Parallel Computing 13 (1990) 235–243
North-Holland

Short Communication

Data mapping of linear programming on fixed-size hypercubes

Gen-Huey CHEN

Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, Republic of China

Hong-Fa HO

Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, Republic of China

Shieu-Hong LIN

Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, Republic of China

Jang-Ping SHEU

Department of Electrical Engineering, National Central University, Chung-Li, Taiwan, Republic of China

Received January 1989
Revised June 1989

Abstract. Although many solution methods are available for the linear programming problem, the simplex method is undoubtedly the most widely used one for its simplicity. In this paper, we propose an implementation of the simplex method on fixed-size hypercubes. A partitioning technique and a mapping technique are presented to fit large-size problem instances into relatively small-size hypercubes. Two cases, pipelined broadcasting allowed and pipelined broadcasting not allowed, are considered. We show that the proposed implementation achieves the linear speedup asymptotically for both cases. Also, under the given mapping method, we discuss how to partition data so as to minimize the total run time and the communication time. We derive sufficient conditions for optimal partitionings. These sufficient conditions are helpful to obtain better partitionings. The optimal partitioning is obtained for a special case.

Keywords. Linear programming, simplex method, data mapping, hypercubes.

1. Introduction

Linear programming is a fundamental problem in operations research and has received much attention for its importance. Mathematically, this problem can be formulated in *standard form* [5] as follows:

$$\begin{aligned} &\text{minimize } z = \boldsymbol{c}\boldsymbol{x}, \\ &\text{subject to } \boldsymbol{A}\boldsymbol{x} = \boldsymbol{d}, \\ &\boldsymbol{x} \geqslant 0 \end{aligned} \tag{1}$$

where $A=[a_{ij}]$ is an $M\times N$ constraint matrix, $0\leqslant i\leqslant M-1$, $0\leqslant j\leqslant N-1$, $d=[d_0, d_1,\ldots, d_{M-1}]^T$ is a positive column vector of length M, $c=[c_0, c_1,\ldots, c_{N-1}]$ is a row vector of length N, and $x=[x_0, x_1,\ldots, x_{N-1}]^T$ is a column vector of length N. The linear programming problem is to find the minimum of z. Dantzig [3] has proposed a well-known solution, the *simplex method*, for this problem. The simplex method is essentially an iterative procedure; it starts from an initial feasible solution, and moves continuously from one feasible solution to another, if improvement can be obtained. In each iteration, operations are needed to determine the pivot column, the pivot row, and the pivot element, and then to update A, c, d, and z [7]. The $(u+1)$th column of A is the *pivot column* if and only if $c_u=\min\{c_i \mid c_i<0\}$. If all the c_i's are nonnegative, then the current value of z is optimal. The $(t+1)$th row of A is the *pivot row* if and only if $d_t/a_{tu}=\min\{d_i/a_{iu} \mid 0\leqslant i\leqslant M-1 \text{ and } a_{iu}>0\}$. Moreover, a_{tu} is called the *pivot element*. After determining the pivot element, A, c, d, and z are updated as follows:

$$a'_{tj}\leftarrow a_{tj}/a_{tu}, \tag{2}$$

$$a'_{ij}\leftarrow a_{ij}-a_{iu}a'_{tj}, \quad i\neq t, \tag{3}$$

$$d'_t\leftarrow d_t/a_{tu}, \tag{4}$$

$$d'_i\leftarrow d_i-a_{iu}d'_t, \quad i\neq t, \tag{5}$$

$$c'_j\leftarrow c_j-c_u a'_{tj}, \tag{6}$$

$$z'\leftarrow z+c_u d'_t, \tag{7}$$

where $j=0, 1,\ldots, N-1$, and a_{ij}, d_i, c_j, and z (a'_{ij}, d'_i, c'_j and z') denote the old (new) values. Let us assume that each binary operation requires the same time and therefore counts one computation step. Then, the total number of (sequential) computation steps required for each iteration is $2MN+2N+3M-1$. Since the feasible solution space is a convex set, the optimum will be reached eventually after a finite number of iterations.

The simplex method has been implemented on several parallel machines. For example, a VLSI wavefront array processor implementation was proposed by Onaga and Nagayasu [4], and a VLSI mesh of trees implementation was proposed by Bertossi and Bonuccelli [1]. Both implementations do not achieve the linear speedup [6], and they need extra hardwares and more complicated control when no sufficient processors are available. The speedup of a parallel algorithm is defined to be the ratio of the fastest worst-case sequential run time to the worst-case parallel run time. A parallel algorithm running on p processors is said to achieve the linear speedup if the speedup is $\theta(p)$. In this paper, we propose an implementation of the simplex method on fixed-size hypercubes. A partitioning technique and a mapping technique are presented to fit large-size problem instances into relatively small-size hypercubes. Two cases: *pipelined broadcasting* allowed and pipelined broadcasting not allowed, are discussed. We show that both cases achieve the linear speedup asymptotically. In addition, we focus our attention on minimizing the total run time and the communication time under the given data mapping method. Sufficient conditions for optimal partitionings are derived. These sufficient conditions are helpful to obtain better partitionings. Also, the optimal partitioning is obtained for a special case.

2. Hypercubes and embedded hypercubes

An *h-dimensional hypercube* contains $p=2^h$ (h is a positive integer) identical processors. Each processor is given an h-bit address $(b_h, b_{h-1},\ldots, b_1)$, $b_i=0, 1$, $1\leqslant i\leqslant h$, and there exists

a link between two processors if and only if their addresses differ in exactly one bit position. The hypercube has a recursive structure as explained below. An h-dimensional hypercube can be regarded as composed of two $(h-1)$-dimensional hypercubes, one with $b_h = 0$ and the other with $b_h = 1$. Similarly, each of the two $(h-1)$-dimensional hypercubes can further be regarded as composed of two $(h-2)$-dimensional hypercubes, one with $b_{h-1} = 0$ and the other with $b_{h-1} = 1$, and so on. Generally, each of the subsets of processors whose addresses differ in k $(1 \leqslant k \leqslant h)$ specified bit positions $b_{i1}, b_{i2}, \ldots, b_{ik}$ and are the same in the remaining $(h-k)$ bit positions forms a k-dimensional hypercube. Such a hypercube is called a *k-dimensional embedded hypercube* on $(b_{i1}, b_{i2}, \ldots, b_{ik})$. The embedded hypercubes are also known as *subcubes*.

One important issue to implement the simplex method on the hypercube is data broadcasting on some embedded hypercubes. That is, some designated processors are necessary to transmit data to all the other processors belonging to the same embedded hypercubes. These data include the pivot element, the pivot row, the pivot column, etc. To broadcast on a k-dimensional embedded hypercube (on $(b_{i1}, b_{i2}, \ldots, b_{ik})$), k communication steps are necessary. Initially, the data to be broadcast are located in the designated processor. In the rth step $(1 \leqslant r \leqslant k)$, each of the processors owning the data sends a duplicate to the processor whose address differs from its address in the bit position b_{ir}. Thus, k communication steps are sufficient to complete the broadcasting. By taking advantage of the property of fast broadening, we can implement the simplex method efficiently on the hypercube.

From the above discussion, two properties of the hypercube immediately follow.

Property 1. Broadcasting on a k-dimensional embedded hypercube requires k communication steps.

An operation is called a semigroup operation if it is associative. Some well-known semigroup operations are addition, multiplication, finding maximum, and finding minimum. The following property is an immediate result of Property 1.

Property 2. Performing semigroup operations on a k-dimensional embedded hypercube (one operand in each processor initially) requires k communication steps and k computation steps. Additional k communication steps are necessary if the computation result is required by every processor.

3. The implementation of the simplex method on fixed-size hypercubes

Cappello [2] has proposed a hypercube algorithm for solving $Ax = b$ via Gaussian elimination with pivoting followed by back substitution, where A is a square matrix. In his approach, data partitioning and data mapping are needed when the given problem instance size is larger than the machine size. In this section, we propose an approach to implement the simplex method on a fixed-size hypercube. The technique we use for data partitioning and data mapping is a direct extension of Cappello's. Suppose that the hypercube is h-dimensional (h is a constant) and therefore contains $p = 2^h$ processors, where $MN > p$ is assumed. We also assume, as Cappello did in [2], that each processor can simultaneously communicate (send data or receive data) with all its neighbor processors within a communication step (we shall remove this assumption later in this paper). Therefore, if a processor wants to broadcast w data items on a k-dimensional embedded hypercube, it can send out these data items in consecutive w communication steps. After $k + w - 1$ communication steps, the broadcasting can be completed. Broadcasting in such a way is called *pipelined broadcasting*.

$$A = \begin{bmatrix} A_{0\,0} & A_{0\,1} & A_{0\,2} & \cdots & A_{0\,2^n-1} \\ A_{1\,0} & A_{1\,1} & A_{1\,2} & \cdots & A_{1\,2^n-1} \\ A_{2\,0} & A_{2\,1} & A_{2\,2} & \cdots & A_{2\,2^n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ A_{2^m-2\,0} & A_{2^m-2\,1} & A_{2^m-2\,2} & \cdots & A_{2^m-2\,2^n-1} \\ A_{2^m-1\,0} & A_{2^m-1\,1} & A_{2^m-1\,2} & \cdots & A_{2^m-1\,2^n-1} \end{bmatrix}$$

(a)

$$d = [d_0, d_1, d_2, \ \cdots \ , d_{2^m-2}, d_{2^m-1}]^T$$

(b)

$$C = [C_0, C_1, C_2, \ \cdots \ , C_{2^n-2}, C_{2^n-1}]$$

(c)

Fig. 1. The partitioning of A, d and c.

Before presenting our approach, let us discuss the necessary data partitioning and data mapping. To balance the computation load, A is distributed evenly over the hypercube; that is, each of the submatrices of A is of size $k_1 \times k_2$, where $k_1 k_2 = MN/p$. Without loss of generality, let us assume $M = k_1 2^m$ and $N = k_2 2^n$ $(m+n=h)$ (if not so, we can extend A appropriately by adding dummy rows and dummy columns to A) in the following discussion. The partitioning of A is shown in Fig. 1(a), where A_{ij}'s $(0 \leqslant i \leqslant 2^m - 1$ and $0 \leqslant j \leqslant 2^n - 1)$ represent the submatrices. The partitioning of c and d is shown in Fig. 1(b) and Fig. 1(c) respectively, where c_j's $(0 \leqslant j \leqslant 2^n - 1)$ and d_i's $(0 \leqslant i \leqslant 2^m - 1)$ represent row subvectors of length k_2 and column subvectors of length k_1 respectively. The mapping of A, c, and d onto the hypercube is as follows:

(1) A_{ij} is placed into the processor with address $(b_h, b_{h-1}, \ldots, b_{n+1}, b_n, \ldots, b_1)$, where $b_h b_{h-1} \ldots b_{n+1}$ is the binary representation of i and $b_n \ldots b_1$ is the binary representation of j.

(2) c_j is placed into the processor with address $(0, 0, \ldots, 0, b_n, \ldots, b_1)$, where $b_n \ldots b_1$ is the binary representation of j.

(3) d_i is placed into the processor with address $(b_h, b_{h-1}, \ldots, b_{n+1}, 1, 1, \ldots, 1)$, where $b_h b_{h-1} \ldots b_{n+1}$ is the binary representation of i.

(4) z is placed into the processor with address $(0, 0, \ldots, 0, 1, \ldots, 1)$ (that is, $b_h = b_{h-1} = \cdots = b_{n+1} = 0$ and $b_n = \cdots = b_1 = 1$).

Table 1 shows the data mapping for $M = 4$, $N = 8$, $h = 4$, $k_1 = 1$, and $k_2 = 2$.

According to the above mapping, four facts are as follows.

Table 1
The data mapping for $M=4$, $N=8$, $h=4$, $k_1=1$ and $k_2=2$; the 4-digit binary numbers indicate the addresses of processors

0000		0001		0010		0011	Z
C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7 d_0
a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}
0100		0101		0110		0111	
							d_1
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}
1000		1001		1010		1011	
							d_2
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	a_{26}	a_{27}
1100		1101		1110		1111	
							d_3
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	a_{36}	a_{37}

Fact 1. For any fixed i, the submatrices A_{ij}'s, $0 \leqslant j \leqslant 2^n - 1$, are mapped onto an n-dimensional embedded hypercube on $(b_n, \ldots, b_1)$. Moreover, each of the processors in the embedded hypercube has the most significant m bits equal to the binary representation of i. For easy description, we refer to this embedded hypercube as *row-i embedded hypercube* in the following discussion.

Fact 2. For any fixed j, the submatrices A_{ij}'s $0 \leqslant i \leqslant 2^m - 1$, are mapped onto an m-dimensional embedded hypercube on $(b_h, b_{h-1}, \ldots, b_{n+1})$. Moreover, each of the processors in the embedded hypercube has the least significant n bits equal to the binary representation of j. In the following discussion, we refer to this embedded hypercube as *column-j embedded hypercube*.

Fact 3. The row vector c is mapped onto the row-0 embedded hypercube.

Fact 4. The column vector d is mapped onto the column-$(2^n - 1)$ embedded hypercube.

Now, it is time to describe the implementation. Since the simplex method consists of a finite number of iterations, we shall concentrate our efforts on the necessary operations of each iteration. These operations include, determining the pivot row, the pivot column, and the pivot element, and updating A, d, c, and z. In the following, we describe their implementation.

Determine the pivot column

The operands required for this operation are c. According to Fact 3, we know that k_2 elements of c are distributed in each processor of the row-0 embedded hypercube. Thus, the most negative element in each processor is determined first. This takes $k_2 - 1$ computation steps. Then, according to Property 2, the most negative element of c can be determined and broadcast to all the processors of the row-0 embedded hypercube in n computation steps and $2n$ communication steps. The pivot column is the $(u+1)$th column of A, where c_u is the most negative element of c. After the pivot column is determined, u, the index of the pivot column, is broadcast on all column-j embedded hypercubes, $0 \leqslant j \leqslant 2^n - 1$, simultaneously. This operation needs m communication steps. In case of no negative elements, the current value of z is optimal.

Determine the pivot row and the pivot element

The operands required for this operation are d and the pivot column. First, each subvector d_i, $0 \leqslant i \leqslant 2^m - 1$, is sent to the processor that holds A_{iy}, where $y = u$ DIV k_2 and u is the index of the pivot column. The computation of y takes only one computation step. Since d_i and A_{iy} are in the same embedded hypercube (the row-i embedded hypercube), the transmission can be pipelined and performed in parallel. Also since the used embedded hypercubes are disjoint, no data congestion is possible. Therefore, $k_1 + n - 1$ communication steps are required. Then , the minimum of d_i/a_{iu}'s with $a_{iu} > 0$ is determined in each processor. This takes at most $3k_1 - 1$ computation steps. Finally, the minimum of these 2^m minima is determined. Since these minima are in the column-y embedded hypercube, $2m$ communication steps and m computation steps are required. The pivot row is the $(t+1)$th row of A, where d_t/a_{tu} is the minimum. Besides, a_{tu} is the pivot element.

Update A, d, c and z

(a) The pivot element is broadcast on the row-(t DIV k_1) embedded hypercube, and then the pivot row is updated according to (2). This takes n communication steps and one computation step.

(b) The elements of the pivot column are broadcast on all row-i embedded hypercubes, $0 \leqslant i \leqslant 2^m - 1$. Since the broadcasting can be pipelined and performed in parallel, $k_1 + n - 1$ communication steps are required.

(c) The elements of the pivot row are broadcast on all column-j embedded hypercubes, $0 \leqslant j \leqslant 2^n - 1$. This takes $k_2 + m - 1$ communication steps.

(d) The value d_t is updated according to (4). This takes one computation step.

(e) The value d_t is broadcast on the column-$(2^n - 1)$ embedded hypercube. This takes m communication steps.

(f) A, c, d, and z are updated according to (3), (5), (6), and (7) respectively. This takes totally $2(k_1k_2 + k_1 + k_2 + 1)$ computation steps.

Thus, the total number of (parallel) computation steps required for each iteration is

$$\begin{aligned}&(k_2 - 1) + n + 1 + (3k_1 - 1) + m + 1 + 1 + 2(k_1k_2 + k_1 + k_2 + 1)\\ &\quad = 2k_1k_2 + 5k_1 + 3k_2 + (m+n) + 3 = 2k_1k_2 + 5k_1 + 3k_2 + h + 3 \end{aligned} \tag{8}$$

and the total number of communication steps required for each iteration is

$$\begin{aligned}&2n + m + (k_1 + n - 1) + 2m + n + (k_1 + n - 1) + (k_2 + m - 1) + m\\ &\quad = 2k_1 + k_2 + 5(m+n) - 3 = 2k_1 + k_2 + 5h - 3. \end{aligned} \tag{9}$$

Since $k_1k_2 = MN/p$, the linear speedup is achieved asymptotically.

Since asymptotical analysis is only of theoretical interest and M, N are finite in practice, we are more interested in how to choose k_1 and k_2 so as to minimize the total run time and the communication time. Let α be the ratio of the unit time for computation to the unit time for communication. The problem can be stated as follows. Given $p = 2^h$, M, N, and α, determine n, m, k_1, and k_2 satisfying

$$h = n + m, \qquad M = k_1 2^m, \qquad N = k_2 2^n$$

so as to minimize

$$\alpha(2k_1k_2 + 5k_1 + 3k_2 + h + 3) + (2k_1 + k_2 + 5h - 3) \quad \text{(Objective 1)} \tag{10}$$

and

$$2k_1 + k_2 + 5h - 3. \quad \text{(Objective 2)}$$

Objective 1. To minimize the total run time. Substituting MN/pk_1 for k_2 with respect to (10), then differentiating it at k_1, and equalizing it to 0, we have

$$5\alpha - (3\alpha + 1)MN/pk_1^2 + 2 = 0. \tag{11}$$

Solving (11), we have $k_1 = (((3\alpha + 1)/(5\alpha + 2))(MN/p))^{1/2}$. The values k_2, n, and m can be determined accordingly. Since k_1 is not an integer, the integer closest to k_1 is chosen.

Objective 2. To minimize the communication time. Set $\alpha = 0$ with respect to (11) and then k_1 becomes $(MN/2p)^{1/2}$. If k_1 is not an integer, the integer closest to k_1 is chosen.

In the above discussion, pipelined broadcasting is allowed. If pipelined broadcasting is not allowed, broadcasting k_1 (k_2) data items on an n (m)-dimensional embedded hypercube requires $k_1 n$ ($k_2 m$) communication steps. Thus, the total number of communication steps required for each iteration changes to

$$2k_1 n + (k_2 + 1)m + 3h. \tag{12}$$

The linear speedup is still achieved asymptotically in this case.

We now proceed to discuss the optimal value of k_1.

Objective 1. To minimize the total run time. We need to minimize

$$\alpha(2k_1k_2 + 5k_1 + 3k_2 + h + 3) + (2k_1 n + (k_2 + 1)m + 3h). \tag{13}$$

Substituting MN/pk_1 for k_2, $\log_2(pk_1/M)$ for n, and $\log_2(M/k_1)$ for m, (13) becomes

$$\alpha(2MN/p + 5k_1 + 3MN/pk_1 + h + 3) + 2k_1 \log_2(pk_1/M) + (MN/pk_1 + 1)\log_2(M/k_1) + 3h. \tag{14}$$

Then, differentiating (14) at k_1 and equalizing it to 0, we have

$$\alpha(5 - 3MN/pk_1^2) + 2\log_2(pk_1/M) - (MN/pk_1^2)\log_2(M/k_1) + (2 - (1/k_1 + MN/pk_1^2))\log_2 e = 0. \tag{15}$$

To solve (15) for k_1 is not easy; however, it is helpful to get a better partition.

Objective 2. To minimize the communication time. Set $\alpha = 0$ with respect to (15) and we can see that the optimal value of k_1 is not easy to be obtained. However, we shall show in the following that the optimal value of k_1 can easily be found for a special case.

When M and N are much greater than p, the constant 1 in (12) is negligible as compared with k_2 and therefore (12) can be simplified to $2k_1 n + k_2 m + 3h$. According to the following theorem, we know that the optimal value of k_1 for the simplified case occurs at $2k_1 = k_2$ when $2M = N$.

Theorem 1. *If the number of communication steps required can be expressed in the form of*

$$ak_1 n + bk_2 m + c, \tag{16}$$

where a, b, and c are constants, then $ak_1 = bk_2$ *minimizes* (16) *when* $aM = bN$.

Proof. Substituting MN/pk_1 for k_2, $\log_2(pk_1/M)$ for n, and $\log_2(M/k_1)$ for m, (16) becomes

$$ak_1 \log_2(pk_1/M) + b(MN/pk_1)\log_2(M/k_1) + c. \tag{17}$$

Differentiating (17) and equalizing it to 0, we have

$$a \log_2(pk_1/M) - b\left(MN/pk_1^2\right) \log_2(M/k_1) + \left(a - b\left(MN/pk_1^2\right)\right) \log_2 e = 0. \tag{18}$$

It is not difficult to check that $ak_1 = bk_2$ is a solution of (18) when $aM = bN$. Thus, this theorem follows. □

4. Concluding remarks

The simplex method is a well-known solution method for the linear programming problem. In this paper, we have proposed an implementation of the simplex method on a fixed-size hypercube. Two cases, pipelined broadcasting allowed and pipelined broadcasting not allowed, were considered. For both cases, the linear speedup is achieved asymptotically. Under the proposed data mapping method, we also derived two sufficient conditions for optimal partitionings. It is not easy to derive optimal partitionings from these two conditions; however, suboptimal partitionings can be obtained with the aid of them. We have obtained the optimal partitioning for a special case. Although the simplex method can also be implemented on other parallel machines [1,4], they do not achieve the asymptotically linear speedup. Moreover, the number of processors they use is dependent upon the problem instance size.

Since the linear programming problem we considered in this paper is in standard form, a basic feasible solution must be provided initially. This can be accomplished using the two-phase technique [5]. In phase 1, we first augment new 'artificial' variables as necessary to secure a starting solution, and then seek the minimization of the sum of the artificial variables. In Phase 2, we use the optimum basic solution of Phase 1 as a starting solution to minimize the objective function of the original problem.

In the proposed implementation, the processor with address $(0, 0, \ldots, 0, 1, \ldots, 1)$ ($b_h = b_{h-1} = \cdots = b_{n+1} = 0$ and $b_n = \cdots = b_1 = 1$) has more computation loads than others. If its computation loads can be shared by the other processors, then the number of computation steps required for each iteration can further be reduced. To do this is not difficult. We can simply let $M = k_1 2^m - 1$ and $N = k_2 2^n - 1$ and reduce the size of A_{0j}'s and $A_{i(2^n-1)}$'s, where $0 \leqslant j \leqslant 2^n - 1$ and $0 \leqslant i \leqslant 2^m - 1$, to $(k_1 - 1) \times (k_2 - 1)$.

Acknowledgment

The authors wish to thank the Editor and the anonymous referees for their many valuable comments and suggestions. The authors are also grateful to the Advanced Technique Center of Electronic Research Service Organization, Industrial Technique Research Institute, Republic of China for supporting this research.

References

[1] A.A. Bertossi and M.A. Bonuccelli, A VLSI implementation of the simplex algorithm, *IEEE Trans. Comput.* **36** (2) (1987) 241–247.

[2] P.R. Cappello, Gaussian elimination on a hypercube automaton, *J. Parallel Distrib. Comput.* **4** (1987) 288–308.

[3] G.B. Dantzig *Linear Programming and Extensions* (Princeton University Press, Princeton, NJ, 1963).

[4] K. Onaga and H. Nagayasu. A wavefront-driven algorithm for linear programming on dataflow processor-arrays, in: *Proc. International Computer Symposium* (1984) 739–746.
[5] C.H. Papadimitriou and K.S. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity* (Prentice-Hall, Englewood Cliffs, NJ, 1983).
[6] M.J. Quinn, *Designing Efficient Algorithms for Parallel Computers* (McGraw-Hill, New York, 1987).
[7] A.H. Taha, *Operations Research: An Introduction* (Macmillan, New York, 1971).