Hypercube). In view of the fact that β is approximately equal to 800τ and $k = 2^n$, it is clear that the results in (5.8) and (5.4) are much better than those in (3.1) and (3.2).

REFERENCES

[1] S. Horiguchi and W. L. Miranker, A parallel algorithm for finding the maximum value, *Parallel Comput.* **10** (1989), 101-108.

[2] Y. Saad and M. H. Shultz, Data communication in hypercubes, *J. Parallel and Distributed Comput.* **6** (1989), 115-135.

[3] V. V. R. Prasad and C. Siva Ram Murthy, Downloading node programs/data into hypercubes, *Parallel Comput.* **17** (1991), 633-642.

Center of Informatics and Computer Technology
25 A, Acad. G.Bonchev str.
1113 Sofia
BULGARIA

Received 28.10.92