An International Journal
computers & mathematics with applications

PERGAMON Computers and Mathematics with Applications 44 (2002) 1431–1437
www.elsevier.com/locate/camwa

Performance Analysis of a Parallel Dantzig-Wolfe Decomposition Algorithm for Linear Programming

JRJUNG LYU*
Department of Industrial Management Science
National Cheng Kung University
Tainan, Taiwan, R.O.C.
jlyu@mail.ncku.edu.tw

HSING LUH
National Cheng Chi University
Tainan, Taiwan, R.O.C.

MING-CHANG LEE
National Cheng Kung University
Tainan, Taiwan, R.O.C.

(Received October 2000; accepted August 2001)

Abstract—This paper employs the Dantzig-Wolfe decomposition principle to solve linear programming models in a parallel-computing environment. Adopting the queuing discipline, we showed that under very general conditions, the proposed algorithm speedup trends toward a limiting value as the number of processors increases. © 2002 Elsevier Science Ltd. All rights reserved.

Keywords—Linear programming, Decomposition algorithms, Parallel processing.

1. INTRODUCTION

Consider a linear programming (LP) problem that can be expressed in the following form:

$$\begin{aligned} \text{Minimize:} \quad & \mathbf{C}^{\mathsf{T}}\mathbf{X}, \\ \text{Subject to:} \quad & \mathbf{AX} = \mathbf{b}, \\ & \mathbf{X} \geq 0. \end{aligned} \tag{1}$$

Suppose the **A** matrix has a special block-angular structure, namely,

$$\mathbf{A} = \begin{bmatrix} \mathbf{L}_1 & \mathbf{L}_2 & \dots & \mathbf{L}_n \\ \mathbf{A}_1 & & & \\ & \mathbf{A}_2 & & \\ & & \ddots & \\ & & & \mathbf{A}_n \end{bmatrix},$$

*Author to whom all correspondence should be addressed.

where all $\mathbf{A}_i$ in the technology matrix $\mathbf{A}$ are independent blocks linked by coupling-equation matrices $\mathbf{L}_i$. In this research, a parallel algorithm, based on the Dantzig-Wolfe decomposition principle (DWDP), was developed to solve the linear programming problems as stated above. Since Dantzig and Wolfe developed the decomposition principle in the early sixties, this method is still widely adopted to cope with large-scale optimization problems. For instance, Ziarati, *et al.* [1] considered a multicommodity flow model for assigning locomotives to train-segments while employing DWDP to solve a very large-scale-scheduling problem. Desaulniers, *et al.* [2] used DWDP to solve the problem of choosing the best crew pairing at Air France. Other examples can be found in [3–5]. Given that larger and more complex mathematical models have become commonplace (see [6]), the importance of DWDP is well recognized by researchers.

An ideal Dantzig-Wolfe decomposition model considers a typical linear programming problem whose technological matrix is block-type. Since each block-type submatrix can be transformed into an independent subproblem while maintaining the global optimum, an ideal Dantzig-Wolfe decomposition algorithm represents a limiting case for the effects of parallelism. When a parallel Dantzig-Wolfe decomposition algorithm is developed, it can be shown that there is no wasted time associated with communication delays where the only source of communication delay is the actual time used in executing the program. This research, therefore, provides an upper bound for algorithm speedup using parallel processing for linear programs.

2. A PARALLEL LP ALGORITHM

In problem (1), let each $\mathbf{A}_i$ have m_i rows and p_i columns and each $\mathbf{L}_i$ be an $m_0 \times p_i$ matrix, for $i = 1, 2, \dots, n$. Letting $m = \sum_{i=0}^{n} m_i$ and $p = \sum_{i=1}^{n} p_i$, $\mathbf{A}$ is, therefore, an $m \times p$ matrix. By partitioning vectors $\mathbf{b}$, $\mathbf{X}$, and $\mathbf{C}$ into sizes corresponding to each $\mathbf{A}_i$, problem (1) can be rewritten as follows.

$$\begin{aligned} \text{Minimize:} \quad & \sum_{i=1}^{n} \mathbf{C}_i^{\mathrm{T}} \mathbf{X}_i, \\ \text{Subject to:} \quad & \sum_{i=1}^{n} \mathbf{L}_i \mathbf{X}_i = \mathbf{b}_0, \\ & \mathbf{A}_i \mathbf{X}_i = \mathbf{b}_i, \\ & \mathbf{X}_i \geq \mathbf{0}. \end{aligned} \tag{2}$$

We can then define the subproblem i, for i=1, 2, ... , n, as:

$$\begin{aligned} \text{Minimize:} \quad & \left(\mathbf{C}_i^{\mathrm{T}} - \boldsymbol{\lambda}_0^{\mathrm{T}} \mathbf{L}_i\right) \mathbf{X}_i, \\ \text{Subject to:} \quad & \mathbf{A}_i \mathbf{X}_i = \mathbf{b}_i, \\ & \mathbf{X}_i \geq \mathbf{0}, \end{aligned} \tag{3}$$

where $\boldsymbol{\lambda}_0^{\mathrm{T}}$ is the vector denoting the simplex multipliers corresponding to the constraint $\sum_{i=1}^{n} \mathbf{L}_i \mathbf{X}_i = \mathbf{b}_0$.

In contrast to subproblem (3), problem (2) is called the master program. Based on the convexity properties of problems (2) and (3), which imply that all solutions can be written as a linear combination of their vertices, a two-level algorithm based on DWDP can be developed. In this algorithm, the master program is on the first level to search for the coefficients of the linear combination. Subproblem (3) is on the second level to solve the possible optimal vertices.

Consider that there exists a distributed computing environment (DCE), which has more than n independent workstations connected by a network and a centralized processor (or the master processor) to serve as the coordinator. Such a framework has proven to be a viable approach to provide concurrent computing power at reasonable costs [7]. Procedure 1 describes an algorithm based on the Dantzig-Wolfe decomposition principle (DWDP) that can be executed on the DCE.

PROCEDURE 1. Parallel DWDP algorithm.

Step 1 Initiate the distributed computing environment by creating n processes in the network and identify one of these processes as the master process to coordinate the computing tasks. Let basis matrix $\mathbf{B} = \mathbf{I}$ be an identity matrix.

Step 2 The master process solves the current basic solution $\mathbf{X_B}$, and determines the simplex multipliers $\boldsymbol{\lambda}^\top = (\boldsymbol{\lambda}_0^\top, \bar{\boldsymbol{\lambda}}) = \mathbf{B}^{-1}\mathbf{C}_\mathbf{B}^\top$ where $\bar{\boldsymbol{\lambda}} = (\bar{\lambda}_1, \bar{\lambda}_2, \dots, \bar{\lambda}_n)$.

Step 3 The master process broadcasts necessary data to each child-process and assigns the i th process to solve $\mathbf{X}_i^*$ of the i th subproblem (as denoted in equation (3)) and calculates $r_i^* = (\mathbf{C}_i^\top - \boldsymbol{\lambda}_0^\top \mathbf{L}_i)\mathbf{X}_i^* - \bar{\lambda}_i$, for $i = 1, 2, \dots, n$.

Step 4 Once the i th process solves the i th subproblem, it sends r_i^* and $\mathbf{X}_i^*$ to the master process. After all of the processes return their solutions, if all $r_i^* \geq 0$, then the optimal is found and the algorithm terminates. Otherwise, go to Step 5.

Step 5 The master process determines which column enters the basis by selecting the minimum value r_i^* of the subproblems. Let $\begin{pmatrix} \mathbf{L}_i \mathbf{X}_i^* \\ \mathbf{e}_i \end{pmatrix}$ be the column that will enter the current basis $\mathbf{B}$, where $\mathbf{e}_i$ is a unit vector.

Step 6 The master process updates $\mathbf{B}^{-1}$ and go to Step 2.

3. PERFORMANCE OF THE PROPOSED ALGORITHM

Assume that the proposed algorithm is implemented in a cluster of n workstations connected in a distributed computing environment and terminates in time t_n. Let t_s be the best possible time required for solving the same problem using a sequential algorithm. The speedup can be defined as the ratio

$$\text{Speedup} = \frac{t_s}{t_n}. \tag{4}$$

Speedup is one of the most common indicators for measuring the efficiency of a parallel algorithm [8]. The attainable speedup of an algorithm is certainly a random variable since it is affected by many factors in a distributed computing environment. Under very general conditions, speedup tends to be a limiting value called mean speedup [9], which can be defined as follows.

$$\text{Mean speedup} = \frac{E\text{ (calculation time using one processor)}}{E\text{ (calculation time using } n \text{ processors)}}, \tag{5}$$

where $E(.)$ is the usual expectation. In a stochastic environment, mean speedup represents the upper bound of the algorithm speedup.

Suppose that a LP problem can be solved within finite iterations in DCE. Let u_{ij} be the computation time for the i th processor to complete its task in the j th stage and c_{ij} be the communication time for a message transmission from the moment of completion of a given task in the j th stage at the i th processor until all processors have received that message. Assume that both u_{ij} and c_{ij} are i.i.d., and that they are independent. In executing the proposed algorithm, since each iteration is independent and has an identical stochastic structure, we can, therefore, omit the subscript j and have the mean speedup as follows.

$$\text{Mean speedup} = \frac{nE(u_1)}{E(t_n)}. \tag{6}$$

In each iteration of the proposed algorithm, a simplex direction search is performed, where the columns generated by subproblems are searching for an entering variable using the simplex method. We can analyze a single iteration as a finite-population G/G/1 queue, where each simplex method can be treated as a "customer" (job) that is "served" (solved) by a processor. Similar to a G/G/1 queue, the job processing may be interrupted before completion. The discipline for this operation is FIFO (first in first out). Let $u_{(n)}$ be the maximum of $u_1, \dots, u_n$ which is the time the last processor finished its computations, or, from the queueing perspective, the arrival time of the n^{th} job. Let a_i be the accumulated communications backlog at the time, when the i^{th} job was finished and c_i is the communication time for the job performed by the i^{th} processor. Thus, we may write $t_n = u_{(n)} + a_n + c_n$. We can then extend (6) as follows.

$$\text{Mean speedup} = \frac{nE(u_1)}{E\left(u_{(n)} + a_n + c_n\right)}. \tag{7}$$

Taking a distributed computing system as the server in a queuing system, we observe that the computation time required is always greater than or equal to the time it takes for all of the jobs to be served. The n^{th} job to enter the queue will not wait longer than the sum of the service times of the $n-1$ jobs that entered the queue before it. We thus have CPU time bounds as follows.

$$c_1 + \cdots + c_n \le t_n \le u_{(n)} + c_1 + \cdots + c_{n-1} + c_n. \tag{8}$$

To derive the lower and upper mean speedup boundaries, the expectations of (8) are inverted, and multiplied by $nE(u_1)$, to obtain the following results:

$$\frac{nE(u_1)}{E\left(u_{(n)} + c_1 + \cdots + c_n\right)} \le \frac{nE(u_1)}{E(t_n)} \le \frac{nE(u_1)}{E(c_1 + \cdots + c_n)}. \tag{9}$$

We can then extend (9) as follows.

$$\frac{nE(u_1)}{E\left(u_{(n)}\right) + nE(c_1)} \le \text{Mean speedup} \le \frac{E(u_1)}{E(c_1)}. \tag{10}$$

Let α be the maximal computation time for a single job and let β be the communication time. The upper and lower mean speedup boundaries are as follows.

$$\frac{nE(u_1)}{E\left(u_{(n)}\right) + nE(c_1)} \le \text{Mean speedup} \le \frac{E(u_1)}{E(c_1)}. \tag{11}$$

Since the distribution of $u_{(n)}$ is F_u, its expectation can be computed by finding the expectation on nonnegative continuous random variable

$$E\left(u_{(n)}\right) = \int_0^\infty \left(1 - F_u^n(y)\right) dy. \tag{12}$$

From (11), it is also clear that a sufficient condition for mean speedup to α/β is

$$\lim_{n\to\infty} \frac{E\left(u_{(n)}\right)}{n} = 0. \tag{13}$$

The sufficient condition for the mean speedup to converge to α/β is given below (the proof is similar to [9]).

PROPOSITION 1. *Let computation time u be a continuous nonnegative random variable with distribution function F_u, and such that $E(u^d) < \infty$ for some $d > 1$. Then, $\lim_{n\to\infty} \mathrm{E}(u_{(n)})/n = 0$; i.e., the mean speedup converges to α/β.*

4. NUMERICAL RESULTS

In this section, we present the numerical experimental results in order to justify the performance of the proposed algorithm. The algorithm presented was implemented in a cluster of distributed network workstations, which were connected via an optical fiber link. We used the parallel virtual machine (PVM) library routines to develop the message-passing environment for distributed

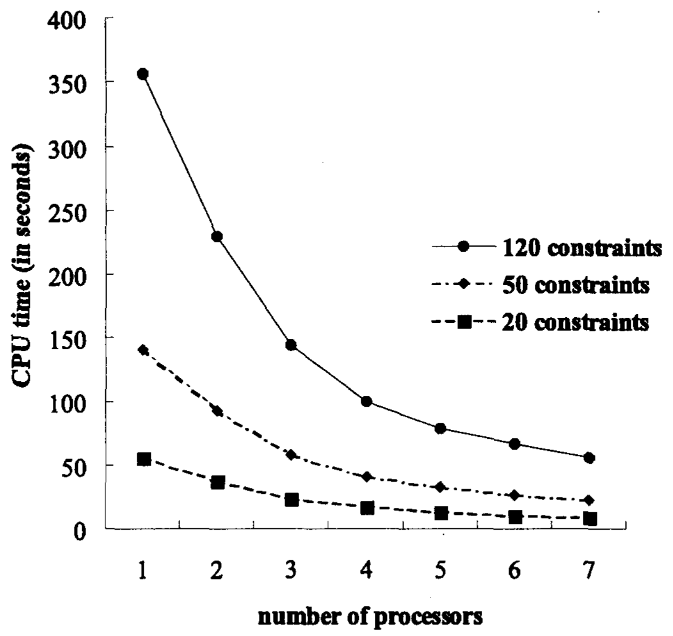

Figure 1. CPU time (in seconds) versus number of processors.

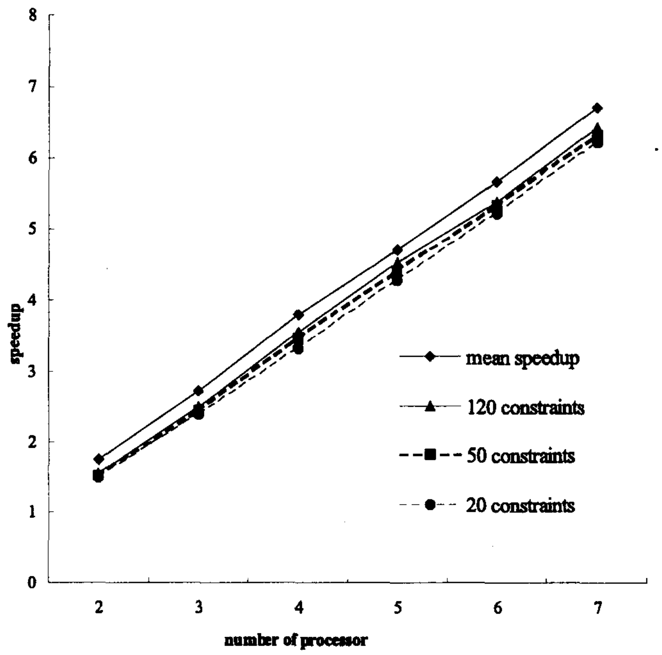

Figure 2. Mean speedup and speedup versus number of processors.

computing. PVM enables a collection of heterogeneous computer systems to be viewed as a single parallel virtual machine and has been widely adopted by researchers [10,11]. The algorithm code was programmed in FORTRAN/77.

The method for numerical experiments in this study was similar to the one proposed by [12], where the number of model constraints are 20, 50, and 120, and the number of model variables is 120. For each problem type, five sets of models, generated using different random-number generator seeds, were solved. The average CPU time (five replications for each instance) required to solve each type of test problem with respect to the different numbers of processors used during experiments is plotted in Figure 1. The speedup earned in the numerical experiments as well as the mean speedup of the proposed algorithm are calculated and plotted in Figure 2.

It is clear that the numerical performance of the proposed algorithm is impressive based on the experimental results demonstrated in Figure 1. When solving a modest sized problem (120 constraints), the CPU time saved in a DCE could be more than six fold. Given that a distributed computing environment is very common in many computer centers and PVM is a shareware, the presented algorithm is useful to many researchers in solving a linear programming problems.

The average of the mean speedups and the empirical speedups obtained in this research are shown in Table 1. The best empirical speedup found in the numerical experiments was 6.34, with respect to the best mean speedup approaches 6.69. In general, the speedups earned in the experiments were close to the mean speedups as expected (Figure 2 and Table 1). It is also interesting to note that near linear speedup was achieved, which means that the proposed algorithm can take full advantage of the distributed computing power as the size of the problem increases.

Table 1. Experimental results of mean speedup and empirical speedup.

Number of processors	Mean speedup	Empirical speedup
2	1.74	1.55
3	2.72	2.42
5	4.70	4.42
6	5.66	5.36
7	6.69	6.34

5. CONCLUSIONS

This study developed a parallel DWDP algorithm on clusters of distributed network workstations and evaluated its performance. It was shown that the mean speedup of the proposed algorithm converges to α/β, where α is the maximal computation time for a single job and β is the communication time. The numerical results are consistent with the analytical analysis. We, therefore, demonstrated that there is a "communications speedup limit" that cannot be exceeded regardless of the number of processors available in the DCE. Since DWDP represents a limiting case for communication delays, we can state that this speedup limit is also an upper boundary for the asymptotic speedup of block-type problems.

On the other hand, the numerical results suggested that the CPU time saved by the proposed parallel algorithm is inspiring. This algorithm could be implemented in a general distributed computing environment and the speedup earned will approach linearity. The parallel DWDP algorithm implementation is therefore useful to many practitioners. As networked computing environments are becoming increasingly more available, greater effort on the development of parallel optimization algorithms will be necessary.

REFERENCES

1. K. Ziarati, F. Soumis, J. Desrosiers and S. Gelinas, Locomotive assignment with heterogeneous consists at CN North America, *European Journal of Operational Research* **97**, 281–292, (1997).
2. G. Desaulniers, J. Desrosiers, Y. Dumas, S. Marc, B. Rioux and M. Solomon, Crew pairing at Air France, *European Journal of Operational Research* **97**, 245–259, (1997).
3. M. Aganagic and S. Mokhtari, Security constrained economic dispatch using nonlinear Dantzig-Wolfe decomposition, *IEEE Transactions on Power Systems* **12**, 105–112, (1997).
4. H. Hafsteinsson, H.R. Levkovitz and G. Mitra, Solving large-scale linear programming problems using an interior point method on a massively parallel SIMD computer, *Parallel Algorithms and Applications* **4**, 301–316, (1994).
5. A. Sanghvi and I. Shavel, Investment planning for hydro-thermal power system expansion: Stochastic programming employing the Dantzig-Wolfe decomposition principle, *IEEE Transactions on Power Systems* **1**, 115–121, (1986).

6. J.W. Chinneck, Computer codes for the analysis of infeasible linear programs, *Journal of the Operational Research Society* **47**, 61–72, (1996).
7. V.S. Sunderam, G.A. Geist, J. Dongarra and R. Manchek, The PVM concurrent computing system: Evolution, experiences, and trends, *Parallel Computing* **20**, 531–545, (1994).
8. J. Lyu, A. Gunasekaran and V. Kachitvichyanukul, Towards a portable and efficient environment for parallel computing, *International Journal of Systems Science* **26** (6), 1333–1341, (1995).
9. P.M. Feldman, R.S. Feldman and D.B. Kim, Predicting speedup for distributed computing on a token ring network, *Journal of Parallel and Distributed Computing* **45**, 53–62, (1997).
10. P. D'Ambra and G. Giunta, Concurrent banded Cholesky factorization on workstation networks using PVM, *Parallel Computing* **21**, 487–494, (1995).
11. V.S. Sunderam, PVM: A framework for parallel distributed computing, *Concurrency, Practice and Experience* **2** (4), 315–339, (1990).
12. D. Pisinger, An expanding-core algorithm for the exact $0-1$ Knapsack problem, *European Journal of Operational Research* **87**, 175–187, (1995).