American Journal of Applied Sciences 1 (2): 90-94, 2004
ISSN 1546-9239

Solving Linear Programming Problems on the Parallel Virtual Machine Environment

[1]JrJung Lyu, [2]Hsing Luh and [1]Ming-Chang Lee
[1]National Cheng Kung University, Taiwan
[2]National Cheng Chi University, Taiwan

Abstract: This study developed a parallel algorithm to efficiently solve linear programming models. The proposed algorithm utilizes the Dantzig-Wolfe Decomposition Principle and can be easily implemented in a general distributed computing environment. The analytical performance of the well-known method, including the speedup upper bound and lower bound limits, was derived. Numerical experiments are also provided in order to verify the complexity of the proposed algorithm. The empirical results demonstrate that the speedup of this parallel algorithm approaches linearity, which means that it can take full advantage of the distributed computing power as the size of the problem increases.

Key words: Linear programming, methodology

INTRODUCTION

Linear Programming (LP) involves a sequence of steps that will lead to the most effective way to allocate scarce resources among competing activities. LP is widely used in a number of areas to help managers make decisions, such as assigning jobs to machines, mixing ingredients for a product, determining a distribution system and other situations. An LP model consists of an objective function to be optimized and mathematical statements of the constraints. Given that many LP models represent large and complex physical systems, a typical medium-sized LP model might have 20,000 variables and 5,000 constraints [1]. The required computing resources for solving a modest LP application are therefore huge.

The availability of cost-effective parallel computers has shown the potential of distributed computing power for many large-scale mathematical programming problems. Some previous studies have developed interesting results in this area [2, 3]. However, exploiting parallelism with a mathematical programming algorithm is not always easy due to the communication complexity between processors, which often becomes a bottleneck during the execution process. Despite many software tools developed for the distributed computing environment, to convert a conventional application into a parallel application remains very difficult. For instance, many Operations Research textbooks [4] have introduced the simplex method, which is an algebraic procedure for solving linear programming problems.

The simplex method improves the feasible solution in an orderly manner by performing a series of elementary row operations until the optimality is achieved. To execute the simple method in a parallel mode is apparently difficult due to its sequential nature.

In this study, we developed a parallel algorithm, based on the Dantzig-Wolfe Decomposition Principle (DWDP), to solve linear programming and other block-type optimization problems. Although [5] introduced the decomposition principle in the early sixties, it is still widely adopted to cope with large-scale optimization problems. Given that larger and more complex mathematical models have become commonplace [6], the importance of the DWDP is well recognized by researchers. Using both analytical studies and numerical analysis, we show that the proposed parallel algorithm can be executed efficiently in a general distributed computing environment.

Description of the algorithm: Consider a linear programming problem that can be expressed in the following form:

$$\text{Minimize } c^T x \qquad (1)$$

$$\text{Subject to: } Ax = b \; x \_ 0$$

where, A is a matrix of order m by k, c and x are both k-dimensional vectors and b is a m-dimensional vector with each component nonnegative. It is observed that the A matrix in many large linear programming problems usually has a special block-angular structure, namely:

$$A = \begin{bmatrix} L_1 & L_2 & \cdots & L_n & L_n \\ A_1 & & & & \\ & A_2 & & & \\ & & . & & \\ & & & . & \\ & & & & . \\ & & & & A_n \end{bmatrix}$$

where, all A_i in the technology matrix A are independent blocks linked by coupling-equation matrices L_i. As the angular-structure appears, the decomposition principle is substantiated by forming an equivalent master program (defined below) and several sub problems which correspond to each sub-matrix A_i. The solution procedure for (1) involves iterations between a set of independent sub problems where their objective functions are formed using parameters derived from the master program.

Suppose each A_i has m_i rows and k_i columns and each L_i is an m0 ×k_i matrix, for i = 1, 2, …, n. By partitioning vectors b, x and c into sizes corresponding to each A_i, problem (1) can be rewritten as follows:

$$\text{Minimize} \sum_{i=1}^{n} c_i^T x_i \qquad (2)$$

$$\text{Subject to}: \sum_{i=1}^{n} L_i x_i = b_0$$

$$X_i \in \Omega_i$$

where, $\Omega_i = \{A_i x_i = b_i,\ x_i \geq 0\}$ for i = 1, 2, …, n. Apparently, the set Ω_i is convex and mutually independent.

We can then define the sub problem i, for i = 1, 2, …, n, as:

$$\text{Minimize } (c_i^T - \lambda_0^T L_i)\, x_i \qquad (3)$$

Subject to: $A_i\, x_i = b_i$,
$x_i \geq 0$

where, λ_0^T is the vector denoting the simplex multipliers corresponding to the constraint $\sum_{i=1}^{n} L_i x_i = b_0$.

In contrast to the sub problem (3), problem (2) calls the master program. Based on the property of convexity of (2) and (3), which implies that all solutions can be written as a linear combination of their vertices, a two-level algorithm for the solution of the linear programming problem can be developed. The master program is on the first level in searching for the coefficients of the linear combination and the subproblem (3) is on the second level of solving the possible optimal vertices. Details of this two-level algorithm, which applies the Dantzig-Wolfe decomposition principle, can be found in [7].

Assume that there exists a Distributed Computing Environment (DCE) in which the processing units are independent machines, connected by a network, and a centralized processor (or the master processor) serves as the coordinator. Such an environment has proven to be a viable approach to provide concurrent computing power at reasonable costs [8]. The design of an algorithm for DCE requires tight load balancing in order to reduce the communication overhead and obtain good performance[9]. A parallel two-level algorithm for linear programming problems that can be implemented in a general DCE is described below.

Algorithm 1: Parallel LP Method

Step 1: Initiate the distributed computing environment by creating n processes in the network and assign one of these processes as the master process to coordinate the computing tasks.

Step 2: Let basis matrix B = I, is an identity matrix.

Step 3: The master process solves the current basic solution X_B, and finds the simplex multipliers $\lambda^T = B^{-1} c_B^T$ where $\lambda^T = (\lambda_0^T, \bar{\lambda})$ and $\bar{\lambda} = (\bar{\lambda}_1, \bar{\lambda}_2, ..., \bar{\lambda}_1)$.

Step 4: The master process broadcasts necessary data to each child-process and assigns the i^{th} child-process to solve the i^{th} sub problem (as denoted in equation (3)) where each child-process calculates
$r_i^* = (c_i^T - \lambda_0^T L_i) x_i^* - \bar{\lambda}_i$ for i = 1, 2, …, n.

Step 5: Once the *i*th child-process solves the i^{th} sub problem, it sends r_i^* and x_i^* to the master process. After all of the processes return their solutions, if all $r_i^* \geq 0$, then the algorithm terminates. Otherwise, go to Step 6.

Step 6: The master process determines which column the basis is entered by selecting the minimum value r_i^* of the sub problems. Let $\begin{pmatrix} L_i x_i^* \\ e_i \end{pmatrix}$ be the column that will enter the current basis B, where e_i is a unit vector.

Step 7: The master process updates B^{-1} and go to Step 3. In the presented algorithm, Step 1 declares the distributed computing environment and creates n child processes. Step 2 assigns the initial basic feasible solution for the master problem. Each child-process uses the simplex multipliers λ^T, found in Step 3, to solve the i^{th} sub problem in Step 4. Note that this step is the most time consuming part in a sequential algorithm because n linear programming models must be solved. The master process collects the solutions obtained from each sub problem, determines the optimal solution x_i^* and the associated optimal objective value r_i^* and checks the optimal condition in Step 5. If the condition is satisfied, x_i^* is the extreme point of Ω_i and the optimum is found. Step 6 constructs the corresponding vector that will enter the basis of the master program if the terminating condition is not satisfied. The solutions x_i^* for the i^{th} sub problems are then sent to the master program, which combines these inputs to update the basic solution matrix and determines a new λ^T. The result is again sent to each child-process and the iteration proceed until an optimality test is satisfied.

Analytical performance: To evaluate the performance of the proposed algorithm we will investigate to performance complexity. The presented algorithm could be implemented into a general distributed computing environment consisting of a network of heterogeneous computers.

Assume that the parallel algorithm uses a cluster of p workstations connected in a DCE and terminated in time T_p. Let T_s be the best possible time required to solve the same problem using a sequence (uni-processor) algorithm. The ratio:

$$S_p = T_s/T_p \quad (4)$$

Is called the algorithm speed up [10]. Speedup is one of the most common indicators for measuring the efficiency of a parallel algorithm. For simplicity, assume that there are enough processors to execute the n sub problem in parallel. If α is the execution time of the inherently sequential proportion of the algorithm, and β is the remaining proportion that could be performed in parallel (the execution time is β/n for a system with n processor) then $T_s = t(\alpha + \beta)$, and $T_p = t\left(\alpha + \frac{\beta}{n}\right)$. Therefore, the speedup could be approximated by:

$$S_p \cong \frac{\alpha + \beta}{\alpha + \frac{\beta}{n}} \quad (5)$$

When α is a proportion of β, that is, $\alpha = w\beta$, then:

$$S_p \cong \frac{nw + n}{nw + 1} \quad (6)$$

We can further derive the speedup upper bound and lower bound limits as follows:

$$\text{Upper bound of } S_p = \lim_{w \to 0} S_p = n \quad (7)$$

$$\text{Upper bound of } S_p = \lim_{w \to \infty} S_p = 1 \quad (8)$$

Now, when the number of sub problems, n, is greater than the number of processors, p, that are available, and assume that n = (p-1) q, the speedup could be approximated by:

$$\begin{aligned} S_p &\cong \frac{nw + n}{nw + 1} \\ &= \frac{(p-1)qw + (p-1)q}{(p-1)qw + 1} < \frac{(p-1)qw + (p-1)q}{(p-1)qw + q} \\ &= \frac{(p-1)w + (p-1)}{(p-1)w + 1} \end{aligned} \quad (9)$$

The speedup upper bound and lower bound limits can now be derived as follows:

$$\text{Upper bound of } S_p = \lim_{w \to 0} S_p = p - 1 \quad (10)$$

$$\text{Lower bound of } S_p = \lim_{w \to \infty} S_p = 1 \quad (11)$$

From the above analysis, it is apparent that the ratio w is critical to the speedup of the parallel algorithm. In the proposed algorithm, since Step 4 is the most computationally intensive part, which is required to solve a linear programming model, the ratio w is therefore very small and the speedup should be approximate to the upper bound limit. That is, the speedup would approach n when there are enough processors in the DCE.

RESULTS AND DISCUSSION

The algorithm presented was implemented on a distributed network of workstations consisting of 26 SUN-lx SPARC workstations. These workstations were connected via an optical fiber link. The code was programmed in FORTRAN/77 using the Parallel Virtual Machine (PVM) system. PVM enables a collection of heterogeneous computer systems to be viewed as a single parallel virtual machine and has been widely adopted by researchers [11].

Three types of randomly generated test problems were solved to investigate the performance of the algorithm. The method for generating the linear programming models was similar to the method proposed by [12], where the number of constrains ranged from 20 to 50 to 124. For each problem type, five sets of models were generated using different random-number generator seeds. The results obtained in Table 1 represent the average CPU time (five replications for each instance) utilized for three different types of test problems using 1, 2, 3, 4, 5 and 6 processors in the DCE.

The main objective of our computational experiments was to assess the problem size and the number of processors on the performance of the proposed parallel algorithm. We also used the numerical results to justify the analytical performance of the algorithm. Table 1 shows the average CPU time with respect to the various numbers of processors. The speedup of the proposed algorithm was also calculated, based on the equation (4), and its correlation with the number of processors is plotted in Fig. 1.

Table 1: Average CPU time (seconds) versus number of processors

Number of constraints	1 processors	2 processors	3 processors	4 processors	5 processors	6 processors
20	136.10	93.25	48.50	38.12	29.86	23.65
50	376.97	248.9	133.55	102.14	78.04	64.70
124	852.64	553.67	291.54	224.45	201.87	162.4

*: The average CPU time is from the mean of five replications for each instance

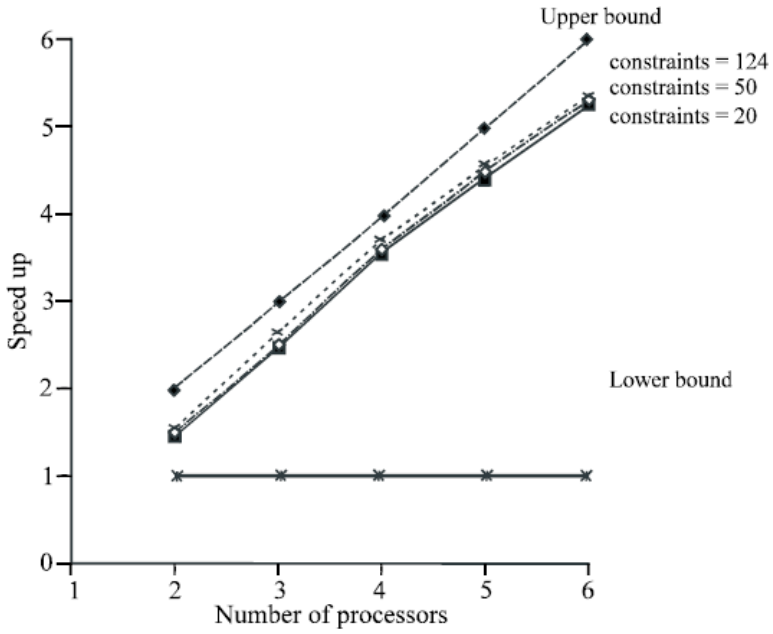

Fig. 1: Speedup versus number of processors

While PVM is easy to implement on a cluster of workstations, the performance of the proposed algorithm is still impressive. The CPU time is apparently shorter when more processors are available. The best speedup obtained was 5.38 for the model with 124 constraints (big problem size), executed in a system with 6 processors. Even for the small sized problem model (20 constraints), the speedup reached 5.25. In general, the speedup increased with the problem size and also with the number of available processors. The near linear speedup was achieved, which was consistent with the complexity derived from the analytical analysis. Despite the communication overhead during the execution, the proposed algorithm was very efficient in solving LP models in a distributed computing environment.

CONCLUSION

Distributed computing on clusters of workstations is attractive and cost-effective to researchers for evolving processing and networking technologies. This study developed a parallel linear programming algorithm and evaluated its performance on a DCE. The numerical results show that the speedup of the proposed algorithm approaches linearity, which is consist with its analytical performance. We conclude that the presented algorithm is efficient and becomes a useful reference solution model for LP applications, especially for large-scale problems. The proposed algorithm was implemented on a PVM system (a portable distributed computing environment). This software is free to the public and has been installed on many networked computing platforms. We are currently working on porting our computer codes into other computing environments and testing the algorithm on a wider variety of test problems. It is safe to state that further study of the development of some mathematical programming algorithms that could also take advantage of distributed computing power requires greater research efforts.

REFERENCES

1. Forrest, J. J. H. And J.A. Tomlin, 1992. Implementing the simplex method for the optimization subroutine library. IBM Systems J., 31: 11-25.
2. Lyu, J., A. Gunasekaran and V. Kachitvichyanukul, 1995. Towards a portable and efficient environment for parallel computing. Int. J. Systems Sci. 26: 1333-1341.
3. Romeijn, H. E. And R. L. Smith, 1999. Parallel algorithms for solving aggregated shortest-path problems. Computers and Operations Res., 26: 941-953.
4. Hiller, F. S., G. J. Lieberman, and G. Lieberman, 1995. Introduction to Operations Res. (New York: McGraw-Hill).
5. Dantzig, G. B. And P. Wolfe, 1960. The decomposition principle for linear programs. Operations Res., 8: 101-111.
6. Chinneck, J. W., 1996. Computer codes for the analysis of infeasible linear programs. J. Operational Res. Soc., 47: 61-72.
7. Martinson, R.K. and J. Tind, 1999. An interior point method in Dantzig-Wolfe decomposition. Computers and Operations Res., 26: 1195-1216.

8. Sunderam, V. S., 1997. Heterogeneous network computing: the next generation. Parallel Computing, 23: 121-135.
9. Sekharan, C. N., V. Goel and R. Sridhar, 1995. Load balancing methods for ray tracing and binary tree computing using PVM. Parallel Computing 21: 1963-1978.
10. Quinn, M. J., 1994. Parallel Computing: Theory and Practice (McGraw Hill, N.Y).
11. Sunderam, V. S., G. A. Geist, J. Dongarra and R. Manchek, 1994. The PVM concurrent computing system: evolution, experiences, and trends. Parallel Computing, 20: 531-545.
12. Pisinger, D., 1995. An expanding-core algorithm for the exact 0-1 Knapsack problem. European J. Operational Res., 87: 175-187.