A PARALLEL PRIMAL-DUAL DECOMPOSITION METHOD FOR MULTI-PART LINEAR PROGRAMS

by

Hyun Jin Park

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Management Sciences

Waterloo, Ontario, Canada, 2001

0-612-65663-2

ABSTRACT

In this thesis, we develop a new parallel primal-dual decomposition algorithm for multi-part linear programming (LP) problems. We first present a parallel decomposition method for two-part models, in which the two master-like subproblems are solved simultaneously in two different processors and generate an upper bound and a lower bound on the original problem at each iteration. These two bounds are monotonically improved and converge to within the prescribed tolerance of the optimal value by exchanging primal and dual information.

Then we extend the basic principles of the two-part algorithm to multi-part models by applying a hierarchical decomposition principle recursively. The original multi-part problem is divided into two aggregated subproblems of lower bound type and upper bound type, and the aggregated subproblems are further divided into two smaller aggregated subproblems of upper bound and lower bound. This bifurcation process continues until there are no subproblems left for further decomposition. The subproblems are solved in different processors simultaneously and work together to reach an optimal point during the iterations by exchanging primal and dual information in the hierarchical way. Convergence and other useful properties of the parallel two-part and multi-part algorithms are proven.

We developed a parallel decomposition solver for problems of two, three or four parts, called WATPAR (WATerloo PARalell), through the use of GAMS, the Regex Library, CPLEX 6.0 and PVM (Parallel Virtual Machine) 3.11 on one IBM RS/6000 workstation and a cluster of four PCs running the Solaris operating system. Several multi-part LP models are tested using

WATPAR, and in each of the tests, the new parallel decomposition algorithm converged to an optimal value of the original problem in a finite number of iterations. For some large problems, the new method showed some speedups.

ACKNOWLEDGEMENTS

I would like to thank and glorify God for allowing me to finish this long journey.

I would like to deeply appreciate my wonderful supervisor, Professor J. David Fuller, for his academic guidance as well as his mentoring and financial support provided throughout this study. My sincere gratitude should go to Professors R. G. Vickson, J. K. Ho, H. Wolkowicz and R. P. Sundarraj for their invaluable feedback and careful review. I also thank Professor E. J. Fragniere and his colleagues in Switzerland for allowing me to use their facilities. Many thanks go to Richard Kyuchul Cho for his friendship for the last couple of years.

I am greatly indebted to my parents for their love, encouragement and prayer. I am also indebted to grandmother-in-law and my parents-in-law for their understanding and support. I should express my gratitude to my sister, brother-in-laws, sister-in-law, uncles and aunts for their help and care.

Finally, the greatest debt of gratitude is owed to my wife Soyoung, whose constant love, patience and support not only inspire me but also allow me to indulge in academic and professional endeavors of this magnitude. Needless to say, without her, I could not have completed this thesis.

To Soyoung and Katherine Eugene Park

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1 Introduction

1.1 Brief History of Decomposition

Over the past several decades, linear programming (LP) has been a useful planning and scheduling tool for economic and management applications. The real world LP problems are very large and usually have some special structures, which could break into several distinct LPs except for a few linking constraints. These linking constraints represent relationships among different periods, regions, stochastic scenarios, etc. The most commonly discussed structures are primal (or dual) block-angular, staircase and block-triangular as shown in Figure 1.1.

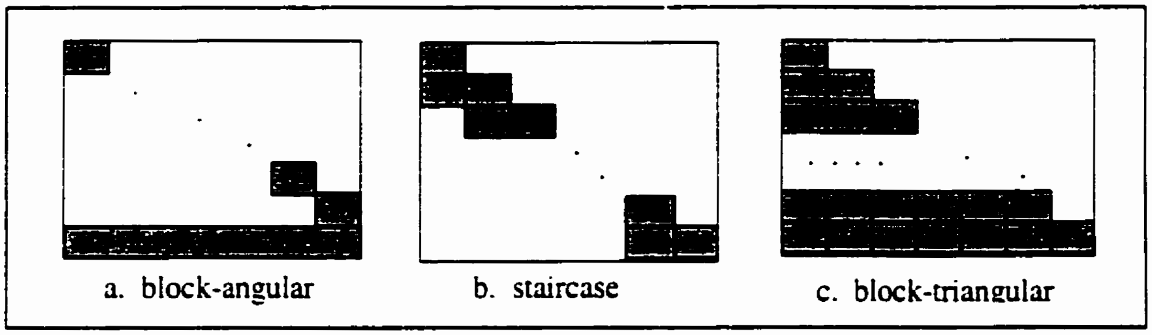

Figure 1.1 Structures of multi-stage problem, from Lan and Fuller [1995]

Based on the ideas of utilizing the special structures, many decomposition algorithms, which decompose a very large problem into several subproblems and solve them iteratively through the exchanges of information, have been proposed since the early 1960's,

There have been several motivations to study decomposition: the prospect of reducing computational time; a method to solve a huge model within a computer's memory limits that do not permit a straightforward solution of the whole model (Fragniere et al. [1998]); and to ease management of a huge model (Murphy and Mudrageda [1999]), by breaking it into its natural constituent parts such as regions, divisions, etc. Following the famous decomposition methods of Dantzig-Wolfe [1960] and Benders' [1962], cross decomposition by Van Roy [1983] and Holmberg [1992], and mean value cross decomposition (Aardal and Ari [1990] and Holmberg [1992]) have been developed to solve large-scale LPs with primal (or dual) block-angular or staircase structures. Another important development was reported by Lan and Fuller [1995a, b] based on Lan [1993] for multi-stage block-triangular structures, in which all subproblems perform like a master problem.

Recently, technological advances in massively parallel processors (MPPs) and distributed computing systems have encouraged development of decomposition algorithms to solve subproblems simultaneously using several different processors. Several parallel computational tests and algorithms for LPs are presented by Ho et al. [1988], and Rosen and Maier [1990] for models with block-angular structure and by Dantzig and Glynn [1990], and Birge et al.[1996] for decomposition of stochastic LPs. Entriken [1996] reported experimental results of solving staircase LPs by using parallel Benders decomposition. In all of these previous studies, subproblems were solved in parallel, but the algorithms had a serial element in the alternation between the master problem and the subproblems. In contrast, our decomposition principle can be used to define an entirely parallel algorithm.

1.2 Motivation and Objectives of the Research

This research is motivated by Lan [1993], in which he suggested an idea of computing the subproblems simultaneously, in a block-triangular problem, by sending proposals and cuts to immediate neighbors in his further research, but after some reflection, it is apparent that this scheme would probably take too many iterations for the last subproblem to hear information of the first subproblem. Thus, this research started from the idea that all the subproblems could be solved at the same time using different processors in order to have benefits of parallelism, by broadcasting all information to all other subproblems for block-triangular structure. It worked well for the two-stage case, however, it failed to extend the idea of the nested decomposition to the multi-stage case from the two-stage algorithm due to complexity of nested algorithms, inconsistency of weighting schemes and infeasiblilty of the solutions. The nested algorithm didn't fit well in parallel decomposition because the nested structure and the information transfers lead to an inherently serial algorithm: it has to solve one subproblem at a time with further weighted primal information of all the previous subproblems on the forward pass and with additional weights on dual information of all the following subproblems on the backward pass.

Realizing that the linking variables and constraints cause those serious problems, we next attempted another broadcasting scheme by including all linking constraints in each subproblem and restricting primal and dual linking variables to convex combinations of known solutions from other subproblems of earlier iterations, since the number of the linking variables and constraints are relatively small. This idea makes it possible to apply the parallel decomposition method to the multi-part structure problems shown in Figure 1.2.

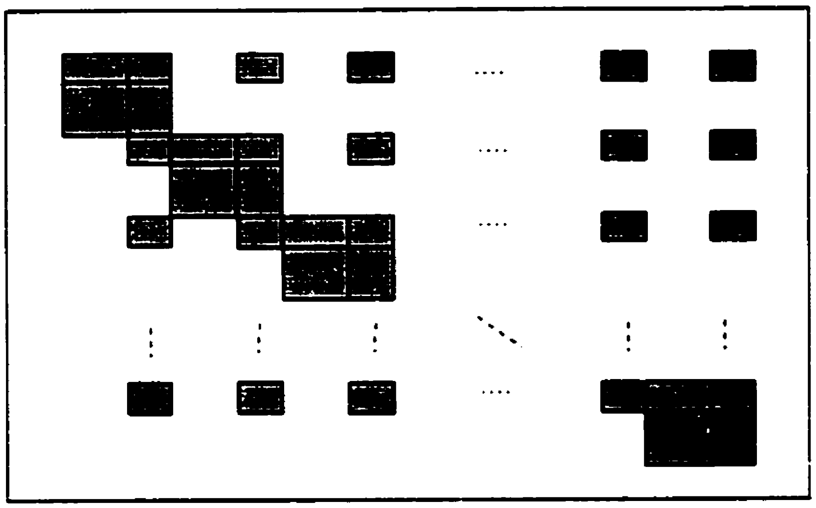

Figure 1.2 General structure of the multi-part models

The parallel primal-dual decomposition algorithm for two part problems converged consistently to the optimal solution in a small number of iterations, however, in some tests of multi-part problems the algorithm (we call this a heuristic parallel algorithm - see section 4.5) did not converge exactly to an optimal solution but repeated the same feasible solution thereafter. After we found some errors in the convergence proof of this multi-part algorithm, we finally reached the ideas of applying the two-part decomposition principle recursively in a hierarchical way since the two-part algorithm seemed to be working fine and no flaws could be found in the convergence proof.

Hence, the objectives of this Ph.D. research are as follows.

1. Develop a new parallel primal-dual decomposition method for two-part and multi-part linear programming models which have the general multi-part matrix structure shown above.

2. Prove and demonstrate the convergence of the parallel two-part and multi-part algorithms as well as other properties of the algorithm.

3. Design and implement a parallel primal-dual decomposition solver for the parallel algorithm on several computers and investigate the computational efficiency of the algorithm.

1.3 Overview of the Thesis

This thesis is organized as follows. The next chapter presents a literature review on both serial decomposition methods and parallel decomposition methods with basic concepts of parallel computing.

Chapter 3 proposes the new decomposition method for two-part linear programs, which divides the original problem into two master-like subproblems, a lower bound subproblem and an upper bound subproblem, and coordinates them to converge to an optimal solution by exchanging primal and dual information. Proofs of the convergence as well as other useful properties of the algorithm are given.

Chapter 4 discusses the development of the parallel algorithm for multi-part models, in which the two-part decomposition principle is recursively applied in a hierarchical manner. It also discusses the heuristic parallel algorithm, a simple extension of the two-part decomposition principle to multi-part problems without the hierarchical decomposition principle.

Chapter 5 presents the design and implementation procedures of the parallel decomposition solver and shows preliminary computational results. In each of the tests, the parallel algorithm converged to within a small tolerance of the optimal solution in a reasonable amount of time. The algorithm was faster than the simplex or interior point methods in a single machine in some very large scale problems. It could solve a huge problem, which could not be solved in one machine due to memory limits. The final chapter indicates the expected contributions and future research directions.

Appendix A presents the test models in GAMS files and Appendices B and C show the core parts of the parallel decomposition solver codes written in the C language.

Chapter 2 Literature Review

This chapter provides an overview of topics found in the literature which are related to this thesis. In the next section, a brief review of decomposition methods for LPs is presented. The second section discusses the basic parallel computing concepts and methods. In section 3, the parallel computational tests and algorithms applied to decomposition methods are reviewed along with the description of their characteristics. The summary is given in the final section.

2.1 An Overview of the Decomposition Methods

Many decomposition schemes, such as Dantzig-Wolfe decomposition, Benders decomposition and cross decomposition by Van Roy, have been developed to solve large-scale mathematical programming problems. A new decomposition algorithm, called primal-dual decomposition, was suggested by Lan and Fuller [1995a]. In this section, these algorithms are briefly reviewed with extensions to the nested algorithm for multi-stages.

Dantzig-Wolfe Decomposition Method

In the decomposition method of Dantzig-Wolfe, the master problem determines an optimal combination of the proposals on hand, submitted by subproblems, by assigning values to the weights. The optimal dual variables, known as prices, are used to adjust the objective function in the subproblems which in turn may produce new proposals to improve the global objective function in the master problem. This mechanism is often called the dual decomposition method.

Dantzig [1963] applied this method in a hierarchical way to a four stage staircase model by dividing the original problem into a first level master problem of stage two and four and two first level subproblems of stage one and stage three. The first level master problem is further divided into a second level master problem of stage four and a second level subproblem of stage two. The first level master problem sends price information to the first level subproblems when the second level subproblem has no more proposal to the second level master problem.

The Nested Dual Decomposition Method for Multi-Stage Models

The Dantzig-Wolfe decomposition method is motivated by the block-angular structure in which the choice of a master problem and subproblems is relatively easy. But, under the system of staircase or block-triangular with multi-stages, the appropriate choice of master and subproblems is not as simple as that of the block-angular structure. This problem can be solved by employing nested decomposition algorithm.

Dual nested decomposition for staircase structure was first introduced by Dantzig [1963], where the Dantzig-Wolfe decomposition principle was applied to the dynamic model structure in a recursive fashion. The algorithm of dual nested decomposition divides the original problem into N period subproblems and solves only one subproblem at a time by applying the Dantzig-Wolfe decomposition principle recursively. This principle makes the period t problem act both as a restricted master problem with respect to periods 1 to t-1 and as a subproblem with respect to t+1 to N.

The price mechanism is used in dual nested decomposition for the coordination between periods. The master problem receives proposals from subproblems and sends a price information to subproblems.

Benders Decomposition Method

In the Benders decomposition method, the master problem allocates shared resources to the subproblems. The subproblems, then, react by utilizing the resources and report to the master problem the prices which reflect how well the subproblems use the shared resources. Then, the master problem adjusts and reallocates the resources according to the price information.

Benders decomposition is efficient for the complicated problem which can be partitioned into two parts: one major part containing linear variables and another part containing the complicating variables such as integer or nonlinear variables. Benders decomposition can be called the primal decomposition method.

The Nested Primal Decomposition Method for Multi-Stage Models

Dantzig [1980] applied a nested decomposition algorithm to the dual of staircase Linear Programming models, which can be considered as a nested decomposition of Benders method. In nested primal decomposition, the resource mechanism is used for the coordination between periods. The master problem, which is the preceding subproblems, receives cuts from the following subproblems and sends a resource vector to the following subproblems.

Cross Decomposition Method by Van Roy

Cross decomposition was first introduced by Van Roy [1983] for mixed integer programs. The cross decomposition method can be described as simultaneously using Benders decomposition (primal decomposition) and Dantzig-Wolfe decomposition (dual decomposition) in an iterative manner. Cross decomposition divides the original problem into two decomposition systems: the primal master problem (PM) and the primal subproblem (PS) in the primal decomposition system; the dual master problem (DM) and the dual subproblem (DS) in the dual decomposition system. The main idea behind this method is to make use of the very close relationship between the PM (DM), and the DS (PS). This relationship is exploited in such a way that only the easy-to-solve subproblems are used as long as they produce a converging sequence of primal and dual solutions. This method may increase computational efficiency. But convergence cannot be guaranteed by the use of subproblems only, and therefore a primal or dual master problem, with all dual or primal solutions generated so far, has to be solved from time to time as the algorithm proceeds. "The need for convergence tests and for involving a master problem often prevents a possible reduction in computer memory requirements since it causes complete sets of primal and dual solutions to be stored even if a master problem is not solved at each iteration" [Aardal and Ari, 1990].

Primal-Dual Decomposition Method by Lan and Fuller

Another new method was developed by Lan and Fuller [1995a] based on Lan [1993] for two-stage models. In this algorithm, the decomposition structure is balanced and convergence is generally rapid. "The algorithm divides the original problem into a pair of restricted primal and dual subproblems, each of which has summary information on all previous iterations of the other" [Lan and Fuller, 1995a]. The solutions of the two subproblems are monotonically improved by coordinating the information of the subproblems and converge to a prescribed tolerance of the optimal value as the iterations go on.

In this algorithm, both subproblems are in equivalent positions and play the role of both the master problem and the subproblem of the traditional method. At each iteration, the stage one subproblem gives the upper bound of the original problem by restricting the dual variables to convex combinations of known dual solutions and the stage two subproblem gives a lower bound by restricting primal variables to convex combinations of known primal solutions. The algorithm iterates until the upper bound and lower bound reach an equilibrium point.

Their tests for eleven problems with the two stage, block-triangular structures showed that the new method is usually faster and more efficient than the traditional methods.

Park [1996] extended this idea to convex, nonlinear programming models. He proved and tested that the new algorithm for the two stage case converged to an optimal solution in a finite number of iterations.

Primal-Dual Nested Decomposition Method for Multi-Stage Models

Lan and Fuller [1995b] also presented a nested primal-dual decomposition algorithm for the multi-stage LP problems with block-triangular matrix structure. In this algorithm, the original multi-stage problem is divided into a sequence of a pair of subproblems for each stage. These subproblems are coordinated by passing the proposals forward and cuts backward; the previous subproblems pass the proposals to the following subproblems forward in stage numbers (which designates time period in many models) time and the following subproblems provide cuts to the previous subproblems backward. This information flow between subproblems is shown in Figure 2.1, from Lan and Fuller [1995b]. The algorithm can be perceived as the combination of the primal and the dual nested decomposition algorithms. As in the two stage case, the first subproblem provides the upper bound to the original problem and the last subproblem provides the lower bound to the original problem. These two bounds are monotonically improved and converge to a prescribed tolerance as the iterations go on.

Park [1996] applied this nesting scheme to multi-stage convex nonlinear programs and showed that the nested algorithm converged in a finite number of iterations, but without a convergence proof.

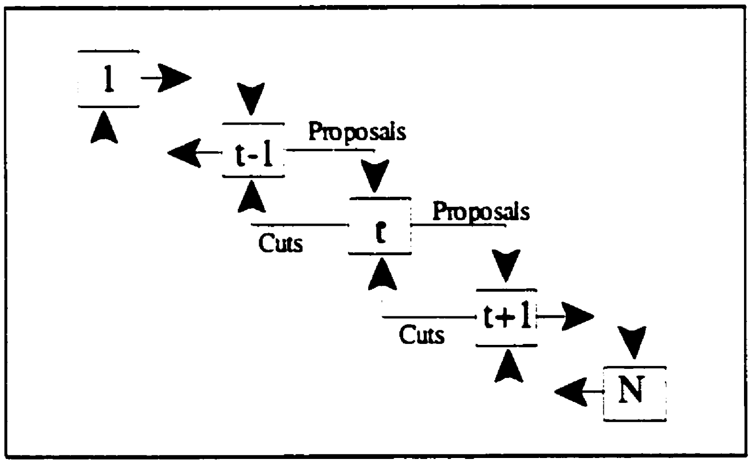

Figure 2.1 Information flow of nested primal-dual decomposition

2.2 An Overview of Parallel Computing

Parallel computing is the use of parallel computers utilizing more than one Central Processing Unit (CPU) at the same time to solve a single large problem faster and more efficiently [Baker and Smith, 1996]. In this section, several parallel computer systems are reviewed briefly and two performance measurements of parallel computing are discussed.

SIMD vs MIMD

Parallel computers can be classified into two groups by Flynn [1966]: SIMD (Single Instruction, Multiple Data) and MIMD (Multiple Instruction, Multiple Data).

A SIMD machine consists of a number of identical processors doing the same things to different data at any given point of time. Typical SIMD machines have large numbers of relatively simple and affordable processors resulting in fine-grained parallelism, which distributes the data as widely as possible with each processor performing the simplest operations.

In MIMD machines, the most widely employed parallel machine architecture, each processor executes a possibly different program on different data under the control of different instruction asynchronously. The MIMD machines generally have fewer but more powerful processors than SIMD machines.

Shared Memory vs Message Passing

Another architectural classification is whether the parallel computer is a shared memory machine or a message passing machine according to how the processors communicate with each other.

Shared memory computers have global memory that can be directly accessed by all processors. Shared memory computers are not very scalable, particularly when the entire global memory is equally accessible to all of large number of processors. They also impose an inherent concern of synchronization, i.e. how different processors can read and write the data in the same location of memory simultaneously without conflict. Currently, most shared memory computers have a local memory distributed to each processor together with a global memory accessible by all processors. Shared memory MIMD computers are often called multiprocessor computers.

Message passing MIMD systems are often called multicomputers. In message passing computers, each processor has its own local memory, but they do not have shared memory, and processors communicate only by sending and receiving messages over a communication network. Each processor in message passing machines executes its own instruction streams and uses its own local data, both stored in its local memory. Then, necessary data can be exchanged by sending messages to each other over a network. Since the network determines the speed and reliability of interprocessor communication and the spatial distribution of the processors, message passing systems can be further characterized into closely coupled systems and loosely coupled systems (or distributed systems) by their communications networks.

Closely coupled systems, such as the architectures with mesh or hypercube networks, have fast and reliable point-to-point links between processors which are physically close to one another. Loosely coupled systems or distributed systems, such as workstations connected by local-area network (LAN) or wide-area network (WAN), have relatively slow and unreliable communication between processors that are physically dispersed. They have coarse-grained parallelism, which distributes the data as large as possible with each processor spending most of its time doing computations and communicating infrequently because of the expensive communication cost.

The local-area network (LAN) systems consist of several workstations connected by a network. The local area network allows communication between any two processors physically apart. In many LAN's, communication is not very reliable, for example, a message may be damaged, arrive out of order, or not arrive at its destination at all, so communication requires a lot of programming effort. Therefore, some software protocols for message passing systems such as Parallel Virtual Machine (PVM) and Message Passing Interface (MPI) are used to implement reliable communication and simpler programming.

PVM, which is more suitable for LAN based systems than MPI, provides an interface which links separate hosts, possibly of varying types, to create a single logical host, so-called *virtual machine*. PVM allows a programmer to take virtually any network of UNIX-based computers and treat them as a single parallel computer. "PVM has a somewhat limited message-passing subset, providing basic send and receive operations and some simple collective communications, but not providing the rich set of features that more formal message passing systems, like the MPI, provide. PVM does, however, provide a complete environment for parallel computing, including the PVM console, and includes features for dynamically adding and deleting machines from your own virtual-machine configuration." (from Baker and Smith, 1996) A LAN limits the physical distance between processors to on the order of a few kilometers. To interconnect processors that are farther apart, a wide-area network (WAN) can be used.

Performance Measurement in Parallel Computing

Generally, the performance of parallel computation can be measured in two ways, speedup and efficiency, even though they depend on hardware, software and algorithms of solving a problem.

The speedup S achieved by a parallel system is defined as the gain in computation speed achieved by using N processors with respect to a single processor:

$$S = T_1 / T_N$$

where T_1 is defined as the time taken to solve a particular problem by the fastest serial algorithm on one processor, and T_N is the parallel runtime taken to solve the same problem by a parallel algorithm or computation on N different processors.

The efficiency E denotes the effective utilization of computing resources. It is the ratio of the speedup to the number of processors used;

$$E = S / N.$$

2.3 Parallel Decomposition

Recent advances in the developments of massively parallel processors (MPPs) and distributed computing systems made it possible to coordinate many small tasks to solve one large problem. This modern technology encourages development of decomposition algorithms to solve subproblems simultaneously using several different processors for algorithmic efficiency.

Several parallel computational tests and algorithms for LPs with block-angular and staircase structures have been presented since block-angular and staircase structures lend themselves naturally to parallel computing. Ho, Lee, and Sundarraj [1988] implemented the Dantzig-Wolfe decomposition algorithm for block-angular linear programs using parallel processing of the subproblems in the CRYSTAL multicomputer at the University of Wisconsin at Madison and showed that significant speedup could be obtained using parallel decomposition. Rosen and Maier [1990] presented another approach to parallel solution using the dual of block-angular linear programs by fixing the dual coupling variables. Gnanendran and Ho [1993] investigated strategies for improving efficiency in distributed Dantzig-Wolfe decomposition by better balancing the load between master and subproblem processors because the parallel efficiency of the distributed approach is critically dependent on the duration of the inherently serial master phase relative to that of the bottleneck subproblem.

Entriken [1996] presented the experimental results of using parallel Benders decomposition to solve staircase multistage problems on a shared memory multiprocessor computer which has 6 processors. He showed that parallel decomposition can solve a large problem with staircase structure faster than the simplex method even when serial decomposition is slower than the simplex method.

Another use of parallel processors for decomposition algorithms is for the area of stochastic programming problems since the equivalent deterministic problem of a stochastic model is typically very large. Dantzig and Glynn [1990] suggested the use of parallel processors to calculate the subproblems of Benders decomposition for stochastic models. Ruszczynski [1993] suggested parallelizing a variant of the nested decomposition algorithm by queuing subproblems for idle processors, and Birge, Donohue, Holmes and Svintsitski [1996] tested and compared the parallel implementation of a nested decomposition algorithm for multistage stochastic linear programs over a serial implementation using PVM on a network of RS/6000 model 320H workstations connected by a local ethernet. Their computational experience on a large test set of practical problems with up to 1.5 million constraints and almost 5 million variables showed that the parallel implementations worked very well but they require careful attention to processor load balancing. Nielsen and Zenios [1997] implemented a version of Benders decomposition algorithm for two-stage stochastic LPs on the parallel CM-5 computer using the interior point method to solve scenario subproblems in parallel. Another computational assessment for stochastic LP decomposition using interior point method was reported by Vladimirou [1998] on an IBM SP2 multiprocessor system.

Fragniere, Gondzio, Sarkissian and Vial [2000] proposed a new tool, called Structure Exploiting Tool [SET], for linking algebraic modelling languages and structure exploiting tools, such as decomposition methods, using the GAMS I/O Library [1996] and the Regex Library [1992]. Fragniere, Gondzio and Vial [1998b] reported a successful parallel implementation of Benders' decomposition on 10 Pentium computers under the Linux operating system for a stochastic financial planning model with one million scenarios, using SET.

2.4 Summary

Unlike Primal-Dual decomposition, most decomposition methods possess an unbalanced structure providing different amounts of information to the master and subproblems and allow only the master problem to converge monotonely. Lan and Fuller [1995] suggest that these two factors may be the main reason for the poor performance and slow convergence of the traditional decomposition methods.

Since no one has developed a parallel algorithm for primal-dual decomposition of LPs, in this thesis, we develop and study such an algorithm.

Chapter 3 Parallel Decomposition of the Two-Part Problem

In this chapter, we develop a parallel decomposition algorithm for two part linear programming problems, i.e. an LP problem which would break into two distinct LPs, except for a few linking constraints that connect the parts. The new method divides the original problem into two subproblems (a lower bound subproblem and an upper bound subproblem), instead of the traditional master and subproblem. The subproblems are derived in a way that is similar to the two subproblems in Lan's [1993] scheme, but here we derive them for a two-part structure for parallel computations, whereas Lan's was for the two-stage structure, for a serial algorithm. Since the subproblems in each part give an upper bound and a lower bound to the original problem, the algorithm arbitrarily selects an upper bound subproblem from one part and a lower bound subproblem from the other part and solves them simultaneously in two different processors. By exchanging primal and dual information at each iteration, these two bounds are monotonically improved and converge to within the prescribed tolerance of the optimal value.

3.1 Model Structure and Assumption

The variables and constraints are grouped into two parts, indicated by the subscript t=1, 2. The objective function is the sum of linear functions and each part's objective function depends only on that part's variables. Each part contains three sets of variables; nonlinking variables, x_t, linking variables, y_t, and artificial variables, v_t. Also, each part consists of three sets of constraints: nonlinking constraints, linking constraints and upper bound constraints on linking variables. The linking constraints in each part may contain linking variables of either part to represent the influence of that part on the other. The general primal and dual structures of the two-part linear program are as follows (superscript T denotes transpose).

$$
\begin{array}{lllcccccccl}
\text{P:} & \max\limits_{x_t, y_t, v_t, t=1,2} & z = & c_1x_1 + d_1y_1 - M_1v_1 + c_2x_2 + d_2y_2 - M_2v_2 & & & & & & & \\
(\pi_1^T) & \text{s.t.} & & A_1x_1 + D_1y_1 & & & & & & \leq & b_1 \\
(\omega_1^T) & & & B_1x_1 + L_{11}y_1 - v_1 & + & L_{12}y_2 & & & & \leq & f_1 \\
(\rho_1^T) & & & y_1 & & & & & & \leq & u_1 \\
(\pi_2^T) & & & & & A_2x_2 + D_2y_2 & & & & \leq & b_2 \\
(\omega_2^T) & & & L_{21}y_1 & + & B_2x_2 + L_{22}y_2 - v_2 & & & & \leq & f_2 \\
(\rho_2^T) & & & & & y_2 & & & & \leq & u_2 \\
 & & & x_1, y_1, v_1, x_2, y_2, v_2 \geq 0 & & & & & & &
\end{array}
$$

$$
\begin{array}{lllcccccccl}
\text{D:} & \min\limits_{\pi_t, \omega_t, \rho_t, t=1,2} & z = & \pi_1b_1 + \omega_1f_1 + \rho_1u_1 + \pi_2b_2 + \omega_2f_2 + \rho_2u_2 & & & & & & & \\
(x_1^T) & \text{s.t.} & & \pi_1A_1 + \omega_1B_1 & & & & & & \geq & c_1 \\
(y_1^T) & & & \pi_1D_1 + \omega_1L_{11} + \rho_1 & + & \omega_2L_{21} & & & & \geq & d_1 \\
(v_1^T) & & & \omega_1 & & & & & & \leq & M_1 \\
(x_2^T) & & & & & \pi_2A_2 + \omega_2B_2 & & & & \geq & c_2 \\
(y_2^T) & & & \omega_1L_{12} & + & \pi_2D_2 + \omega_2L_{22} + \rho_2 & & & & \geq & d_2 \\
(v_2^T) & & & & & \omega_2 & & & & \leq & M_2 \\
 & & & \pi_1, \omega_1, \rho_1, \pi_2, \omega_2, \rho_2 \geq 0 & & & & & & &
\end{array}
$$

where $x_t \in \mathbf{R}^{n_t}$, $y_t \in \mathbf{R}^{r_t}$, $v_t \in \mathbf{R}^{q_t}$ are the vectors of variables for part t. The dual variable vectors for the constraints of part t of P are denoted by row vectors π_t, ω_t and ρ_t where $\pi_t \in \mathbf{R}^{m_t}$, $\omega_t \in \mathbf{R}^{q_t}$ and $\rho_t \in \mathbf{R}^{r_t}$. L_{ts} is a $q_t \times r_s$ matrix. A_t is a $m_t \times n_t$ matrix, B_t is a $q_t \times n_t$ matrix, D_t is a $m_t \times r_t$ matrix, and c_t, d_t, $M_t > 0$, b_t, f_t, and $u_t > 0$ are vectors of suitable dimensions for t=1, 2. Each part has a set of primal (or dual) nonlinking constraints containing A_t, and a set of upper bound constraints. The two parts are linked through the primal (or dual) linking constraints containing the matrix L_{ts}. The primal (or dual) linking variables are y_t (or ω_t) which appear in linking constraints of both parts. Each linking primal constraint has a corresponding dual linking variable and similarly, there is a linking dual constraint corresponding to each linking primal variable. Each primal artificial variable induces a corresponding upper bound constraint to a dual linking variable. Similarly, there is a dual artificial variable corresponding to each upper bound constraint for a primal linking variable. These artificial variables can be adjusted to satisfy the linking constraints, thus allowing each part to act independently. However, because of the high cost of the artificial variables, it may be desirable, in the optimal solution, for the parts to cooperate. Thus, the artificial variables (primal and dual) are an important aspect of the notion of "parts" of a linear program. The primal artificial variables may in fact represent a real aspect of the situation such as unfillable demand or emergency purchase in inventory constraints; whether real or artificial, these variables ensure the feasibility of linking constraints.

The two-stage (or block-triangular) structure is a special case for which part one has no linking constraints and y_2 does not exist, i.e. B_1, L_{11}, L_{12}, D_2 and L_{22} are all zero. This structure arises, e.g., in a two period model, in which the linking constraints represent the influence of decisions in the first period on those in the second.

An assumption is made in order to simplify the algorithm, and to guarantee convergence.

Assumption: The set of nonlinking constraints in each part, together with the upper bound constraints and nonnegativity constraints, define bounded feasible regions for the x_t, y_t vectors.

The assumption ensures the boundedness of the solutions to the subproblems defined below, which simplifies the algorithm. The assumption is also used in the proof of convergence of the algorithm. Note that the optimal values of the artificial variables are also bounded because the costly nature of the artificial variables keep them as close as possible to their lower bounds of zero during the whole optimization process. (This is shown formally in the proof of Theorem 3.8 in section 3.5).

3.2 Subproblem Structures

The new decomposition method relies on the definition of a subproblem for each part; the subproblem for one part gives an upper bound on the optimal value of the original problem and the subproblem for the other part provides a lower bound for the original problem. Because of the symmetry of the parts in the general problem statement, the choice of subproblem type for a part appears to be arbitrary, but for a given instance of the two-part LP, the modeller may have reasons to make a particular choice. In what follows, the subproblem for part one is of the lower bound type, and part two provides an upper bound.

The lower bound subproblem from part one, denoted by $\underline{\mathrm{SP}}_1^k$ at iteration k, can be constructed by restricting the primal variable vectors x_2 and y_2 to convex combinations of known solutions of x_2 and y_2, obtained from the second subproblem at earlier iterations, thus allowing us to drop out the sets of nonlinking constraints and upper bound constraints of part two from the original primal problem. The subproblem $\underline{\mathrm{SP}}_1^k$ keeps both parts' primal linking constraints. The upper bound subproblem in dual form from part two, denoted by $\overline{\mathrm{SD}}_2^k$, can be constructed in the same way by restricting the dual variable vectors π_1 and ω_1 to convex combinations of known values of π_1 and ω_1 from the first subproblem at earlier iterations, thus allowing us to drop out the sets of dual nonlinking constraints and dual upper bound constraints of part one from the original dual problem, but keeping both parts' dual linking constraints.

The primal and dual forms of the lower bound subproblem at iteration k are presented below first.

$$\begin{array}{llllll}
\underline{\mathrm{SP}_1^k}: & \max\limits_{x_1,y_1,v_1,\lambda^{k-1},v_2} & z_1^k = c_1x_1 + d_1y_1 - M_1v_1 + (c_2X_2^{k-1} + d_2Y_2^{k-1})\lambda^{k-1} - M_2v_2 & & & \\
(\pi_1^T) & s.t. & A_1x_1 + D_1y_1 & & & \le b_1 \\
(\omega_1^T) & & B_1x_1 + L_{11}y_1 - v_1 & + & L_{12}Y_2^{k-1}\lambda^{k-1} & \le f_1 \\
(\rho_1^T) & & y_1 & & & \le u_1 \\
(\omega_2^T) & & L_{21}y_1 & + & (B_2X_2^{k-1} + L_{22}Y_2^{k-1})\lambda^{k-1} - v_2 & \le f_2 \\
(\phi_1) & & & & e^{k-1}\lambda^{k-1} & = 1 \\
 & & & x_1, y_1, v_1, v_2, \lambda^{k-1} \ge 0 & &
\end{array}$$

$$\begin{array}{llllll}
\underline{\mathrm{SD}_1^k}: & \min\limits_{\pi_1,\omega_1,\pi_1,\omega_2,\phi_1} & z_1^k = \pi_1b_1 + \omega_1f_1 + \rho_1u_1 + \omega_2f_2 + \phi_1 & & & \\
(x_1^T) & s.t. & \pi_1A_1 + \omega_1B_1 & & & \ge c_1 \\
(y_1^T) & & \pi_1D_1 + \omega_1L_{11} + \rho_1 & + & \omega_2L_{21} & \ge d_1 \\
(v_1^T) & & \omega_1 & & & \le M_1 \\
(\lambda^{k-1^T}) & & \omega_1L_{12}Y_2^{k-1} + & & \omega_2(B_2X_2^{k-1} + L_{22}Y_2^{k-1}) + \phi_1e^{k-1} & \ge c_2X_2^{k-1} + d_2Y_2^{k-1} \\
(v_2^T) & & & & \omega_2 & \le M_2 \\
 & & & \pi_1, \omega_1, \rho_1, \omega_2 \ge 0 & &
\end{array}$$

where X_2^{k-1} is a $n_2 \times (k\text{-}1)$ matrix and Y_2^{k-1} is a $r_2 \times (k\text{-}1)$ matrix, i.e. $X_2^{k-1} = (x_2^1, x_2^2, \dots, x_2^{k-1})$ and $Y_2^{k-1} = (y_2^1, y_2^2, \dots, y_2^{k-1})$, come from the ($k$-1) previous primal solutions of SP$_2$; λ^{k-1} is a (k-1)-dimensional column vector variable, i.e. $\lambda^{k-1} = (\lambda_1^{k-1}, \lambda_2^{k-1}, \dots, \lambda_{k-1}^{k-1})^T$; ϕ_1 is an unrestricted scalar variable; and e^{k-1} is the (k-1)-dimensional row vector with all components equal to 1. At the first iteration, k=1, there is no information from part two, so there is no λ variable, and the optimal value is not in general a lower bound (unless x_2= 0, y_2= 0 is feasible in part two).

Because SP_1^k is a restriction of P and the restriction can be loosened at each iteration by the inclusion of another x_2 in X_2^{k-1} and y_2 in Y_2^{k-1}, it follows that the optimal values of the subproblems form a nondecreasing sequence of numbers, all less than or equal to the optimal value of P, z^*: $z_1^2 \leq \ldots \leq z_1^{k-1} \leq z_1^k \leq z^*$.

The dual and primal forms of the upper bounding subproblem are as follows.

$$\begin{array}{llllll}
\overline{\mathrm{SD}_2^k}: & \min\limits_{\mu^{k-1}, \rho_1, \pi_2, \omega_2, \rho_2} & \multicolumn{4}{l}{\bar{z}_2^k = \mu^{k-1}(\Pi_1^{k-1} b_1 + \Omega_1^{k-1} f_1) + \rho_1 u_1 + \pi_2 b_2 + \omega_2 f_2 + \rho_2 u_2} \\
(y_1^T) & s.t. & \mu^{k-1}(\Pi_1^{k-1} D_1 + \Omega_1^{k-1} L_{11}) + \rho_1 & + & \omega_2 L_{21} & \geq d_1 \\
(x_2^T) & & & & \pi_2 A_2 + \omega_2 B_2 & \geq c_2 \\
(y_2^T) & & \mu^{k-1} \Omega_1^{k-1} L_{12} & + & \pi_2 D_2 + \omega_2 L_{22} + \rho_2 & \geq d_2 \\
(v_2^T) & & & & \omega_2 & \leq M_2 \\
(\theta_2) & & \mu^{k-1} e^{k-1^T} = 1 & & & \\
& & & \multicolumn{3}{l}{\mu^{k-1}, \rho_1, \pi_2, \omega_2, \rho_2 \geq 0}
\end{array}$$

$$\begin{array}{llll}
\overline{\mathrm{SP}_2^k}: & \max\limits_{y_1, x_2, y_2, v_2, \theta_2} & \bar{z}_2^k = d_1 y_1 + c_2 x_2 + d_2 y_2 - M_2 v_2 + \theta_2 & \\
(\mu^{k-1^T}) & s.t. & (\Pi_1^{k-1} D_1 + \Omega_1^{k-1} L_{11}) y_1 + \Omega_1^{k-1} L_{12} y_2 + e^{k-1^T} \theta_2 & \leq \Pi_1^{k-1} b_1 + \Omega_1^{k-1} f_1 \\
(\rho_1^T) & & y_1 & \leq u_1 \\
(\pi_2^T) & & A_2 x_2 + D_2 y_2 & \leq b_2 \\
(\omega_2^T) & & L_{21} y_1 + B_2 x_2 + L_{22} y_2 - v_2 & \leq f_2 \\
(\rho_2^T) & & y_2 & \leq u_2 \\
& & y_1, x_2, y_2, v_2 \geq 0 &
\end{array}$$

where Π_1^{k-1} and Ω_1^{k-1} are a (k-1)$\times m_1$ matrix and a (k-1)$\times q_1$ matrix, i.e. $\Pi_1^{k-1} = (\pi_1^{1T}, \pi_1^{2T}, \ldots,$

$\pi_1^{k-1T})^T$ and $\Omega_1^{k-1} = (\omega_1^{1T}, \omega_1^{2T}, \dots, \omega_1^{k-1T})^T$ come from (k-1) previous dual solutions of SP_1; μ^{k-1} is a (k-1)-dimensional row vector variable, i.e. $\mu^{k-1} = (\mu_1^{k-1}, \mu_2^{k-1}, \dots, \mu_{k-1}^{k-1})$; θ_2 is an unrestricted scalar variable; and e^{k-1} is again the (k-1)-dimensional row vector with all components equal to 1. Because $\overline{SD_2^k}$ is a restriction of D and the restriction can be loosened at each iteration, by the inclusion of another π_1 in Π_1^{k-1} and ω_1 in Ω_1^{k-1}, it follows that $z^* \leq z_2^k$ $z_2^{k-1} \leq \dots \leq z_2^2$.

When k=1, there are no variables θ_2 or ϕ_1, and there are no cuts, nor are there λ or μ variables, because the algorithm begins by solving both SP_1 and SP_2 simultaneously, so there are no solutions yet available from both subproblems.

After both subproblems are solved concurrently, SP_1 sends dual information (called proposals) to SP_2, and receives primal information (proposals) from SP_2, while SP_2 sends primal information to SP_1, and receives dual proposals from SP_1. This communication scheme continues until the gap between their objective function values reduces to within a prescribed tolerance. Figure 3.1 shows this communication scheme.

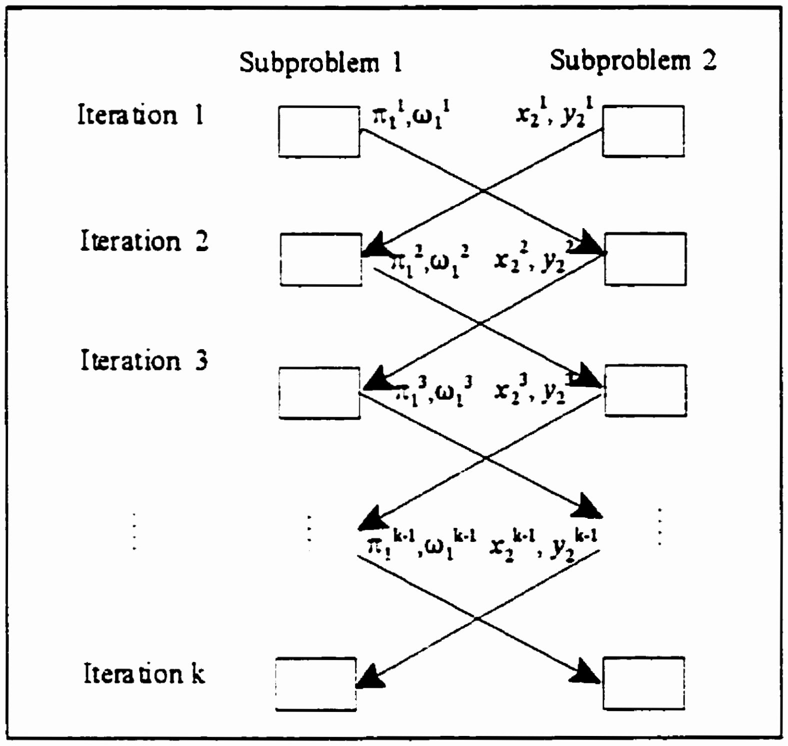

Figure 3.1 Information flow for parallel two-part algorithm

If only one processor is available, then the decomposition principle can be implemented in a serial fashion since any parallel algorithm can be implemented serially with the disadvantages of slower speed, larger computer memory or disk requirement (on one computer than on any one of a set of computers run in parallel), and the poor usage of available information. For example, the scheme in Figure 3.1 could be implemented serially by solving subproblem one, then subproblem two, then exchanging information, and repeating, i.e. by moving left to right and down in Figure 3.1. However, it seems advantageous to use proposal information as soon as it is available, so another serial implementation would solve one subproblem, then pass a proposal to the other subproblem, which is then solved, and a proposal is passed back to the first subproblem. The latter scheme has been implemented by Lan and Fuller [1995a] for the two stage model structure, a special case of the two part model structure, e.g. two period models. Lan and Fuller choose part one as the upper bound subproblem. The serial algorithm first solves the stage one subproblem and sends a primal proposal to the second stage subproblem, then it solves a new second stage subproblem with a new column generated, and sends dual proposals to the stage one subproblem. The stage one subproblem is then solved with a new cut. The stage one subproblem provides upper bounds on the optimal value of the whole problem, and the stage two subproblem provides lower bounds. This sequential process continues until the objective values of both subproblems have come within a predetermined tolerance.

3.3 The Decomposition Algorithm

In this section, the procedure of the parallel decomposition algorithm is formally described. Various properties of this algorithm will be discussed in the next section. The scalar $\varepsilon > 0$ is defined by the user to stop the algorithm when the upper bound z_2^k and the lower bound z_1^k are less than ε apart. Claims about feasibility and optimality are justified in the next section. In the statement of the algorithm, the symbols X_2^{k-2}, Y_2^{k-2}, Π_1^{k-2} and Ω_1^{k-2} appear, and for the value k=2, they have special meanings: X_2^0, and Y_2^0 are null matrices having no columns, while Π_1^0 and Ω_1^0 are null matrices having no rows. The send and receive statements synchronize the two processors' iteration counters.

DO IN PARALLEL

***Processor* 1**

Step 0. Set k=1, ε>0, and determine whether P is infeasible or the Assumption is violated.

- solve $\underline{SP}_1^1$; if it is infeasible or unbounded, send stop signal to SP_2 and stop, P is infeasible or the Assumption is violated;
- if stop signal from SP_2 is received, stop, P is infeasible;
- otherwise, record optimal dual solution $\pi_1{}^1$, $\omega_1{}^1$

Step 1. Set k=k+1, exchange information, modify SP_1 and solve it.

- set $\Pi_1{}^{k-1}=(\Pi_1{}^{k-2\,T}, \pi_1{}^{k-1\,T})^T$ and $\Omega_1{}^{k-1}=(\Omega_1{}^{k-2\,T}, \omega_1{}^{k-1T})^T$;
- send $(\pi_1{}^{k-1}D_1+\omega_1{}^{k-1}L_{11})$, $\omega_1{}^{k-1}L_{12}$, and $(\pi_1{}^{k-1}b_1+\omega_1{}^{k-1}f_1)$ to SP_2;
- receive $(c_2x_2{}^{k-1}+d_2y_2{}^{k-1})$, $L_{12}y_2{}^{k-1}$ and $(B_2x_2{}^{k-1}+L_{22}y_2{}^{k-1})$ from SP_2;
- solve $\underline{SP}_1^k$; record optimal $z_1{}^k, x_1{}^k, y_1{}^k, v_1{}^k, v_2{}^k, \lambda^{k-1}$ and $\pi_1{}^k, \omega_1{}^k$;
- send $z_1{}^k$ and receive $z_2{}^k$.

Step 2. Test for convergence: if $z_2{}^k - z_1{}^k \le \varepsilon$, go to step 3; otherwise, go to step 1.

Step 3. Send λ^{k-1} to and receive μ^{k-1} from SP_2. Calculate the optimal primal and dual solutions for part 1 as $(x_1^k, y_1^k, v_1^k, v_2^k)$ and $(\sum_{i=1}^{k-1}\mu_i^{k-1}\pi_1^i, \sum_{i=1}^{k-1}\mu_i^{k-1}\omega_1^i)$.

***Processor* 2**

Step 0. Set k=1, ε>0, and determine whether P is infeasible or the Assumption is violated.

- solve $\underline{\mathrm{SP}}_2^1$; if it is infeasible or unbounded, send stop signal to SP_1 and stop, P is infeasible or the Assumption is violated;
- if stop signal from SP_1 is received, stop, P is infeasible;
- otherwise, record optimal primal solution x_2^1, y_2^1

Step 1. Set k=k+1, exchange information, modify SP_2 and solve it.

- set $X_2^{k-1} = (X_2^{k-2}, x_2^{k-1})$ and $Y_2^{k-1} = (Y_2^{k-2}, y_2^{k-1})$;
- send $(c_2x_2^{k-1} + d_2y_2^{k-1})$, $L_{12}y_2^{k-1}$ and $(B_2x_2^{k-1} + L_{22}y_2^{k-1})$ to SP_1;
- receive $(\pi_1^{k-1}D_1 + \omega_1^{k-1}L_{11})$, $\omega_1^{k-1}L_{12}$, and $(\pi_1^{k-1}b_1 + \omega_1^{k-1}f_1)$ from SP_1;
- solve $\overline{\mathrm{SP}}_2^k$; record optimal z_2^k, x_2^k, y_2^k and π_2^k, ω_2^k, ρ_1^k, ρ_2^k, μ^{k-1} ;
- send z_2^k and receive z_1^k.

Step 2. Test for convergence: if $z_2^k - z_1^k \le \varepsilon$, go to step 3; otherwise, go to step 1.

Step 3. Send μ^{k-1} to and receive λ^{k-1} from SP_1. Calculate the optimal primal and dual solutions for part 2 as $(\sum_{i=1}^{k-1} x_2^i\lambda_i^{k-1}, \sum_{i=1}^{k-1} y_2^i\lambda_i^{k-1})$ and $(\rho_1^k, \pi_2^k, \omega_2^k, \rho_2^k)$; stop.

END

Note that x_2^k, y_2^k and π_2^k, ω_2^k would not actually be passed because there can be a more efficient communication scheme by exchanging smaller size vectors such as $L_{12}y_2^{k-1}$, $\omega_1^{k-1}L_{12}$ and $c_2x_2^{k-1} + d_2y_2^{k-1}$, etc. If there are some good feasible solutions available to the original problem, they may be used as a warm start since they always satisfy the constraints of $\underline{\mathrm{SP}}_1^1$ and $\overline{\mathrm{SP}}_2^1$ which don't have λ nor θ, nor cuts, thus saving efforts to find initial feasible solutions for the subproblems at the first iteration. Also, the algorithm can jump to k=2 for SP_1 and SP_2 with small duality gap if good primal and dual feasible solutions to the original problem are available.

Each subproblem works as both the master problem and the subproblem in the traditional decomposition methods: each accumulates proposals from the other, so is like a master problem yet each contains full details on only its own part, so is like a subproblem. The parallel algorithm approximates the optimal value of the original problem by calculating a nonincreasing upper bound and a nondecreasing lower bound at each iteration. This procedure terminates when the two bounds are considered to be close enough according to the prescribed tolerance. The optimal primal (or dual) solution of SP_2 (or SD_1) in Step 3 may be obtained in an alternate method by adding optimal allocated resource constraints to the subproblem of an extra, final iteration and solving the new subproblem rather than storing all the previous solutions and weighting them (Ho and Loute [1996]).

It is assumed that the LP solver takes care of any degeneracy (e.g. CPLEX uses perturbation methods). Therefore, a nondegeneracy assumption is not needed for the convergence proof in the next section.

3.4 Properties of the Algorithm

In this section, the important properties of the algorithm are discussed. The central result is the guarantee of convergence of the algorithm, Theorems 8 and 9.

Theorem 3.1*a* below shows that the part one subproblem cannot be unbounded for

k=1, and Theorem 3.1*b* rules out the possibility that the second subproblem can be unbounded for k=1. Theorem 3.1*c* ensures that the unboundedness (violation of the Assumption) is detected at k=1.

Theorem 3.1*a* *For k=1, subproblem $\underline{\mathrm{SP}}_1^k$ is either infeasible or has a bounded optimal solution.*

Proof: When k=1, $\underline{\mathrm{SP}}_1^k$ has no λ variable. By the Assumption, and the negative objective coefficients of the artificial variables, $\underline{\mathrm{SP}}_1^1$ can't have an unbounded solution. Therefore, $\underline{\mathrm{SP}}_1^1$ is either infeasible or has a finite optimal solution.

Theorem 3.1*b* *For k=1, subproblem $\overline{\mathrm{SP}}_2^k$ is either infeasible or has a bounded optimal solution.*

Proof: When k=1, the subproblem $\overline{\mathrm{SP}}_2^k$ has no cuts or θ variable. The possibility of unboundedness is ruled out by the Assumption, and the large negative objective coefficients of the artificial variables. Therefore, $\overline{\mathrm{SP}}_2^1$ is either infeasible or has a bounded optimal solution.

Suppose a modeller unintentionally submits a model that violates the Assumption and has an unbounded optimal value. Then, the algorithm detects the violation of the Assumption and stops.

Theorem 3.1c *For $k=1$, if the algorithm returns a message of an unbounded optimal value, then the Assumption is violated and the algorithm stops.*

Proof: When $k=1$, there is no λ or θ variable. If the set defined by the nonlinking constraints and upper bounds of $\underline{SP}_1^1$ has an unbounded ray, $\underline{SP}_1^1$ is unbounded, too, because the linking constraints can be satisfied by the ray through a choice of the variables v_1 and v_2. A similar observation can be made for $\overline{SP}_2^1$. Therefore, at least one subproblem has unbounded optimal value for $k=1$, then the Assumption is violated and the algorithm stops.

The following theorems show that subproblems $\underline{SP}_1^k$ and $\overline{SP}_2^k$ have finite optimal solutions, at each iteration for $k>1$. Recall that the algorithm proceeds for $k>1$ only if $\underline{SP}_1^1$ and $\overline{SP}_2^1$ have finite optimal solutions.

The following theorems show that subproblems $\underline{SP}_1^k$ and $\overline{SP}_2^k$ have finite optimal solutions, at each iteration for $k>1$. Recall that the algorithm proceeds for $k>1$ only if $\underline{SP}_1^1$ and $\overline{SP}_2^1$ have finite optimal solutions.

Theorem 3.2a *When $k>1$, subproblem $\underline{SP}_1^k$ has a finite optimal solution.*

Proof: For the same reasons as in the proof of Theorem 1a, together with the constraints $e^{k-1}\lambda^{k-1}=1,\ \lambda^{k-1}\geq 0$, $\underline{SP}_1^k$ can't be unbounded. Next, if $\underline{SP}_1^k$ were infeasible, the set of nonlinking constraints and upper bound constraints in $\underline{\mathrm{SP}}_1^k$ should be infeasible because the artificial variables in the linking constraints guarantee that these constraints are always satisfied.

However, for the algorithm to proceed to iteration $k>1$, the same set of nonlinking constraints and upper bound constraints, which was present in $\underline{\mathrm{SP}}_1^1$ at step 0, must be feasible, a contradiction. Thus, $\underline{\mathrm{SP}}_1^k$ is feasible for $k>1$. Therefore, $\underline{\mathrm{SP}}_1^k$ has a finite optimal solution at each iteration $k>1$.

Theorem 3.2*b* *For* $k>1$, *each subproblem* $\overline{\mathrm{SP}}_2^k$ *has a finite optimal solution.*

Proof: For $k>1$, the feasibility of $\overline{\mathrm{SP}}_2^k$ is discussed first and then the boundedness will be shown. There exists a feasible solution $(y_1^f, x_2^f, y_2^f, v_2^f)$ which satisfies all constraints except possibly the set of first constraints (or cuts) in $\overline{\mathrm{SP}}_2^k$ because if not, the algorithm would have stopped at step 0. The first set of constraints is satisfied if θ_2 is chosen as

$$\theta_2 = \min_{i=1,2,\dots,k-1} \pi_1^i b_1 + \omega_1^i f_1 - (\pi_1^i D_1 + \omega_1^i L_{11})\, y_1^f - \omega_1^i L_{12}\, y_2^f .$$

Next, the boundedness can be shown by the following reasoning:

$$\max_{y_1, x_2, y_2, v_2, \theta_2} \{ d_1 y_1 + c_2 x_2 + d_2 y_2 - M_2 v_2 + \theta_2 \,|$$
$$(\pi_1^i D_1 + \omega_1^i L_{11})\, y_1 + \omega_1^i L_{12}\, y_2 + \theta_2 \le \pi_1^i b_1 + \omega_1^i f_1, \;\; for \;\; i = 1, 2, \dots, k-1,$$
$$y_1 \le u_1,\; A_2 x_2 + D_2 y_2 \le b_2,\;\; L_{21} y_1 + B_2 x_2 + L_{22} y_2 - v_2 \le f_2,\;\; y_2 \le u_2, y_1, x_2, y_2, v_2 \ge 0\}$$

$$\begin{aligned}
&\max_{y_1, x_2, y_2, v_2, \theta_2} \{ d_1 y_1 + c_2 x_2 + d_2 y_2 - M_2 v_2 \mid \\
\le\ & (\pi_1^i D_1 + \omega_1^i L_{11}) y_1 + \omega_1^i L_{12} y_2 + \theta_2 \le \pi_1^i b_1 + \omega_1^i f_1, \ \textit{for}\ i = 1, 2, \ldots, k-1, \\
& y_1 \le u_1, A_2 x_2 + D_2 y_2 \le b_2, L_{21} y_1 + B_2 x_2 + L_{22} y_2 - v_2 \le f_2, y_2 \le u_2, \ y_1, x_2, y_2, v_2 \ge 0 \} \\
+\ & \\
&\max_{y_1, x_2, y_2, v_2, \theta_2} \{ \theta_2 \mid (\pi_1^i D_1 + \omega_1^i L_{11}) y_1 + \omega_1^i L_{12} y_2 + \theta_2 \le \pi_1^i b_1 + \omega_1^i f_1, \ \textit{for}\ i = 1, 2, \ldots, k-1, \\
& y_1 \le u_1, A_2 x_2 + D_2 y_2 \le b_2, L_{21} y_1 + B_2 x_2 + L_{22} y_2 - v_2 \le f_2, y_2 \le u_2, \ y_1, x_2, y_2, v_2 \ge 0 \} \\
\le\ & \max_{y_1, x_2, y_2, v_2} \{ d_1 y_1 + c_2 x_2 + d_2 y_2 - M_2 v_2 \mid y_1 \le u_1, A_2 x_2 + D_2 y_2 \le b_2, \\
& \qquad L_{21} y_1 + B_2 x_2 + L_{22} y_2 - v_2 \le f_2, \ y_2 \le u_2, \ y_1, x_2, y_2, v_2 \ge 0 \} \\
+\ & \max_{i=1,2,\ldots,k-1} \ \max_{y_1, x_2, y_2, v_2} \{ \pi_1^i b_1 + \omega_1^i f_1 - (\pi_1^i D_1 + \omega_1^i L_{11}) y_1 - \omega_1^i L_{12} y_2 \mid y_1 \le u_1, \\
& A_2 x_2 + D_2 y_2 \le b_2, \ L_{21} y_1 + B_2 x_2 + L_{22} y_2 - v_2 \le f_2, \ y_2 \le u_2, \ y_1, x_2, y_2, v_2 \ge 0 \}
\end{aligned}$$

The latter maxima are finite, by the Assumption. Therefore, SP_2^k has a bounded optimal solution for $k>1$.

Theorem 3.3 shows that the original problem P is infeasible if and only if one of the subproblems is determined to be infeasible at step 0.

Theorem 3.3 *Problem* P *is infeasible iff the algorithm detects infeasibility during step* 0.

Proof: (The "if" part) If the algorithm on processor 1 reports that P is infeasible at step 0, then the set of nonlinking constraints and upper bound constraints of stage one is infeasible, because the linking constraints can always be satisfied due to the artificial variables. This implies that P is infeasible. Similarly, if the algorithm on processor 2 reports that P is infeasible at step 0, then the set $\{(x_2, y_2) \mid A_2 x_2 + D_2 y_2 \le b_2, \ y_2 \le u_2, \ x_2, \ y_2 \ge 0\}$ is empty, which means that P is infeasible.

(The "only if" part) If P is infeasible, then since the linking constraints can always be satisfied due to the artificial variables, it follows that one or both of the following sets of constraints are infeasible:

(a) the nonlinking constraints and the upper bound constraints of stage one,

$A_1 x_1 + D_1 y_1 \leq b_1, \qquad y_1 \leq u_1$, or

(b) the nonlinking constraints and the upper bound constraints of stage two,

$A_2 x_2 + D_2 y_2 \leq b_2, \qquad y_2 \leq u_2$.

The infeasibility of (a) will be detected at step 0 on processor 1. The infeasibility of (b) will be reported at step 0 on processor 2.

The next theorem shows that an optimal solution of $\underline{\mathrm{SP}}_1^k$ provides a primal feasible solution for the original problem P and $\overline{\mathrm{SD}}_2^k$ provides a dual feasible solution to D. Theorem 3.5 shows that the algorithm issues both monotonically improved lower bounds and upper bounds as the iterations proceed.

Theorem 3.4 $\underline{\mathrm{SP}}_1^k$ *provides a primal feasible solution* $(x_1^k, y_1^k, v_1^k, \sum_{i=1}^{k-1} x_2^i\lambda_i^{k-1}, \sum_{i=1}^{k-1} y_2^i\lambda_i^{k-1}, v_2^k)$ *to* P *and* $\overline{\mathrm{SD}}_2^k$ gives *a dual feasible solution* $(\sum_{i=1}^{k-1}\mu_i^{k-1}\pi_1^i, \sum_{i=1}^{k-1}\mu_i^{k-1}\omega_1^i, \rho_1^k, \pi_2^k, \omega_2^k, \rho_2^k)$ *to* D *when* $k>1$.

Proof: The expressions $s_x^k = \sum_{i=1}^{k-1} x_2^i\lambda_i^{k-1}$ and $s_y^k = \sum_{i=1}^{k-1} y_2^i\lambda_i^{k-1}$ are convex combinations of x_2 and y_2 values respectively, each of which satisfies the constraints $A_2 x_2 + D_2 y_2 \leq b_2$, $y_2 \leq u_2$ with nonnegativity constraints. Because $(x_1^k, y_1^k, v_1^k, v_2^k, S_x^k, S_y^k)$ solves $\underline{\mathrm{SP}}_1^k$, the part one constraints and linking constraints of P are satisfied. Therefore, $(x_1^k, y_1^k, v_1^k, v_2^k, S_x^k, S_y^k)$ satisfies all constraints of P. The proof for the dual is similar.

Note that $\sum_{i=1}^{k-1} x_2^i \lambda_i^{k-1}$ and $\sum_{i=1}^{k-1} y_2^i \lambda_i^{k-1}$ are not actually calculated on processor 1 at any iteration; they are calculated on processor 2, when convergence is achieved, by passing λ^{k-1} from processor 1 to processor 2. Similarly, $\sum_{i=1}^{k-1} \mu_i^{k-1} \pi_1^i$ and $\sum_{i=1}^{k-1} \mu_i^{k-1} \omega_1^i$ are not actually calculated on processor 2 at any iteration; they are calculated on processor 1 by passing μ^{k-1} from processor 2 to processor 1, when convergence is achieved.

Theorem 3.5 *If the algorithm proceeds to iterations* $k>1$, *then*

$$z_1^2 \leq \ldots \leq z_1^{k-1} \leq z_1^k \leq z^* \leq z_2^k \leq z_2^{k-1} \leq \ldots \leq z_2^2.$$

Proof: Since $\underline{\mathrm{SP}}_1^k$ for $k>1$ is a restriction of P and the feasible region of subproblem $\underline{\mathrm{SP}}_1^k$ includes that of the previous subproblem (by restricting the newest variable, $\lambda_{k-1}^{k-1} = 0$), it follows that $z_1^2 \leq \ldots \leq z_1^{k-1} \leq z_1^k \leq z^*$. Similarly, $\overline{\mathrm{SD}}_2^k$ is a restriction of the dual of P, that is loosened at each iteration, which proves the remainder of the claim.

The following theorem verifies the calculation of the primal and dual optimal solutions of P.

Theorem 3.6 *Suppose* $(x_1^k, y_1^k, v_1^k, \sum_{i=1}^{k-1} x_2^i \lambda_i^{k-1}, \sum_{i=1}^{k-1} y_2^i \lambda_i^{k-1}, v_2^k)$ *and* $(\sum_{i=1}^{k-1} \mu_i^{k-1} \pi_1^i, \sum_{i=1}^{k-1} \mu_i^{k-1} \omega_1^i, \rho_1^k, \pi_2^k, \omega_2^k, \rho_2^k)$ *are the optimal solutions of* $\underline{SP_1^k}$ *and* $\overline{SD_2^k}$, *respectively at iteration k. They are optimal primal and dual solutions of P and D if and only if* $z_1^k = z_2^k$.

Proof: Basic duality theory, and the feasibility of $(x_1^k, y_1^k, v_1^k, \sum_{i=1}^{k-1} x_2^i \lambda_i^{k-1}, \sum_{i=1}^{k-1} y_2^i \lambda_i^{k-1}, v_2^k)$ *and* $(\sum_{i=1}^{k-1} \mu_i^{k-1} \pi_1^i, \sum_{i=1}^{k-1} \mu_i^{k-1} \omega_1^i, \rho_1^k, \pi_2^k, \omega_2^k, \rho_2^k)$ in P and D ensure the result.

Because the two subproblems are solved simultaneously, it takes two iterations of the parallel method for one subproblem to respond to the information of the other subproblem. Therefore, the subproblem of the next iteration can have the same solution as the current iteration. Theorem 3.7 shows that if the feasible solution of current iteration is feasible in the next two consecutive iterations of the parallel algorithm, then it is an optimal solution to the original problem.

Theorem 3.7 *For any k>1, if the primal optimal solution of* $\overline{SP_2^{k-1}}$, $(y_1^{k-1}, x_2^{k-1}, y_2^{k-1}, v_2^{k-1}, \theta_2^{k-1})$, *is feasible in* $\overline{SP_2^k}$ *and* $\overline{SP_2^{k+1}}$, *then the optimum is reached.*

Proof: For any $k>1$, suppose that the primal optimal solution of $\overline{SP_2^{k-1}}$, $(y_1^r, x_2^r, y_2^r, v_2^r, \theta_2^r)$ is feasible in $\overline{SP_2^k}$ and $\overline{SP_2^{k+1}}$. Since $\overline{SP_2^{k+1}}$ is a restriction of $\overline{SP_2^k}$, which is a restriction of $\overline{SP_2^{k-1}}$, $(y_1^r, x_2^r, y_2^r, v_2^r, \theta_2^r)$ is optimal in $\overline{SP_2^k}$ and $\overline{SP_2^{k+1}}$. Then, the unchanged proposal, (x_2^r, y_2^r), could be passed to $\underline{\mathrm{SP}_1^k}$, $\underline{\mathrm{SP}_1^{k+1}}$ and $\underline{\mathrm{SP}_1^{k+2}}$, which would all be the same problem. Therefore, the optimal solution of $\underline{\mathrm{SP}_1^k}$ is also on optimum of $\underline{\mathrm{SP}_1^{k+1}}$ and $\underline{\mathrm{SP}_1^{k+2}}$. The algorithm will be repeated with the same solutions of both subproblems thereafter.

From $\underline{\mathrm{SD}_1^k}$ (for clear presentation we put subscripts with parentheses on the lower right side of variables except nonlinking variables, which indicates the subproblem in which the value was solved for)

$$\phi_1^k \geq c_2 x_2^{k-1} + d_2 y_{2,(2)}^{k-1} - \omega_{1,(1)}^k L_{12} y_{2,(2)}^{k-1} - \omega_{2,(1)}^k (B_2 x_2^{k-1} + L_{22} y_{2,(2)}^{k-1}),$$

and from $\overline{\mathrm{SP}_2^{k-1}}$ $z_2^{k-1} = d_1 y_{1,(2)}^{k-1} + c_2 x_2^{k-1} + d_2 y_{2,(2)}^{k-1} - M_2 v_{2,(2)}^{k-1} + \theta_2^{k-1}$, so we have

$$\phi_1^k \geq z_2^{k-1} - d_1 y_{1,(2)}^{k-1} + M_2 v_{2,(2)}^{k-1} - \theta_2^{k-1} - \omega_{1,(1)}^k L_{12} y_{2,(2)}^{k-1} - \omega_{2,(1)}^k (B_2 x_2^{k-1} + L_{22} y_{2,(2)}^{k-1}). \tag{3.1}$$

From $\overline{\mathrm{SP}_2^{k+1}}$

$$\theta_2^{k+1} \leq \pi_1^k b_1 + \omega_{1,(1)}^k f_1 - (\pi_1^k D_1 + \omega_{1,(1)}^k L_{11}) y_{1,(2)}^{k+1} - \omega_{1,(1)}^k L_{12} y_{2,(2)}^{k+1},$$

and from $\underline{\mathrm{SD}_1^k}$ $z_1^k = \pi_1^k b_1 + \omega_{1,(1)}^k f_1 + \rho_{1,(1)}^k u_1 + \omega_{2,(1)}^k f_2 + \phi_1^k$. so we have

$$\theta_2^{k+1} \leq z_1^k - \rho_{1,(1)}^k u_1 - \omega_{2,(1)}^k f_2 - \phi_1^k - (\pi_1^k D_1 + \omega_{1,(1)}^k L_{11}) y_{1,(2)}^{k+1} - \omega_{1,(1)}^k L_{12} y_{2,(2)}^{k+1}. \tag{3.2}$$

Subtracting (3.1) from (3.2) gives

$$\begin{aligned}\theta_2^{k+1} \leq {} & z_1^k - \rho_{1,(1)}^k u_1 - \omega_{2,(1)}^k f_2 - z_2^{k-1} + d_1 y_{1,(2)}^{k-1} - M_2 v_{2,(2)}^{k-1} + \theta_2^{k-1} + \omega_{1,(1)}^k L_{12} y_{2,(2)}^{k-1} \\ & + \omega_{2,(1)}^k (B_2 x_2^{k-1} + L_{22} y_{2,(2)}^{k-1}) - (\pi_1^k D_1 + \omega_{1,(1)}^k L_{11}) \; y_{1,(2)}^{k+1} - \omega_{1,(1)}^k L_{12} y_{2,(2)}^{k+1}\end{aligned} \quad . \tag{3.3}$$

Since $-M_2 \leq -\omega_{2,(1)}^k$ and $\pi_1^k D_1 + \omega_{1,(1)}^k L_{11} + \rho_{1,(1)}^k + \omega_{2,(1)}^k L_{21} \geq d_1$ from $\underline{\mathrm{SD}_1^k}$

and $-u_1 \leq -y_{1,(2)}^{k-1}$, $-f_2 \leq -L_{21} y_{1,(2)}^{k-1} - B_2 x_2^{k-1} - L_{22} y_{2,(2)}^{k-1} + v_{2,(2)}^{k-1}$ from $\overline{\mathrm{SP}_2^{k-1}}$, (3.3) gives

$$\begin{aligned}\theta_2^{k+1} + (\pi_1^k D_1 + \omega_{1,(1)}^k L_{11})\ y_{1,(2)}^{k+1} + \omega_{1,(1)}^k L_{12}\, y_{2,(2)}^{k+1} \le z_1^k - z_2^{k-1} + \theta_2^{k-1} - \rho_{1,(1)}^k\, y_{1,(2)}^{k-1} - \omega_{2,(1)}^k\, v_{2,(2)}^{k-1} + \\ - \omega_{2,(1)}^k (L_{21}\, y_{1,(2)}^{k-1} + B_2 x_2^{k-1} + L_{22}\, y_{2,(2)}^{k-1} - v_{2,(2)}^{k-1}) \\ + (\pi_1^k D_1 + \omega_{1,(1)}^k L_{11} + \rho_{1,(1)}^k + \omega_{2,(1)}^k L_{21})\, y_{1,(2)}^{k-1} \\ + \omega_{1,(1)}^k L_{12}\, y_{2,(2)}^{k-1} + \omega_{2,(1)}^k (B_2 x_2^{k-1} + L_{22}\, y_{2,(2)}^{k-1})\end{aligned},$$

so it becomes

$$\theta_2^{k+1} + (\pi_1^k D_1 + \omega_{1,(1)}^k L_{11})\, y_{1,(2)}^{k+1} + \omega_{1,(1)}^k L_{12}\, y_{2,(2)}^{k+1} \le z_1^k - z_2^{k-1} + \theta_2^{k-1} + (\pi_1^k D_1 + \omega_{1,(1)}^k L_{11})\ y_{1,(2)}^{k-1} + \omega_{1,(1)}^k L_{12}\, y_{2,(2)}^{k-1}$$

Since $\theta_2^{k+1} = \theta_2^{k-1}$, $y_{1,(2)}^{k+1} = y_{1,(2)}^{k-1}$, $y_{2,(2)}^{k+1} = y_{2,(2)}^{k-1}$ and by Theorem 3.5, it should be $z_1^{\,k} = z_2^{\,k-1}$ and the optimal value has been reached.

The convergence of the algorithm is proved using the following lemma and theorems.

Lemma *Let* $\{z_1^k\}_{k=1}^{\infty}$, $\{z_2^k\}_{k=1}^{\infty}$ *and* $\{w_i^k\}_{k=1}^{\infty}$ *be infinite sequences, where* $z_1^{\,k} \in \mathbf{R}^1$, $z_2^{\,k} \in \mathbf{R}^1$, *and* $w_i^{\,k} \in W_i \subset \mathbf{R}^{n_i}$ *for* $i=1,2,\ldots,I$, *where* W_i *is closed and bounded. Suppose that there exists* $Q>0$, *for all* k *and all* $j>k$, *such that*

$$0 \le z_2^{\,j} - z_1^{\,k} \le Q \sum_{i=1}^{I} \left\| w_i^j - w_i^k \right\|.$$

Then, there exists at least one subsequence indexed by $S = \{k_n\}_{n=1}^{\infty} \subseteq \mathrm{N}$ *such that* $\lim_{n\to\infty} (z_2^{k_{n+1}} - z_1^{k_n}) = 0$.

Proof: Recall that, for any infinite sequence of vectors chosen from a closed and bounded set, there exists at least one convergent infinite subsequence. Therefore, there exists a convergent infinite subsequence of $\{w_1^k\}_{k=1}^{\infty}$ indexed by $S_1 \subseteq \mathrm{N}$. Similarly, there exists a convergent infinite subsequence of $\{w_2^k\}_{k=1}^{\infty}$, indexed by $S_2 \subseteq S_1$. Proceeding in a similar fashion, we may conclude that there exist subsequences indexed by S=S_I such that $\{w_i^k\}$ converges for every i=1, 2,...., I, when $k \in S$. The inequalities in the assumption hold for any j>k, so they hold in particular for adjacent elements of the subsequence S, labeled k_n, k_{n+1}. Because the difference $z_2^{k_{n+1}} - z_1^{k_n}$ is bounded below by 0 and above by an expression which converges to 0 through the subsequence S, the claim is proven.

Theorem 3.8 *Consider the primal and dual sequences* $\{(y_1^j, x_2^j, y_2^j, v_2^j, \theta_2^j)\}_{j=1}^{\infty}$ and $\{(\pi_1^k, \omega_1^k, \rho_1^k, \omega_2^k, \phi_1^k)\}_{k=1}^{\infty}$ *generated by* $\overline{\mathrm{SP}}_2^j$ *and* $\underline{\mathrm{SD}}_1^k$, *with the objective values* $z_2^j = d_1\, y_1^j + c_2\, x_2^j + d_2 y_2^j - M_2\, v_2^j + \theta_2^j$ *and* $z_1^k = \pi_1^k\, b_1 + \omega_1^k f_1 + \rho_1^k\, u_1 + \omega_2^k f_2 + \phi_1^k$. *There exists an infinite subsequence* $S = \{k_n\}_{n=1}^{\infty} \subseteq \mathrm{N}$ *such that the objectives converge to equal values in the limit, i.e.* $(z_2^{k_{n+1}} - z_1^{k_n}) \longrightarrow 0$ *as* $n \longrightarrow \infty$.

Proof: Consider any iteration j>k. From $\underline{\mathrm{SD}}_1^k$, we have

$$z_1^k \geq \pi_1^k\, b_1 + \omega_1^k\, f_1 + \rho_1^k\, u_1 + \omega_2^k\, f_2 + c_2\, x_2^{k-1} + d_2\, y_2^{k-1} - \omega_1^k\, L_{12}\, y_2^{k-1} - \omega_2^k (B_2\, x_2^{k-1} + L_{22}\, y_2^{k-1}) \qquad (3.4)$$

From $\overline{\mathrm{SP}}_2^j$, we have

$$z_2^j \leq d_1\, y_1^j + c_2\, x_2^j + d_2\, y_2^j - M_2\, v_2^j + \pi_1^k\, b_1 + \omega_1^k\, f_1 - \pi_1^k\, D_1\, y_1^j - \omega_1^k\, L_{11}\, y_1^j - \omega_1^k\, L_{12}\, y_2^j \qquad (3.5)$$

Subtracting (3.4) from (3.5) gives

$$\begin{aligned} z_2^j - z_1^k \leq\, & d_1\, y_1^j + c_2\, x_2^j + d_2\, y_2^j - M_2\, v_2^j - \pi_1^k\, D_1\, y_1^j - \omega_1^k\, L_{11}\, y_1^j - \omega_1^k\, L_{12}\, y_2^j \\ & - \rho_1^k\, u_1 - \omega_2^k\, f_2 - c_2\, x_2^{k-1} - d_2\, y_2^{k-1} + \omega_1^k\, L_{12}\, y_2^{k-1} + \omega_2^k\, B_2\, x_2^{k-1} + \omega_2^k\, L_{22}\, y_2^{k-1} \end{aligned} \qquad (3.6)$$

By multiplying ω_2^k (≥ 0) of $\underline{\mathrm{SD}_1^k}$ to both sides of $L_{21} y_1^{k-1} + B_2 x_2^{k-1} + L_{22} y_2^{k-1} - v_2^{k-1} \leq f_2$ from $\overline{\mathrm{SP}_2^{k-1}}$, we have

$$\omega_2^k L_{21} y_1^{k-1} + \omega_2^k B_2 x_2^{k-1} + \omega_2^k L_{22} y_2^{k-1} - \omega_2^k v_2^{k-1} \leq \omega_2^k f_2 . \tag{3.7}$$

Substituting (3.7) into (3.6) yields

$$\begin{aligned} z_2^j - z_1^k \leq{} & d_1 y_1^j + c_2 x_2^j + d_2 y_2^j - M_2 v_2^j - \pi_1^k D_1 y_1^j - \omega_1^k L_{11} y_1^j - \omega_1^k L_{12} y_2^j \\ & - \rho_1^k u_1 - c_2 x_2^{k-1} - d_2 y_2^{k-1} + \omega_1^k L_{12} y_2^{k-1} - \omega_2^k L_{21} y_1^{k-1} + \omega_2^k v_2^{k-1} \end{aligned} \tag{3.8}$$

By multiplying y_1^j (≥ 0) of $\overline{\mathrm{SP}_2^j}$ to both sides of $\pi_1^k D_1 + \omega_1^k L_{11} + \rho_1^k + \omega_2^k L_{21} \geq d_1$ from $\underline{\mathrm{SD}_1^k}$, we have

$$\pi_1^k D_1 y_1^j + \omega_1^k L_{11} y_1^j + \rho_1^k y_1^j + \omega_2^k L_{21} \; y_1^j \geq d_1 y_1^j . \tag{3.9}$$

Combining (3.9) with (3.8) gives

$$\begin{aligned} z_2^j - z_1^k \leq{} & c_2 x_2^j + d_2 y_2^j - M_2 v_2^j - \omega_1^k L_{12} y_2^j - \rho_1^k u_1 - c_2 x_2^{k-1} - d_2 y_2^{k-1} \\ & + \omega_1^k L_{12} y_2^{k-1} - \omega_2^k L_{21} y_1^{k-1} + \omega_2^k v_2^{k-1} + \rho_1^k y_1^j + \omega_2^k L_{21} y_1^j \end{aligned} \tag{3.10}$$

From $\underline{\mathrm{SD}_1^k}$, if $\omega_2^k \leq M_2$ is multiplied by v_2^{k-1}(≥ 0) of $\overline{\mathrm{SP}_2^{k-1}}$, then it follows that

$$\omega_2^k v_2^{k-1} \leq M_2 v_2^{k-1}. \tag{3.11}$$

From $\overline{\mathrm{SP}_2^j}$, if $y_1^j \leq u_1$ is multiplied by ρ_1^k (≥ 0) of $\underline{\mathrm{SD}_1^k}$, then we have

$$\rho_1^k y_1^j \leq \rho_1^k u_1 . \tag{3.12}$$

Combining (3.11) and (3.12) with (3.10) gives

$$\begin{aligned} z_2^j - z_1^k \leq{} & c_2 x_2^j + d_2 y_2^j - M_2 v_2^j - \omega_1^k L_{12} y_2^j - c_2 x_2^{k-1} - d_2 y_2^{k-1} \\ & + \omega_1^k L_{12} y_2^{k-1} - \omega_2^k L_{21} y_1^{k-1} + \omega_2^k L_{21} y_1^j + M_2 v_2^{k-1} \end{aligned} \tag{3.13}$$

Rearranging (3.13), and using Theorem 3.5 gives

$$0 \le z_2^j - z_1^k \le \omega_2^k L_{21}(y_1^j - y_1^{k-1}) + c_2(x_2^j - x_2^{k-1}) + (d_2 - \omega_1^k L_{12})(y_2^j - y_2^{k-1}) - M_2(v_2^j - v_2^{k-1}) .$$

The expression on the right is a sum of products of two vectors, therefore

$$z_2^j - z_1^k \le \|\omega_2^k L_{21}\| \bullet \|y_1^j - y_1^{k-1}\| + \|c_2\| \bullet \|x_2^j - x_2^{k-1}\| + \|d_2 - \omega_1^k L_{12}\| \bullet \|y_2^j - y_2^{k-1}\| + \|-M_2\| \bullet \|v_2^j - v_2^{k-1}\| . \quad (3.14)$$

Since ω_1 and ω_2 have upper bounds of M_1 and M_2, we can let $Q_1 = \max\limits_{0 \le \omega_1 \le M_1} \|d_2 - \omega_1 L_{12}\|$,

$Q_2 = \max\limits_{0 \le \omega_2 \le M_2} \|\omega_2 L_{21}\|$ and $Q = \max\{Q_1, Q_2, \|c_2\|, \|-M_2\|\}$.

Then, since $M_2 > 0$, it follows that $Q > 0$. Therefore,

$$0 \le z_2^j - z_1^k \le Q\{\|y_1^j - y_1^{k-1}\| + \|x_2^j - x_2^{k-1}\| + \|y_2^j - y_2^{k-1}\| + \|v_2^j - v_2^{k-1}\|\} . \quad (3.15)$$

However, the artificial variables, v_2^k (≥ 0), are also drawn from a closed, bounded set by the following reasoning. From P, we have $v_2 \ge L_{21} y_1 + B_2 x_2 + L_{22} y_2 - f_2$; since v_2 is very costly, the optimization process forces equality at the optimum. An upper bound on v_2^k can be obtained by maximizing each component by selection of y_1, x_2, y_2 from the bounded set defined by $A_2 x_2 + D_2 y_2 \le b_2$, $y_1 \le u_1$, $y_2 \le u_2$ together with $y_1 \ge 0$, $x_2 \ge 0$, $y_2 \ge 0$. Call the upper bound $v_{2\max}$. Then, $0 \le v_2^k \le v_{2\max}$.

Therefore, by the Lemma there exists a subsequence $\{k_n\}_{n=1}^{\infty}$ such that $(z_2^{k_{n+1}} - z_1^{k_n}) \to 0$ *as* $n \to \infty$.

Theorem 3.9 *The objectives of the entire sequence of optimal solutions converge to equal values, in the limit, i.e.* $(z_2^j - z_1^{j-1}) \to 0$ *as* $j \to \infty$.

Proof: For any particular value of $j \ge k_1$, there is an earlier iteration number, from the subsequence, $\{k_n\}_{n=1}^{\infty}$, which is closest to j, i.e. $k_{n_j} = \max\{k_n \mid k_n \le j, j \in \mathbb{N}\}$. By Theorem 3.5,

$z_2^j - z_1^{j-1} \leq z_2^{k_{*,j}} - z_1^{k_{*,j-1}}$, and by Theorem 3.8, the right side of the inequality converges to zero. Therefore, $(z_2^j - z_1^{j-1}) \longrightarrow 0$ *as* $j \rightarrow \infty$ through $j \in \mathbf{N}$.

Corollary *If the parallel primal-dual decomposition algorithm proceeds to iteration k>1, then with the given tolerance ε > 0, it stops in a finite number of iterations.*

Proof: The corollary follows directly from the stopping criterion ε>0 and Theorem 3.9.

3.5 Summary and Observations on the Algorithm

1. The original problem is divided into two types of subproblems in each part: a lower bound type subproblem and an upper bound type subproblem.

2. The lower bound subproblem in primal form is constructed by restricting primal variables, x_2 and y_2, to convex combinations of known values, received from the upper bound subproblem, thus dropping out redundant primal nonlinking constraints and primal upper bound constraints. The upper bound subproblem in dual form is constructed by restricting dual variables, π_1 and ω_1, to convex combinations of known values, received from the first subproblem, then dropping out redundant dual nonlinking constraints and dual upper bound constraints.

3. The parallel algorithm has a perfectly balanced structure by the two master-like subproblems instead of the master and subproblem structure as in the traditional decomposition methods.

4. Information on primal and dual solutions is sent and received between the two subproblems at each iteration. The dual solution of the first subproblem is passed to

the second subproblem to make a cut and the primal solutions of the second subproblem are passed to the first subproblem as the proposal information.

5. The new algorithm approximates the objective function value by issuing monotonically improved lower bounds and upper bounds by the first subproblem and second subproblem, respectively during the iterations.

6. This algorithm performs the convergence test by referring to both subproblems simultaneously and converges to a given tolerance in a finite number of steps.

7. If, in the upper bounding subproblem, $\overline{SP_2^k}$, all cuts except the most recent are eliminated, then the algorithm reduces to the familiar Dantzig-Wolfe decomposition algorithm.

8. If, in the lower bounding subproblem, $\underline{SP_1^k}$, the weight on the most recent proposal is required to equal 1 (and the other weights are zero), then the algorithm reduces to the familiar Benders decomposition algorithm.

9. The parallel decomposition algorithm can be extended to more than two part problems. Chapter 4 discusses a parallel algorithm for the multi-part problems.

Chapter 4 Parallel Decomposition of the Multi-Part Model

In this chapter, two parallel decomposition methods for multi-part linear programming problems are presented. The first is developed by applying a hierarchical decomposition principle recursively. We first divide the original multi-part problem into two aggregated subproblems of lower bound type and upper bound type, by applying the basic algorithm of the two-part method, which was discussed in the previous chapter. Then, the aggregated subproblems are further divided into two smaller aggregated subproblems of upper bound and lower bound. This results in some primal subproblems accumulating both proposals and cuts. This bifurcation process continues until there are no subproblems left for further decomposition. The subproblems are solved in different processors simultaneously and work together to reach an optimal point during the iterations by exchanging information in the hierarchical way. In Chapter 5, we report successful convergence on several test problems.

The second method is less complex than the first. It defines some subproblems to be of the lower bound type from Chapter 3, i.e. utilizing primal proposals from other subproblems, and the other subproblems are of the upper bound type. Unfortunately, this method fails to converge in some tests. We include it because it may be useful as a heuristic.

4.1 Definition of a Multi-Part Linear Program

The definition of the two-part problems is now extended to that of multi-part problems. Consider the following primal and dual forms of LP problem of N parts:

$$
\begin{array}{llllll}
\text{P:} & \max \sum_{t=1}^{N} (c_t x_t + d_t y_t - M_t v_t) & & & & \\
(\pi_t^T) & s.t. & A_t x_t + D_t y_t & \le b_t, & t = 1, 2, \ldots, N \\
(\omega_t^T) & & B_t x_t + \sum_{s=1}^{N} L_{ts} y_s - v_t & \le f_t, & t = 1, 2, \ldots, N \\
(\rho_t^T) & & y_t & \le u_t, & t = 1, 2, \ldots, N \\
 & & x_t, y_t, v_t \ge 0 & for & t = 1, 2, \ldots, N
\end{array}
$$

$$
\begin{array}{llllll}
\text{D:} & \min \sum_{t=1}^{N} (\pi_t b_t + \omega_t f_t + \rho_t u_t) & & & & \\
(x_t^T) & s.t. & \pi_t A_t + \omega_t B_t & \ge c_t, & t = 1, 2, \ldots, N \\
(y_t^T) & & \pi_t D_t + \sum_{s=1}^{N} \omega_s L_{st} + \rho_t & \ge d_t, & t = 1, 2, \ldots, N \\
(v_t^T) & & \omega_t & \le M_t, & t = 1, 2, \ldots, N \\
 & & \pi_t, \omega_t, \rho_t \ge 0 & for & t = 1, 2, \ldots, N
\end{array}
$$

where the subscripts t and s indicate the part number, N is the total number of parts, x_t, y_t and v_t are the vectors of n_t, r_t, and q_t variables for part t, for t=1, 2, ..., N. The dual variable vectors for the constraints of part t of P are denoted by row vectors π_t, ω_t and ρ_t which are the vectors of m_t, q_t, and r_t variables for part t. L_{ts} is a $q_t \times r_s$ matrix, A_t is an $m_t \times n_s$ matrix, B_t is a $q_t \times n_s$ matrix, D_t is an $m_t \times r_s$ matrix, and c_t, d_t, $M_t > 0$, b_t, f_t, and $u_t > 0$ are vectors of suitable dimensions for t=1, 2, ..., N and s=1, 2, ..., N. The general structure of the multi-part model is also shown in Table I.

The same assumption as in the two-part case is made in order to simplify the algorithm and to guarantee convergence:

Assumption: For each part, t=1, 2, ..., N, the set of nonlinking constraints, together with upper bound constraints and nonnegativity constraints, define a bounded feasible region for the x_t, y_t vectors.

Table 4.1 General structure of multi-part model

Dual Vector	Primal Vector	x_1	y_1	v_1	x_2	y_2	v_2	....	x_N	y_N	v_N	
	Dimension	n_1	r_1	q_1	n_2	r_2	q_2	....	n_N	r_N	q_N	RHS
π_1	m_1	A_1	D_1									b_1
ω_1	q_1	B_1	L_{11}	-1		L_{12}		....		L_{1N}		f_1
ρ_1	r_1		1									u_1
π_2	m_2				A_2	D_2						b_2
ω_2	q_2		L_{21}		B_2	L_{22}	-1	....		L_{2N}		f_2
ρ_2	r_2						1					u_2
....	....	.	.	.	.	.	.	.	.	.	.	.
π_N	m_N								A_N	D_N		b_N
ω_N	q_N		L_{N1}			L_{N2}		....	B_N	L_{NN}	-1	f_N
ρ_N	r_N									1		u_N
	Objective	c_1	d_1	$-M_1$	c_2	d_2	$-M_2$	...	c_N	d_N	$-M_N$	

4.2 The Structure of Subproblems for the First Method

4.2.1 The Bifurcation Process

We first divide the original multi-part problem into an aggregated lower bound subproblem, denoted by P_L, and an aggregated upper bound subproblem, denoted by P_U, by applying the basic idea of parallel decomposition of two-part models to multi-part models. Again, we divide the aggregated lower bound subproblem into its aggregated lower bound subproblem, denoted by P_{LL}, and its aggregated upper bound subproblem, denoted by P_{LU}, and the aggregated upper bound subproblem into its aggregated lower bound subproblem, denoted by P_{UL}, and its aggregated upper bound subproblem, denoted by P_{UU}. This bifurcation process continues until there are no subproblems left for further decomposition. Figure 4.1 shows this bifurcation process for N=9 with the number of parts at each level. Since the total number of parts is not a power of two in this example, the choice of the parts number at each level is arbitrary. However, to keep the number of bifurcation levels as small as possible, each bifurcation can be done such that the number of parts in the two subproblems are equal or different by one, depending on whether the total number of parts is even or odd.

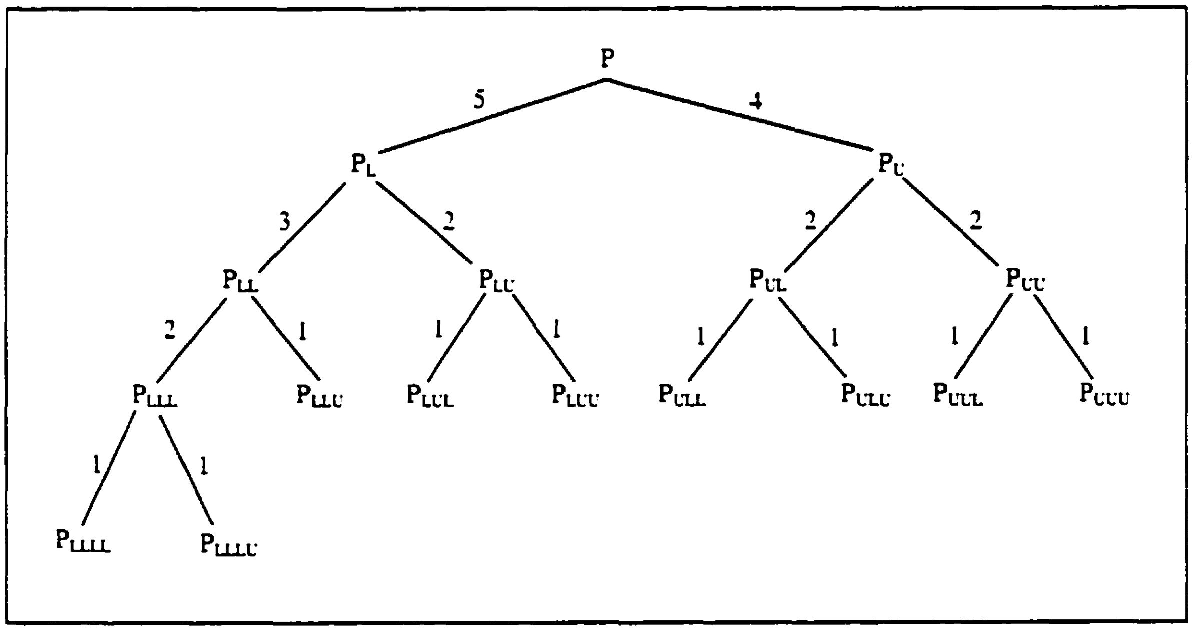

Figure 4.1 The bifurcation process for N=9 (9-parts)

At each level, a new linking primal variable (λ vector) is included in the lower bound type subproblem and a new linking dual variable (μ vector) is included in the upper bound type subproblem. The lower bound subproblem with all new linking primal variables, called the lowest bound subproblem ($P_{LL...L}$), consists of that part's variables, plus fractional weighting variables for proposals from other parts, and artificial variables, and it includes linking constraints for all parts. The upper bound subproblem with all new linking dual variables, called the uppermost subproblem ($P_{UU...U}$), has that part's variables, all parts' linking variables, and extra constraints (cuts) constructed with dual variable proposals from all other parts. Other subproblems except the two subproblems mentioned above include that part's variables, some fractional weighting variables for primal proposals from some other parts and artificial variables, and they also include some linking variables in some linking constraints for some other parts, and extra constraints (cuts) constructed with dual variable proposals from some other parts.

Each subproblem is solved simultaneously in each processor and exchanges primal and dual information with the immediate neighbor subproblem, which is bifurcated from the same aggregated subproblem, until each pair of subproblems reach an optimal solution of each aggregated subproblem. Then, the algorithm checks the pairs of aggregated subproblems at the previous level in the bifurcation tree. If they converge to the same value, the algorithm checks again the optimality of the pairs of the next level and if not, the algorithm performs information exchange, updates and simultaneously solves all subproblems descended from that bifurcation. This hierarchical process iterates until the root level subproblems, P_L and P_U, converge to an optimal solution of P, i.e. the optimal value of the lowest bound subproblem ($P_{LL...L}$) gets close enough to the optimal value of the uppermost subproblem ($P_{UU...U}$) to satisfy a prescribed tolerance. It turns out that it is possible to stop short of optimality for all pairs beyond the root bifurcation, yet still have convergence of the whole process to the optimal value of P. This is explained further below.

Since it is very difficult and long to state the general *N* part decomposition method in algebraic terms in detail, we will focus on a 4-part case as shown in Figure 4.2. The extension to any number of parts *N* can be straightforward from the demonstration of this 4-part case.

P_{UU} gives an upper bound to the original problem because P_U in the first level gives an upper bound to the original problem, and the upper bound subproblem of P_U, which is P_{UU}, provides an upper bound to P_U. With the same reasoning, P_{LL} provides a lower bound to the original problem. The upper bounds provided by P_{UU} are nonincreasing, as the iterations proceed, and the lower bounds from P_{LL} are nondecreasing. The algorithm proceeds through iterations of parallel solution of P_{LL} and P_{LU}, by exchanges of primal and dual proposals, converging towards the optimal solution of P_L. Simultaneously, P_{UL} and P_{UU} are solved iteratively, in parallel, converging towards the optimal solution of P_U.

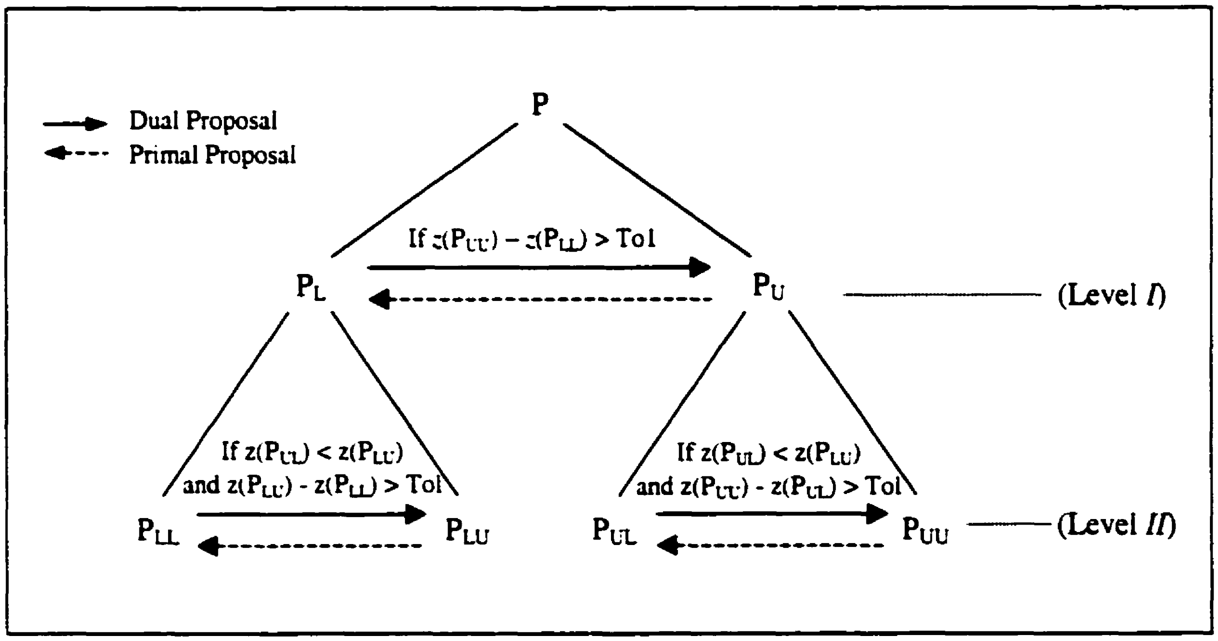

Figure 4.2 4-part decomposition principle and information flow.
("Tol" is the predetermined small tolerance for judging convergence)

An algorithm could be defined to exchange primal and dual proposals at the first level, i.e. between P_L and P_U, only when both level *II* pairs of subproblems have converged to the optimal values of P_L and P_U. It should be clear, based on the convergence of the two-part case, that convergence could be proved for such an algorithm. However, we have implemented a different scheme which requires fewer iterations at the second level before information exchange at the first level. A careful examination of the convergence proof for two part models reveals that convergence is assured if the two parts pass feasible (not necessarily optimal) solutions such that $z_1^k \leq z_2^k$. Applying this observation to P_L and P_U in the implemented algorithm, we get a dual feasible solution to P_L from P_{LU}, a primal feasible solution from P_{UL}, and we wait until $z(P_{LU}) \leq z(P_{UL})$ before exchanging proposals between P_L and P_U. Figure 4.2 shows the criteria for the iterations to *continue* with primal and dual exchanges, at each level: second level exchanges between a pair of subproblems continue if $z(P_{UL}) < z(P_{LU})$ and the pair has not converged to within a predetermined tolerance of the optimal value of its first level problem; first level exchanges continue as long as the upper bound, $z(P_{UU})$, has not converged to the lower bound, $z(P_{LL})$.

The parallel decomposition method would be balanced among the processors if the number of parts in the original problem is a power of 2. In other cases, a tree like in Figure 4.1 would have some end nodes at different levels than other end nodes. This could lead to much idle time for the processors that solve the subproblems at higher level end nodes. However, one can consider a balancing strategy that assigns a large or difficult subproblem to a higher end level node in order to decrease idle time of the processors. Another balancing scheme could have two or more subproblems of higher level end nodes assigned to one processor, to be solved serially.

In contrast, in the Lan-Fuller method, for the 9-stage case as shown in Figure 4.3., the original problem (P) is first divided into a subproblem of stage 1 (P_1) and an aggregated subproblem of stage 2 to 9 ($P_{2\text{-}9}$), then the aggregated subproblem is further divided into a subproblem of stage 2 (P_2) and an aggregated subproblem of stage 3 to 9 ($P_{3\text{-}9}$). This nested partitioning process continues until the aggregated subproblem of stage 8 and 9 ($P_{8\text{-}9}$) is divided into a subproblem of stage 8 (P_8) and a subproblem of stage 9 (P_9). The nested partitioning process of Lan-Fuller has depth of 8, generalized as (N-1) depth, while the hierarchical partitioning process of the new decomposition algorithm has depth of 4, generalized as floor($\log_2 N$) or 1+floor($\log_2 N$) (where floor(•) is the function that returns the largest integer less than or equal to the argument), assuming that each bifurcation produces subproblems whose numbers of parts are equal or different by one.

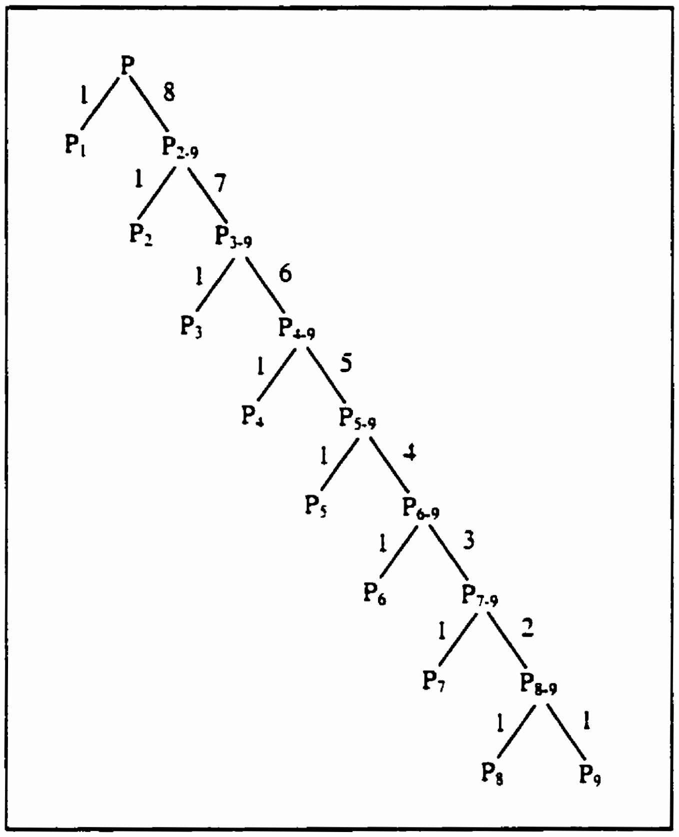

Figure 4.3 9-stage decomposition principle in Lan-Fuller method.

4.2.2 Precise Description of the Subproblems

A precise statement of the subproblems is given below (new notation is defined after the statements of the subproblems). Note that there are 3 different iteration counters involved for 4-part decomposition, one for the first level counter, *k*, and two for the second level counters, *i* and *j*, because the pairs of subproblems can have different numbers of iterations to converge towards the optimal solutions of P_U and P_L respectively.

$SP^{k,i}_{LL}$ (also called $SP^{k,i}_{1}$ because it is assigned to processor number 1) is constructed by restricting the primal variables of the aggregated upper bound subproblem (parts 3 and 4) into convex combinations of known solutions of the previous k-1 iterations and by further restricting the primal variables of part 2 into convex combinations of known solutions of the previous i-1 iterations.

$\mathrm{SP}_{LL}^{k,i}$ $(\mathrm{SP}_1^{k,i})$:

$$\max_{\substack{x_1,\, y_1,\, \text{all } v_t,\\ \lambda_{1.I}^{k-1},\, \lambda_{1.II}^{i-1}}} \quad c_1 x_1 + d_1 y_1 - M_1 v_1 + (c_2 X_2^{i-1} + d_2 Y_2^{i-1})\,\lambda_{1.II}^{i-1} + \sum_{t=3}^{4} (c_t X_t^{k-1} + d_t Y_t^{k-1})\,\lambda_{1.I}^{k-1} - \sum_{t=2}^{4} M_t v_t$$

$$\begin{array}{llllll}
s.t. & A_1 x_1 + D_1 y_1 & & & & \le b_1 \\
 & B_1 x_1 + L_{11} y_1 - v_1 & + \; L_{12} Y_2^{i-1} \lambda_{1.II}^{i-1} & + \; (L_{13} Y_3^{k-1} + L_{14} Y_4^{k-1}) \lambda_{1.I}^{k-1} & & \le f_1 \\
 & y_1 & & & & \le u_1 \\
 & L_{21} y_1 & + \; (B_2 X_2^{i-1} + L_{22} Y_2^{i-1}) \lambda_{1.II}^{i-1} & + \; (L_{23} Y_3^{k-1} + L_{24} Y_4^{k-1}) \lambda_{1.I}^{k-1} & - \; v_2 & \le f_2 \\
 & L_{31} y_1 & + \; L_{32} Y_2^{i-1} \lambda_{1.II}^{i-1} & + \; (B_3 X_3^{k-1} + L_{33} Y_3^{k-1} + L_{34} Y_4^{k-1}) \lambda_{1.I}^{k-1} & - \; v_3 & \le f_3 \\
 & L_{41} y_1 & + \; L_{42} Y_2^{i-1} \lambda_{1.II}^{i-1} & + \; (L_{43} Y_3^{k-1} + B_4 X_4^{k-1} + L_{44} Y_4^{k-1}) \lambda_{1.I}^{k-1} & - \; v_4 & \le f_4 \\
 & & e^{i-1} \lambda_{1.II}^{i-1} = 1 & & & \\
 & & & e^{k-1} \lambda_{1.I}^{k-1} = 1 & & \\
 & & x_1, y_1, v_1, v_2, v_3, v_4, \lambda_{1.II}^{i-1}, \lambda_{1.I}^{k-1} \ge 0 & & &
\end{array}$$

$\mathrm{SD}_{LL}^{k,i}$ $(\mathrm{SD}_1^{k,i})$:

$$\min_{\substack{\pi_1,\, \text{all } \omega_t,\\ \rho_1,\, \phi_{1.I},\, \phi_{1.II}}} \quad \pi_1 b_1 + \omega_1 f_1 + \rho_1 u_1 + \omega_2 f_2 + \omega_3 f_3 + \omega_4 f_4 + \phi_{1.II} + \phi_{1.I}$$

$$\begin{array}{lll}
s.t. & \pi_1 A_1 + \omega_1 B_1 & \ge c_1 \\
 & \pi_1 D_1 + \omega_1 L_{11} + \rho_1 + \omega_2 L_{21} + \omega_3 L_{31} + \omega_4 L_{41} & \ge d_1 \\
 & \omega_1 L_{12} Y_2^{i-1} + \omega_2 (B_2 X_2^{i-1} + L_{22} Y_2^{i-1}) + \omega_3 L_{32} Y_2^{i-1} + \omega_4 L_{42} Y_2^{i-1} + e^{i-1} \phi_{1.II} & \ge c_2 X_2^{i-1} + d_2 Y_2^{i-1} \\
 & \omega_1 (L_{13} Y_3^{k-1} + L_{14} Y_4^{k-1}) + \omega_2 (L_{23} Y_3^{k-1} + L_{24} Y_4^{k-1}) + \omega_3 (B_3 X_3^{k-1} + L_{33} Y_3^{k-1} + L_{34} Y_4^{k-1}) & \\
 & \qquad + \omega_4 (L_{43} Y_3^{k-1} + B_4 X_4^{k-1} + L_{44} Y_4^{k-1}) + e^{k-1} \phi_{1.I} & \ge \sum_{t=3}^{4} c_t X_t^{k-1} + d_t Y_t^{k-1} \\
 & \omega_t & \le M_t \quad \text{for } t = 1,2,3,4 \\
 & \pi_1, \; \rho_1, \; \omega_1, \; \omega_2, \; \omega_3, \; \omega_4 \ge 0 &
\end{array}$$

$\mathrm{SD}_{LU}^{k,i}$ (or $\mathrm{SD}_2^{k,i}$ because it is assigned to processor 2) is constructed by restricting the primal variables of the aggregated upper bound subproblem (parts 3 and 4) into convex combinations of known solutions of the previous *k*-1 iterations and by converting it into the dual problem and further restricting the dual variables of part 1 into convex combinations of known solutions of the previous *i*-1 iterations.

$\mathrm{SD}_{LU}^{k,i}$ $(\mathrm{SD}_{2}^{k,i})$:

$$
\begin{array}{lll}
\min\limits_{\substack{\mu_{2,II}^{i-1},\pi_2,\omega_2,\rho_2\\ \omega_3,\omega_4,\rho_1,\Phi_{2,I}}} & \mu_{2,II}^{i-1}(\Pi_1^{i-1}b_1+\Omega_1^{i-1}f_1)+\pi_2 b_2+\omega_2 f_2+\rho_2 u_2+\omega_3 f_3+\omega_4 f_4+\rho_1 u_1+\Phi_{2,I} & \\
s.t. & \mu_{2,II}^{i-1}(\Pi_1^{i-1}D_1+\Omega_1^{i-1}L_{11}) + \omega_2 L_{21} + \omega_3 L_{31}+\omega_4 L_{41}+\rho_1 & \ge d_1 \\
 & \pi_2 A_2+\omega_2 B_2 & \ge c_2 \\
 & \mu_{2,II}^{i-1}\Omega_1^{i-1}L_{12} + \pi_2 D_2+\omega_2 L_{22}+\rho_2 + \omega_3 L_{32}+\omega_4 L_{42} & \ge d_2 \\
 & \mu_{2,II}^{i-1}\Omega_1^{i-1}(L_{13}Y_3^{k-1}+L_{14}Y_4^{k-1})+\omega_2(L_{23}Y_3^{k-1}+L_{24}Y_4^{k-1})+\omega_3(B_3X_3^{k-1}+L_{33}Y_3^{k-1}+L_{34}Y_4^{k-1}) & \\
 & \qquad \omega_4(L_{43}Y_3^{k-1}+B_4X_4^{k-1}+L_{44}Y_4^{k-1})+e^{k-1}\Phi_{2,I} & \ge \sum\limits_{t=3}^{4} c_t X_t^{k-1}+d_t Y_t^{k-1} \\
 & \omega_t & \le M_t \quad \text{for } t=2,3,4 \\
 & \mu_{2,II}^{i-1}e^{i-1}=1 & \\
 & \mu_{2,II}^{i-1},\pi_2,\omega_2,\omega_3,\omega_4,\rho_1,\rho_2\ge 0 &
\end{array}
$$

$SP_{LL}^{k,i}$ $(SP_2^{k,i})$:

$$
\begin{array}{lll}
\max\limits_{\substack{y_1,x_2,y_2,v_2,\\ v_3,v_4,\theta_{2,II},\lambda_{2,I}^{k-1}}} & d_1y_1+c_2x_2+d_2y_2-M_2v_2+\sum\limits_{t=3}^{4}(c_tX_t^{k-1}+d_tY_t^{k-1})\lambda_{2,I}^{k-1}-M_3v_3-M_4v_4+\theta_{2,II} & \\
s.t. & (\Pi_1^{i-1}D_1+\Omega_1^{i-1}L_{11})y_1+\Omega_1^{i-1}L_{12}y_2+\Omega_1^{i-1}(L_{13}Y_3^{k-1}+L_{14}Y_4^{k-1})\lambda_{2,I}^{k-1} + e^{i-1}\theta_{2,II} & \le \Pi_1^{i-1}b_1+\Omega_1^{i-1}f_1 \\
 & y_1 & \le u_1 \\
 & A_2x_2+D_2y_2 & \le b_2 \\
 & L_{21}y_1+B_2x_2+L_{22}y_2-v_2 + (L_{23}Y_3^{k-1}+L_{24}Y_4^{k-1})\lambda_{2,I}^{k-1} & \le f_2 \\
 & y_2 & \le u_2 \\
 & L_{31}y_1 + L_{32}y_2 + (B_3X_3^{k-1}+L_{33}Y_3^{k-1}+L_{34}Y_4^{k-1})\lambda_{2,I}^{k-1} - v_3 & \le f_3 \\
 & L_{41}y_1 + L_{42}y_2 + (L_{43}Y_3^{k-1}+B_4X_4^{k-1}+L_{44}Y_4^{k-1})\lambda_{2,I}^{k-1} - v_4 & \le f_4 \\
 & e^{k-1}\lambda_{2,I}^{k-1} = 1 & \\
 & y_1,x_2,v_2,v_3,v_4,\lambda_{2,I}^{k-1}\ge 0 &
\end{array}
$$

$\mathrm{SP}_{LL}^{k,i}$ (or $\mathrm{SP}_{2}^{k,i}$) is constructed by restricting the dual variables of the aggregated lower bound subproblem (parts 1 and 2) into convex combinations of known solutions of the previous k-1 iterations and by converting it to primal form, and further restricting the primal variables of part 4 into convex combinations of known solutions of previous j-1 iterations.

$\mathrm{SP}_{UL}^{k,j}$ $(\mathrm{SP}_3^{k,j})$:

$$\max_{\substack{y_1, y_2, x_3, y_3, \\ v_3, v_4, \lambda_{3,II}^{j-1}, \theta_{3,I}}} \quad d_1 y_1 + d_2 y_2 + c_3 x_3 + d_3 y_3 - M_3 v_3 + (c_4 X_4^{j-1} + d_4 y_4^{j-1}) \lambda_{3,II}^{j-1} - M_4 v_4 + \theta_{3,I}$$

$$\begin{aligned}
s.t.\ & (\Pi_1^{k-1} D_1 + \Omega_1^{k-1} L_{11} + \Omega_2^{k-1} L_{21})\, y_1 + (\Omega_1^{k-1} L_{12} + \Pi_2^{k-1} D_2 + \Omega_2^{k-1} L_{22})\, y_2 + (\Omega_1^{k-1} L_{13} + \Omega_2^{k-1} L_{23})\, y_3 \\
& \qquad + (\Omega_1^{k-1} L_{14} + \Omega_2^{k-1} L_{24}) Y_4^{j-1} \lambda_{3,II}^{j-1} + e^{k-1} \theta_{3,I} \le \sum_{t=1}^{2} \Pi_t^{k-1} b_t + \Omega_t^{k-1} f_t \\
& y_1 \le u_1 \\
& y_2 \le u_2 \\
& A_3 x_3 + D_3 y_3 \le b_3 \\
& L_{31} y_1 + L_{32} y_2 + B_3 x_3 + L_{33} y_3 - v_3 + L_{34} Y_4^{j-1} \lambda_{3,II}^{j-1} \le f_3 \\
& y_3 \le u_3 \\
& L_{41} y_1 + L_{42} y_2 + L_{43} y_3 + (B_4 X_4^{j-1} + L_{44} Y_4^{j-1}) \lambda_{3,II}^{j-1} - v_4 \le f_4 \\
& e^{j-1} \lambda_{3,II}^{j-1} = 1 \\
& y_1, y_2, x_3, y_3, v_3, v_4, \lambda_{3,II}^{k-1} \ge 0
\end{aligned}$$

$\mathrm{SD}_{UL}^{k,j}$ $(\mathrm{SD}_3^{k,j})$:

$$\min_{\substack{\mu_{3,I}^{k-1}, \pi_3, \omega_3, \rho_3, \\ \omega_4, \rho_2, \rho_1, \Phi_{3,II}}} \quad \sum_{t=1}^{2} \mu_{3,I}^{k-1} (\Pi_t^{k-1} b_t + \Omega_t^{k-1} f_t) + \pi_3 b_3 + \omega_3 f_3 + \rho_3 u_3 + \omega_4 f_4 + \sum_{t=1}^{2} \rho_t u_t + \Phi_{3,II}$$

$$\begin{aligned}
s.t.\ & \mu_{3,I}^{k-1} (\Pi_1^{k-1} D_1 + \Omega_1^{k-1} L_{11} + \Omega_2^{k-1} L_{21}) + \omega_3 L_{31} + \omega_4 L_{41} + \rho_1 \ge d_1 \\
& \mu_{3,I}^{k-1} (\Omega_1^{k-1} L_{12} + \Pi_2^{k-1} D_2 + \Omega_2^{k-1} L_{22}) + \omega_3 L_{32} + \omega_4 L_{42} + \rho_2 \ge d_2 \\
& \pi_3 A_3 + \omega_3 B_3 \ge c_3 \\
& \mu_{3,I}^{k-1} (\Omega_1^{k-1} L_{13} + \Omega_2^{k-1} L_{23}) + \pi_3 D_3 + \omega_3 L_{33} + \rho_3 + \omega_4 L_{43} \ge d_3 \\
& \mu_{3,I}^{k-1} (\Omega_1^{k-1} L_{14} + \Omega_2^{k-1} L_{24}) Y_4^{j-1} + \omega_3 L_{34} Y_4^{j-1} + \omega_4 (B_4 X_4^{j-1} + L_{44} Y_4^{j-1}) + e^{j-1} \Phi_{3,II} \ge c_4 X_4^{j-1} + d_4 Y_4^{j-1} \\
& \mu_{3,I}^{k-1} e^{k-1} = 1 \\
& \mu_{3,I}^{k-1}, \pi_3, \omega_3, \rho_3, \rho_1, \rho_2, \omega_4 \ge 0
\end{aligned}$$

$\mathrm{SP}_{UL}^{k,j}$ (or $\mathrm{SP}_4^{k,j}$) is constructed by restricting the dual variables of the aggregated lower bound subproblem (parts 1 and 2) into convex combinations of known solutions of the previous k-1 iterations and by further restricting the dual variables of part 3 into convex combinations of known solutions of the previous j-1 iterations, and expressing the result in primal form.

$\mathrm{SD}_{UU}^{k,j}$ $(\mathrm{SD}_{4}^{k,j})$:

$$\min_{\substack{\mu_{4,I}^{k},\mu_{4,II}^{j},\pi_4\\ \omega_4, all\,\rho_t}} \mu_{4,I}^{k-1}\sum_{t=1}^{2}(\Pi_t^{k-1}b_t+\Omega_t^{k-1}f_t)+\mu_{4,II}^{j-1}(\Pi_3^{j-1}b_3+\Omega_3^{j-1}f_3)+\pi_4 b_4+\omega_4 f_4+\sum_{t=1}^{4}\rho_t u_t$$

$$\begin{array}{llll}
s.t. & \mu_{4,I}^{k-1}(\Pi_1^{k-1}D_1+\Omega_1^{k-1}L_{11}+\Omega_2^{k-1}L_{21})+\mu_{4,II}^{j-1}\Omega_3^{j-1}L_{31} & +\ \omega_4 L_{41}+\rho_1 & \geq d_1\\
& \mu_{4,I}^{k-1}(\Omega_1^{k-1}L_{12}+\Pi_2^{k-1}D_2+\Omega_2^{k-1}L_{22})+\mu_{4,II}^{j-1}\Omega_3^{j-1}L_{32} & +\ \omega_4 L_{42}+\rho_2 & \geq d_2\\
& \mu_{4,I}^{k-1}(\Omega_1^{k-1}L_{13}+\Omega_2^{k-1}L_{23})+\mu_{4,II}^{j-1}(\Pi_3^{j-1}D_3+\Omega_3^{j-1}L_{33}) & +\ \omega_4 L_{43}+\rho_3 & \geq d_3\\
& & \pi_4 A_4+\omega_4 B_4 & \geq c_4\\
& \mu_{4,I}^{k-1}(\Omega_1^{k-1}L_{14}+\Omega_2^{k-1}L_{24})+\mu_{4,II}^{j-1}\Omega_3^{j-1}L_{34} & +\ \pi_4 D_4+\omega_4 L_{44}+\rho_4 & \geq d_4\\
& & \omega_4 & \leq M_4\\
& \mu_{4,II}^{k-1}e^{k-1}=1 & & \\
& \mu_{4,I}^{k-1}e^{k-1}=1 & & \\
& \mu_{4,II}^{j-1},\mu_{4,I}^{k-1},\pi_4,\omega_4,\rho_4,\rho_1,\rho_2,\rho_3\geq 0 & &
\end{array}$$

$\mathrm{SP}_{UU}^{k,j}$ $(\mathrm{SP}_{4}^{k,j})$

$$\max_{\substack{all\ y_t,x_4,v_4\\ \theta_{4,I},\theta_{4,II}}} d_1 y_1+d_2 y_2+d_3 y_3+c_4 x_4+d_4 y_4-M_4 v_4+\theta_{4,I}+\theta_{4,II}$$

$$\begin{array}{ll}
s.t.\ (\Pi_1^{k-1}D_1+\Omega_1^{k-1}L_{11}+\Omega_2^{k-1}L_{21})y_1+(\Omega_1^{k-1}L_{12}+\Pi_2^{k-1}D_2+\Omega_2^{k-1}L_{22})y_2+(\Omega_1^{k-1}L_{13}+\Omega_2^{k-1}L_{23})y_3 & \\
\qquad +(\Omega_1^{k-1}L_{14}+\Omega_2^{k-1}L_{24})y_4+e^{k-1}\theta_{4,I} & \leq\sum_{t=1}^{2}\Pi_t^{k-1}b_t+\Omega_t^{k-1}f_t\\
\Omega_3^{j-1}L_{31}y_1+\Omega_3^{j-1}L_{32}y_2+(\Pi_3^{j-1}D_3+\Omega_3^{j-1}L_{33})y_3+\Omega_3^{j-1}L_{34}y_4+e^{j-1}\theta_{4,II} & \leq\Pi_3^{j-1}b_3+\Omega_3^{j-1}f_3\\
A_4 x_4+D_4 y_4 & \leq b_4\\
L_{41}y_1+L_{42}y_2+L_{43}y_3+B_4 x_4+L_{44}y_4-v_4 & \leq f_4\\
y_t & \leq u_t\\
 & for\ t=1,2,,3,4\\
y_1,y_2,y_3,x_4,y_4,v_4\geq 0 &
\end{array}$$

Note that when k=1, i=1 and j=1, there are no corresponding λ or ϕ variables, and there are no corresponding cuts, nor are there corresponding μ or θ variables, because the algorithm begins by solving all the subproblems simultaneously, so there are no solutions yet available from other subproblems. Also, when i and j are reset to i=1 and j=1 at the start of each iteration k>1, no information is made available from second level iterations because the new proposals exchanged at the first level create different subproblems at the second level (but see the next subsection about partial use of old second level proposals).

The definitions of various symbols are given below.

e^{k-1} is a $1 \times (k-1)$ row vector with all entries equal to 1.

e^{i-1} and e^{j-1} are a $1 \times (i-1)$ row vector and a $1 \times (j-1)$ row vector, with all entries equal to 1, respectively.

$\theta_{t,I}$ and $\theta_{t,II}$ are scalar variables derived from level I and level II respectively in the subproblem of part t, t = 2, 3, 4.

$\phi_{t,I}$ and $\phi_{t,II}$ are scalar variables derived from level I and level II respectively in the subproblem of part t, t = 1, 2, 3.

$\lambda_{t,I}^{k-1}$ is a $(k-1) \times 1$ column vector variable whose components weight primal proposals from the aggregated upper bound (level I) subproblem in the subproblem of part t, t = 1, 2.

$\lambda_{1,II}^{i-1}$ and $\lambda_{3,II}^{j-1}$ are a $(i-1) \times 1$ column vector variable and a $(j-1) \times 1$ column vector variable, whose components weight primal proposals from the corresponding upper bound subproblem at level II in the subproblem of part 1 and 3, respectively.

$\mu_{t,I}^{k-1}$ is a $1 \times (k-1)$ row vector variable whose components weight dual proposals from the aggregated lower bound subproblem (level I) in the subproblem of part t, t = 3, 4.

$\mu_{2,II}^{i-1}$ and $\mu_{4,II}^{j-1}$ are a $1 \times (i-1)$ row vector variable and a $1 \times (j-1)$ row vector variable, whose components weight dual proposals from the corresponding lower bound subproblem at level *II* in the subproblem of part 2 and 4, respectively.

X_t^{k-1} is an $n_t \times (k-1)$ matrix and Y_t^{k-1} is a $r_t \times (k-1)$ matrix, i.e. $X_t^{k-1} = (x_t^1, x_t^2, \ldots, x_t^{k-1})$ and $Y_t^{k-1} = (y_t^1, y_t^2, \ldots, y_t^{k-1})$, coming from the first $(k-1)$ primal solutions of subproblem *t*, t=3, 4.

X_2^{i-1} is a $n_2 \times (i-1)$ matrix and Y_2^{i-1} is a $r_2 \times (i-1)$ matrix, i.e. $X_2^{i-1} = (x_2^1, x_2^2, \ldots, x_2^{i-1})$ and $Y_2^{i-1} = (y_2^1, y_2^2, \ldots, y_2^{i-1})$, coming from the first $(i-1)$ primal solutions of subproblem 2. At the start of each iteration *k*, *i* is reset to i=1 and, X_2^{i-1} and Y_2^{i-1} are reset to null matrices.

X_4^{j-1} is a $n_4 \times (j-1)$ matrix and Y_4^{j-1} is a $r_4 \times (j-1)$ matrix, i.e. $X_t^{k-1} = (x_t^1, x_t^2, \ldots, x_t^{j-1})$ and $Y_4^{j-1} = (y_4^1, y_4^2, \ldots, y_4^{j-1})$, coming from the first $(j-1)$ primal solutions of subproblem 4. At the start of each iteration *k*, *j* is reset to j=1 and, X_4^{j-1} and Y_4^{j-1} are reset to null matrices.

Π_t^{k-1} is a $(k-1) \times m_t$ matrix and Ω_t^{k-1} is a $(k-1) \times q_t$ matrix, i.e. $\Pi_t^{k-1} = (\pi_t^{1^T}, \pi_t^{2^T}, \ldots, \pi_t^{k-1^T})^T$ and $\Omega_t^{k-1} = (\omega_t^{1^T}, \omega_t^{2^T}, \ldots, \omega_t^{k-1^T})^T$, coming from the first $(k-1)$ dual solutions of subproblem *t*, t=1, 2.

Π_1^{i-1} is a $(i-1) \times m_1$ matrix and Ω_1^{i-1} is a $(i-1) \times q_1$ matrix, i.e. $\Pi_1^{i-1} = (\pi_1^{1^T}, \pi_1^{2^T}, \ldots, \pi_1^{i-1^T})^T$ and $\Omega_1^{i-1} = (\omega_1^{1^T}, \omega_1^{2^T}, \ldots, \omega_1^{i-1^T})^T$, coming from the first $(i-1)$ dual solutions of subproblem 1. At the start of each iteration *k*, *i* is reset to i=1 and, Π_1^{i-1} and Ω_1^{i-1} are reset to null matrices.

Π_3^{j-1} is a $(j-1) \times m_3$ matrix and Ω_3^{j-1} is a $(j-1) \times q_3$ matrix, i.e. $\Pi_3^{j-1} = (\pi_3^{1^T}, \pi_3^{2^T}, \ldots, \pi_3^{j-1^T})^T$ and $\Omega_3^{j-1} = (\omega_3^{1^T}, \omega_3^{2^T}, \ldots, \omega_3^{j-1^T})^T$, coming from the first $(j-1)$ dual solutions of subproblem 3. At the start of each iteration *k*, *j* is reset to j=1 and, Π_3^{j-1} and Ω_3^{j-1} are reset to null matrices.

$SP_1^{k,j}$ has the same structure as the part one subproblem in the parallel two part decomposition method in Chapter 3, except for two more parts, thus having one more proposal from the aggregated upper bound subproblem (parts 3 and 4) at each first level iteration. Also, $SP_4^{k,j}$ has the same structure as the part two subproblem in the parallel two part decomposition except for two more parts, thus having one more cut from the aggregated lower bound subproblem (parts 1 and 2) at each first level iteration. The subproblems $SP_2^{k,j}$ and $SP_3^{k,j}$ utilize both proposals and cuts.

For first level iterations, the coordination is made through broadcasting proposals and cuts. The proposals, which have the primal information of the previous subproblems, are broadcasted to the other two subproblems and the cuts, which have dual information, are broadcasted to the other two subproblems. During the second level iterations, the coordination is made through exchanging primal and dual proposals only between the two subproblems bifurcated from the same aggregated subproblem.

At the first iteration of the parallel method, there is no information flow among the subproblems since no information is available, while in contrast Lan's serial method begins with the first subproblem having no information from other subproblems, but all other subproblems are solved with proposals or cuts from other subproblems, even in the first iteration.

From the second iteration of the parallel method, information is exchanged between the aggregated subproblems of level *I*, then level *II* subproblems are formed and solved with new proposals or cuts exchanged between the immediate neighbor subproblems bifurcated from the same aggregated subproblem as shown in figure 4.2 in the previous section. Lan's serial method solves the first subproblem and sends primal information to the second subproblem, then it solves the second subproblem and sends primal information to the third subproblem. By doing the same procedure to the third subproblem, Lan's method solves the last subproblem and sends back dual information to the third subproblem and continues to solve and send dual information as in the following figure, thus shows its nature of serial computation.

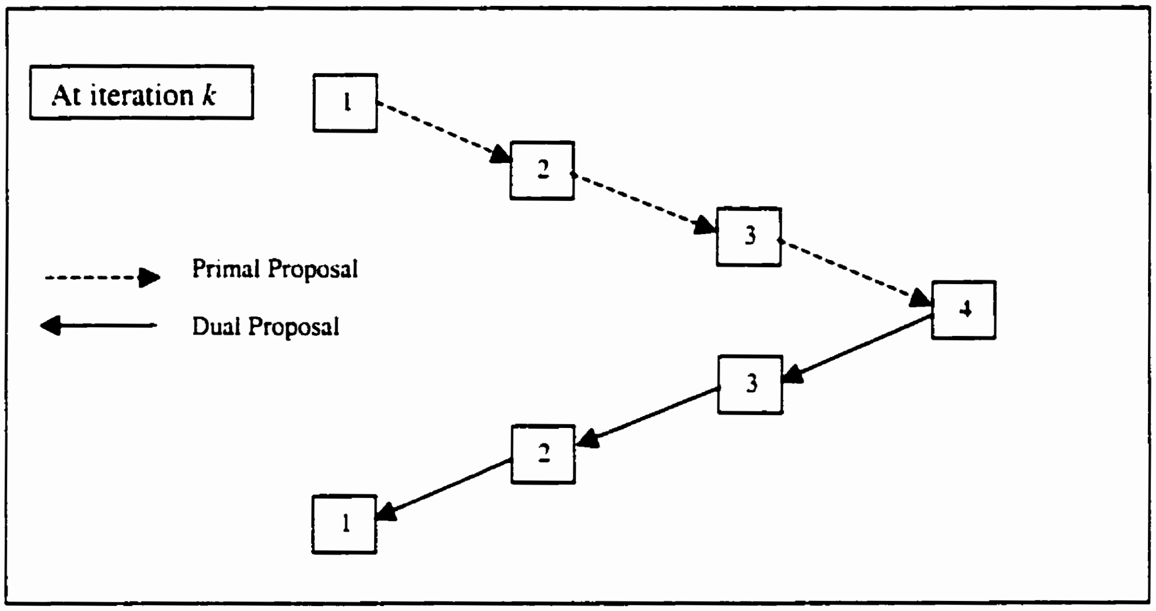

Figure 4.4 Information flow of 4-stage decomposition in Lan's method.

When the iteration counter $k>1$, SP_4^{k,j^*}, in which j^* denotes the second level iteration counter satisfying the stopping criteria of level *II*, i.e. $z_4^{k,j} - z_3^{k,j} \leq \varepsilon$ or $z_2^{k,i} \leq z_3^{k,j}$, gives upper bounds to the original problem and SP_1^{k,i^*}, in which i^* denotes the second level iteration counter satisfying the stopping criteria of level *II*, i.e. $z_2^{k,i} - z_1^{k,i} \leq \varepsilon$ or $z_2^{k,i} \leq z_3^{k,j}$, provides lower bounds for the original problem. The two subproblems perform the convergence test at each iteration *k*.

In contrast with the new algorithm, Dantzig's hierarchical decomposition method can be applied only to staircase structures and has traditional master and subproblems. Also, it is very difficult to apply parallel decomposition since it has to solve lower level master problems and subproblems serially as well as upper level master problem and subproblems.

4.2.3. A Strategy to use more Information from Second Level Iteration

Another strategy in utilizing more information can be defined in order to speed up the convergence of the algorithm. $SP_1^{k,i}$ and $SP_4^{k,j}$ can keep adding proposals and cuts coming from all the previous first and second level iterations because they are still feasible in the nonlinking constraints of $SP_2^{k,i}$ and $SP_3^{k,j}$ respectively, no matter what cuts and proposals are included in $SP_2^{k,i}$ and $SP_3^{k,j}$ respectively. Thus, they can produce nondecreasing lower bounds and nonincreasing upper bounds at every iteration of *k*, *i* and *j* (this is proven in section 4.4). However, when we tried to keep all available information from the first and second level iterations in $SP_2^{k,i}$ and $SP_3^{k,j}$, there were problems in tests.

In the next section, we define the algorithm for the case that all second level information is "forgotten" every time that the first level proceeds to another iteration; however the algorithm that is implemented in code uses the modified strategy defined above.

4.3 The Parallel Decomposition Algorithm for the First Method

In this section, the procedure of the parallel decomposition algorithm for multi-part problems is discussed. Various properties of this algorithm will be discussed in the next section.

The first step determines that the whole problem is feasible or not by detecting the infeasibility of subproblems, as proven in the next section. If any subproblem is infeasible, then the algorithm stops because the original problem is determined to be infeasible, and if each subproblem has its own feasible solutions, then the algorithm proceeds to the next steps because the original problem is feasible.

In Steps 1 and 2, the scalar $\varepsilon>0$ is defined by the user, and the algorithm solves each subproblem, exchanges the information between each pair of subproblems in the hierarchical manner and tries to reach the prescribed tolerance between the upper bound and the lower bound. $z_1^{k,i}$ and $z_2^{k,i}$ represent the objective values of $SP_1^{k,i}$ and $SP_2^{k,i}$ respectively, at first level iteration k and second level iteration i. $z_3^{k,j}$ and $z_4^{k,j}$ represent the objective values of $SP_3^{k,j}$ and $SP_4^{k,j}$ respectively, at first level iteration k and second level iteration j. Up_Opt and Low_Opt are set to 1 if P_U and P_L reach optimality; otherwise 0, respectively.

DO IN PARALLEL

Processor 1

Step 0. Set level *I* counter $k=1$, level *II* counter $i=1$, $\varepsilon>0$, and determine whether P is infeasible.

- solve $SP_1^{1,1}$: if it is infeasible, send a stop signal to all other subproblems and stop, P is infeasible;

- if a stop signal from any other subproblem is received, stop, P is infeasible;

- otherwise, record optimal dual solution π_1^1, ω_1^1

Step 1. Set k=k+1, i=0 and Up_Opt=0.

- set $\Pi_1^{k-1} = (\Pi_1^{k-2\,T}, \pi_1^{k-1\,T})^T$ and $\Omega_1^{k-1} = (\Omega_1^{k-2\,T}, \omega_1^{k-1T})^T$;

- send $(\pi_1^{k-1}D_1+\omega_1^{k-1}L_{11})$, $\omega_1^{k-1}L_{12}$, $\omega_1^{k-1}L_{13}$, $\omega_1^{k-1}L_{14}$ and $(\pi_1^{k-1}b_1+\omega_1^{k-1}f_1)$ to SP$_3$ and SP$_4$;

- receive $(c_tx_t^{k-1}+d_ty_t^{k-1})$, $L_{st}y_t^{k-1}$ and $(B_tx_t^{k-1}+L_{tt}y_t^{k-1})$ from SP$_t$ for t=3, 4 and s=1, ..., 4, $s \neq t$;

Step 1.1. Set i=i+1.

- send $(\pi_1^{i-1}D_1+\omega_1^{i-1}L_{11})$, $\omega_1^{i-1}L_{12}$, ω_1^{i-1} and $(\pi_1^{i-1}b_1+\omega_1^{i-1}f_1)$ to SP$_2$;

- receive $(c_2x_2^{i-1} + d_2y_2^{i-1})$, $L_{s2}y_2^{i-1}$ and $(B_2x_2^{i-1} + L_{22}y_2^{i-1})$ from SP$_2$ for s=1, 3, 4;

- update and solve $SP_1^{k,i}$; record optimal $z_1^{k,i}$, x_1^i, y_1^i, v_t^i, $\lambda_{1,I}^{k-1}$, $\lambda_{1,II}^{i-1}$ and π_1^i, ω_1^i;

- send $z_1^{k,i}$ to SP$_2$ and receive $z_2^{k,i}$ from SP$_2$;

- if Up_Opt = 0, receive Up_Opt and $z_3^{k,j}$ from SP$_3$;

Step 1.2. Test for level *II* convergence or exit.

- if $z_2^{k,i} - z_1^{k,i} \leq \varepsilon$ or if $z_2^{k,i} \leq z_3^{k,j}$, then set $z_1^{k,i*} = z_1^{k,i}$, $x_1^k = x_1^i$, $y_1^k = y_1^i$, $v_t^k = v_t^i$, $\pi_1^k = \pi_1^i$, $\omega_1^k = \omega_1^i$ and go to Step 2; otherwise, go to Step 1.1.

Step 2. Test for level *I* convergence.

- Send $z_1^{k,i*}$ to all other subproblems and receive $z_4^{k,j*}$ from SP$_4$;

- if $z_4^{k,j*} - z_1^{k,i*} \leq \varepsilon$, go to step 3; otherwise, go to step 1.

Step 3. Send $\lambda_{1,I}^{k-1}$ to SP$_3$ and SP$_4$, and $\lambda_{1,II}^{i-1}$ to SP$_2$, and receive $\mu_{4,I}^{k-1}$ from SP$_4$. Calculate the optimal primal and dual solutions for part 1 as $(x_1^k, y_1^k, v_1^k, v_2^k, v_3^k, v_4^k)$ and

$(\mu_{4,l}^{k-1}\Pi_l^{k-1}, \mu_{4,l}^{k-1}\Omega_l^{k-1})$.

Processor 2

Step 0. Set level *I* counter k=1, level *II* counter i=1, ε>0, and determine whether P is infeasible.

- solve $SP_2^{1,1}$; if it is infeasible, send a stop signal to all other subproblems and stop, P is infeasible;

- if a stop signal from any other subproblem is received, stop, P is infeasible;

- otherwise, record optimal dual solution π_2^1, ω_2^1

Step 1. Set k=k+1, i=0, Up_Opt=0 and Low_Opt=0;.

- set $\Pi_2^{k-1}=(\Pi_2^{k-2\,T}, \pi_2^{k-1\,T})^T$ and $\Omega_2^{k-1}=(\Omega_2^{k-2\,T}, \omega_2^{k-1T})^T$;

- send $(\pi_2^{k-1}D_2+\omega_2^{k-1}L_{22})$, $\omega_2^{k-1}L_{21}$, $\omega_1^{k-1}L_{23}$, $\omega_2^{k-1}L_{24}$ and $(\pi_2^{k-1}b_2+\omega_2^{k-1}f_2)$ to SP_3 and SP_4;

- receive $(c_tx_t^{k-1}+d_ty_t^{k-1})$, $L_{st}y_t^{k-1}$ and $(B_tx_t^{k-1}+L_{tt}y_t^{k-1})$ from SP_t for t=3, 4 and s=1, ... 4, $s\neq t$;

Step 1.1. Set i=i+1.

- send $(c_2x_2^{i-1} + d_2y_2^{i-1})$, $L_{s2}y_2^{i-1}$ and $(B_2x_2^{i-1} + L_{22}y_2^{i-1})$ from SP_2 for s=1, 3, 4;

- receive $(\pi_1^{i-1}D_1+\omega_1^{i-1}L_{11})$, $\omega_1^{i-1}L_{12}$, ω_1^{i-1} and $(\pi_1^{i-1}b_1+\omega_1^{i-1}f_1)$ from SP_1;

- update and solve $SP_2^{k,i}$; record optimal $z_2^{k,i}$, x_2^i, y_2^i, and π_2^i, ω_2^i;

- set $X_2^{i-1}=(X_2^{i-2}, x_2^{i-1})$ and $Y_2^{i-1}=(Y_2^{i-2}, y_2^{i-1})$;

- send $z_2^{k,i}$ to SP_1 and receive $z_1^{k,i}$ from SP_1;

- if $z_2^{k,i}$- $z_1^{k,i}\leq\varepsilon$, then set Low_Opt = 1; send Low_Opt and $z_2^{k,i}$ to SP_3;

- if Up_Opt = 0, receive Up_Opt and $z_3^{k,j}$ from SP$_3$;

Step 1.2. Test for level *II* convergence or exit.

- if $z_2^{k,i} - z_1^{k,i} \leq \varepsilon$ or if $z_2^{k,i} \leq z_3^{k,j}$, then set $\pi_2^k = \pi_2^i$, $\omega_2^k = \omega_2^i$ and go to Step 2; otherwise, go to Step 1.1.

Step 2. Test for level *I* convergence.

- receive $z_1^{k,i*}$ from SP$_1$ and receive $z_4^{k,j*}$ from SP$_4$;
- if $z_4^{k,j*} - z_1^{k,i*} \leq \varepsilon$, go to step 3; otherwise, go to step 1.

Step 3. Receive $\lambda_{1,II}^{i-1}$ from SP$_1$, and receive $\mu_{4,I}^{k-1}$ from SP$_4$. Calculate the optimal primal and dual solutions for part 2 as $(\lambda_{1,II}^{i-1} X_2^{i-1}, \lambda_{1,II}^{i-1} Y_2^{i-1})$ and $(\mu_{4,I}^{k-1} \Pi_2^{k-1}, \mu_{4,I}^{k-1} \Omega_2^{k-1})$.

Processor 3

Step 0. Set level *I* counter k=1, level *II* counter j=1, ε>0, and determine whether P is infeasible.

- solve $\mathrm{SP}_3^{1,1}$; if it is infeasible, send a stop signal to all other subproblems and stop, P is infeasible;
- if a stop signal from any other subproblem is received, stop, P is infeasible;
- otherwise, record optimal dual solution x_3^1, y_3^1

Step 1. Set k=k+1, j=0, Up_Opt=0 and Low_Opt=0;

- set $X_3^{k-1} = (X_3^{k-2}, x_3^{k-1})$ and $Y_3^{k-1} = (Y_3^{k-2}, y_3^{k-1})$;
- send $(c_3 x_3^{k-1} + d_3 y_3^{k-1})$, $L_{s3} y_3^{k-1}$ and $(B_3 x_3^{k-1} + L_{33} y_3^{k-1})$ for s=1, 2, 4, to SP$_1$ and SP$_2$;
- receive $(\pi_t^{k-1} D_t + \omega_t^{k-1} L_{tt})$, $\omega_t^{k-1} L_{ts}$ and $(\pi_t^{k-1} b_t + \omega_t^{k-1} f_t)$ from SP$_t$ for t=1, 2, s=1,..., 4, $s \neq t$;

Step 1.1. Set j=j+1.

- send $(\pi_3^{j-1}D_3+\omega_3^{j-1}L_{33})$, $\omega_3^{j-1}L_{3s}$ and $(\pi_3^{j-1}b_3+\omega_3^{j-1}f_3)$ for s=1, 2, 4, to SP$_4$;

- receive $(c_4x_4^{j-1}+d_4y_4^{j-1})$, $L_{s4}y_4^{j-1}$ and $(B_4x_4^{j-1}+L_{44}y_4^{j-1})$ from SP$_4$ for s=1, 2, 3;

- update and solve $\mathrm{SP}_3^{k,j}$; record optimal z_3^{kj}, x_3^j, y_3^j, and π_3^j, ω_3^j;

- set $\Pi_3^{j-1}=(\Pi_3^{j-2\,T}, \pi_3^{j-1\,T})^T$ and $\Omega_3^{j-1}=(\Omega_3^{j-2\,T}, \omega_3^{j-1T})^T$;

- send z_3^{kj} to SP$_4$ and receive z_4^{kj} from SP$_4$;

- if z_4^{kj}- $z_3^{kj} \leq \varepsilon$, then set Up_Opt = 1; send Up_Opt and z_3^{kj} to SP$_2$;

- if Low_Opt = 0, receive Low_Opt and $z_2^{k,i}$ from SP$_2$;

Step 1.2. Test for level *II* convergence or exit.

- if z_4^{kj}- $z_3^{kj} \leq \varepsilon$ or if $z_2^{k,i} \leq z_3^{kj}$, then set $x_3^k=x_3^j$, $y_3^k=y_3^j$ and go to Step 2; otherwise, go to Step 1.1.

Step 2. Test for level *I* convergence.

- receive $z_1^{k,i*}$ from SP$_1$ and z_4^{kj*} from SP$_4$;

- if z_4^{kj*} - $z_1^{k,i*} \leq \varepsilon$, go to step 3; otherwise, go to step 1.

Step 3. Receive $\lambda_{1.I}^{k-1}$ from SP$_1$, and receive $\mu_{4.II}^{j-1}$ from SP$_4$. Calculate the optimal primal and dual solutions for part 3 as $(\lambda_{1.I}^{k-1} X_3^{k-1}, \lambda_{1.I}^{k-1} Y_3^{k-1})$ and $(\mu_{4.II}^{j-1} \Pi_3^{j-1}, \mu_{4.II}^{j-1} \Omega_3^{j-1})$.

Processor 4

Step 0. Set level *I* counter k=1, level *II* counter j=1, ε>0, and determine whether P is infeasible.

- solve $\mathrm{SP}_4^{1,1}$; if it is infeasible, send a stop signal to all other subproblems and stop, P is infeasible;

- if a stop signal from any other subproblem is received, stop, P is infeasible;

- otherwise, record optimal dual solution x_4^1, y_4^1

Step 1. Set k=k+1, j=0 and Low_Opt=0;

- set $X_4^{k-1}=(X_4^{k-2}, x_4^{k-1})$ and $Y_4^{k-1}=(Y_4^{k-2}, y_4^{k-1})$;

- send $(c_4x_4^{k-1} + d_4y_4^{k-1})$, $L_{s4}y_4^{k-1}$ and $(B_4x_4^{k-1}+L_{44}y_4^{k-1})$ for s=1, 2, 3, to SP$_1$ and SP$_2$;

- receive $(\pi_t^{k-1}D_t+\omega_t^{k-1}L_{tt})$, $\omega_t^{k-1}L_{ts}$ and $(\pi_t^{k-1}b_t+\omega_t^{k-1}f_t)$ from SP$_t$ for t=1, 2, s=1,..., 4, $s \neq t$;

Step 1.1. Set j=j+1.

- send $(c_4x_4^{j-1}+d_4y_4^{j-1})$, $L_{34}y_4^{j-1}$, y_4^{j-1} and $(B_4x_4^{j-1}+L_{44}y_4^{j-1})$ to SP$_3$;

- receive $(\pi_3^{j-1}D_3+\omega_3^{j-1}L_{33})$, $\omega_3^{j-1}L_{3s}$ and $(\pi_3^{j-1}b_3+\omega_3^{j-1}f_3)$ for s=1, 2, 4, from SP$_3$;

- update and solve $SP_4^{k,j}$; record optimal $z_4^{k,j}$, x_4^j, y_4^j, and π_4^j, ω_4^j, ρ_t^j for t =1,..., 4;

- send $z_4^{k,j}$ to SP$_3$ and receive $z_3^{k,j}$ from SP$_3$;

- if Low_Opt = 0, receive Low_Opt and $z_2^{k,i}$ from SP$_2$;

Step 1.2. Test for level *II* convergence or exit.

- if $z_4^{k,j} - z_3^{k,j} \le \varepsilon$ or if $z_2^{k,i} \le z_3^{k,j}$, then set $z_4^{k,j*}=z_4^{k,j}$, $x_4^k=x_4^j$, $y_4^k=y_4^j$, $\pi_4^k=\pi_4^j$, $\omega_4^k=\omega_4^j$, ρ_t^k $=\rho_t^j$ for t =1,..., 4 and go to Step 2; otherwise, go to Step 1.1.

Step 2. Test for level *I* convergence.

- send $z_4^{k,j*}$ to all other subproblems and receive $z_1^{k,i*}$ from SP$_1$;

- if $z_4^{k,j*} - z_1^{k,i*} \le \varepsilon$, go to step 3; otherwise, go to step 1.

Step 3. Receive $\lambda_{1,l}^{k-1}$ from SP$_1$. Calculate the optimal primal and dual solutions for part 4 as $(\lambda_{1,l}^{k-1} X_4^{k-1}, \lambda_{1,l}^{k-1} Y_4^{k-1})$ and $(\pi_4^k, \omega_4^k, \rho_t^k)$ for t =1,..., 4.

END

The steps 1 to 3 of each processor solve the subproblems simultaneously and search for the optimum by exchanging the primal and dual solutions in the hierarchical manner. Since the feasibility of the original problem and subproblems are ensured by the step 0, each subproblem after step 0 always has a feasible solution, (this will be discussed in the next section). The algorithm terminates when the difference between $z_4^{k,j*}$ and $z_1^{k,i*}$ gets less than the predetermined convergence tolerance $\varepsilon>0$.

Note that when the algorithm converges within the given tolerance and stops, it may not give a basic feasible solution to the original problem, but slightly interior, due to the nature of convex combinations. However, the basic feasible solution could be recovered by developing a similar scheme used in purification [Kortanek and Zhu, 1988] or crossover facility of CPLEX barrier method, i.e., the optimal solution of the algorithm is adjusted by moving some primal variables to upper or lower bounds (to become nonbasic variables) and if necessary, the simplex method finds a basic optimal solution of the original problem in a small number of iterations since the solution fed into the simplex method is already feasible and close to an optimal solution of the original problem.

4.4 Properties of the Algorithm for the First Method

Several properties of the parallel decomposition algorithm are discussed in this section. The arguments are similar to those of the serial case of Lan and Fuller [1995b].

The first theorem verifies that the algorithm rules out the possibility of unboundedness of the problem P and of any of the primal subproblems $SP_1^{k,i}$, $SP_2^{k,i}$, $SP_3^{k,j}$ and $SP_4^{k,j}$.

Theorem 4.1 *Problem P and all subproblems* $SP_1^{k,i}$, $SP_2^{k,i}$, $SP_3^{k,j}$ and $SP_4^{k,j}$ *are bounded.*

Proof : The Assumption guarantees that the optimal value of P is bounded. Then, the boundedness of each subproblem is proven as follows: by the Assumption, the non-artificial variables x_t and y_t are bounded, and the artificial variables, v_t, cannot cause unboundedness because the artificial variables are nonnegative and have large negative objective coefficients. Also, the λ variables are bounded because of nonnegativity and the sum to one constraint. The θ variables are also bounded by an argument to similar to that in the proof of Theorem 3.2b in the previous chapter. Therefore, the optimal value of each subproblem is bounded.

Theorem 4.2 states that Step 0 of the algorithm in each processor accurately detects the feasibility of the whole problem P.

Theorem 4.2 *Problem P is infeasible if and only if the first step of the algorithm in each processor reports infeasibility of P.*

Proof : (The "if" part) If a subproblem is found to be infeasible at step 1 in any processor, then the nonlinking constraints and upper bounds for the subproblem are infeasible because the linking constraints can always be satisfied for some choice of the artificial variables. Infeasibility of the nonlinking constraint and upper bound constraint implies that P is infeasible.

(The "only if" part) If problem P is infeasible, then at least one part's set of nonlinking constraints and upper bound constraints is infeasible because the linking constraints can't be violated. This infeasibility will be detected at step 0.

The next result guarantees that in Step 1 of the algorithm, all subproblems have feasible solutions.

Theorem 4.3 *Once step 0 reports that P is feasible, all subsequent subproblems are feasible.*

Proof : In subproblems SP_2, SP_3 and SP_4, the cuts are added to the subproblems of previous iterations and this addition of cuts can't affect the feasibility of subproblems because the cuts can always be satisfied by adjusting the value of the free variable θ.

In the subproblems SP_1, SP_2 and SP_3, the primal proposals are added to the subproblems of previous iteration and this addition of primal proposals does not change the feasibility of subproblems because the λ variables appear with nonzero coefficients only in the linking constraints and these linking constraints are always satisfied by artificial variables.

The next theorem shows that the algorithm provides primal and dual feasible solutions for the original problem P when it proceeds to Step 1 and it justifies the calculations of primal and dual solutions.

Theorem 4.4 *For any $k>1$, $i>1$ and $j>1$, with λ weights from $SP_1^{k,i}$, the algorithm gives the following primal feasible solution to the original problem and with the dual μ weights of $SP_4^{k,J}$, if provides the following dual feasible solution for P:*

$$\pi_1^* = \mu_{4,J}^{k-1}\Pi_1^{k-1},\ \omega_1^* = \mu_{4,J}^{k-1}\Omega_1^{k-1}, \rho_1^* = \rho_1^k,\ \pi_2^* = \mu_{4,J}^{k-1}\Pi_2^{k-1},\ \omega_2^* = \mu_{4,J}^{k-1}\Omega_2^{k-1},\ \rho_2^* = \rho_2^k$$

$$\pi_3^* = \mu_{4,Jl}^{j-1}\Pi_3^{j-1}, \omega_3^* = \mu_{4,Jl}^{j-1}\Omega_3^{j-1}, \rho_3^* = \rho_3^k,\ \pi_4^* = \pi_4^k, \omega_4^* = \omega_4^k, \rho_4^* = \rho_4^k.$$

Proof : For $t = 2, \dots, 4$, each x_t^* and y_t^* is a convex combination of known solutions of x_t and y_t in the previous iterations, which satisfies the nonlinking constraints $A_t x_t + D_t y_t \le b_t$ and upper bound constraints $y_t \le u_t$ of part t. Since x_1^* and y_1^* together with x_t^* and y_t^* solves SP$_1$, all linking constraints in P are also satisfied. So, SP$_1$ gives a feasible solution to the original problem P. The proof of the dual part is similar. □

The following theorem states that, at each iteration $k>1$, the optimal values of SP$_4^{k,j*}$ and SP$_1^{k,i*}$ give nonincreasing upper bounds and nondecreasing lower bounds to the original problem P.

Theorem 4.5 *In the processor 1 and 4 with k>1, the optimal values of $SP_1^{k,i*}$ form a nondecreasing series of lower bounds on the optimal value of P and the optimal values of $SP_4^{k,j}$ form a nonincreasing series of upper bounds on the optimal value of P,* i.e. $z_1^{2,i*} \le z_1^{3,i*} \le \dots \le z_1^{k,i*} \le z^* \le z_4^{k,j*} \le \dots \le z_4^{3,j*} \le z_4^{2,j*}$

Proof : Since SP$_1^{k,i*}$ is a restriction of the whole problem P and the feasible regions of successive subproblems SP$_1^{k,i*}$ include that of previous subproblems at each iteration by inclusion of another positive λ variable, it gives $z_1^{2,i*} \le z_1^{3,i*} \le \dots \le z_1^{k,i*} \le z^*$.

Similarly, SD$_4^{k,j*}$ is a restriction of the dual of the whole problem P and it is loosened at each iteration by inclusion of another positive μ variable, so the feasible region of SD$_4^{k,j*}$ gets bigger at each iteration. It provides that $z^* \le z_4^{k,j*} \le \dots \le z_4^{3,j*} \le z_4^{2,j*}$, so proves the theorem. □

Corollary 4.1 *In the processor 1 and 4 with k>1 and i>1, j>1, the optimal values of $SP_1^{k,i}$*

form a nondecreasing series of lower bounds on the optimal value of P if all proposals are accumulated and the optimal values of $SP_4^{k,j}$ *form a nonincreasing series of upper bounds on the optimal value of P if all cuts are included,*

i.e. $z_1^{2,2} \leq z_1^{2,3} \leq \ldots \leq z_1^{2,i^*} \leq z_1^{3,2} \leq z_1^{3,3} \leq \ldots z_1^{3,i^*} \leq \ldots \leq z_1^{k,i^*} \leq z^*$

$$\leq z_4^{k,j^*} \leq \ldots \leq z_4^{3,j^*} \leq \ldots \leq z_4^{3,3} \leq z_4^{3,2} \leq z_4^{2,j^*} \leq \ldots \leq z_4^{2,3} \leq z_4^{2,2}$$

Proof : Since $SP_1^{k,i}$ is a restriction of SP_1^{k,i^*} and the feasible regions of successive subproblems $SP_1^{k,i}$ include that of previous subproblems at each iteration of k and i, by inclusion of another positive λ variables, it gives $z_1^{2,2} \leq z_1^{2,3} \leq \ldots \leq z_1^{2,i^*} \leq z_1^{3,2} \leq z_1^{3,3} \leq \ldots z_1^{3,i^*} \leq \ldots \leq z_1^{k,i^*} \leq z^*$.

Similarly, $SD_4^{k,j}$ is a restriction of the dual of SP_4^{k,j^*} and it is loosened at each iteration of k and j, by inclusion of another positive μ variables, so the feasible region of $SD_4^{k,j}$ gets bigger at each iteration. It provides that $z^* \leq z_4^{k,j^*} \leq \ldots \leq z_4^{3,j^*} \leq \ldots \leq z_4^{3,3} \leq z_4^{3,2} \leq z_4^{2,j^*} \leq \ldots \leq z_4^{2,3} \leq z_4^{2,2}$. □

4.5 A Heuristic Decomposition Algorithm – the Second Method

In this section, a heuristic parallel decomposition algorithm for multi-part linear programming problems is presented. The heuristic parallel algorithm divides the original multi-part problem into several small subproblems of either lower bound type or upper bound type from each part, by extending the basic algorithm of the two-part method without the hierarchical decomposition principle. The subproblems communicate with each other by sending and receiving primal and dual solutions, and work together to reach an optimal point during the iterations. The present approach gives simple subproblem structures and algorithm, however it does not give any guarantee for convergence; in tests, mentioned briefly in Chapter 5, this heuristic sometimes fails to converge.

4.5.1 The Structure of Subproblems for the Second Method

The heuristic algorithm divides the original multi-part problem into small lower bound subproblems and upper bound subproblems by extending the basic algorithm of parallel decomposition of two-part models to multi-part models; thus it has only one iteration counter. Each part has a primal form of either a lower bound subproblem or an upper bound subproblem. Each lower bound subproblem consists of that part's variables, plus fractional weighting variables for proposals from other parts and artificial variables, and it includes linking constraints for all parts. Each upper bound subproblem has that part's variables, all parts's linking variables, and extra constraints (cuts) constructed with dual variable proposals from all other parts. Note that to proceed with the algorithm, it should include at least one lower bound subproblem and at least one upper bound subproblem.

The algorithm proceeds to solve all the subproblems simultaneously and broadcasts primal information (proposals) to the lower bound subproblems and dual information (cuts) to the upper bound subproblems. After all the lower bound subproblems receive primal information and all the upper bound subproblems receive dual information from other subproblems, the algorithm solves all the subproblems simultaneously again. This procedure continues until the algorithm satisfies some stopping criteria. Figure 4.3 shows the communication scheme between subproblems mentioned above, for the case of a five part LP having three lower bounding subproblems and two upper bounding subproblems. Note that there could be various assignments of subproblem type (lower, upper) to part number are possible. Figure 4.3 illustrates one possibility.

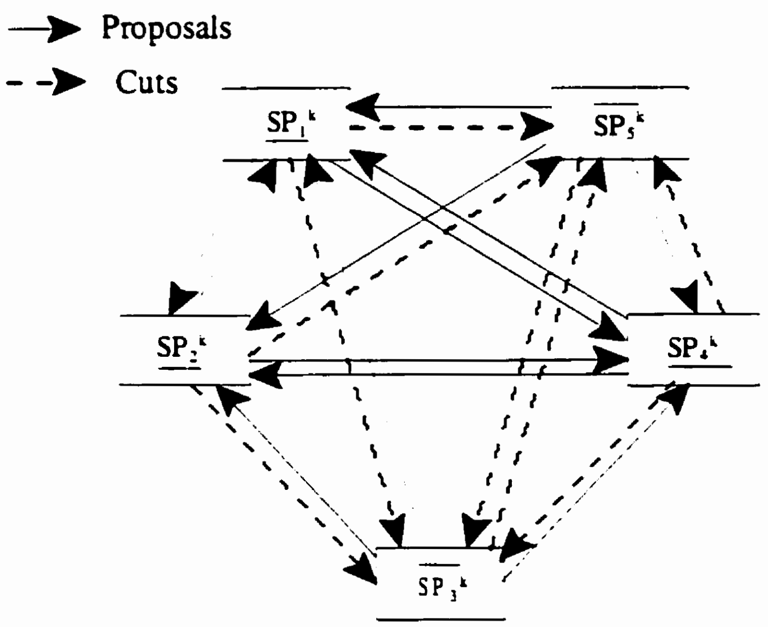

Figure 4.5 Information flows for the heuristic parallel decomposition algorithm

A precise statement of the heuristic algorithm for the general *N*-part case is given as follows. The primal and dual forms of the lower bound subproblem for part *t*, denoted by $\underline{\mathrm{SP}_t^k}$ and $\underline{\mathrm{SD}_t^k}$, are defined as (new notation is defined after the statements of the subproblems)

$$
\begin{array}{llll}
\underline{\mathrm{SP}_t^k}: & \max\limits_{x_t,\, y_t,\, v_t,\, \lambda_{it}^{k-1},\, v_i,\, i\neq t} & \underline{z_t^k} = c_t x_t + d_t y_t - M_t v_t + \sum\limits_{i=1, i\neq t}^{N} \{(c_i X_i^{k-1} + d_i Y_i^{k-1})\lambda_{it}^{k-1} - M_i v_i\} & \\
(\pi_t^T) & s.t. & A_t x_t + D_t y_t & \leq b_t \\
(\omega_t^T) & & B_t x_t + L_{tt} y_t - v_t + \sum\limits_{s=1, s\neq t}^{N} L_{ts} Y_s^{k-1} \lambda_{st}^{k-1} & \leq f_t \\
(\omega_i^T) & & L_{it} y_t + B_i X_i^{k-1} \lambda_{it}^{k-1} + \sum\limits_{s=1, s\neq t}^{N} L_{is} Y_s^{k-1} \lambda_{st}^{k-1} - v_i & \leq f_i,\ \mathit{for\ all}\ i \neq t \\
(\rho_t^T) & & y_t & \leq u_t \\
(\phi_i) & & e^{k-1} \lambda_{it}^{k-1} & = 1,\ \mathit{for\ all}\ i \neq t \\
 & & x_t,\, y_t,\, v_t,\, \lambda_{it}^{k-1},\, v_i \geq 0,\ \mathit{for}\ i = 1, 2, \ldots, N,\ i \neq t &
\end{array}
$$

$$
\begin{array}{llll}
\underline{\mathrm{SD}_t^k}: & \min\limits_{\pi_t,\, w_t,\, \rho_t,\, \omega_i,\, \phi_i,\, i\neq t} & \underline{z_t^k} = \pi_t b_t + \sum\limits_{i=1}^{N} \omega_i f_i + \rho_t u_t + \sum\limits_{i=1, i\neq t}^{N} \phi_i & \\
(x_t^T) & \text{s.t.} & \pi_t A_t + \omega_t B_t & \geq c_t \\
(y_t^T) & & \pi_t D_t + \sum\limits_{s=1}^{N} \omega_s L_{st} + \rho_t & \geq d_t \\
(\lambda_{it}^T) & & \omega_i B_i X_i^{k-1} + \sum\limits_{s=1} \omega_s L_{si} Y_i^{k-1} + \phi_i & \geq c_i X_i^{k-1} + d_i Y_i^{k-1},\ \mathit{for\ all}\ i \neq t \\
(v_i^T) & & \omega_i & \leq M_i,\ \mathit{for\ all}\ i \\
 & & \pi_t \geq 0,\ \rho_t \geq 0,\ \omega_i \geq 0\ \mathit{for}\ i = 1, 2, \ldots, N &
\end{array}
$$

and the primal and dual forms of the upper bound subproblems are

$$\begin{array}{llll}
\overline{\mathrm{SP}_t^k}: & \max\limits_{x_t,\, y_t,\, v_t,\, y_i,\, \theta_i,\, i\neq t} & \overline{z_t^k} = c_t x_t + \sum\limits_{i=1}^{N} d_i y_i + \sum\limits_{i=1, i\neq t}^{N} \theta_i - M_t v_t & \\
(\pi_t^T) & \text{s.t.} & A_t x_t + D_t y_t & \le b_t \\
(\omega_t^T) & & B_t x_t + \sum\limits_{s=1}^{N} L_{ts} y_s \quad - v_t & \le f_t \\
(\mu_{it}^{k-1^T}) & & \Pi_i^{k-1} D_i\, y_i + \sum\limits_{s=1}^{N} \Omega_i^{k-1} L_{is}\, y_s + \theta_i & \le \Pi_i^{k-1} b_i + \Omega_i^{k-1} f_i ,\ \textit{for all } i\neq t \\
(\rho_i^T) & & y_i & \le u_i ,\quad \textit{for all } i \\
 & & x_t , v_t , y_i \ge 0,\ i = 1,\dots, N &
\end{array}$$

$$\begin{array}{llll}
\overline{\mathrm{SD}_t^k}: & \min\limits_{\pi_t,\, \omega_t,\, \rho_t,\, \rho_i,\, \mu_{it}^{k-1},\, i\neq t} & \overline{z_t^k} = \pi_t b_t + \omega_t f_t + \rho_t + \sum\limits_{i=1, i\neq t}^{N} \{\mu_{it}^{k-1} (\Pi_i^{k-1} b_i + \Omega_i^{k-1} f_i) + \rho_i\} & \\
(x_t^T) & \text{s.t.} & \pi_t A_t + \omega_t B_t & \ge c_t \\
(y_t^T) & & \pi_t D_t + \omega_t L_{tt} + \rho_t + \sum\limits_{s=1, s\neq t}^{N} \mu_{st}^{k-1}\, \Omega_s^{k-1}\, L_{st} & \ge d_t \\
(y_i^T) & & \omega_t L_{ti} + \mu_{it}^{k-1}\, \Pi_i^{k-1} D_i + \sum\limits_{s=1, s\neq t} \mu_{st}^{k-1}\, \Omega_s^{k-1}\, L_{si} + \rho_i & \ge d_i ,\ \textit{for all } i\neq t \\
(v_t^T) & & \omega_t & \le M_t \\
(\theta_i) & & \mu_{it}^{k-1}\, e^{k-1^T} & = 1,\ \textit{for all } i\neq t \\
 & & \pi_t \ge 0,\ \omega_t \ge 0,\ \rho_i \ge 0,\ \mu_{it}^{k-1} \ge 0 \quad \textit{for } i = 1,\dots, N,\ \ i\neq t &
\end{array}$$

Note that when $k=1$, there are no θ or ϕ variables, and there are no cuts, nor are there λ or μ variables. The definitions of new symbols are given below.

θ_i and ϕ_i are scalar variables.

λ_{it}^{k-1} is a $(k-1)\times 1$ column vector variable whose components weight primal proposals from part i in the lower bounding subproblem of part t.

μ_{it}^{k-1} is a $1\times(k-1)$ row vector variable whose components weight dual proposals from part i in the upper bounding subproblem of part t.

X_t^{k-1} is a $n_t \times (k\text{-}1)$ matrix and Y_t^{k-1} is a $r_t \times (k\text{-}1)$ matrix, i.e. $X_t^{k-1} = (x_t^1, x_t^2, \ldots, x_t^{k-1})$ and

$Y_t^{k-1} = (y_t^1, y_t^2, \ldots, y_t^{k-1})$, coming from the first (k-1) primal solutions of SP_t.

Π_t^{k-1} is a $(k\text{-}1) \times m_t$ matrix and Ω_t^{k-1} is a $(k\text{-}1)\times q_t$ matrix, i.e. $\Pi_t^{k-1} = (\pi_t^{1^T}, \pi_t^{2^T}, \ldots, \pi_t^{k-1^T})^T$

and $\Omega_t^{k-1} = (\omega_t^{1^T}, \omega_t^{2^T}, \ldots, \omega_t^{k-1^T})^T$, coming from the first (k-1) dual solutions of SP_t.

The lower bound subproblems have the same structures as the part one subproblem (lower bound subproblem) in the parallel two part decomposition method in Chapter 3, except for more parts, thus having (N-2) more proposals from other subproblems at each iteration. The upper bound subproblems also have the same structures as the part two subproblem (upper bound subproblem) in the parallel two part decomposition except for more parts, thus having (N-2) more cuts from other subproblems at each iteration.

The heuristic parallel decomposition structure exhibits an equivalent position of subproblems, and all subproblems have access to information on all other subproblems and work together to optimize the whole problem.

The coordination is made through broadcasting proposals and cuts during the iteration. The proposals, which have the primal information of the previous subproblems, are broadcasted to all other lower bound subproblems and the cuts, which have dual information, are broadcasted to upper bound subproblems.

At the first iteration of the heuristic method, there is no information flow among the subproblems since no information is available, while Lan's serial method begins with the first subproblem having no information from other subproblems, but all other subproblems are solved with proposals or cuts from other subproblems, even in the first iteration.

When the iteration counter $k>1$, since all upper bound subproblems give upper bounds to the original problem and all lower bound subproblems provide lower bounds for the original problem, the best upper bound and lower bound can be chosen for the convergence test from the subproblems at each iteration. Although the subproblems generate the nonincreasing upper bounds and nondecreasing lower bounds, the algorithm can not be guaranteed to converge within a prescribed tolerance. The heuristic algorithm can get stuck and repeat the same solution without improvement after some number of iterations, so it is terminated with a feasible solution of the original problem when all the lower bound subprolems and all the upper bound subproblems have the same objective values respectively in three consecutive iterations.

4.5.2 The Heuristic Decomposition Algorithm for the Second Method

In this section, the procedure of the heuristic decomposition algorithm for multi-part problems is discussed. Various properties of this algorithm will be discussed in the next subsection.

Step 0 determines that the whole problem is feasible or not by detecting the infeasibility of subproblems, as proven in the next subsection. If any subproblem is infeasible, then the algorithm stops because the original problem is determined to be infeasible, and if each subproblem has its own feasible solutions, then the algorithm proceeds to Step 1 because the original problem is feasible.

In Step 1, the scalar $\varepsilon>0$ is defined by the user, and the algorithm solves each subproblem, exchanges the information among the subproblems and tries to reach the prescribed tolerance between the best upper bound issued by $\overline{\mathrm{SP}}_u^k$ and the best lower bound issued by $\underline{\mathrm{SP}}_l^k$.

Before the algorithm starts, the user selects either the lower bound subproblem, or the upper bound subproblem, from each part $t=1, \ldots, N$, while ensuring that at least one subproblem is the lower bound type, and at least one is the upper bound type.

```
DO IN PARALLEL
  Processor t for $t=1, \ldots, N$
    Step 0. Set $k=1$, $\varepsilon>0$, and determine whether P is infeasible.
        - solve $\mathrm{SP}_t^1$; if it is infeasible, send a stop signal to $\mathrm{SP}_s$ for all $s \neq t$ and stop, P is infeasible;
        - if a stop signal from $\mathrm{SP}_s$ for all $s \neq t$ is received, stop, P is infeasible;
        - otherwise, record optimal primal and dual solutions $x_t^1$, $y_t^1$ and $\pi_t^1$, $\omega_t^1$.
    Step 1. Set $k=k+1$, exchange information, modify $\mathrm{SP}_t$ and solve it.
        - set $X_t^{k-1} = (X_t^{k-2}, x_t^{k-1})$ and $Y_t^{k-1} = (Y_t^{k-2}, y_t^{k-1})$;
        - set $\Pi_t^{k-1} = (\Pi_t^{k-2\,T}, \pi_t^{k-1\,T})^T$ and $\Omega_t^{k-1} = (\Omega_t^{k-2\,T}, \omega_t^{k-1T})^T$;
        - send $(c_t x_t^{k-1} + d_t y_t^{k-1})$, $L_{it} y_t^{k-1}$ for all $i$, and $B_t x_t^{k-1}$ to all lower bound subproblems;
        - send $\pi_t^{k-1} D_t$, $\omega_t^{k-1} L_{ti}$ for all $i$, and $(\pi_t^{k-1} b_t + \omega_t^{k-1} f_t)$ to all upper bound subproblems;
        - if $t$ is a lower bound subproblem, receive $(c_s x_s^{k-1} + d_s y_s^{k-1})$, $L_{is} y_s^{k-1}$ and $B_s x_s^{k-1}$ from $\mathrm{SP}_s$
          for all $i$ and $s \neq t$;
```

- if t is an upper bound subproblem, receive $\pi_s^{k-1}D_s$, $\omega_s^{k-1}L_{si}$, and $(\pi_s^{k-1}b_s + \omega_s^{k-1}f_s)$ from SP_s for all i and $s \neq t$;

- solve SP_t^k; record optimal z_t^k, x_t^k, y_t^k, and π_t^k, ω_t^k; record optimal v_s^k, λ_{it}^{k-1} if t is a lower bound subproblem, and ρ_s^k, μ_{it}^{k-1} if t is an upper bound subproblem for all s and $i \neq t$;

- broadcast the optimal z_t^k to SP_s for all $s \neq t$; receive the optimal z_s^k from SP_s for all $s \neq t$.

Step 2. Test for convergence: select the best upper bound, $\overline{z}_u^k$ from $\overline{SP}_u^k$, and the best lower bound, $\underline{z}_l^k$ from $\underline{SP}_l^k$.

- if $(\overline{z}_u^k - \underline{z}_l^k) \leq \varepsilon$, then go to step 3;

- if $(\overline{z}_u^k - \underline{z}_l^k) > \varepsilon$ and $\underline{z}_l^k = \underline{z}_t^k$ for all lower bound subproblems and $\overline{z}_u^k = \overline{z}_t^k$ for all upper bound subproblems in three consecutive iterations, go to step 3;

- otherwise, go to step 1.

Step 3. Calculate the optimal (or feasible) primal and dual solutions, and terminate the algorithm.

- If $t=l$, broadcast λ_{sl}^{k-1} to SP_s for all $s \neq t$; if $t=u$, broadcast μ_{su}^{k-1} to SP_s for all $s \neq t$;

- receive λ_{tl}^{k-1} from SP_l for $t \neq l$ and μ_{tu}^{k-1} from SP_u for $t \neq l$;

- If $t=l$, calculate the optimal (or feasible) primal and dual solutions for part t as (x_t^k, y_t^k, v_s^k) for all s and $(\mu_{tu}^{k-1}\Pi_t^{k-1}, \mu_{tu}^{k-1}\Omega_t^{k-1})$;

- If $t=u$, calculate the optimal (or feasible) primal and dual solutions for part t as $(X_t^{k-1}\lambda_{tl}^{k-1}, Y_t^{k-1}\lambda_{tl}^{k-1})$ and $(\pi_t^k, \omega_t^k, \rho_s^k)$ for all s;

- otherwise, calculate the optimal (or feasible) primal and dual solutions for part t as

$$(X_t^{k-1}\lambda_{tt}^{k-1}, Y_t^{k-1}\lambda_{tt}^{k-1}) \text{ and } (\mu_{tu}^{k-1}\Pi_t^{k-1}, \mu_{tu}^{k-1}\Omega_t^{k-1})\text{; Stop.}$$

END

The heuristic algorithm solves the subproblems simultaneously and searches for the optimum by broadcasting the primal and dual solutions for $k > 1$. Since the feasibility of the original problem and subproblems are ensured by the step 1, each subproblem always has a feasible solution for P, (which will be discussed in the next subsection). The algorithm terminates with an optimal solution of the original problem when the difference between the best upper bound z_u^k and the best lower bound z_l^k of the whole problem gets less than the predetermined convergence tolerance $\varepsilon > 0$. However, in the cases of the same repeated objective values three times in a row in all upper bound and all lower bound subproblems respectively, the heuristic algorithm terminates with a feasible but not optimal solution of the original problem.

4.5.3 Properties of the Algorithm for the Second Method

Several properties of the parallel decomposition algorithm are discussed in this section. The arguments are similar to those in section 4.4.

Theorem 4.7 verifies that the algorithm rules out the possibility of unboundedness of the problem P and of any of the primal subproblems SP_t^k.

Theorem 4.7 *Problem P and all subproblems SP_t^k are bounded.*

Proof : The Assumption guarantees that the optimal value of P is bounded. Then, the boundedness of each subproblem is proven as follows: by the Assumption, the non-artificial variables x_t and y_t are bounded, and the artificial variables, v_t, cannot cause unboundedness because the artificial variables have nonnegativity and large negative objective coefficients. Also, the λ_{it}^{k-1} variables are bounded because of nonnegativity and the constraint $e^{k-1}\lambda_{it}^{k-1} = 1$. The θ_t variables in upper bound subproblems are also bounded by the similar argument to the proof of Theorem 3.2b in the previous chapter. Therefore, the optimal value of each subproblem is bounded.

Theorem 4.8 states that Step 0 of the algorithm accurately detects the feasibility of the whole problem P.

Theorem 4.8 *Problem P is infeasible if and only if Step 0 of the algorithm reports infeasibility of P.*

Proof : (The "if" part) If a subproblem is found to be infeasible at step 0, then at least one nonlinking constraint and upper bound constraint for a subproblem is infeasible because the linking constraints can always be satisfied for some choice of the artificial variables. Then, infeasibility of the nonlinking constraint and upper bound constraint implies that P is infeasible.

(The "only if" part) If problem P is infeasible, then at least one part's set of nonlinking constraints and upper bound constraints is infeasible because the linking constraints can't be violated. This infeasibility will be detected at step 0.

The next result guarantees that in Step 1 of the algorithm, all subproblems have feasible solutions.

Theorem 4.9 *Once the first step reports that P is feasible, all subsequent subproblems are feasible.*

Proof : In the upper bound subproblems, the cuts are added to the subproblems of previous iteration and this addition of cuts can't affect the feasibility of subproblems because the cuts can always be satisfied by adjusting the value of θ_t, which is a free variable.

In the lower bound subproblems, the primal proposals are added to the subproblems of previous iteration and this addition of primal proposals does not change the feasibility of subproblems because the λ variables appear with nonzero coefficients only in the linking constraints and these linking constraints are always satisfied by artificial variables.

The next theorem shows that the algorithm provides a primal feasible solution for the original problem P when it proceeds to step 1 and it justifies the calculations of primal and dual solutions in step 3.

Theorem 4.10 *For any t and k>1, the lower bound subproblem $\underline{SP}_t^k$ gives the folloing primal feasible solution to the original problem and the dual form of the upper bound subproblem $\overline{SP}_t^k$ provides the following dual feasible solution for P:*

$x_1^* = X_1^{k-1}\lambda_{1t}^{k-1}$, $y_1^* = Y_1^{k-1}\lambda_{1t}^{k-1}$, $x_2^* = X_2^{k-1}\lambda_{2t}^{k-1}$, $y_2^* = Y_2^{k-1}\lambda_{2t}^{k-1}$ ....

$x_t^* = x_t^k$, $y_t^* = y_t^k$ ....

$x_{N-1}^* = X_{N-1}^{k-1}\lambda_{N-1t}^{k-1}$, $y_{N-1}^* = Y_{N-1}^{k-1}\lambda_{N-1,t}^{k-1}$, $x_N^* = X_N^{k-1}\lambda_{Nt}^{k-1}$, $y_N^* = Y_N^{k-1}\lambda_{Nt}^{t-1}$

$\pi_1^* = \mu_{1t}^{k-1}\Pi_1^{k-1}$, $\omega_1^* = \mu_{1t}^{k-1}\Omega_1^{k-1}$, $\pi_2^* = \mu_{2t}^{k-1}\Pi_2^{k-1}$, $\omega_2^* = \mu_{2t}^{k-1}\Omega_2^{k-1}$ ....

$\pi_t^* = \pi_t^k$, , $\omega_t^* = \omega_t^k$ ....

$\pi_{N-1}^* = \mu_{N-1,t}^{k-1}\Pi_{N-1}^{k-1}$, $\omega_{N-1}^* = \mu_{N-1,t}^{k-1}\Omega_{N-1}^{k-1}$, $\pi_N^* = \mu_{Nt}^{k-1}\Pi_N^{k-1}$, $\omega_N^* = \mu_{Nt}^{k-1}$ Ω_N^{k-1}.

Proof : For $i=1, 2, \dots, N$ and $i \neq t$, each x_i^* and y_i^* is a convex combination of known solutions of x_i and y_i in the preveious iterations, which satisfies the nonlinking constraints $A_i x_i + D_i y_i \leq b_i$ and upper bound constraints $y_i \leq u_i$ of part i. Since x_t^* and y_t^* together with x_i^* and y_i^* solves $\underline{SP_t^k}$, all linking constraints in P are also satisfied. So, $\underline{SP_t^k}$ gives a feasible solution to the original problem P. The proof of dual part is similar.

The following theorem states that, at each iteration $k>1$, the optimal value of an upper bound subproblem and a lower bound subproblem give nonincreasing upper bounds and nondecreasing lower bounds to the original problem P.

Theorem 4.11 *In Step 1 for any lower bound subproblem (indexed by $t \neq s$), the optimal values of $\underline{SP_t^k}$, $\underline{z_t^k}$, form a nondecreasing series of lower bounds on the optimal value of P and for any upper bound subproblem (indexed by s), the optimal values of $\overline{SP_s^k}$, $\overline{z_s^k}$, form a nonincreasing* *series of upper bounds on the optimal value of P,*

i.e. $\underline{z}_t^2 \le \underline{z}_t^3 \le \dots \le \underline{z}_t^k \le z_t^* \le \overline{z}_t^k \le \dots \le \overline{z}_t^3 \le \overline{z}_t^2$.

Proof : Since $\underline{SP_t^k}$ is a restriction of the whole problem P and the feasible regions of successive subproblems $\underline{SP_t^k}$ include that of previous subproblems at each iteration by inclusion of another positive variables, λ_{it}^{k-1}, it gives $\underline{z}_t^2 \le \underline{z}_t^3 \le \dots \le \underline{z}_t^k \le z_t^*$.

Similarly, $\overline{SD_t^k}$ is a restriction of the dual of the whole problem P and it is loosened at each iteration by inclusion of another positive variable, μ_{it}^{k-1}, so the feasible region of $\overline{SD_t^k}$ gets bigger at each iteration. It provides that $z_t^* \le \overline{z}_t^k \le \dots \le \overline{z}_t^3 \le \overline{z}_t^2$, so proves the theorem.

Chapter 5 Preliminary Implementation and Results

This chapter reports on some implementation procedures and preliminary computational results of the parallel primal-dual decomposition algorithm for the multi-part problems through the use of GAMS [1992], the Regex Library [1992], PVM (Parallel Virtual Machine) 3.11 [1994] and CPLEX 6.0 [1997] on an IBM RS/6000 workstation and a cluster of four PCs (Personal Computers) running the Solaris operating system. Several multi-part LP models are implemented and in each of the tests, the new parallel decomposition algorithm (by the first method) converges to within a small tolerance of the optimal value in a finite number of iterations. The algorithm of the second method sometimes fails to converge. Therefore, we discuss only the first method in this chapter, except for a brief mention of the results for the second method, at the end of the chapter.

5.1 The Implementation Procedure

In order to demonstrate the convergence and report the computational results of the new parallel decomposition algorithm, we coded the multi-part decomposition algorithm into C programs using GAMS, the Regex Library, the CPLEX Callable Library and PVM which are executable in one IBM RS/6000 with 128MB RAM and a cluster of four Pentium PCs, each with 300MHz processor and 128MB RAM. We chose the C language for coding because it can utilize both the CPLEX Callable Library and PVM, thus allowing implementation of the parallel decomposition algorithm without worrying about the details of coding a linear programming solver and parallel computing software. The network connectivity is shown in Figure 5.1. The

RS/6000 and the 4 PCs communicate with each other via the University of Waterloo LAN but the PCs use their own LAN connected by Ethernet cards with the speed of 10MB.

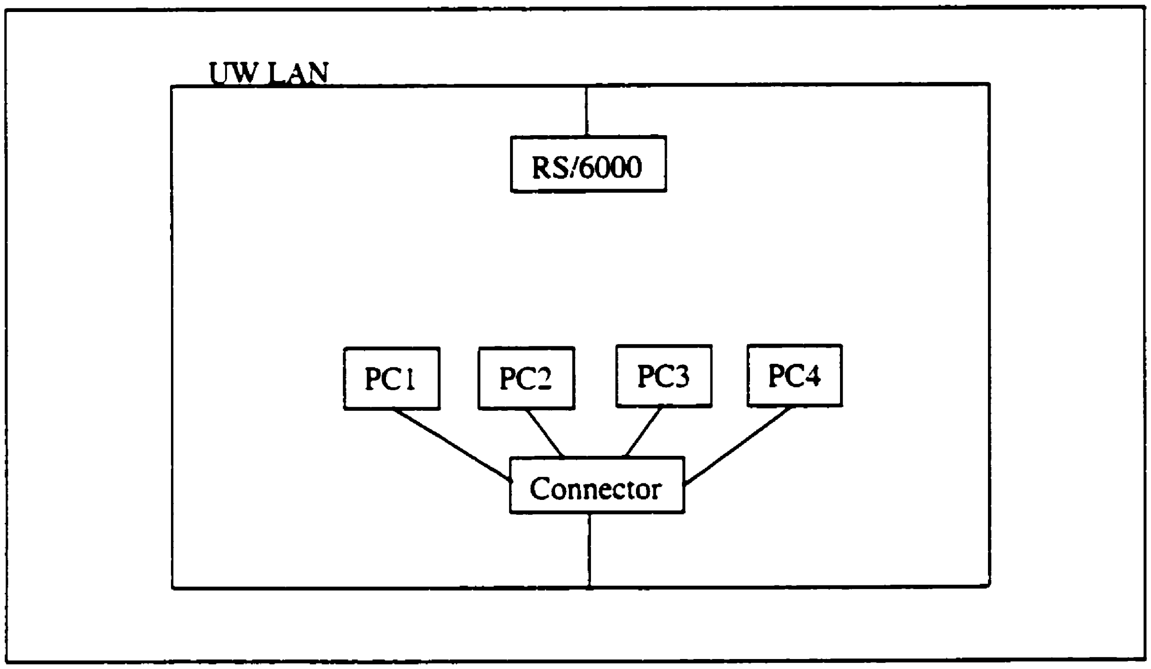

Figure 5.1 Network connectivity

The implementation procedure has two phases: a decomposition phase on the RS/6000 and a solution phase on the PCs. The decomposition phase includes formulating a model, decomposing the model into subproblems and distributing the subproblems to each computer. The solution phase includes receiving, generating and solving the subproblems simultaneously in each computer with the exchanges of primal and dual proposals at each iteration until an upper bound and a lower bound of the original problem have come within a prescribed tolerance.

5.1.1 Decomposition Phase

In the decomposition phase on the RS/6000, the model is formulated into GAMS and all data of the model, such as objective coefficients, constraint coefficients and right hand sides etc., are stored in a GAMS dictionary with the names of generic constraints or variables and the elements of their attached sets. Then, WSET (Waterloo Structure Exploiting Tool), which adopted and modified the basic feature of SPI (Structure Passing Interface) of the original SET (Structure Exploiting Tool) (Fragniere et al. [2000]), reads all the data from the GAMS dictionary using the GAMS I/O Library (GAMS [1996]) and partitions the whole problem into subproblems according to pre-defined rules in a file, called the SET file, provided by the user. These rules employ a public domain Regex library [1992] with the notion of regular expression (or pattern matcher). The partitioned data of the subproblems are sent to each computer over a local network, using PVM.

For a clear presentation of the decomposition phase, we will illustrate with the following four-region energy planning model with 10 periods ("chrisjin4.gms") is used. In the GAMS model description, the names of linking variables and linking constraints start with the capital letter "L" and the names of nonlinking variables and nonlinking constraints start with the capital letters "NL" in order to correctly identify linking elements and nonlinking elements in the later stage of partition (These particular letters are not required by our software - WSET could recognize other character strings to distinguish linking from nonlinking entities). The solver is changed to our parallel decomposition solver, called WATPAR (WATerloo PARallel) instead of CPLEX.

```
SETS
   R        region         /A,B,C,D/
   T        time periods   /1,2,3,4,5,6,7,8,9,10/
   ALIAS    (R,RR);
PARAMETERS
   COAL(R)   total coal reserves
       / A   140,      B   105,      C   130,      D   184 /
   WATER(R)  annual water availability
       / A   120,      B    72,      C    97,      D   105 /;
TABLE
   DEMELEC(R,T)    demand for electricity
         1     2     3     4     5     6     7     8     9    10
   A    23    25    28    31    35    36    40    42    38    39
   B    21    23    27    28    30    37    36    40    42    41
   C    10    12    13    15    18    23    24    21    23    25
   D    25    21    26    29    27    31    33    34    41    39 ;
TABLE
   DEMWATER(R,T)   demand for water
         1     2     3     4     5     6     7     8     9    10
   A    60    62    65    70    72    70    71    73    75    74
   B    45    44    42    44    46    49    47    45    48    49
   C    22    25    30    37    38    45    39    40    41    44
   D    33    31    37    45    48    46    51    55    50    52;
PARAMETERS
    COALCOST(R)          / A  6,         B  4,         C  8,         D  11    /
    HYELCOST(R)          / A  7,         B  9,         C  8,         D  6     / ;
TABLE
    TRANSCOST(R,RR)
         A     B      C    D
    A   100    1     1.5  1.7
    B    1    100     2   1.6
    C   1.5    2     100  1.9
    D   1.7   1.6    1.9  100 ;
SCALAR DISFACT   /.9/;
VARIABLES     COST    discounted cost ;
POSITIVE VARIABLES
    LFLOWS(R,RR,T),   NLPRCOAL(R,T),   NLPRWATER(R,T),   NLPRHYEC(R,T),   NLIMP(R,T),   NLEXP(R,T);
EQUATIONS
     OBJECTIVE         define discounted cost,      LTRADE(R,T)      electricity trade
     NLRESERVE(R)      coal reserve limit,      NLWATAVAIL(R,T) upper limit on water
     NLSUPWATER(R,T) supply & demand for water,
     NLSUPELEC(R,T)  supply & demand for electricity;
OBJECTIVE.. COST =E= -(SUM(T,(DISFACT**ORD(T))*
                     SUM(R,COALCOST(R)*NLPRCOAL(R,T)+HYELCOST(R)*NLPRHYEC(R,T)
                     +SUM(RRS(ORD(R)NE ORD(RR)), TRANSCOST(R,RR)*LFLOWS(R,RR,T))))));
LTRADE(R,T)..      -(SUM(RR,LFLOWS(RR,R,T)-LFLOWS(R,RR,T))-NLIMP(R,T)+NLEXP(R,T)) =L= 0;
NLRESERVE(R)..       SUM(T,NLPRCOAL(R,T)) =L= COAL(R);
NLWATAVAIL(R,T)..   NLPRWATER(R,T) =L= WATER(R);
NLSUPWATER(R,T)..  -(NLPRWATER(R,T)-NLPRHYEC(R,T)) =L= -DEMWATER(R,T);
NLSUPELEC(R,T)..   -(NLPRCOAL(R,T)+NLPRHYEC(R,T)+NLIMP(R,T)-NLEXP(R,T)) =L=-DEMELEC(R,T);
LFLOWS.UP('A',R,T) = 10.00;      LFLOWS.UP('B',R,T) = 7.00;
LFLOWS.UP('C',R,T) = 12.00;      LFLOWS.UP('D',R,T) = 9.00;
OPTION LP =WATPAR ;  MODEL CHRISJIN  ALL/; SOLVE CHRISJIN USING LP Maximizing COST;
```

Figure 5.2 Example of GAMS model formulation

Once "chrisjin4.gms" is executed, all information of the model is stored in the GAMS dictionary. WSET reads the data using the GAMS I/O Library and partitions the whole model into subproblems according to the user defined rules in the following SET file.

```
NB_SUB_PB 8
# Selecting the rows
ROWSPB \(L\w*(A,\w*)\)
ROWSPB \(NL\w*(A,\w*)\|NL\w*(A)\)

ROWSPB \(L\w*(B,\w*)\)
ROWSPB \(NL\w*(B,\w*)\|NL\w*(B)\)

ROWSPB \(L\w*(C,\w*)\)
ROWSPB \(NL\w*(C,\w*)\|NL\w*(C)\)

ROWSPB \(L\w*(D,\w*)\)
ROWSPB \(NL\w*(D,\w*)\|NL\w*(D)\)

# Selecting the columns

COLSPB \(L\w*(A,\w*,\w*)\)
COLSPB \(NL\w*(A,\w*)\)

COLSPB \(L\w*(B,\w*,\w*)\)
COLSPB \(NL\w*(B,\w*)\)

COLSPB \(L\w*(C,\w*,\w*)\)
COLSPB \(NL\w*(C,\w*)\)

COLSPB \(L\w*(D,\w*,\w*)\)
COLSPB \(NL\w*(D,\w*)\)

OBJROW OBJECTIVE
OBJCOL COST
ROWMASTER NOTREQUIRED
COLMASTER NOTREQUIRED
WITHGNUPLOT
PD
```

Figure 5.3 Example of SET file

The pattern to be matched is defined as "\(...\)". "\w*" means "any character string" and "|" means "or". The first line (`NB_SUB_PB 8`) shows that the original model is partitioned into 4 subproblems, each with 2 sub-parts (linking and nonlinking variables and constraints). The first subproblem for region 'A' has a set of linking constraints whose names start with the character 'L' and whose first index set value is 'A' and anything for the second index. The first subproblem for region 'A' also has nonlinking constraints defined as follows: they either start with the character 'NL' and have region 'A' for the first index; or they start with 'NL' and have 'A' as the only index. The linking variables for the region 'A' subproblem start with 'L' and have 'A' as the first index, while the nonlinking variables start with 'NL' and have 'A' as the first index. The other three subproblems for region B, C, and D have the linking (or nonlinking) constraints and variables defined by similar pattern matching rules. Since the primal-dual decomposition algorithm doesn't need any master problem, row master and column master are stated as "notrequired". "Withgnuplot" shows the following pictures: Figure 5.3 for the original problem with no structure extracted; and Figure 5.4 for the original problem with multi-part structure using WSET. The dark spots indicate the locations of the nonzero elements of the matrix.

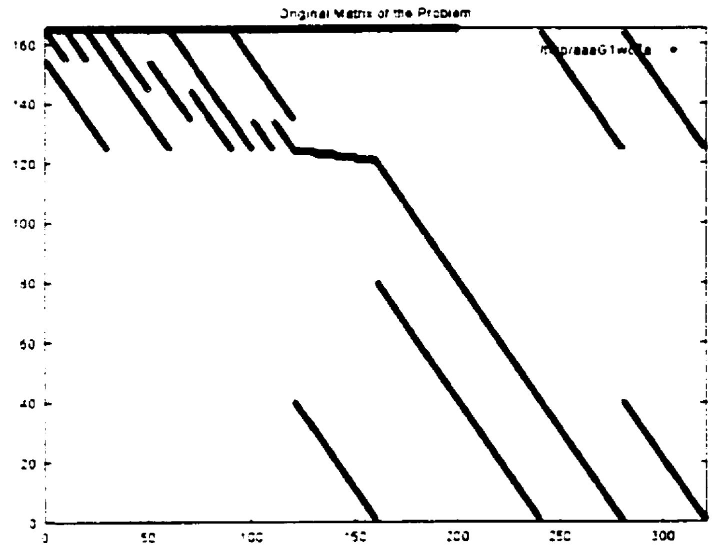

Figure 5.4 Example of anonymous matrix generated by GAMS

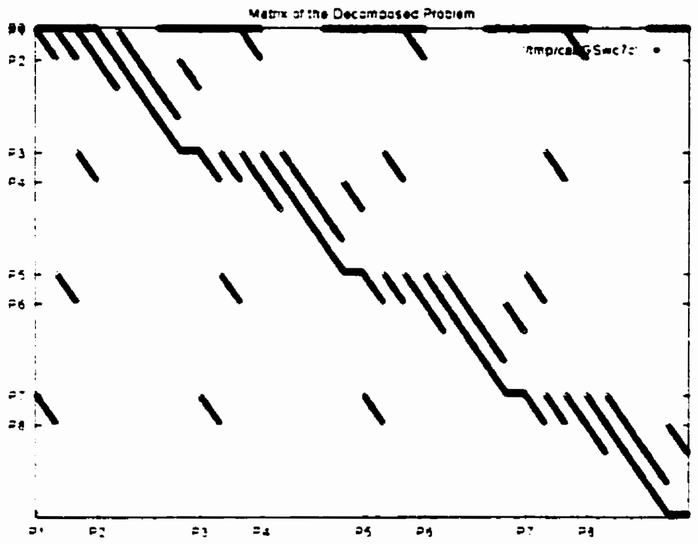

Figure 5.5 Example of multi-part structure generated by WSET

When the "PD" (Parallel Decomposition) subroutine is called, it spawns the executable files in the four PCs by the pvm_spawn() routine from the PVM library, asking them to start their own processes, and then sends each subproblem's data, partitioned by the above schemes, to each machine by the pvm_send() routine. The constraint matrix is represented by three nonzero vectors: one vector stores nonzero constraint coefficients by column, another vector has row location numbers of those coefficients and the other vector indicates the row number of the first nonzero element in each column. However the CPLEX callable library needs an additional vector indicating the number of nonzero elements in each column, so the last vector of CPLEX format should be retrieved from the three vectors of the subproblem in each PC. The data are in the format of column wise vectors, so the columns which include that part's nonlinking constraints are first sent and necessary columns for other parts' linking constraints are sent afterwards.

5.1.2 Solution phase

In the solution phase on PCs, each machine starts with its own process and receives the subproblem's data from the RS/6000 by the routine pvm_receive(). Using the CPLEX Callable Library, each LP subproblem is loaded in each computer. For instance in SP_3, the columns having that part's nonlinking constraints (the columns in L_{33}, D_3, L_{43} and B_3, A_3) are loaded first, then necessary columns for other parts' linking constraints (the columns in L_{31}, L_{41} and L_{32}, L_{42}) are loaded and finally the unnecessary linking constraints of other parts (parts 1 and 2) are deleted since column wise vectors include all linking constraints' data. Each subproblem is solved simultaneously without any information exchange at the first iteration and exchanges necessary primal or dual information with other machines and solves each new subproblem again until the gap between the objective function values of the upper-upper bound subproblem and the lower-lower bound subproblem reaches a prescribed tolerance. Note that if any of the subproblems is unbounded, the whole process stops at the first iteration by checking the optimal status generated by CPXSolution().

Since CPLEX provides the dual values corresponding to the primal constraints by calling the CPXsolution() routine in the CPLEX Callable Library, we don't have to solve for the dual variables separately, so the implementation efforts are greatly simplified. The subproblems can be solved by the simplex method or the barrier method. For the barrier method, the dual and basis information can be obtained by crossover at the last step using CPXhybbaropt().

The new primal and dual proposals are multiplied by corresponding matrices and vectors in each computer as discussed in Chapter 3. Thus, only small sized vectors are exchanged in order to minimize communication load. When the algorithm is executed to the end of the cycle and the difference between the upper bound and the lower bound of the original problem is still larger than a predetermined small tolerance, then the algorithm will start another cycle with a new cut using the CPXaddrows() routine and a new proposal using the CPXaddcols() routine from the CPLEX Callable Library.

Since the size of the whole C codes of WSET and WATPAR are very large, only the core parts of the codes are presented with detailed explanation in appendix A.

5.2 The Test Problems and Results of the Experiment

In order to make sure that our algorithm converges to an optimal solution in a finite number of iterations, we have tested several models such as the small "Toy Energy Planning" model (TEP) explained in the previous section (Figure 5.1), a North American Energy Planning model (NAEP) [Fuller, 1992], a Hydro-Electric Power Generation model (HEPG1) [Birge et al., 1999] and a huge Financial Planning model (FP) [Fragniere et al., 1998b]. Because TEP and NAEP are multi-regional energy planning models, they can be naturally partitioned into regions. HEPG1 and FP are stochastic linear programming models, so they can be divided into scenarios. HEPG1 is rewritten with different formulation and data, in the format of a scenario formulation with the nonanticipativity constraints as linking constraints; this is called HEPG2. The full details of models except NAEP (whose description is very lengthy and complex) are available in Appendix B.

Table 5.1 presents the statistics of the five test problems. The linking rows and columns are those constraints and variables which have nonzero entries in the matrices L_{ij} when the original problems are decomposed into four parts. The parenthesis in "Linking Cols (or "Linking Rows") shows the ratio of the number of linking columns (or rows) to the number of columns (or rows).

Table 5.1 Statistics of the test problems

Problems	# of regions /scenarios	Columns	Rows	Linking Cols.	Linking Rows	Nonzeroes
TEP	4	321	165	120 (37%)	40 (24%)	841
NAEP	7	3949	2425	234 (5.9%)	234 (9.6%)	12033
HEPG1	45	153721	34471	858 (0.5%)	858 (2.4%)	572015
HEPG2	16	273281	62555	130 (0.04%)	130 (0.2%)	1015893
FP	117649	333346	137258	5 (0.001%)	5 (0.003%)	1254924

The linking variables in TEP are 'LFLOWS (R,RR,T)' representing the amount of electricity that flows from region R to region RR at time T, and the linking constraints are the electricity demand constraints in each region R at time T, 'LTRADE(R,T)'. NAEP has 7 regions of Western Canada (WC), Eastern Canada (EC), Western U.S. (WU), Northern U.S. (NU), Eastern U.S. (EU), Southern U.S. (SU) and the Rest of the World (RW). The 7 regions were grouped into 4 parts: WC, EC and RW, WU and NU, and EU and SU. The linking variables and constraints of NAEP are those representing energy flows among the 4 parts. HEPG1 is formulated using time index and scenario index as shown in Figure 5.5, so it can be divided into the following 4 parts: scenarios 1 to 12, scenarios 13 to 24, scenarios 25 to 36 and scenarios 37 to 45. HEPG2 has 16 scenarios formulated with nonanticipativity constraints, thus each part having 4 scenarios. FP has 6 periods, each period with 7 scenarios and divided by the first period's scenarios, so the linking variables are the asset variables, 'LX0(A)', and cash variable, 'LC0', of period 0 since those 5 variables appear in all the first period's constraints. The linking constraints are the balance constraints of financial flows in period 1, 'LBALF_1(MS)', which include the linking variables.

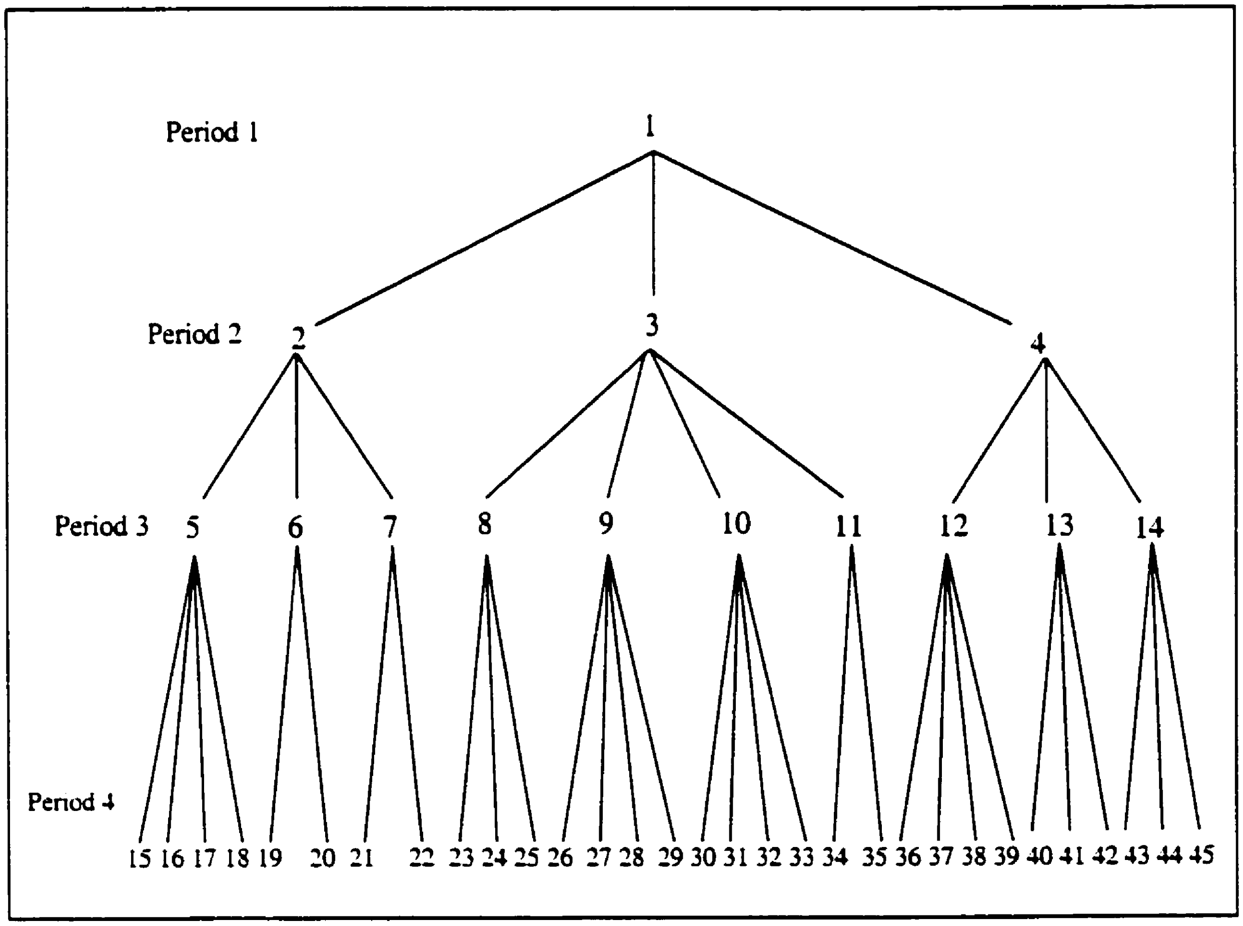

Figure 5.6 A Scenario tree for HEPG1

We first solve them directly using the simplex method and the barrier method of CPLEX 6.0 on the RS/6000 in order to compare the solution of the direct method with that of our decomposition algorithm. We measure the solution time on the RS/6000 as elapsed time, which includes the model generation time by GAMS and the solver time by CPLEX, in order to give an idea of how long the whole process takes. Note that HEPG2 cannot be solved by the barrier method, due to insufficient memory, and FP is also too large to be solved, due to the memory limit, for both the simplex method and the barrier method.

Table 5.2 The test statistics of direct methods on RS/6000 (Time is measured in seconds).

	Simplex Method			Barrier Method		
Problems	Sol. Time	Iter. No.	Obj.Values	Sol. Time	Iter. No.	Obj.Values
TEP	0.46	226	-4154.840	0.45	12	-4154.840
NAEP	102.52	2837	-3016.200	104.45	37	-3016.200
HEPG1	435	33245	-144008.474	958	22	-144008.474
HEPG2	873	55387	-30307.573	N/A	N/A	N/A
FP	N/A	N/A	N/A	N/A	N/A	N/A

We also solved the original problems in one PC by transferring all data, using WSET, in order to compare the solution time with that of parallel decomposition in the same environment. Since the artificial variables are added in the linking constraints in this method, the iteration numbers could be different from those in Table 5.2. In Table 5.3, "Simp. Iter." or "Bar. Iter" shows the number of iterations using the simplex method or the barrier method of CPLEX and

"WSET" is the time taken on the RS/6000 for partitioning the original problem into subproblems after model generation by GAMS; in this case, there is no partitioning, but WSET is used to pass the problem to CPLEX on the PC. "Setup" in Table 5.3 is the time for receiving subproblem data and loading the problem for the first iteration, and "Sol" represents the whole time to solve the problem in one PC, after setup is complete.

Table 5.3 Statistics of Simplex and Barrier methods on one PC using WSET

	Simplex Solver					Barrier Solver				
	Simp. Iter.	Time			Obj. Value	Bar. Iter.	Time			Obj. Value
		WSET	Setup	Sol.			WSET	Setup	Sol.	
TEP	220	0.83	0.06	0.19	-4154.840	14	0.82	0.06	0.21	-4154.840
NAEP	2727	8.23	0.68	1.56	-3016.200	51	8.13	0.68	4.14	-3016.200
HEPG1	32418	380.87	21.46	59.37	-144008.474	38	381.77	21.37	175.56	-144008.474
HEPG2	55387	596.65	65.70	155.11	-30307.573	23	593.61	67.30	820.07	-30307.573
FP	459177	627.76	84.3	37692	7803892.41	33	622.62	85.8	936.02	7803892.41

Note that there are slight differences between the two methods in 'WSET' and 'Setup' since the WSET time is dependent on the computer and communication network loads, and the Setup is different for the two methods.

The following table gives the sizes of the subproblems that are actually solved at the first iteration of the decomposition algorithm.

Table 5.4 The subproblem sizes

	Rows				Columns			
	LL	LU	UL	UU	LL	LU	UL	UU
TEP	71	61	51	41	120	140	160	180
NAEP	1032	822	1230	1584	625	466	708	883
HEPG1	10050	9756	9426	6894	41850	42084	42084	31836
HEPG2	15736	15736	15684	15658	68450	68502	68476	68502
FP	39221	39219	39217	19609	95245	95243	95241	47625

The lower-lower bound subproblem keeps adding proposals coming from all the previous upper and lower level iterations and the upper-upper bound subproblem accumulates the cuts coming from all the previous upper and lower level iterations. However, the lower-upper bound and upper-lower bound subproblems don't keep all the information; they forget the lower level information when they have upper level information exchange due to the problems mentioned in the previous chapter. The lower-lower bound and lower-upper bound subproblems used primal simplex method and the upper-lower bound and upper-upper bound subproblems utilized dual simplex method.

The stopping criterion used by the decomposition algorithm is that the relative duality gap (i.e. the gap as a fraction of the average of the upper and lower bounds) is smaller than or equal to 1.0×10^{-6}, which is a very rigorous condition. By this stopping criterion, all tested problems using the parallel decomposition algorithm converged to optimal solutions and they are exactly the same as those obtained by the direct methods by CPLEX 6.0. Tables 5.5 and 5.6 show the results of the parallel decomposition method using the simplex and barrier methods for solving subproblems, respectively. Note that different choices of subproblem type may give slightly different solution time and decomposition iteration number. The results shown here are one instance of what we have tried. Our experiences indicate that correctly identifying the parts, in order to reduce the number of linking variables and constraints is very important since the poor performance we had at an earlier stage turned out to be due to a poor definition of the parts. For example, the performance of NAEP was improved by 3 minutes with a different grouping of regions, which gave a smaller number of linking variables and constraints. Also, HEPG1 was solved 10 minutes faster by redefining the linking variables and constraints. In our first attempt, we defined all the water level variables, Llevel (hydro_u, K), as the linking variables and all the water balance constraints, LwatBal(hydro_u,K, K), as the linking constraints. However, it turned out that the water level variables of scenarios 1 to 3 and nodes 37 to 45 are not truly linking variables since they don't appear in other parts and the water balance constraints in nodes 1 to 12 are not truly linking constraints because they don't include other parts' linking variables.

In "PD Steps" of Tables 5.5 and 5.6, "P #" means the number of information exchanges between P_L and P_U, and "P_L #" and "P_U #" represent the number of information exchanges in the second level between P_{LL} and P_{LU} and between P_{UL} and P_{UU}, respectively. Each subproblem's simplex iteration number is presented as the sum of all iterations, and "PD Time" is the longest elapsed time taken, including setup time and idle time, among the 4 subproblems by the parallel decomposition in 4 PCs even though all processors finish almost at the same time. The speedup is measured as the ratio of the elapsed time taken by one processor over that of four processors using just "PD time" since WSET could be implemented with a more advanced coding scheme or different strategies such as storing data in each local database. All the following reports are based on the primal simplex method and the barrier method with primal crossover, however other methods such as the dual simplex method or dual crossover can be used to solve the subproblems. The simplex method was faster than the barrier method in all test problems except FP in which the barrier method was much faster.

Table 5.5 Performance of Parallel Decomposition with Simplex solver

	PD Steps			Simplex Iteration				Time (sec)		Speedup	Dual Gap
	P #	P_L #	P_U #	LL	LU	UL	UU	WSET	PD		(%)
TEP	6	6	6	191	168	149	123	1.01	0.36	0.53	0.000000
NAEP	35	67	47	1360	1841	1494	1972	18.08	5.70	0.27	0.000000
HEPG1	2	3	2	8350	8607	8542	6483	649.78	36.94	1.61	0.000000
HEPG2	4	4	4	13305	12803	13736	12801	997.45	94.53	1.64	0.000000
FP	3	3	3	267063	279306	276563	71452	1159.1	3708	10.16	0.000000

Table 5.6 Performance of Parallel Decomposition with Barrier solver

	PD Steps			Barrier Iteration				Total Time		Speedup	Dual Gap
	P #	P_L #	P_U #	LL	LU	UL	UU	WSET	PD		(%)
TEP	6	6	6	80	56	109	102	1.01	0.41	0.5	0.000000
NAEP	33	93	43	2174	1628	1597	1218	18.58	69.60	0.06	0.000000
HEPG1	2	3	2	21	84	54	44	647.18	146.48	1.20	0.000000
HEPG2	4	4	4	79	1171	105	88	995.23	341.45	2.40	0.000000
FP	3	3	3	87	84	76	107	1167.7	270.02	3.40	0.000000

Note that the relatively large number of linking variables and columns of NAEP seem to be the cause of its poor performance. Since the new algorithm converged to an optimal solution in a small number of PD steps (with the possible exception of NAEP), there was no chance for error propagation (Ho [1984]) and the test problems showed very good accuracy without dual gaps.

The statistics of detailed solution time and idle time for each subproblem are presented in Table 5.7 and 5.8. "Simplex solution time" or "Barrier solution time" shows the sum of the time taken only to solve the subproblems each iteration with Simplex or Barrier method and "Idle Time" presents the sum of waiting time for one processor to receive necessary information of other(s) at each iteration. Since PVM doesn't provide facility to measure pure communication time, this idle time includes pure communication time and pure waiting time (i.e. waiting for another processor to finish a subproblem that takes longer). "Setup time" gives the longest time taken among the four processors in receiving initial data and loading it into CPLEX.

Table 5.7 Time measurement of the parallel algorithm with Simplex solver

	Setup time	Simplex solution time				Idle time			
		LL	LU	UL	UU	LL	LU	UL	UU
TEP	0.174	0.031	0.032	0.025	0.019	0.091	0.045	0.05	0.038
NAEP	1.11	1.21	1.60	1.17	1.43	3.61	3.18	2.59	0.86
HEPG1	25.40	7.35	6.45	7.30	4.15	0.25	2.28	1.23	6.93
HEPG2	41.80	24.51	15.76	14.87	11.75	23.93	11.29	12.55	18.38
FP	73.89	3441	3442	3347	322	254	264	359	3383

Table 5.8 Time measurement of the parallel algorithm with Barrier solver

	Setup time	Barrier solution time				Idle time			
		LL	LU	UL	UU	LL	LU	UL	UU
TEP	0.174	0.15	0.13	0.19	0.17	0.09	0.04	0.03	0.04
NAEP	0.61	67.56	45.69	13.92	22.11	1.21	14.14	54.45	45.69
HEPG1	27.55	34.39	109.63	69.44	34.01	76.12	0.02	39.83	80.62
HEPG2	69.17	161.44	264.75	254.78	167.80	128.79	1.55	7.14	99.58
FP	78.01	181.41	195.53	174.81	108.03	20.49	1.82	20.31	115.5

Since FP has a lot of idle time, especially in the UU processor, due to its unbalanced structure, if it is run on a daily basis in practice, it can be implemented with a better load balancing strategy such as assigning more scenarios to the UU processor.

We also analyzed the speed-ups and efficiencies for FP obtained from 1 to 4 processors as shown in Figures 5.6 and 5.7 respectively. Figure 5.6 shows that the parallel decomposition algorithm can speed up the solution time for utilizing both simplex and barrier methods in a larger FP model. With the two processors, P_U had the first period's scenarios 1 to 3 and P_L had 4 to 7. With the three processors, FP was divided into 3 parts: P_U (scenarios 1 to 3), P_{LU} (scenarios 4 and 5) and P_{LL} (scenarios 6 and 7). Note that WATPAR can also handle another alternative of 3 parts, P_{UU} . P_{UL} and P_L, by indicating the subproblem type in each processor. Figure 5.6 shows that the parallel decomposition algorithm can speed up the solution time for utilizing both simplex and barrier methods in a larger FP model. Figure 5.7 indicates that the parallel decomposition algorithm for FP can have efficiencies greater than 1 by using the simplex method but not for the barrier method.

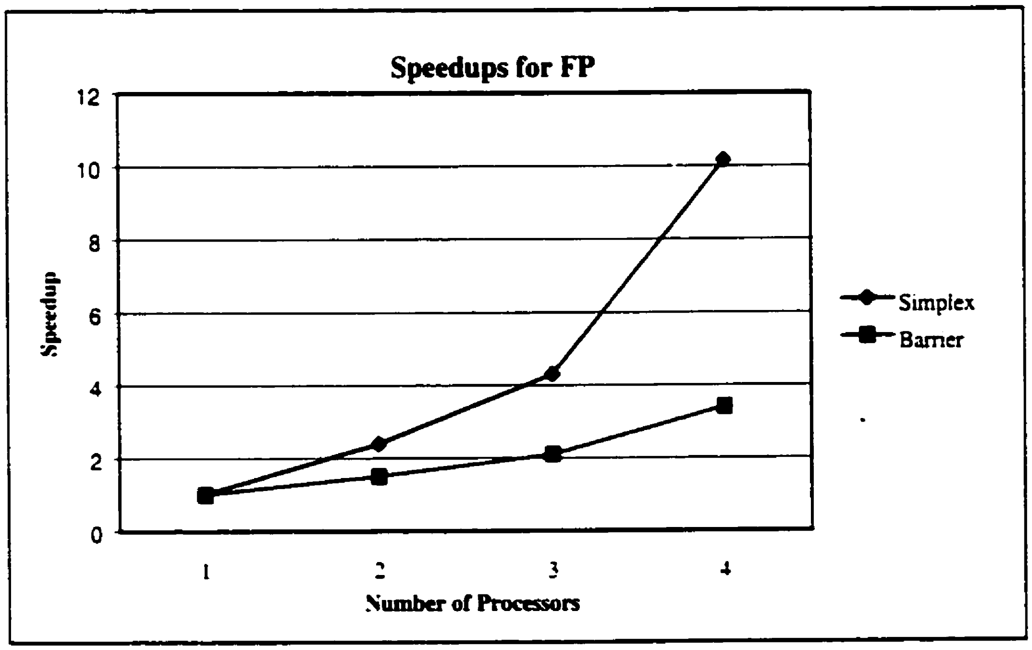

Figure 5.7 Speedups for FP with Simplex and Barrier

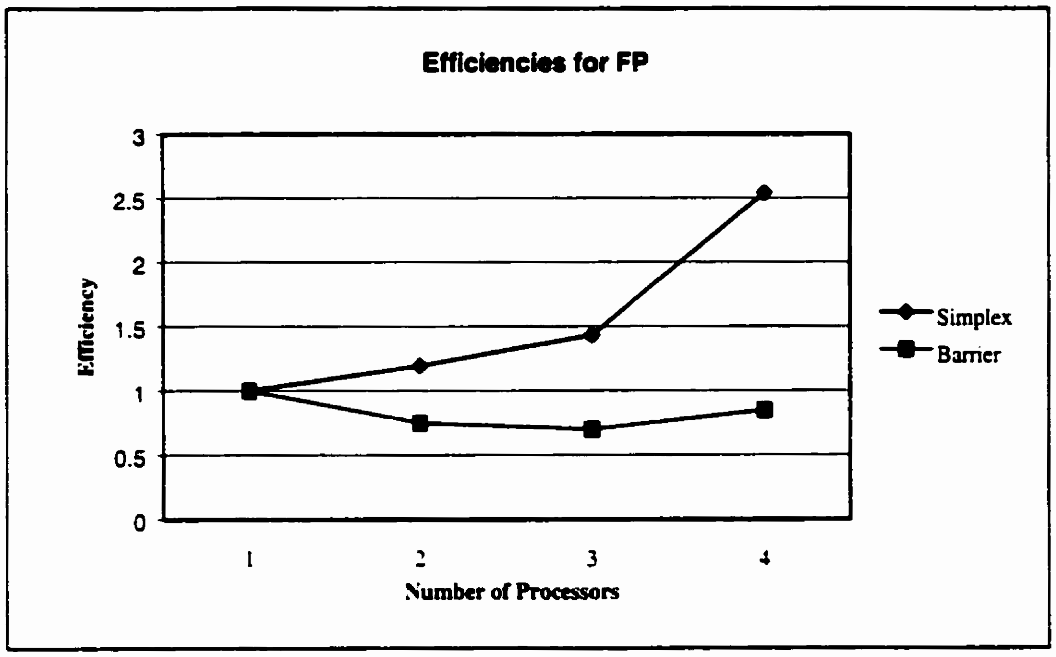

Figure 5.8 Efficiencies for FP with Simplex and Barrier

In some large problems, the parallel primal-dual decomposition showed some encouraging test results, however much more testing and development as well as efficient coding schemes are needed.

For the heuristic method, experiments were done at the very early stage of this research, and we have only the records of TEP and NAEP test results. The tests were done with several different data sets and they showed that the heuristic algorithm did not always converge to an optimal solution of the original problem. When it did not converge, it repeated the same upper bounds and lower bounds of the original problem for three consecutive iterations and the duality gap was between 0.5 % and 9 % in those test problems.

Chapter 6 Conclusions and Future Research

6.1 Conclusions

In this thesis, the main objective is to develop a new parallel decomposition method for multi-part linear programming models. We focus on the proofs and demonstration of the convergence of the two-part and multi-part algorithms. We report in some test results that the algorithms converge to an optimal solution of the original problems in a finite number of decomposition iterations.

The idea of this thesis originally came from Lan [1993] for multi-stage nested decomposition, however due to some infeasibility problems on applying the nested decomposition in parallel, we developed another parallel decomposition scheme for multi-part structured models.

The parallel decomposition algorithm for two-part models consistently converged to an optimal solution (and a convergence proof was formed), but our first algorithm for multi-part models did not always converge (we call this algorithm a heuristic). Finally, we came up with a convergent parallel primal-dual decomposition algorithm for multi-part LPs, which converged in all tests, by applying the two-part decomposition principle recursively in a hierarchical way.

The distinctions between Lan's algorithm and the new parallel decomposition algorithm are summarized in Table 6.1.

Table 6.1 The distinction between Lan's algorithm and the new parallel decomposition algorithm

	Lan's algorithm	New Parallel algorithm
Applied structure	Block-triangular	Multi-part
Construction of subproblems	One small and everything else	Half and half
Depth of subproblems	More depth (N-1)	Less depth (floor($\log_2 N$) or 1+floor($\log_2 N$))
Information exchange	Sending nearest neighbor	Broadcasting to others
Weighting scheme	Weights carried over	No carryover of weights
Subproblem computation	Serial computation (forward and backward)	Parallel computation
Decomposition algorithm	Nested	Hierarchical

In this thesis. we have completed several theoretical and implementation tasks, namely, we

1. developed the parallel decomposition algorithm for two-part models,
2. proved the convergence of the two-part method as well as other useful properties,
3. developed the parallel multi-part decomposition algorithm,
4. proved several useful properties of the multi-part method,
5. developed a variant (heuristic) decomposition algorithm for multi-part models,
6. modified SET in order to extract the multi-part structure from GAMS,
7. developed a software for a parallel primal-dual decomposition solver using PVM and CPLEX.

8. tested several models for convergence and speedups of the multi-part decomposition algorithm.

The parallel decomposition algorithm for two-part LPs solves two master-like subproblems, an upper bound subproblem and a lower bound subproblem, simultaneously. Each subproblem works as both the master problem and the subproblem in the traditional decomposition methods: each accumulates proposals from the other, so is like a master problem yet each contains full details on only its own part, so is like a subproblem. Each subproblem has a balanced structure by having the same amount of primal and dual information and has an equivalent position by conducting the convergence test at the same time. The parallel algorithm approximates the optimal value of the original problem by calculating a nonincreasing upper bound and a nondecreasing lower bound at each iteration. This procedure terminates when the two bounds are considered to be close enough according to the prescribed tolerance.

The two-part decomposition principle can be extended to more than two parts by applying the decomposition principle recursively in a hierarchical way. First, the original problem is divided into an aggregated lower bound subproblem and an aggregated upper bound subproblem. Then, each aggregated subproblem can be again divided into its aggregated lower bound subproblem and its aggregated upper bound subproblem respectively, and this hierarchical decomposing process is repeated until there are no more subproblems to decompose. In the case of four parts, the original problem can have two aggregated subproblems in the first level, an aggregated lower bound subproblem (P_L) and an aggregated upper bound subproblem (P_U), and in the second level, the P_L can be further decomposed into its lower bound subproblem and its upper bound subproblem, referred to as the lower-lower bound subproblem (P_{LL}) and the lower-upper bound subproblem (P_{LU}), while P_U can be further decomposed into, the upper-lower bound subproblem (P_{UL}) and upper-upper bound subproblem (P_{UU}).

The subproblem P_{UU} gives nonincreasing upper bounds to the original problem and P_{LL} provides nondecreasing lower bounds to the original problem as the iterations proceed, and the values of P_{UU} and P_{LL} converge to an optimal solution of P. The algorithm proceeds through iterations of parallel solution of P_{LL} and P_{LU}, by exchanges of primal and dual proposals, converging towards the optimal solution of P_L. Simultaneously, P_{UL} and P_{UU} are solved iteratively, in parallel, converging towards the optimal solution of P_U. The algorithm exchanges primal and dual proposals at the first level, i.e. between P_L and P_U, either when both level *II* pairs of subproblems have converged to the optimal values of P_L and P_U, or when the optimal value of P_{UL}, $z(P_{UL})$, is greater than or equal to the optimal value of P_{LU}, $z(P_{LU})$, i.e., $z(P_{UL}) \geq z(P_{LU})$.

We developed a parallel decomposition solver for four-part problems, called WATPAR (WATerloo PARallel). It can extract multi-part structure from an original GAMS problem and divides it into four subproblems on an RS/6000. Then it sends the subproblems to four PCs using PVM and solves them simultaneously using CPLEX. The solver is designed to utilize any number of processors from one to four.

In the preliminary tests, the algorithm converges to an optimal solution in a finite number of iterations and shows faster convergence than direct methods such as the simplex and interior point methods for some large test problems.

There are several benefits of the new decomposition method. It can improve computational speed for some real world large problems by exploiting several parallel processors. The parallel decomposition algorithm can give a solution to a model, so huge that it cannot be solved in one machine as the whole model due to a computer's memory limits. It can also provide a convenient modeling approach and easy model management with less effort, in which submodels may be developed and managed by different teams or different computers, and linked by the algorithm when a global optimum is desired (i.e. without having to merge the databases into one big, hard-to-manage model).

6.2 Future Research

The studies carried out in this thesis have suggested several possibilities for future study both in theories and implementation.

From the theoretical point of view, the parallel primal-dual decomposition method can be generalized to a broader class of mathematical programs such as mixed integer programming and nonlinear programs, which could have more benefits from parallel decomposition. The heuristic algorithm could be developed further as a warm start for the decomposition method since the heuristic algorithm seems to have better improvements in the beginning due to more proposals and cuts, and it gives feasible solutions to the original problem. Different weighting schemes could be developed by splitting the first level λ (or μ) variable into two separate λ (or μ) variables to choose different proposals coming from SP_3 and SP_4 (or SP_1 and SP_2) respectively. Also, there should be more study on load balancing strategies among the processors to reduce idle times, which can be a main success factor of parallel computing.

Due to some difficulties in coding and time limits, we could not develop the parallel decomposition solver for more than 4 parts. However, it should be extended to handle any number of parts. This could be done with more advanced coding techniques such as the object oriented programming paradigm. In its current form, the parallel decomposition solver does not calculate optimal primal and dual vectors. To be of practical use to modellers, these should be calculated and sent back to GAMS to allow generation of a solution report. Implementing the parallel algorithm among physically distributed computers with local databases for locally developed subproblems and communicating with one another by emails or web browsers would be another interesting subject from a practical point of view.

Needless to say, more testing and implementations are required to assess the efficiency of the parallel decomposition algorithm for various sized and structured problems.

APPENDIX A GAMS codes of the test models

We present the full details of GAMS models such as HEPG1, HEPG2 and FP. The data of HEPG1 and HEPG2 are generated randomly. A part of data is given here to present the data format.

A.1 Hydro-Electric Generation Planning Model 1 (HEGP1)

```
** Written by J.R. Birge and C. Supatgiat, U. Michigan, 1998*
** Last modified H. Jin Park

$include "columbiaB.dat"
$include "columbiaB.prn"

VARIABLES
       COST;

POSITIVE variables

Ntherm_gen(therm_u,modes,K) therm production in each mode in each month (MWh)
Ngen_flow(hydro_u,modes,K)  hydro water release for generation in each mode in each
month (m3)
Llevel(hydro_u,K) water level at the end of period (Mm3)
NspillFlow(hydro_u,modes,K) water spill over the period due to over capacity (m3)
Nexchange(areas,areas,modes,K) amount power transfer from (1st) to (2nd) in each
mode, each node (MWh)
Nrat_flow(hydro_u,modes,K) slack in minimum flow constraints. It gets penalty in
the objective function.
;

EQUATIONS
   OBJECTIVE
   NexpCapa(areas,modes,K)
   Nimp_capa(areas,modes,K)
   Nmeet_load(areas,modes,K)
   LwatBal(hydro_u,K,K)
   NhydroMinC(hydro_u,modes,K)
 ;

OBJECTIVE ..COST=E=-(SUM(K $(ord(K) GT 1),prob(K)*SUM(modes,SUM(therm_u,
             SUM(periods,duration(modes,K,periods))*unitCost(therm_u)*Ntherm_gen(th
       erm_u,modes,K)))))+(SUM(K $(ord(K) GT 1),
       prob(K)*SUM(modes,SUM(hydro_u,Nrat_flow(hydro_u,modes,K)))))/3.6);

NexpCapa(areas,modes,K) $(ord(K) GT 1)..SUM(areas1$(ord(areas1) NE ord(areas)),
Nexchange(areas,areas1,modes,K)) =L= export(areas) * num_hour(K)
       * SUM(periods $duration(modes,K,periods), duration(modes,K,periods));

Nimp_capa(areas,modes,K) $(ord(K) GT 1) .. SUM(areas1$(ord(areas1) NE
       ord(areas)),Nexchange(areas1,areas,modes,K)) =L= import(areas) *
       num_hour(K) * SUM(periods,duration(modes,K,periods));

Nmeet_load(areas,modes,K) $(ord(K) GT 1) ..
       -(SUM(hydro_u,hydro_isA(hydro_u,areas) * conversion(hydro_u) *
                                        Ngen_flow(hydro_u,modes,K) / 3600.0)
        + SUM(therm_u,therm_isA(therm_u,areas) * Ntherm_gen(therm_u,modes,K)) +
      SUM(areas1$(ord(areas1) NE ord(areas)),Nexchange(areas1,areas,modes,K)) -
      SUM(areas1$(ord(areas1) NE ord(areas)),Nexchange(areas,areas1,modes,K)))
    =L=-(SUM(periods,demand(modes,K,periods))*1000.0*fracArea(modes,areas));

LwatBal(hydro_u,K,predlevel) $(ord(K) GT 1 and pred(K) EQ pred1(predlevel)) ..
    -(initial(hydro_u)$(pred(K) EQ 0)+Llevel(hydro_u,predlevel)$(pred(K) GT 0)
      +(SUM(hydro_u1,links(hydro_u1,hydro_u)*
                                        SUM(modes,(Ngen_flow(hydro_u1,modes,K
                                  )
      +NspillFlow(hydro_u1,modes,K))) / 1000000.0)) + inflow(hydro_u,K) *
      (num_hour(K) * 3600.0 / 1000000.0) - SUM(modes,(Ngen_flow(hydro_u,modes,K)
      + NspillFlow(hydro_u,modes,K)) / 1000000.0)) =L= -Llevel(hydro_u,K);

NhydroMinC(hydro_u,modes,K) $(ord(K) GT 1) .. - (Ngen_flow(hydro_u,modes,K) +
      NspillFlow(hydro_u,modes,K) + Nrat_flow(hydro_u,modes,K)) =L=
   -(Fmin(hydro_u)*SUM(periods,duration(modes,K,periods))*3600.0*num_hour(K));

Llevel.UP(hydro_u,K)$(ord(K) GT 1) = Vmax(hydro_u);
Llevel.LO(hydro_u,K)$(ord(K) GT 1) = Vmin(hydro_u);

Ntherm_gen.UP(therm_u,modes,k) =
      (1.0-therm_eff(therm_u)*0.01)*capacity(therm_u)*num_hour(K)
                                 *SUM(periods,duration(modes,K,periods));
Ngen_flow.UP(hydro_u,modes,K)$(ORD(K) GT 1) =
                   (1.0 - hydro_eff(hydro_u) * 0.01) * Fmax(hydro_u)
             *SUM(periods,duration(modes,K,periods)) * 3600.0 * num_hour(K);

OPTION LP =WATPAR;

MODEL MFT /ALL/;
SOLVE MFT USING LP MAXIMIZE COST;
```

***ColumbiaB.dat**

```
SETS   therm_u   /therA, ....., therU/
       hydro_u   /hydrA, ....., hydrZ /
       K  / 0*45 /
       periods   /  1, 2, 3, 4  /
       predlevel(K) /0*14/
       modes     / 1*10 /
       areas     / reg1*reg16 /
       ALIAS    (hydro_u, hydro_u1)
       ALIAS    (areas, areas1)       ;
parameters      num_hour(K)
                Prob(K)  ;
parameter unitCost(therm_u)
          /  therA   0.8, ....., therU  0.8    /;
parameter capacity(therm_u)
          /  therA   9500, ....., therU  21    /;
parameter initial(hydro_u)
          /  hydrA    650, ....., hydrX  580  /;
parameter conversion(hydro_u)
          /  hydrA   4, .....,  hydrZ  5  /;
parameter Vmax(hydro_u)
          /  hydrA   2100, ....., hydrZ  2750 /;
parameter Vmin(hydro_u)
          /  hydrA    230, ....., hydrZ   210 /;
parameter hydro_eff(hydro_u)
```

```
          /  hydrA  25,     ...., hydrZ  15 /;
parameter Fmax(hydro_u)
          /  hydrA   21, ...., hydrZ   25   /;
parameter Fmin(hydro_u)
          /  hydrA   5, ....,  hydrZ   5  / ;
table links(hydro_u, hydro_u1)
      hydrA .... hydrL
hydrA  0.5  ....  0.2
....       ....
hydrZ  0.3  ....  0.1  ;
parameter prob(K)  / 1     1,
                     2   0.4,     3    0.2,     4   0.4,
                                ....
                    43  0.005,  44   0.005,  45  0.01     / ;
parameter pred(K)  / 1  0,
                     2*4  1,
                     ....
                     34*35  11, 36*39  12, 40*42 13, 43*45 14  /;

parameter pred1(K)   / 0  0, 1  1,  2  2,   3  3,  4  4, 5  5, 6  6, 7  7
                       8  8, 9  9, 10  10, 11 11, 12 12 ,13  13, 14  14/   ;

            Loop(K,
             num_hour(K) = ROUND(UNIFORM(3,8))
             );

table demand(modes, K, periods)
       1.1    2*4.2  5*14.3   15*45.4
1       9      19     28       39
....          ....
10      7      12     27       34
 ;
table duration(modes, K, periods)
       1.1   2*4.2    5*14.3   15*45.4
1      0.3    0.2      0.25     0.3
....                ....
10     0.2    0.3      0.23     0.24
;
table fracArea(modes, areas)
   reg1 reg2  ....  reg15 reg16
1  0.05 0.08  ....   0.1  0.1
....        ....
10 0.09 0.01  ....   0.02 0.18
;
parameter import(areas) / reg1   25000, ...., reg16  27700  /
table hydro_isA(hydro_u, areas)
       reg1 reg2 reg3 .... reg13 reg14 reg15 reg16
hydrA   1
hydrB       1
hydrC            1
....                    ....
hydrX                              1
hydrY                                     1
hydrZ                                           1     ;
table therm_isA(therm_u, areas)
       reg1 reg2 reg3 .... reg14 reg15 reg16
therA   1
therB       1
therC            1
....                 ....
therS                         1
```

```
therT                             1
therU                                   1;

parameter export(areas)  / reg1  22500, ...., reg16 27700   / ;
```

***ColumbiaB.prn**

```
Table inflow(hydro_u,K)
        1   2   3 ....  43  44  45
hydrA  52  66  81 ....   57    66  85
.....           ....
hydrZ  57  69  85 ....   58  70  81 ;
```

A.2 Hydro-Eletric Generation Planning Model 2 (HEGP2)

```
* written by J.R. Birge and C. Supatgiat, U. Michigan, 1998*
* Last modified H. Jin Park

$include "columbia.dat"

VARIABLES
COST;

POSITIVE VARIABLES
Ntherm_gen(therm_u,modes,periods,senarios)
* therm production in each mode in each month (MWh)
Ngen_flow(hydro_u,modes,periods,senarios)
* hydro water release for generation in each mode in each month (m3)
Llevel(hydro_u,periods,senarios)
* water level at the end of period (Mm3)
NspillFlow(hydro_u,modes,periods,senarios)
* water spill over the period due to over capacity (m3)
Nexchange(areas,areas,modes,periods,senarios)
* amount power transfer from (1st) to (2nd) in each mode , each node (MWh)
Nrat_flow(hydro_u,modes,periods,senarios);
* slack in minimum flow constraints. It gets penalty in the objective function.

EQUATIONS
   OBJECTIVE
   NexpCapa(areas,modes,periods,senarios)
   Nimp_capa(areas,modes,periods,senarios)
   Nmeet_load(areas,modes,periods,senarios)
   NwatBal(hydro_u,periods,senarios)

   NhydroMinC(hydro_u,modes,periods,senarios)
   LnonAnt(hydro_u,periods,senarios)
   LnonAnt21(hydro_u,periods,senarios)
   LnonAnt22(hydro_u,periods,senarios)

   LnonAnt31(hydro_u,periods,senarios), LnonAnt32(hydro_u,periods,senarios)
   LnonAnt33(hydro_u,periods,senarios), LnonAnt34(hydro_u,periods,senarios)

   LnonAnt41(hydro_u,periods,senarios), LnonAnt42(hydro_u,periods,senarios)
   LnonAnt43(hydro_u,periods,senarios), LnonAnt44(hydro_u,periods,senarios)
   LnonAnt45(hydro_u,periods,senarios), LnonAnt46(hydro_u,periods,senarios)
   LnonAnt47(hydro_u,periods,senarios), LnonAnt48(hydro_u,periods,senarios)
;

OBJECTIVE .. COST =E= - (SUM(senarios, prob(senarios)*SUM(periods,
        SUM(modes,duration(modes,periods)* SUM(therm_u, unitCost(therm_u) *
             Ntherm_gen(therm_u,modes,periods,senarios)))))
```

```
      + SUM(senarios, prob(senarios) * SUM(periods, SUM(modes,SUM(hydro_u,
            Nrat_flow(hydro_u,modes,periods,senarios)))))/3.6);

NexpCapa(areas,modes,periods,senarios) .. SUM(areas1$(ord(areas1) NE
 ord(areas)),Nexchange(areas,areas1,modes,periods,senarios)) =L= export(areas) *
      num_hour(periods,senarios) * duration(modes,periods);

Nimp_capa(areas,modes,periods,senarios) .. SUM(areas1$(ord(areas1) NE
      ord(areas)),Nexchange(areas1,areas,modes,periods,senarios)) =L=
      import(areas) * num_hour(periods,senarios) * duration(modes,periods);

Nmeet_load(areas,modes,periods,senarios) ..
             -(SUM(hydro_u,hydro_isA(hydro_u,areas) * conversion(hydro_u) *
                    Ngen_flow(hydro_u,modes,periods,senarios) / 3600.0)
                      + SUM(therm_u,therm_isA(therm_u,areas) *
      Ntherm_gen(therm_u,modes,periods,senarios)) + (SUM(areas1$(ord(areas1) NE
             ord(areas)),Nexchange(areas1,areas,modes,periods,senarios)))
             - SUM(areas1$(ord(areas1) NE
      ord(areas)),Nexchange(areas,areas1,modes,periods,senarios)))
     =L= -(demand(modes,periods) * 1000.0 * fracArea(modes,areas));

NwatBal(hydro_u,periods,senarios) ..
       -(initial(hydro_u)$(ord(periods) EQ 1) + Llevel(hydro_u,periods-1,
      senarios)$(ord(periods) GT 1)  + (SUM(hydro_u1, links(hydro_u1,hydro_u) *
      SUM(modes, (Ngen_flow(hydro_u1,modes,periods,senarios)
      + NspillFlow(hydro_u1,modes,periods,senarios))) / 1000000.0))
      + inflow(hydro_u,periods,senarios) * (num_hour(periods,senarios) * 3600.0
            / 1000000.0) - SUM(modes,(Ngen_flow(hydro_u,modes,periods,senarios)
      + NspillFlow(hydro_u,modes,periods,senarios)) / 1000000.0)) =L=
                                          - Llevel(hydro_u,periods,senarios);

NhydroMinC(hydro_u,modes,periods,senarios)..
                          - Ngen_flow(hydro_u,modes,periods,senarios) +
                            NspillFlow(hydro_u,modes,periods,senarios) +
                                    Nrat_flow(hydro_u,modes,periods,senarios))
                     =L= -(Fmin(hydro_u) *duration(modes,periods) * 3600.0 *
                               num_hour(periods,senarios));

LnonAnt(hydro_u,peri1,senarios)..        Llevel(hydro_u,peri1,senarios) =l=
                                                  Llevel(hydro_u,peri1,senarios++1);

LnonAnt21(hydro_u,peri2,sen1)..   Llevel(hydro_u,peri2,sen1) =l=
                                                  Llevel(hydro_u,peri2,sen1++1);
LnonAnt22(hydro_u,peri2,sen2)..   Llevel(hydro_u,peri2,sen2) =l=
                                                  Llevel(hydro_u,peri2,sen2++1);

LnonAnt31(hydro_u,peri3,sen3)..   Llevel(hydro_u,peri3,sen3) =l=
                                                  Llevel(hydro_u,peri3,sen3++1);
LnonAnt32(hydro_u,peri3,sen4)..   Llevel(hydro_u,peri3,sen4) =l=
                                                  Llevel(hydro_u,peri3,sen4++1);
LnonAnt33(hydro_u,peri3,sen5)..   Llevel(hydro_u,peri3,sen5) =l=
                                                  Llevel(hydro_u,peri3,sen5++1);
LnonAnt34(hydro_u,peri3,sen6)..   Llevel(hydro_u,peri3,sen6) =l=
                                                  Llevel(hydro_u,peri3,sen6++1);

LnonAnt41(hydro_u,peri4,sen7)..   Llevel(hydro_u,peri4,sen7) =l=
                                                  Llevel(hydro_u,peri4,sen7++1);
LnonAnt42(hydro_u,peri4,sen8)..   Llevel(hydro_u,peri4,sen8) =l=
                                                  Llevel(hydro_u,peri4,sen8++1);
LnonAnt43(hydro_u,peri4,sen9)..   Llevel(hydro_u,peri4,sen9) =l=
                                                  Llevel(hydro_u,peri4,sen9++1);
```

```
LnonAnt44(hydro_u,peri4,sen10).. Llevel(hydro_u,peri4,sen10) =l=
                                         Llevel(hydro_u,peri4,sen10++1);
LnonAnt45(hydro_u,peri4,sen11) ..        Llevel(hydro_u,peri4,sen11) =l=
                                         Llevel(hydro_u,peri4,sen11++1);
LnonAnt46(hydro_u,peri4,sen12) ..        Llevel(hydro_u,peri4,sen12) =l=
                                         Llevel(hydro_u,peri4,sen12++1);
LnonAnt47(hydro_u,peri4,sen13) ..        Llevel(hydro_u,peri4,sen13) =l=
                                         Llevel(hydro_u,peri4,sen13++1);
LnonAnt48(hydro_u,peri4,sen14) ..        Llevel(hydro_u,peri4,sen14) =l=
                                         Llevel(hydro_u,peri4,sen14++1);

Llevel.UP(hydro_u,periods,senarios) = Vmax(hydro_u);
Llevel.LO(hydro_u,periods,senarios) = Vmin(hydro_u);

Ntherm_gen.UP(therm_u,modes,periods,senarios) =(1.0 - therm_eff(therm_u) * 0.01)
       * capacity(therm_u) * num_hour(periods,senarios)* duration(modes,periods);

Ngen_flow.UP(hydro_u,modes,periods,senarios) = (1.0 - hydro_eff(hydro_u) * 0.01)
       *Fmax(hydro_u) *duration(modes,periods)*3600.0*num_hour(periods,senarios);

OPTION LP =WATPAR;
MODEL MFT /ALL/;
SOLVE MFT USING LP MAXIMIZE COST;
```

***Columbia.dat**

```
SETS  therm_u  /therA, therB, .... , therU/
      hydro_u  /hydrA, hydrB, .... , hydrZ /

      periods  /  T1*T5  /
      senarios  /S1*S16/

      peri1(periods)      /T1/,      peri2(periods)      /T2/
      peri3(periods)      /T3/,      peri4(periods)      /T4/

      sen1(senarios)   /S1*S8/,      sen2(senarios)   /S9*S16/
      sen3(senarios)   /S1*S4/,      sen4(senarios)   /S5*S8/
      sen5(senarios)   /S9*S12/,      sen6(senarios)   /S13*S16/

      sen7(senarios)   /S1*S2/,      sen8(senarios)   /S3*S4/
      sen9(senarios)   /S5*S6/,      sen10(senarios)   /S7*S8/
      sen11(senarios)   /S9*S10/,    sen12(senarios)   /S11*S12/
      sen13(senarios)   /S13*S14/,   sen14(senarios)   /S15*S16/

      modes     / 1*10 /
      areas     / reg1*reg16 /
      ALIAS    (hydro_u, hydro_u1)
      ALIAS    (areas, areas1)     ;

parameter unitCost(therm_u)
          / therA  0.8, .... , therU  0.8 /;
parameter capacity(therm_u)
          / therA  9500, therB  9800, .... , therU  7900  /;
parameter therm_eff(therm_u)
            therA  21, therB  17, .... , therU  21    /;

parameter initial(hydro_u)
          / hydrA   500, hydrB 700, .... , hydrZ 690 /;
parameter conversion(hydro_u)
          / hydrA  4, hydrB  6, .... , hydrZ  5  /;
parameter Vmax(hydro_u)
```

```
          /  hydrA  1100, hydrB  1200, .... , hydrZ  1250     /;
parameter Vmin(hydro_u)
          /  hydrA   530, hydrB   540, .... , hydrZ   630    /;
parameter hydro_eff(hydro_u)
          /  hydrA  25, hydrB  26, ..... , hydrZ   25        /;
parameter Fmin(hydro_u)
          /  hydrA   5, hydrB   4, .... , hydrZ   5       / ;

table links(hydro_u, hydro_u1)
            hydrA hydrB hydrC .... hydrJ hydrK hydrL
hydrA        0.5   0.2   0.25 ....
....                              ....
hydrZ         0.1          ....          0.3   0.45  ;

parameter prob(senarios) / S1  0.02,  S2  0.02, S3  0.016, S4  0.024
                     ....   S13  0.07,  S14  0.09, S15 0.126, S16  0.124     /;
table num_hour(periods,senarios)
     s1   S2   S3   ....   S15   S16
T1   3    3    3    ....    3    3
....                ....
T5  3.7  2.7  5.2   ....   3.5  1.9  ;

table inflow(hydro_u, periods, senarios)
      T1.S1*S16  T2.S1*S8 ....   T5.S16
hydrA     33        17      ....     29
....     ....      ....     ....    ....
hydrZ     65        22      ....     30           ;
table demand(modes, periods)
       T1    T2    T3    T4     T5
1     6.3   6.3   6.2   6.8    7.3
.... ....  ....  ....  ....   ....
10    3.7   4.9   5.7   7.4    7.9  ;
table duration(modes, periods)
        T1     T2     T3      T4      T5
1      0.3    0.2    0.25    0.3     0.1
....  ....   ....    ....    ....    ....
10     0.2    0.3    0.23    0.24   0.08  ;

table fracArea(modes, areas)
      reg1   reg2  ....  reg15    reg16
1     0.05   0.08  ....   0.1      0.1
....  ....   ....  ....  ....     ....
10    0.09   0.01  ....  0.02     0.18    ;

parameter import(areas)
                        / reg1   25000,   .... , reg16  27700 /;
table hydro_isA(hydro_u, areas)
       reg1 reg2 .... reg15 reg16
hydrA   1
hydrB        1
....              ....
hydrZ                   1              ;

table therm_isA(therm_u, areas)
      reg1 reg2 .... reg15 reg16
therA   1        ....
therB                  1
....             ....
therU                        1         ;
parameter export(areas)
                         / reg1  22500, .... , reg16 27700  / ;
```

A.3 Financial Planning model (FP)

```
* A GAMS financial planning model inspired by J. Birge and G. Infanger.
* Modified and tested by E. Fragniere , J. Gondzio and J.-P. Vial,
* Maximum size:  9 periods (10 stages), 10 scenarios (more can easily be added)
* Default size:  6 periods ( 7 stages), scenarios 1*7
* This is a nice big model: for the default of 6 periods and 7 scenarios
* Last Modified by H. Jin Park

OPTION RESLIM = 21600;      OPTION ITERLIM = 500000;
OPTION LIMROW = 100;        OPTION LIMCOL = 100;       OPTION SOLPRINT = OFF;

SCALARS
  RC    Cash rate of return      /1.05/,
  WI    Initial capital          /50/,
  WF    Goal                     /75/,
  MAXPER 'max # of periods'      / 9 /,
  NPER  '# of periods used'      / 6 /;
* don't change MAXPER unless you know what you are doing!
* changing NPER is the intent, but do that in your own copy of this model

abort$(MAXPER gt 9)    'You should not have changed this!';
abort$(NPER gt MAXPER) 'Maximum # of periods allowed exceeded';
abort$(NPER lt 1)      'Must run model for at least one period';

SETS
  A              'Assets'     / USAB, FORS, CORP, GOVE /,
  MS             'master set of scenarios'        / s1 * s10, id /,
  id(MS)         'identity: replicates previous period'    / id /,
  P1(MS)         'scenarios used in period 1'              / s1 * s7 /,
  P2(MS)         'scenarios used in period 2'
  P3(MS)         'scenarios used in period 3'
  P4(MS)         'scenarios used in period 4'
  P5(MS)         'scenarios used in period 5'
  P6(MS)         'scenarios used in period 6'
*  P7(MS)         'scenarios used in period 7'
*  P8(MS)         'scenarios used in period 8'
*  P9(MS)         'scenarios used in period 9'
;

* In order to make the number of periods flexible,
* we allow only the identity scenario for periods
* after the ones we wish to consider.
P2(MS) = P1(MS)$(NPER ge 2) + id(MS)$(NPER lt 2);
P3(MS) = P1(MS)$(NPER ge 3) + id(MS)$(NPER lt 3);
P4(MS) = P1(MS)$(NPER ge 4) + id(MS)$(NPER lt 4);
P5(MS) = P1(MS)$(NPER ge 5) + id(MS)$(NPER lt 5);
P6(MS) = P1(MS)$(NPER ge 6) + id(MS)$(NPER lt 6);
*P7(MS) = P1(MS)$(NPER ge 7) + id(MS)$(NPER lt 7);
*P8(MS) = P1(MS)$(NPER ge 8) + id(MS)$(NPER lt 8);
*P9(MS) = P1(MS)$(NPER ge 9) + id(MS)$(NPER lt 9);

TABLE RR(MS,A)  'Asset rates of return'
      USAB   FORS   CORP   GOVE
s1    1.27   1.16   0.99   1.02
s2    1.20   1.41   1.04   1.05
s3    1.06   0.91   1.11   1.10
s4    1.23   0.83   1.05   1.04
```

```
s5    1.09    1.10    0.95    0.98
s6    1.15    1.28    1.25    1.18
s7    0.83    0.97    1.02    1.07
s8    0.83    0.77    0.91    0.98
s9    1.09    1.18    1.18    1.16
s10   1.20    1.18    1.18    1.16
id    1       1       1       1    ;

VARIABLES
  EU                                   'Expectation of the utility function';
POSITIVE VARIABLES
  LC0                                  'cash period 0',
  C1(MS)                               'cash period 1',
  C2(MS,MS)                            'cash period 2',
  C3(MS,MS,MS)                         'cash period 3',
  C4(MS,MS,MS,MS)                      'cash period 4',
  C5(MS,MS,MS,MS,MS)                   'cash period 5',
*   C6(MS,MS,MS,MS,MS,MS)               'cash period 6',
*   C7(MS,MS,MS,MS,MS,MS,MS)            'cash period 7',
*   C8(MS,MS,MS,MS,MS,MS,MS,MS)         'cash period 8',
  LX0(A)                               'assets period 0',
  X1(MS,A)                             'assets period 1',
  X2(MS,MS,A)                          'assets period 2',
  X3(MS,MS,MS,A)                       'assets period 3',
  X4(MS,MS,MS,MS,A)                    'assets period 4',
  X5(MS,MS,MS,MS,MS,A)                 'assets period 5',
*   X6(MS,MS,MS,MS,MS,MS,A)             'assets period 6',
*   X7(MS,MS,MS,MS,MS,MS,MS,A)          'assets period 7',
*   X8(MS,MS,MS,MS,MS,MS,MS,MS,A)       'assets period 8',
*   U (MS,MS,MS,MS,MS,MS,MS,MS,MS)      'surplus, final period',
*   V (MS,MS,MS,MS,MS,MS,MS,MS,MS)      'deficit, final period1';

CART(MS)
U (MS,MS,MS,MS,MS,MS)        'surplus, final period',
V (MS,MS,MS,MS,MS,MS)        'deficit, final period';

EQUATIONS
  OBJECTIVE                            'computes EU (expected utility)',
  BALF_0                                   'Balance of Financial Flows period 0',
  LBALF_1(MS)                          'Balance of Financial Flows period 1',
  BALF_2(MS,MS)                        'Balance of Financial Flows period 2',
  BALF_3(MS,MS,MS)                     'Balance of Financial Flows period 3',
  BALF_4(MS,MS,MS,MS)                  'Balance of Financial Flows period 4',
  BALF_5(MS,MS,MS,MS,MS)               'Balance of Financial Flows period 5',
*   BALF_6(MS,MS,MS,MS,MS,MS)           'Balance of Financial Flows period 6',
*   BALF_7(MS,MS,MS,MS,MS,MS,MS)        'Balance of Financial Flows period 7',
*   BALF_8(MS,MS,MS,MS,MS,MS,MS,MS)     'Balance of Financial Flows period 8',
*   compUV(MS,MS,MS,MS,MS,MS,MS,MS,MS)  'surplus and deficit, final period';
COMPUV(MS,MS,MS,MS,MS,MS)

;

OBJECTIVE..
         EU =E= SUM ((P1,P2,P3,P4,P5,P6),
                        5*U(P1,P2,P3,P4,P5,P6)-20*V(P1,P2,P3,P4,P5,P6)) ;
*OBJECTIVE..
*          EU =E= SUM ((P1,P2,P3,P4,P5,P6,P7,P8,P9),
*                    5*U(P1,P2,P3,P4,P5,P6,P7,P8,P9)
*                 -20*V(P1,P2,P3,P4,P5,P6,P7,P8,P9) );

BALF_0.. SUM (A, LX0(A)) + LC0 =l= WI;
```

```
LBALF_1(P1)$(NPER gt 1)..
       -(SUM (A, LX0(A)*RR(P1,A)) + LC0*RC) =l= -(SUM (A, X1(P1,A))+ C1(P1)) ;
BALF_2(P1,P2)$(NPER gt 2)..
        -(SUM (A, X1(P1,A)*RR(P2,A)) + C1(P1)*RC)  =l=
        -(SUM (A, X2(P1,P2,A))       + C2(P1,P2));

BALF_3(P1,P2,P3)$(NPER gt 3)..
        -(SUM (A, X2(P1,P2,A)*RR(P3,A)) + C2(P1,P2)*RC)  =l=
        -(SUM (A, X3(P1,P2,P3,A))       + C3(P1,P2,P3));
BALF_4(P1,P2,P3,P4)$(NPER gt 4)..
        -(SUM (A, X3(P1,P2,P3,A)*RR(P4,A)) + C3(P1,P2,P3)*RC)  =l=
        -(SUM (A, X4(P1,P2,P3,P4,A))       + C4(P1,P2,P3,P4));
BALF_5(P1,P2,P3,P4,P5)$(NPER gt 5)..
        -(SUM (A, X4(P1,P2,P3,P4,A)*RR(P5,A)) + C4(P1,P2,P3,P4)*RC)  =L=
        -(SUM (A, X5(P1,P2,P3,P4,P5,A))       + C5(P1,P2,P3,P4,P5));
COMPUV(P1,P2,P3,P4,P5,P6)..
        WF + U(P1,P2,P3,P4,P5,P6)
           - V(P1,P2,P3,P4,P5,P6)  =L=
        (sum (A, LX0(A)*RR(P1,A))              + LC0*RC) $(NPER eq 1) +
        (sum (A, X1(P1,A)*RR(P2,A))            + C1(P1)*RC) $(NPER eq 2) +
        (sum (A, X2(P1,P2,A)*RR(P3,A))         + C2(P1,P2)*RC) $(NPER eq 3) +
        (sum (A, X3(P1,P2,P3,A)*RR(P4,A))      + C3(P1,P2,P3)*RC) $(NPER eq 4)
+
        (sum (A, X4(P1,P2,P3,P4,A)*RR(P5,A))
               + C4(P1,P2,P3,P4)*RC) $(NPER eq 5)
+
        (sum (A, X5(P1,P2,P3,P4,P5,A)*RR(P6,A))
               + C5(P1,P2,P3,P4,P5)*RC) $(NPER eq 6)
*+
*        [sum ((A, X6(P1,P2,P3,P4,P5,P6,A)*RR(P7,A))
*               + C6(P1,P2,P3,P4,P5,P6)*RC] $(NPER eq 7)
;

LX0.UP(A) = WI;  LC0.UP = WI;

OPTION LP = WATPAR;
MODEL PORT / ALL /;
SOLVE PORT USING LP MAXIMIZING EU;
```

APPENDIX B The C codes of WSET

We present a simplified code of a core part for WSET (Waterloo Structure Exploiting Tool), which describes how the multi-part structure can be extracted from GAMS and how each subproblem's data can be sent to the parallel machines. This code is a modification of the original SET, which has much more complicated coding scheme with several files. Assuming that all files and libraries are linked properly and all variables are defined correctly, we give only the modification of the original set.

```
LpSubProb* short SplitData_PD (LpGams* LpG, BlockSt* Decomp, LpSubProb** LpSub)
{
  ..... DEFINE VARIABLES AND ALLOCATE APPOPRIATE MEMORY .....

   /* Retrieve data from nb_blocks Decomposition structure of original SET. */
   int  nb_blocks        = Decomp->nb_blocks;
   int  objective_row    = Decomp->objective_row;
   int  objective_column = Decomp->objective_column;

   int* rowinfo  = Decomp->rowinfo;
   int* row_perm = Decomp->row_permutation;
   int* inv_rw_p = Decomp->inverse_rowperm;

   int* colinfo  = Decomp->colinfo;
   int* col_perm = Decomp->col_permutation;
   int* inv_cl_p = Decomp->inverse_colperm;

   /*  Determine the dimensions of SubProblems; number of variables,constraints,
    *  and nonzeroes of nb_subs part for WSET.
    *  Remove column corresponding to objective variable from Sub_0.
    *  Retrieve the index of the objective row in a permuted matrix. */

   nb_subs = nb_blocks/2;

   for (i = 0; i <= nb_subs; i++) {
      if(i==0)
      {
       SPb_n[i]  = Decomp->LastCol[i] - Decomp->FirstCol[i];
       SPb_m[i]  = Decomp->LastRow[i] - Decomp->FirstRow[i] ;
      }
      else
      {
       SPb_n[2*i]  = Decomp->LastCol[2*i] - Decomp->FirstCol[2*i]  ;
       SPb_m[2*i]  = Decomp->LastRow[2*i] - Decomp->FirstRow[2*i]  ;
       SPb_n[2*i-1]  = Decomp->LastCol[2*i-1] - Decomp->FirstCol[2*i-1]  ;
       SPb_m[2*i-1]  = Decomp->LastRow[2*i-1] - Decomp->FirstRow[2*i-1]  ;
       SPb_n[i] = SPb_n[2*i-1]+SPb_n[2*i];
       SPb_m[i] = SPb_m[2*i];

       for(j=1; j <= nb_subs; j++)
```

```
        {
         SPb_m[i] = SPb_m[i]+Decomp->LastRow[2*j-1]-Decomp->FirstRow[2*j-1] ;
        }
      }
      SPb_nz[i] = 0;
     }
   SPb_n[0]--;
   obj_row = row_perm[objective_row];
   printf("obj_row = %d\n", obj_row);
   for (i = 0; i < Decomp->n; i++) {
      j = colinfo[i];
      SPb_nz[j] += LpG->jcol[i+1] - LpG->jcol[i];
   }

   j = objective_column;
   SPb_nz[0] -= LpG->jcol[j+1] - LpG->jcol[j];
   for (i = 1; i <= nb_blocks; i++) {
      SPb_m[i] += SPb_m[0];
   }

   for(i = 0; i <= nb_subs; i++)
    {
      LOWSPb_m[i] = 0;
      if(i == 0)
        SPb_nz[0]= 0;
      else
        SPb_nz[i] = SPb_nz[2*i-1] + SPb_nz[2*i];
    }

   /* Find the number of linking variables and constraints for each part*/

   for(i=1 ; i <=nb_subs ; i++)
   {
    LSPb_n[i] = Decomp->LastCol[2*i-1] - Decomp->FirstCol[2*i-1];
    LSPb_m[i] = Decomp->LastRow[2*i-1] - Decomp->FirstRow[2*i-1];
   }

   /* Find the total number of linking variables and constraints for upper
parts*/

   for(i=1;i<=nb_subs;i++)
   {
     HISPb_m[i] = HISPb_m[i-1]+LSPb_m[i];
     HISPb_n[i] = HISPb_n[i-1]+LSPb_n[i];
   }

   /* Find the total number of linking constraints for lower parts */

  for(k=1; k <= nb_subs; k++)
  {
    LOWSPb_m[k] = HISPb_m[nb_subs] - HISPb_m[k];
  }

 phd = 1;

 * First main loop. Split Data of GAMS LP problem into SubProblems.
 * Set up all column related data in SubProblem LP matrices.
 * Notation:
 * i   and j    denote row and column indices in the GAMS LP matrix;
 * irw and jcl  denote row and column indices in the permuted matrix;
 * isp and jsp  denote row and column indices in the SubProblem;
 *  k             denotes an index of SubProblem to which column j belongs.
 *  Starting from the first column, decide the row part number of the nonzero
 *  coefficient: nonlinking constraint's coefficient belongs to the same part
 *  number which the column belongs to; linking constraint's coefficient
 *  should have a column part number and row part number.
 *  Note that there are 2*nb_subs subproblems in original SET but should be
 *  converted to nb_subs subproblems. iii and ppp accumulates the number of
 *  rows above each part, so can find correct row number.
 */

  for (jcl = 0; jcl < Decomp->n; jcl++) {
     j = inv_cl_p[jcl];
     k = colinfo[j];

     if ( j != objective_column ) {
        phd = (k+1)/2; k = phd ;

        for (ipos = LpG->jcol[j]; ipos < LpG->jcol[j+1]; ipos++) {
           i   = LpG->irow[ipos];
           irw = row_perm[i];

          if(rowinfo[i] <= 2*phd)
          {
           if ( rowinfo[i] <= 0 )
              isp = irw;
           else if (rowinfo[i] < 2*phd-1 )
           {
            ppp=0;
            for(iii=0; iii < (rowinfo[i]-1)/2 ; iii++)
            {ppp= ppp + Decomp->LastRow[2*iii+1]- Decomp->FirstRow[2*iii+1];}
             isp = irw + Decomp->LastRow[0] - Decomp->FirstRow[rowinfo[i]]+ppp;
           }
              else
              {
               ppp=0;
               for(iii=0; iii < phd-1 ; iii++)
                ppp= ppp + Decomp->LastRow[2*iii+1]- Decomp-FirstRow[2*iii+1];

               isp = irw + Decomp->LastRow[0] - Decomp->FirstRow[2*k-1]+ppp;
              }
          }
          else
          {
            ppp=0;
            for(iii= 1; iii < (rowinfo[i]+1)/2; iii++)
              ppp = ppp + Decomp->LastRow[2*iii-1]- Decomp->FirstRow[2*iii-1];

           isp = Decomp->LastRow[2*phd] - Decomp->FirstRow[2*phd] + irw +
                   Decomp->LastRow[0] - Decomp->FirstRow[rowinfo[i]]+ ppp ;
          }

          jsp = SPb_n[k];

          LpSub[k]->rwnmbs[SPb_nz[k]] = isp;
          LpSub[k]->coeffs[SPb_nz[k]] = LpG->aa[ipos];

          SPb_nz[k]++;
        }

        LpSub[k]->lo_bnds[jsp] = LpG->lb[j];
        LpSub[k]->up_bnds[jsp] = LpG->ub[j];
        LpSub[k]->clpnts[jsp + 1] = SPb_nz[k];
        SPb_n[k]++;
      }
    }

  /* Check the consistency of data. */

    for (k = 0; k <= nb_blocks; k++) {
      if ( SPb_n[k] != LpSub[k]->n ) {
         printf("Error: SubPb %d has incorrect number of columns %d.\n",
           k, SPb_n[k]);
          }

      for (j = 0; j < LpSub[k]->n; j++) {
         if ( LpSub[k]->lo_bnds[j] > LpSub[k]->up_bnds[j] ) {
           printf("Error: SubPb %d, column %d: LO_bnd exceeds UP_bnd.\n", k, j);
          }
      }
      if ( SPb_nz[k] != LpSub[k]->nz ) {
         printf("Error: SubPb %d has incorrect number of entries %d.\n",
           k, SPb_nz[k]);
      }
  }
  if ( Dcmp_ErrDim > 0 ) {
     printf("Error in SplitData_DW: Wrong SubProblems dimensions.\n");
     return 0;
  }
  if ( Dcmp_ErrBnd > 0 ) {
     printf("Error in SplitData_DW: LpSub is infeasible (LO_bnd exceeds
                         UP_bnd).\n");
     return 0;
  }

  /*  Second main loop for constraints. Split Data of GAMS LP problem into
   *  SubProblems.
   *  Set up all row related data in SubProblem LP matrices.
   *  Notation:
   *  i   denotes row index in the GAMS LP matrix;
   *  irw denotes row index in the permuted matrix;
   *  isp denotes row index in the SubProblem;
   *  k   denotes an index of SubProblem to which row i belongs.
   *  Starting from the first row, decide the row part number of rhs and sign
   *   of constraints
   *  Note that linking constraints should be included in all nb_subs parts.
   */

 ppp = 0;

 for(k=1; k<=nb_subs; k++)
 {
    for (irw = 0; irw < Decomp->m-1; irw++)
    {
      i = inv_rw_p[irw];
      if ( rowinfo[i] <= 0 )
      {
        isp = irw;

           LpSub[k]->row_type[isp] = LpG->isign[i];
            if ( i == objective_row )
              LpSub[k]->row_type[isp] = 3;
           LpSub[k]->rhs[isp]      = LpG->rhs[i];
        LpSub[k]->ranges[isp]    = 1.0e+30;

        SPb_m[k]++;
   }
   else
   {
    if (rowinfo[i] < 2*k-1 )
     {
      for(phd=1; phd <= nb_subs; phd++)
       {
        if(rowinfo[i] == 2*phd-1)
         {
          ppp = 0;
          for(iii=0; iii < phd-1 ; iii++)
           ppp= ppp + Decomp->LastRow[2*iii+1]- Decomp->FirstRow[2*iii+1];

          isp = irw + Decomp->LastRow[0] - Decomp->FirstRow[rowinfo[i]]+ppp;

          ppp = 0;

          LpSub[k]->row_type[isp]  = LpG->isign[i];
          LpSub[k]->rhs[isp]       = LpG->rhs[i];
          LpSub[k]->ranges[isp]    = 1.0e+30;

          SPb_m[k]++;
         }
       }
     }
     else
     {
       for(phd=1; phd <= nb_subs; phd++)
        {
         if(rowinfo[i] == 2*phd-1)
          {
            ppp =0;
            for(iii=0; iii < phd-1 ; iii++)
             ppp= ppp + Decomp->LastRow[2*iii+1]- Decomp->FirstRow[2*iii+1];

             if(phd == k)
             {
              isp = irw + Decomp->LastRow[0] - Decomp->FirstRow[rowinfo[i]]
                    + ppp;
             }
             else
             {
              isp = irw + Decomp->LastRow[0] - Decomp->FirstRow[rowinfo[i]]
                + (Decomp->LastRow[2*k] - Decomp->FirstRow[2*k]) + ppp;
             }

            ppp = 0;

            LpSub[k]->row_type[isp]  = LpG->isign[i];
            LpSub[k]->rhs[isp]       = LpG->rhs[i];
            LpSub[k]->ranges[isp]    = 1.0e+30;

            SPb_m[k]++;
          }
        }

       if(rowinfo[i]==2*k)
       {
          for(iii=0; iii < (rowinfo[i]+1)/2 - 1 ; iii++)
            ppp= ppp + Decomp->LastRow[2*iii+1]- Decomp->FirstRow[2*iii+1];

          isp = irw + Decomp->LastRow[0] - Decomp->FirstRow[rowinfo[i]-1]+ppp;

          ppp = 0;

          LpSub[k]->row_type[isp] = LpG->isign[i];
          LpSub[k]->rhs[isp]      = LpG->rhs[i];
          LpSub[k]->ranges[isp]   = 1.0e+30;

          SPb_m[k]++;
        }
      }
    }
   }
}

/* Store linking variable information in different vector */

 for(k=1; k<=nb_subs;k++)
  {
   for(j=0; j<LSPb_n[k];j++)
    {
     if(LpSub[k]->rwnmbs[LpSub[k]->clpnts[j]] == 0)
       LpSub[k]->Lobjcoef[j]= -LpSub[k]->coeffs[LpSub[k]->clpnts[j]];
     else
       LpSub[k]->Lobjcoef[j]=0.00;

     LpSub[k]->Lup_bnds[j] = LpSub[k]->up_bnds[j];
          LpSub[k]->Llo_bnds[j] = LpSub[k]->lo_bnds[j];
   }
  }

 /* Define distributed machines and executable files in each machine*/
  where[1] = "pardist1.uwaterloo.ca";
  where[2] = "pardist2.uwaterloo.ca";
  where[3] = "pardist3.uwaterloo.ca";
  where[4] = "pardist4.uwaterloo.ca";

  if(nb_subs == 1) /* If only one processor is used */
   exefile[1] = "/u/hjpark/phdthesis/chrisjin1";
  else
   exefile[1] = "/u/hjpark/phdthesis/chrisjin";

  exefile[2] = "/u/hjpark/phdthesis/chrisjin";
  exefile[3] = "/u/hjpark/phdthesis/chrisjin";
  exefile[4] = "/u/hjpark/phdthesis/chrisjin";

  /* Ask each machine to start each process */
  for(k=1; k<=nb_subs;k++)
  {
   cc=pvm_spawn(exefile[k],0,1, where[k],1,&tid[k]);
   if(cc != 1)
    printf(" ERROR in spawning executables  ERROR code : %d !!! \n",cc);
  }

  tid[0] = pvm_mytid();

  for(k=0; k<=nb_subs;k++)
    aaa[k] = tid[k];
```

```
/* Pack initial data and send */
for(k=1; k <= nb_subs; k++)
{
  pvm_initsend(PvmDataDefault);
  info = pvm_pkint(&nb_subs, 1, 1);
  info = pvm_pkint(subtype, nb_subs+1, 1);
}

for(k=1; k <= nb_subs; k++)
 info = pvm_send(tid[k], 111);

for(k=1; k<=nb_subs;k++)
{
  pvm_initsend(PvmDataDefault);

  info = pvm_pkint (aaa,nb_subs+1,1);

  info = pvm_pklong (SPb_m, nb_subs+1, 1);
  info = pvm_pklong (SPb_n, nb_subs+1, 1);
  info = pvm_pklong (SPb_nz, nb_subs+1, 1);
  info = pvm_pklong (LSPb_m, nb_subs+1, 1);
  info = pvm_pklong (LSPb_n, nb_subs+1, 1);
  info = pvm_pklong (LpSub[k]->rwnmbs, SPb_nz[k], 1);
  info = pvm_pkdouble (LpSub[k]->coeffs, SPb_nz[k], 1);
  info = pvm_pklong (LpSub[k]->clpnts, SPb_n[k]+1, 1);
  info = pvm_pkdouble (LpSub[k]->lo_bnds, SPb_n[k], 1);
  info = pvm_pkdouble (LpSub[k]->up_bnds, SPb_n[k], 1);
  info = pvm_pklong (LpSub[k]->row_type, SPb_m[k],1);
  info = pvm_pkdouble (LpSub[k]->rhs,SPb_m[k],1);
  info = pvm_pkdouble (LpSub[k]->ranges, SPb_m[k],1);
  if (info)
   printf(" ERROR in packing data to define the subproblem !!!\n");
}

for(k=1; k <= nb_subs; k++)
{
  info = pvm_send(tid[k],1);
  if (info)
   printf(" ERROR in sending data to define the subproblem !!!\n");
}

pvm_initsend(PvmDataDefault);
for(k=1; k <=nb_subs; k++)
{
 info = pvm_pkdouble(LpSub[k]->Lobjcoef, LSPb_n[k],1);
 info = pvm_pkdouble(LpSub[k]->Lup_bnds, LSPb_n[k],1);
 info = pvm_pkdouble(LpSub[k]->Llo_bnds, LSPb_n[k],1);
 info = pvm_pklong(LpSub[k]->clpnts, LSPb_n[k]+1, 1);
 info = pvm_pklong(LpSub[k]->rwnmbs, LpSub[k]->clpnts[LSPb_n[k]],1);
 info = pvm_pkdouble(LpSub[k]->coeffs, LpSub[k]->clpnts[LSPb_n[k]],1);
}

for(k=1; k<=nb_subs ; k++)
{
 if(k != 1)
 info = pvm_send(tid[k],10);
}

pvm_initsend(PvmDataDefault);
for(k=1; k <=nb_subs; k++)
```

```
      {
       info = pvm_pklong(LpSub[k]->clpnts, LSPb_n[k]+1, 1);
       info = pvm_pklong(LpSub[k]->rwnmbs, LpSub[k]->clpnts[LSPb_n[k]],1);
       info = pvm_pkdouble(LpSub[k]->coeffs, LpSub[k]->clpnts[LSPb_n[k]],1);
      }

      for(k=1; k<=nb_subs ; k++)
      {
       if(k == 1)
        info = pvm_send(tid[k],10);
      }

      return 1 ;
   }
```

Appendix C C codes of WATPAR

We present the simplified C source codes for the parallel decomposition solver WATPAR for each subproblem of 4 part cases. The first subproblem gives more detailed explanation than the other 3 subproblems since the others are very similar except handling primal or dual information and updating the subproblems. Processors 1 to 4 solves lower-lower bound subproblem, upper-upper bound subproblem, lower-upper bound subproblem, and upper-lower bound subproblem respectively.

C.1 Processor 1 : Lower-Lower Bound Subproblem

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include  "/.software/arch/pvm-3/distribution/include/pvm3.h"
#include  "/.software/arch/cplex-6.0.1/distribution/cplex.h"
#include  "LpSub.h"

int     nb_subs, cur_block = 1;
int     iter_count = 1, lev2_count=0;
double  tol = 0.0001;

..... DEFINE GLOBAL VARIABLES AND ALLOCATE MEMORIES .....

double myseconds(double acumt)  /* Time measurement function */
{
 double ldt;
 gettimeofday(&tv2, (struct timezone*)0);
 ldt=(double) (tv2.tv_sec - tv1.tv_sec)*1000000+tv2.tv_usec - tv1.tv_usec;
 acumt = acumt + ldt ;
 return (acumt);
}

double CompuPiB(LpSubProb **LpSub)  /* Compute PiB to send */
{
 int i, j ;
 LpSub[cur_block]->PiB = 0.000;

  for(i=0; i<SPb_m[cur_block]-LOWSPb_m[cur_block]-HISPb_m[cur_block-1]-1; i++)
  {
   if(pi[i] != 0.0 && LpSub[cur_block]->rhs[i+HISPb_m[cur_block-1]+1] != 0.0)
   LpSub[cur_block]->PiB+=pi[i]*LpSub[cur_block]->rhs[i+HISPb_m[cur_block-1]+1];
  }

  for(j=LSPb_n[cur_block]; j < SPb_n[cur_block]; j++)
  {
   if(cstat[j] == 0)
   {
    if(LpSub[cur_block]->lo_bnds[j] != 0.0 && dj[j] != 0.0)
     LpSub[cur_block]->PiB=LpSub[cur_block]->PiB+
                                             LpSub[cur_block]->lo_bnds[j]*
                                        (dj[j]);
   }

   else if(cstat[j] == 2)
   {
    if(LpSub[cur_block]->up_bnds[j] != 0.0 && dj[j] != 0.0)
     LpSub[cur_block]->PiB=LpSub[cur_block]->PiB+LpSub[cur_block]->
                                                           up_bnds[j]*dj[j];

   }
  }
  return(LpSub[cur_block]->PiB);
 }

 double *CompuPiD(LpSubProb **LpSub) /* Compute PiD to send */
 {
  int j,p;

  for(j=0; j<LSPb_n[cur_block] ; j++)
  {
   LpSub[cur_block]->PiD[j]=0.0;
   for(p=LpSub[cur_block]->clpnts[j] ; p<LpSub[cur_block]->clpnts[j+1]; p++)
    if(pi[LpSub[cur_block]->rwnmbs[p]-1-HISPb_m[cur_block-1]] != 0.0 &&
      LpSub[cur_block]->rwnmbs[p]>HISPb_m[cur_block-1] &&
      LpSub[cur_block]->rwnmbs[p] < SPb_m[cur_block]-LOWSPb_m[cur_block])
        LpSub[cur_block]->PiD[j] += pi[LpSub[cur_block]->rwnmbs[p]-1-
                               HISPb_m[cur_block-1]]*LpSub[cur_block]->coeffs[p];

  }
  return(LpSub[cur_block]->PiD);
 }

 double *CompuOMeL(LpSubProb **LpSub) /* Compute OmegaL */
 {
  int j, k, p;
  for(j=0; j < HISPb_n[nb_subs]; j++)
    LpSub[cur_block]->OMeL[j] = 0.0;

  for(k=1; k <= nb_subs ; k++)
  {
   if(k < cur_block)
   {
    for(j=0; j<LSPb_n[k]; j++)
    {
     for(p=LpSub[k]->clpnts[j] ;p<LpSub[k]->clpnts[j+1] ;p++)
      if(LpSub[k]->rwnmbs[p] >= SPb_m[k]-LOWSPb_m[cur_block-1] &&
         LpSub[k]->rwnmbs[p] < SPb_m[k] && pi[LpSub[k]->rwnmbs[p]-SPb_m[k] +
                             LOWSPb_m[cur_block-1]+LSPb_m[cur_block-1]] != 0.0)
      LpSub[cur_block]->OMeL[j+HISPb_n[k-1]] += pi[LpSub[k]->rwnmbs[p]-
       SPb_m[k]+LOWSPb_m[cur_block-1]+LSPb_m[cur_block-1]]*LpSub[k]->coeffs[p];
    }
   }

   if(k >= cur_block)
   {
    for(j=0; j<LSPb_n[k]; j++)
    {
     for(p=LpSub[k]->clpnts[j] ;p<LpSub[k]->clpnts[j+1] ;p++)
    if(LpSub[k]->rwnmbs[p] > HISPb_m[cur_block-1] &&
       LpSub[k]->rwnmbs[p] <= HISPb_m[cur_block] &&
       pi[LpSub[k]->rwnmbs[p]-HISPb_m[cur_block-1]-1] != 0.0)
      LpSub[cur_block]->OMeL[j+HISPb_n[k-1]] += pi[LpSub[k]->rwnmbs[p]-
                                      HISPb_m[cur_block-1]-1]*LpSub[k]-
                               >coeffs[p];
   }
  }
 }
 return(LpSub[cur_block]->OMeL);
}

/* Optimize, obtain solutions and write them to output file */
int SolveProb(CPXENVptr env, CPXLPptr lp)
{
  int i, j, k, info, bufid, status, sitcnt, solstat; double dt1;

  gettimeofday(&tv1, (struct timezone*)0);
   status = CPXprimopt(env,lp);
   if ( status ) {
      char  errmsg[1024];
      CPXgeterrorstring (env, status, errmsg);
      fprintf (stderr, "%s", errmsg);
      fprintf (stderr, "Failed to optimize LP.\n");
      return(status);
  }
  gettimeofday(&tv2, (struct timezone*)0);

  dt1=(double) (tv2.tv_sec - tv1.tv_sec) * 1000000 + tv2.tv_usec - tv1.tv_usec;

  fprintf(fp, "Real solution time :  %8f\n",  dt1/1000000);
  at1 = at1 +  dt1;
  fprintf(fp, "TOTAL solution time :  %8f\n",  at1/1000000);

  cur_numrows = CPXgetnumrows (env, lp);
  cur_numcols = CPXgetnumcols (env, lp);

  status=CPXsolution (env, lp, &solstat, &objval[cur_block], x, pi, slack, dj);
  if ( status ) {
      fprintf (stderr, "Failed to obtain solution 1.\n");
      return(status);
  }

  status = CPXgetbase (env, lp, cstat, rstat);
  if ( status ) {
      fprintf (stderr, "Failed to obtain basis.\n");
      return(status);
  }

  sitcnt = CPXgetitcnt(env,lp);
  fprintf(fp1,"\n***** Iteration Number : %d,  %d  \n", iter_count, lev2_count);
  fprintf (fp,"\nSolution status = %d\n", solstat);
  fprintf (fp,"Objective value  = %f\n\n", objval[cur_block]);
  fprintf(fp1, " Simplex Iteration Count : %d \n", sitcnt);

  for (i = 0; i < cur_numrows; i++) {
      fprintf (fp,"Row %d:  Slack = %f  Pi = %f  rstat =%d\n", i, slack[i],
pi[i],rstat[i]);

  for (j = 0; j < cur_numcols; j++) {
      fprintf (fp,"Column %d:  Value = %f  Reduced cost = %f  cstat = %d\n",
                j, x[j], dj[j],cstat[j]);

  return(status);
}

short
LowerBound (LpSubProb **LpSub)
{

  .... Declare and allocate space for the variables and arrays ....

 /* Receive initial data for linking variables of other parts */
   gettimeofday(&tvl, (struct timezone*)0);
   bufid = pvm_recv(ptid,10);
   rdtl = myseconds(rdtl);
   fp = fopen("/u/hjpark/phdthesis/chrisjin.out","a");
    fprintf(fp, " Receiving Set up Data Time : %8f\n", rdtl/1000000);
   fclose(fp);

   for(k=1; k<= nb_subs;k++)
   {
    info = pvm_upklong(LpSub[k]->clpnts, LSPb_n[k]+1, 1);
    info = pvm_upklong(LpSub[k]->rwnmbs, LpSub[k]->clpnts[LSPb_n[k]],1);
    info = pvm_upkdouble(LpSub[k]->coeffs, LpSub[k]->clpnts[LSPb_n[k]],1);
   }

 /* In order to load the subproblem to CPLEX, convert data (sense) to CPLEX
      format and retrieve the vector which indicates the number of nonzero
      elements (matcnt) in each column and objective function coefficients
      (objcoeffs)*/
   for(i=0; i < LpSub[cur_block]->m ;i++)
   {
    if(LpSub[cur_block]->row_type[i] == 0)
             sense[i]='E';
      else if(LpSub[cur_block]->row_type[i] == 1)
                  sense[i]='G';
          else if(LpSub[cur_block]->row_type[i] == 2)
                      sense[i]='L';
            else sense[i]='R';
   }

   for(j=0; j<LpSub[cur_block]->n;j++){
     matcnt[j]=(int) (LpSub[cur_block]->clpnts[j+1]
                                     -LpSub[cur_block]->clpnts[j]);
       if(LpSub[cur_block]->rwnmbs[LpSub[cur_block]->clpnts[j]] == 0)
        objcoeffs[j]= -LpSub[cur_block]->coeffs[LpSub[cur_block]->clpnts[j]];
       else
         objcoeffs[j]=0.000;
   }

   /* Initialize the CPLEX environment */

   env = CPXopenCPLEXdevelop (&status);
   if ( env == NULL ) {
    char  errmsg[1024];
       fprintf (stderr, "Could not open CPLEX environment.\n");
       CPXgeterrorstring (env, status, errmsg);
       fprintf (stderr, "%s", errmsg);
       goto TERMINATE;
    }
   /* Use advanced basis at each iteration */
   status = CPXsetintparam(env, CPX_PARAM_ADVIND, 1);
   if ( status ) {
      fprintf (stderr, "Failed to scale.\n");
      goto TERMINATE;
   }

  /* For barrier method, use option 1 to solve subproblems */
   status = CPXsetintparam(env, CPX_PARAM_BARALG, 1);
   if ( status ) {
      fprintf (stderr, "Failed to scale.\n");
      goto TERMINATE;
   }

   /* Create the problem. */
   lp = CPXcreateprob (env, &status, "3-region");
     if ( lp == NULL ) {
       fprintf (stderr, "Failed to create LP.\n");
       goto TERMINATE;
   }

  /* Now copy the problem data into CPLEX */
   status=CPXcopylp(env,lp,(int) LpSub[cur_block]->n,(int)LpSub[cur_block]->m,
     -1,objcoeffs, LpSub[cur_block]->rhs,sense,(int*)(LpSub[cur_block]-clpnts),
       matcnt, (int*) LpSub[cur_block]->rwnmbs,LpSub[cur_block]->coeffs,
       LpSub[cur_block]->lo_bnds, LpSub[cur_block]->up_bnds, NULL);
     if ( status ) {
      fprintf (stderr, "Failed to copy problem data.\n");
      goto TERMINATE;
   }

   /* Delete the first row because the original vector has always the objective
       function coefficients in row zero */
   status = CPXdelrows (env, lp, 0, 0);
   if ( status ) {
      fprintf (stderr, "Failed to delete row 0\n");
      goto TERMINATE;
   }

 /* Add an artificial variable to each linking constraints */
   for(k=1; k<=nb_subs; k++)
   {
    for(i=0; i < LSPb_m[k]; i++)
      LpSub[k]->arti_obj[i] = -arti ;
   }

   for(k=1; k<=nb_subs; k++)
   {
    if(LSPb_m[k] > 0)
     status = CPXnewcols(env, lp, (int) LSPb_m[k], LpSub[k]->arti_obj, NULL,
                                                                   NULL, NULL,
                                                            NULL);
   }

   for(i=0; i<LSPb_m[cur_block]; i++)
    status = CPXchgcoef(env,lp,(int) i,(int) (LpSub[cur_block]->n)+i,arti_coef);

   for(i=LpSub[cur_block]->m-LOWSPb_m[cur_block]-1; i<LpSub[cur_block]->m-1;i++)
    status=CPXchgcoef(env,lp,(int) i,(int)(LpSub[cur_block]->n)+
    LSPb_m[cur_block]+i-LpSub[cur_block]->m+LOWSPb_m[cur_block]+1 , arti_coef);
```

```
/* Optimize the problem and obtain solution. */
 status = SolveProb(env, lp);
 if(status) {
   goto TERMINATE;
 }

/* Add the sum-to-one constraints for Lamda variables */
for(k=1; k <= nb_subs; k++)
{
  if (subtype[k] == 1 )
   status = CPXnewrows (env, lp, 1, &Lam_rhs, &Lam_sense, NULL, NULL);
}


objval[2] = objval[1] + 10;
while(iter_count<150 && (objval[2]-objval[1]>tol)) /* Upper level iteration */
{
 iter_count = iter_count + 1;
 totlev2C = totlev2C + lev2_count;  /* Find the total number of proposals */

 fprintf(fp,"\n ***** Iteration Number :  %d  ***** \n", iter_count);

 /* Compute the dual proposals and send them to aggregated upper bound
     subproblem (processors 2 and 4) */
 LpSub[cur_block]->PiB = CompuPiB(LpSub);
 LpSub[cur_block]->PiD = CompuPiD(LpSub);
 LpSub[cur_block]->OMeL = CompuOMeL(LpSub);

 pvm_initsend(PvmDataRaw);
 info = pvm_pkdouble (&LpSub[cur_block]->PiB, 1, 1);
 info = pvm_pkdouble (LpSub[cur_block]->PiD, LSPb_n[cur_block], 1);
 info = pvm_pkdouble (LpSub[cur_block]->OMeL, HISPb_n[nb_subs],1);
  if(info < 0)
   fprintf(fp,"ERROR in packing data\n");

 for(k=1; k <= nb_subs; k++)
 {
  if(subtype[k] == 2)
   bufid = pvm_send(tid[k],5);
 }

 /* Receive primal proposals from aggregated upper bound subproblem */
 for(k=1; k<=nb_subs ; k++)
 {
  if(subtype[k] == 2 )
  {
   gettimeofday(&tvl, (struct timezone*)0);
   bufid = pvm_recv(tid[k], 2);
   rdtl = myseconds(rdtl);
   fprintf(fp, " Receiving Set up Data Time : %8f\n", rdtl/1000000);

   info = pvm_upkdouble (&LpSub[k]->CX, 1, 1);
   info = pvm_upkdouble (LpSub[k]->LY, HISPb_m[nb_subs],1);
  }
 }

 /* Add the sum of primal proposals and add one column to the subproblem */
 LpSub[2]->CX += LpSub[4]->CX ;

 for(i=0; i < HISPb_m[nb_subs] ; i++)
    LpSub[2]->LY[i] +=  LpSub[k]->LY[i];
```

```
status = CPXnewcols (env,lp, 1, &LpSub[2]->CX, &Lam_lo, &Lam_up, NULL, NULL);
if(status)
  fprintf(fp," ERROR in defining a new column!!!\n");

for(i=HISPb_m[0]; i <HISPb_m[cur_block] ; i++)
{
 status = CPXchgcoef (env, lp, i, LpSub[cur_block]->n + HISPb_m[nb_subs]+
           +iter_count-2+totlev2C, LpSub[2]->LY[i]);
}

for(i=LpSub[1]->m - LOWSPb_m[1]-1; i<LpSub[1]->m-1; i++)
{
 status = CPXchgcoef(env, lp, i, LpSub[cur_block]->n+HISPb_m[nb_subs]
            + iter_count-2 + totlev2C ,
      LpSub[2]->LY[i-LpSub[1]->m + LOWSPb_m[1]+1+HISPb_m[cur_block]]);
}

status = CPXchgcoef (env, lp, (int) (LpSub[cur_block]->m-1),
      LpSub[1]->n+HISPb_m[1]+LOWSPb_m[1]+iter_count-2+totlev2C, Lam_coef) ;

/* Lower level iteration. 'exitlev2' is a signal coming from lower-upper
   bound subproblem asking to exit lower level iteration */
while(lev2_count < 150 && (objval[3] - objval[1] > 0.0001 || lev2_count <1)
           && (lev2_count < 1 || exitlev2 <= 0 ))
{
 lev2_count = lev2_count + 1;

 if(lev2_count > 1)
 {
  LpSub[cur_block]->PiB = CompuPiB(LpSub);
  LpSub[cur_block]->OMeL = CompuOMeL(LpSub);
  LpSub[cur_block]->PiD = CompuPiD(LpSub);
 }

 for(i = 0 ; i < LSPb_m[cur_block]; i++)
   LpSub[cur_block]->Omega[i]  = pi[i];

 pvm_initsend(PvmDataRaw);
 info = pvm_pkdouble (&LpSub[cur_block]->PiB, 1, 1);
 info = pvm_pkdouble (LpSub[cur_block]->PiD, LSPb_n[cur_block], 1);
 info = pvm_pkdouble (LpSub[cur_block]->OMeL, HISPb_n[nb_subs],1);
 info = pvm_pkdouble (LpSub[cur_block]->Omega, LSPb_m[cur_block], 1);
  if(info < 0)
    fprintf(fp,"ERROR in packing data\n");

 bufid = pvm_send(tid[3],5);

 gettimeofday(&tv1, (struct timezone*)0);
 bufid = pvm_recv(tid[3], 2);
 rdt1 = myseconds(rdt1);
 fprintf(fp, " Receiving Set up Data Time : %8f\n", rdt1/1000000);

 info = pvm_upkdouble (&LpSub[3]->CX, 1, 1);
 info = pvm_upkdouble (LpSub[3]->LY, HISPb_m[nb_subs],1);

 /* Add a primal proposal from subproblem 3 (lower-upper bound subproblem) */
 status = CPXnewcols(env,lp, 1, &LpSub[3]->CX, &Lam_lo, &Lam_up, NULL, NULL);
 if(status)
  fprintf(fp," ERROR in defining a new column!!!\n");
```

```
    for(i=HISPb_m[0]; i <HISPb_m[cur_block] ; i++)
      status = CPXchgcoef (env, lp, i, LpSub[cur_block]->n +
            HISPb_m[nb_subs]+iter_count-2+totlev2C+lev2_count, LpSub[3]->LY[i]);

    for(i=LpSub[1]->m - LOWSPb_m[1]-1; i<LpSub[1]->m-1; i++)
    {
     status = CPXchgcoef(env, lp, i, LpSub[cur_block]->n+HISPb_m[nb_subs]
              + iter_count-2 + totlev2C+lev2_count ,
        LpSub[3]->LY[i-LpSub[1]->m + LOWSPb_m[1]+1+HISPb_m[cur_block]]);
    }

    status = CPXchgcoef (env, lp, (int) (LpSub[cur_block]->m-1+1),
       LpSub[1]->n+HISPb_m[1]+LOWSPb_m[1]+iter_count-2+totlev2C+lev2_count,
        Lam_coef) ;

    status = SolveProb(env, lp);

    pvm_initsend(PvmDataRaw);
    info = pvm_pkdouble (&objval[cur_block],1 ,1);
    bufid = pvm_send (tid[3],7);

    gettimeofday(&tv1, (struct timezone*)0);
    bufid = pvm_recv (tid[3],7);
    rdt1 = myseconds(rdt1);
    fprintf(fp, " Receiving Set up Data Time : %8f\n", rdt1/1000000);

    info = pvm_upkdouble (&objval[3],1,1);
    info = pvm_upkint (&exitlev2, 1,1);
    if(info < 0)
     fprintf(fp, " ERROR in unpacking objval[3]\n", 4);
  }
}

 pvm_initsend(PvmDataRaw);
 info = pvm_pkdouble (&objval[cur_block],1 ,1);
 for(k=1; k <=nb_subs; k++)
  {
   if(k != cur_block)
   bufid = pvm_send (tid[k],8);
   if(bufid <0)
     fprintf(fp, "ERROR in sending objval[%d]\n",k);
  }

 for(k=1; k <=nb_subs; k++)
  {
   if(k != cur_block )
   {
    gettimeofday(&tv1, (struct timezone*)0);
    bufid = pvm_recv (tid[k],8);
    rdt1 = myseconds(rdt1);
    fprintf(fp, " Receiving Set up Data Time : %8f\n", rdt1/1000000);

    info = pvm_upkdouble (&objval[k],1,1);
    if(info < 0)
     fprintf(fp, " ERROR in unpacking objval[%d]\n",k);
   }
  }
 }

  pvm_initsend(PvmDataDefault);
  info = pvm_pkdouble (&objval[cur_block], 1,1);
```

```
    bufid = pvm_send(ptid, 7);


    TERMINATE:
    .... Free up the problem as allocated by CPXcreateprob ....

  return ;
 }

main(void)
{
 ..... DEFINE ALL THE NECESSARY VARIABLES AND ALLOCATE NECESSARY MEMORIES .....

   /* Record program start time */
   gettimeofday(&tv1, (struct timezone*)0);
   idt = (double) (tv1.tv_sec) * 1000000 + tv1.tv_usec;

   ptid = pvm_parent();

   /* Record the idle time to receive the initial data */
   gettimeofday(&tv1, (struct timezone*)0);
   bufid = pvm_recv (ptid, 111);    /* Receive data */
   rdt1 = myseconds(rdt1);
   fp = fopen("/u/hjpark/phdthesis/chrisjin.out","w");
    fprintf(fp, " Receiving Set up Data Time : %8f\n", rdt1/1000000);
   fclose(fp);

   bufid = pvm_upkint  (&nb_subs, 1, 1);  /* Unpack data */
    if(bufid < 0)
    {
     fp = fopen("/u/hjpark/phdthesis/chrisjin.out","a");
     fprintf(fp, "ERROR in unpacking nb_subs\n");
     fclose(fp);
    }

   bufid = pvm_upkint  (subtype, nb_subs+1,1);
    if(bufid < 0)
    {
     fp = fopen("/u/hjpark/phdthesis/chrisjin.out","a");
     fprintf(fp, "ERROR in unpacking subtype\n");
     fclose(fp);
    }

   gettimeofday(&tv1, (struct timezone*)0);
   bufid = pvm_recv (ptid, 1);                    /* Receive initial problem data */
   rdt1 = myseconds(rdt1);
   fp = fopen("/u/hjpark/phdthesis/chrisjin.out","a");
    fprintf(fp, " Receiving Set up Data Time : %8f\n", rdt1/1000000);
   fclose(fp);

   info = pvm_upkint  (aaa, nb_subs+1,1);
   for(k=1; k<=nb_subs ;k++)
    tid[k] = aaa[k];

   info = pvm_upklong (SPb_m, nb_subs+1, 1);
   info = pvm_upklong (SPb_n, nb_subs+1, 1);
   info = pvm_upklong (SPb_nz, nb_subs+1, 1);
   info = pvm_upklong (LSPb_m, nb_subs+1, 1);
   info = pvm_upklong (LSPb_n, nb_subs+1, 1);
    if (info)
    {
```

```
    fp = fopen("/u/hjpark/phdthesis/chrisjin.out","a");
    fprintf(fp," ERROR in unpacking data to define the subproblem !!!\n");
    fclose(fp);
   }

  /* Find the number of accumulated linking variables and constraints */
   HISPb_m = (long *) malloc((nb_subs + 1)*sizeof(long));
   HISPb_n = (long *) malloc((nb_subs + 1)*sizeof(long));

   HISPb_m[0] = 0;
   HISPb_n[0] = 0;

   for(k=1; k<=nb_subs; k++)
   {
     HISPb_m[k] = HISPb_m[k-1]+LSPb_m[k];
     HISPb_n[k] = HISPb_n[k-1]+LSPb_n[k];
   }

   for(k=1; k <= nb_subs; k++)
    LOWSPb_m[k] = HISPb_m[nb_subs] - HISPb_m[k];

   info = pvm_upklong ( LpSub[cur_block]->rwnmbs, SPb_nz[cur_block], 1);
   info = pvm_upkdouble ( LpSub[cur_block]->coeffs, SPb_nz[cur_block], 1);
   info = pvm_upklong ( LpSub[cur_block]->clpnts, SPb_n[cur_block]+1, 1);
   info = pvm_upkdouble ( LpSub[cur_block]->lo_bnds, SPb_n[cur_block], 1);
   info = pvm_upkdouble ( LpSub[cur_block]->up_bnds, SPb_n[cur_block], 1);
   info = pvm_upklong ( LpSub[cur_block]->row_type, SPb_m[cur_block],1);
   info = pvm_upkdouble (LpSub[cur_block]->rhs,SPb_m[cur_block],1);
   info = pvm_upkdouble (LpSub[cur_block]->ranges, SPb_m[cur_block],1);
   if (info < 0)
    {
     fp = fopen("/u/hjpark/phdthesis/chrisjin.out","a");
     fprintf(fp," ERROR in unpacking data to define the subproblem !!!\n");
     fclose(fp);
    }

    LowerBound (LpSub);

    /* Take the program end time and measure total program time */
    gettimeofday(&tv1, (struct timezone*)0);

   edt = (double) (tv1.tv_sec) * 1000000 + tv1.tv_usec;

   fp = fopen("/u/hjpark/phdthesis/chrisjin.out","a");
     fprintf(fp, "Total program time :  %8f\n",  (edt-idt)/1000000);
   fclose(fp);
  }
```

****

C.2 Processor 2 : Upper-Upper Bound Subproblem

```
.... LINK LIBRARIES AND HEADER FILES. DEFINE VARIABLES AND ALLOCATE MEMORIES ...

 * Compute the primal information (LY and BX) to send */
short CompuLY (LpSubProb **LpSub) {
 int i,j, p;

 for(i=0; i < HISPb_m[nb_subs]; i++)
```

```
  LpSub[2]->LY[i]=0.0;

 for(j=0; j < LSPb_n[2]; j++)
 {
  if(x[j] != 0.0)
  {
   for(p = LpSub[2]->clpnts[j]; p < LpSub[2]->clpnts[j+1];p++)
   {
    if(LpSub[2]->rwnmbs[p] > 0 && LpSub[2]->rwnmbs[p]  <= HISPb_m[2])
      LpSub[2]->LY[LpSub[2]->rwnmbs[p]-1] += LpSub[2]->coeffs[p] * x[j];

    else if(LpSub[2]->rwnmbs[p]>=SPb_m[2]-LOWSPb_m[2]
                                          && LpSub[2]->rwnmbs[p] < SPb_m[2])
      LpSub[2]->LY[LpSub[2]->rwnmbs[p]-1-SPb_m[2]+LOWSPb_m[2]+1+HISPb_m[2]] +=
                                          LpSub[2]->coeffs[p] * x[j];
   }
  }
 }

 for(j=LSPb_n[2]; j < SPb_n[2]; j++)
 {
  if(x[j] != 0.0)
   for(p = LpSub[2]->clpnts[j]; p < LpSub[2]->clpnts[j+1];p++)
     if(LpSub[2]->rwnmbs[p]> HISPb_m[1] && LpSub[2]->rwnmbs[p] <= HISPb_m[2])
     LpSub[2]->LY[LpSub[2]->rwnmbs[p]-1] += LpSub[2]->coeffs[p] * x[j];
 }

 return;
}

short SolveProb(CPXENVptr env, CPXLPptr lp)
{
 .... SOLVE THE SUBPROBLEM AND WRITE RESULTS TO OUTPUT FILE ....
 return(lstatus);
} ,

 * Add lower level cut using dual proposal of upper-lower bound subproblem
   (processor 4). A new cut is always added at the end of the constraints, so
   finding the total number of constraints is important*/
short Addlev2Cuts(LpSubProb **LpSub, CPXENVptr env, CPXLPptr lp)
{
   int i, j, k, lbufid, linfo, temp3, lstatus;

   gettimeofday(&tv1, (struct timezone*)0);
   lbufid = pvm_recv(tid[nb_subs],5);
   rdt1 = myseconds(rdt1);
    fprintf(fp, " Receiving Set up Data Time : %8f\n", rdt1/1000000);

   linfo = pvm_upkdouble (&LpSub[nb_subs]->PiB, 1 , 1);
   linfo = pvm_upkdouble (LpSub[nb_subs]->PiD, LSPb_n[nb_subs], 1);
   linfo = pvm_upkdouble (LpSub[nb_subs]->OMeL, HISPb_n[nb_subs],1);
   if(linfo < 0)
    fprintf(fp," ERROR in unpacking LpSub->PiD \n");

   lstatus=CPXnewrows (env, lp, 1, &LpSub[nb_subs]->PiB, &cut_sense,NULL,NULL);

   for(j=0; j < HISPb_n[1] ;j++)
   {
     lstatus = CPXchgcoef (env, lp,(int) (SPb_m[2]-LOWSPb_m[2]
        -HISPb_m[1] -1+totlev2C+lev2_count + iter_count-2), (int) (j+SPb_n[2]
               +LSPb_m[2]),LpSub[nb_subs]->OMeL[j]) ;
   }

   for(j=SPb_n[2]+LSPb_m[2]+HISPb_n[1];
                 j<SPb_n[2]+LSPb_m[2]+HISPb_n[nb_subs]-LSPb_n[2];j++)
   {
    lstatus = CPXchgcoef (env, lp, (int) (SPb_m[2]-LOWSPb_m[2]
                -HISPb_m[1] -1+totlev2C+lev2_count + (iter_count-2)), j,
            LpSub[nb_subs]->OMeL[j+HISPb_n[2]-SPb_n[2]-LSPb_m[2]-HISPb_n[1]]) ;
   }

   for(j=0; j<LSPb_n[2];j++)
   {
    lstatus = CPXchgcoef(env,lp, (int) (SPb_m[2]-LOWSPb_m[2]
                -HISPb_m[1] -1+totlev2C+lev2_count + (iter_count-2)), j,
                 LpSub[nb_subs]->OMeL[j+HISPb_n[1]]);
   }

   for(j=SPb_n[2]+LSPb_m[2]+HISPb_n[nb_subs-1]-LSPb_n[2];
              j<SPb_n[2]+LSPb_m[2]+HISPb_n[nb_subs]-LSPb_n[2]; j++)
   {
    lstatus = CPXchgcoef (env, lp, (int) (SPb_m[2]-LOWSPb_m[2]
                -HISPb_m[1]-1+totlev2C+lev2_count + iter_count-2), j,
       LpSub[nb_subs]->PiD[j-SPb_n[2]-LSPb_m[2]-HISPb_n[nb_subs-1]+LSPb_n[2]]) ;
   }

   lstatus = CPXchgcoef (env, lp, (int) (SPb_m[2]-LOWSPb_m[2]-HISPb_m[1]
                  -1+totlev2C+lev2_count + iter_count-2),
            LpSub[2]->n+LSPb_m[2]+HISPb_n[nb_subs]-LSPb_n[2]+1, Theta_coef);

   return;
}

short
UpperBound (LpSubProb **LpSub)
{
 .... DEFINE LOCAL VARIABLES AND ALLOCATE MEMEORIES .....

 /* Receive information of linking variables and constraints of other parts */
   for(k=1; k<= nb_subs;k++)
   {
    info = pvm_upkdouble(LpSub[k]->Lobjcoef, LSPb_n[k],1);
    info = pvm_upkdouble(LpSub[k]->Lup_bnds, LSPb_n[k],1);
    info = pvm_upkdouble(LpSub[k]->Llo_bnds, LSPb_n[k],1);
    info = pvm_upklong(LpSub[k]->clpnts, LSPb_n[k]+1, 1);
    info = pvm_upklong(LpSub[k]->rwnmbs, LpSub[k]->clpnts[LSPb_n[k]],1);
    info = pvm_upkdouble(LpSub[k]->coeffs, LpSub[k]->clpnts[LSPb_n[k]],1);
   }

... CONVERT ORIGINAL DATA sense, objcoeffs and matcnt TO CONFORM CPLEX ....

.... INITIALIZE THE CPLEX ENVIRONMENT ....

     * Now copy the problem data into the lp */

   status=CPXcopylp(env,lp,(int)LpSub[2]->n,(int) LpSub[2]->m,
       -1,objcoeffs, LpSub[2]->rhs,sense, (int*) (LpSub[2]->clpnts), matcnt,
       (int*) LpSub[2]->rwnmbs, LpSub[2]->coeffs,LpSub[2]->lo_bnds,
                                                  LpSub[2]->up_bnds, NULL);

   /* Make the upper-upper bound subproblem format of the first iteration by
       deleting row zero and other parts' linking constraints. Add artificial
    variables and add other part's linking variables*/
status = CPXdelrows (env, lp, 0, 0);
if ( status ) {
   fprintf (stderr, "Failed to delete row 0\n");
   goto TERMINATE;
}

if(HISPb_m[1] >= 1)
 status = CPXdelrows (env, lp, 0, HISPb_m[1]-1);

if(nb_subs > 2)
  status = CPXdelrows (env, lp, (int) (SPb_m[2]-HISPb_m[1]-LOWSPb_m[2]-1),
                   (int) (SPb_m[2]-HISPb_m[1]-2));

status=CPXnewcols(env,lp,(int) LSPb_m[2], arti_obj, NULL, NULL, NULL, NULL);

for(i=0; i<LSPb_m[2]; i++)
    status = CPXchgcoef(env, lp, i, (int) (LpSub[2]->n)+i, arti_coef);

for(k=1;k<=nb_subs;k++)
{
 if(k != 2)
 {
  if(LSPb_n[k] > 0)
   status = CPXnewcols (env,lp,(int) LSPb_n[k], LpSub[k]->Lobjcoef,
                 LpSub[k]->Llo_bnds, LpSub[k]->Lup_bnds, NULL, NULL);
 }

 if(k < 2)
  for(j=0; j < LSPb_n[k] ; j++)
  {
   if(LSPb_n[k] > 0)
   for(p=LpSub[k]->clpnts[j]; p < LpSub[k]->clpnts[j+1]; p++)
   {
    if(LpSub[k]->rwnmbs[p] >= SPb_m[k]-LOWSPb_m[k] &&
          LpSub[k]->rwnmbs[p] < SPb_m[k]-LOWSPb_m[2])
      status = CPXchgcoef(env, lp, LpSub[k]->rwnmbs[p]-SPb_m[k]+LOWSPb_m[k],
                SPb_n[2]+LSPb_m[2]+HISPb_n[k-1]+j,LpSub[k]->coeffs[p]);
   }
  }

 if(k > 2)
  for(j=0; j < LSPb_n[k] ; j++)
  {
   for(p=LpSub[k]->clpnts[j]; p < LpSub[k]->clpnts[j+1]; p++)
   {
    if(LpSub[k]->rwnmbs[p] > HISPb_m[1]
       && LpSub[k]->rwnmbs[p] <= HISPb_m[cur_block])
      status = CPXchgcoef(env, lp, LpSub[k]->rwnmbs[p]-HISPb_m[1]-1,
            SPb_n[2]+LSPb_m[2]+HISPb_n[k-1]-LSPb_n[2]+j,LpSub[k]->coeffs[p]);
   }
  }
 }
}

/* Optimize the problem and obtain solution. */
status = SolveProb(env, lp);
if(status) {
  goto TERMINATE;
}
```

```
 /* Add theta variable */
 for(k=1; k <= nb_subs; k++)
  if (subtype[k] == 2 )
   status = CPXnewcols (env,lp, 1, &Theta_coef, &Theta_lo, NULL,NULL, NULL);

objval[1]= objval[2]-10;

while ((iter_count<150) && (objval[2] - objval[1] > tol ) )
{
 iter_count = iter_count + 1;

 totlev2C = totlev2C + lev2_count;

 fprintf(fp,"\n ***** Iteration Number :  %d  ***** \n", iter_count);

 /* Compute CX to send aggregated lower bound subproblem */
 LpSub[2]->CX=0.00;
 for(j=0; j < SPb_n[2]; j++)
  if(x[j] != 0.0 && objcoeffs[j] != 0.0)
   LpSub[2]->CX += objcoeffs[j]*x[j];

 status = CompuLY(LpSub);

 pvm_initsend(PvmDataRaw);
 info = pvm_pkdouble (&LpSub[2]->CX, 1, 1);
 info = pvm_pkdouble (LpSub[2]->LY, HISPb_m[nb_subs],1);

 for(k=1; k <= nb_subs; k++)
  if(subtype[k] == 1)
   info = pvm_send(tid[k],2);

 for(k=1; k<=nb_subs; k++)
  {
   if(subtype[k] == 1)
   {
    gettimeofday(&tv1, (struct timezone*)0);
    bufid = pvm_recv(tid[k],5);
    rdt1 = myseconds(rdt1);
    fprintf(fp, " Receiving Set up Data Time : %8f\n", rdt1/1000000);

    info = pvm_upkdouble (&LpSub[k]->PiB, 1, 1);
    info = pvm_upkdouble (LpSub[k]->PiD, LSPb_n[k], 1);
    info = pvm_upkdouble (LpSub[k]->OMeL, HISPb_n[nb_subs],1);
   }
  }

 /* Add the dual proposals coming from the aggregated lower bound subproblem
   (processors 1 and 3) to make one upper level cut */
 for(k=1; k<=nb_subs ; k++)
 {
  if(k > 2 && subtype[k] == 1 )
  {
   LpSub[1]->PiB = LpSub[1]->PiB + LpSub[3]->PiB ;

   for(i=HISPb_n[1]; i < HISPb_n[2] ; i++)
     LpSub[1]->OMeL[i] = LpSub[1]->OMeL[i]+ LpSub[3]->OMeL[i];

   for(i=HISPb_n[3]; i < HISPb_n[nb_subs] ; i++)
     LpSub[1]->OMeL[i] = LpSub[1]->OMeL[i]+ LpSub[3]->OMeL[i];

   for(j=0; j<LSPb_n[1]; j++)
```

```
      LpSub[1]->PiD[j] = LpSub[1]->PiD[j]+LpSub[3]->OMeL[j];

    for(j=0; j<LSPb_n[3]; j++)
      LpSub[3]->PiD[j] = LpSub[3]->PiD[j]+LpSub[1]->OMeL[j+HISPb_n[2]];
   }
  }

 /* Add a cut for upper level iteration */
  status = CPXnewrows (env, lp, 1, &LpSub[1]->PiB, &cut_sense, NULL, NULL);

  for(j=SPb_n[2]+LSPb_m[2]+HISPb_n[1];
                        j<SPb_n[2]+LSPb_m[2]+HISPb_n[nb_subs]-LSPb_n[2];j++)
    status = CPXchgcoef (env, lp, (int) (SPb_m[2]-LOWSPb_m[2]
               -HISPb_m[1] -1+(iter_count-2)+totlev2C), j,
                LpSub[1]->OMeL[j+HISPb_n[2]-SPb_n[2]-LSPb_m[2]-HISPb_n[1]]) ;

  for(j=0; j<LSPb_n[2];j++)
    status = CPXchgcoef(env,lp, (int) (SPb_m[2]-LOWSPb_m[2]
               -HISPb_m[1] -1+(iter_count-2)+totlev2C), j,
                LpSub[1]->OMeL[j+HISPb_n[1]]);

  for(j=SPb_n[2]+LSPb_m[2];
                        j<SPb_n[2]+LSPb_m[2]+HISPb_n[1]; j++)
    status = CPXchgcoef (env, lp, (int) (SPb_m[2]-LOWSPb_m[2]- HISPb_m[1]-
            1+iter_count-2+totlev2C),j,LpSub[1]->PiD[j-SPb_n[2]-LSPb_m[2]]) ;

     if(nb_subs > 2 && subtype[3] == 1)
     {
      for(j=SPb_n[2]+LSPb_m[2]+LSPb_n[1];
                     j<SPb_n[2]+LSPb_m[2]+LSPb_n[1]+LSPb_n[3]; j++)
      {
       status = CPXchgcoef(env,lp,(int)(SPb_m[2]-LOWSPb_m[2]-HISPb_m[1]-1+
              iter_count-2+totlev2C),j,LpSub[3]->PiD[j-SPb_n[2]-LSPb_m[2]-
                                                            LSPb_n[1]]) ;
      }

      for(j=0; j<LSPb_n[4];j++)
       status = CPXchgcoef(env,lp, (int) (SPb_m[2]-LOWSPb_m[2]
               -HISPb_m[1] -1+(iter_count-2)+totlev2C),
                   j+SPb_n[2]+LSPb_m[2]+LSPb_n[1]+LSPb_n[3],
                   LpSub[1]->OMeL[j+HISPb_n[3]]);

    }

   status = CPXchgcoef (env, lp, (int) (SPb_m[2]-LOWSPb_m[2]-HISPb_m[1]-
            1+(iter_count-2)+totlev2C),
             (int) (LpSub[2]->n+LSPb_m[2]+HISPb_n[nb_subs]-LSPb_n[2]),
                   Theta_coef);

 nbufid = 0; lev2_count = 0;  com_sub = 3; exitlev2=0; lev2_block = 4;

 objval[2] = objval[lev2_block] + 5;
 objval[lev2_block] = objval[com_sub] - 3;

 while(lev2_count < 150 && (objval[2]-objval[lev2_block] > 0.0001 ||
      lev2_count < 1) && (lev2_count < 1 ||  exitlev2 <=0 ))
 {
  lev2_count = lev2_count + 1;

  LpSub[2]->CX=0.00;
  for(j=0; j < SPb_n[2]; j++)
```

```
      LpSub[2]->CX += objcoeffs[j]*x[j];

    status = CompuLY(LpSub);

    for(j=0; j < LSPb_n[2]; j++)
      alty[j+HISPb_n[1]] = x[j];

    pvm_initsend(PvmDataRaw);

    info = pvm_pkdouble (&LpSub[2]->CX,1,1);
    info = pvm_pkdouble (LpSub[2]->LY, HISPb_m[nb_subs],1);
    if(subtype[nb_subs] == 2)
     info = pvm_pkdouble (alty, HISPb_n[nb_subs],1);
     info = pvm_send(tid[lev2_block],2);

    Addlev2Cuts(LpSub, env, lp);

    status = SolveProb (env, lp);
     if ( status ) {
       goto TERMINATE;
     }

    pvm_initsend(PvmDataRaw);
    info = pvm_pkdouble (&objval[cur_block],1 ,1);
    bufid = pvm_send (tid[lev2_block],7);

    gettimeofday(&tv1, (struct timezone*)0);
    bufid = pvm_recv (tid[lev2_block],7);

    rdt1 = myseconds(rdt1);
     fprintf(fp, " Receiving Set up Data Time : %8f\n", rdt1/1000000);

    info = pvm_upkdouble (&objval[lev2_block],1,1);
    info = pvm_upkint (&exitlev2, 1,1);
    }
   }

   .... SEND AND RECEIVE OBJECTIVE VALUES ....
}

  TERMINATE:

  .... Free up the problem as allocated by CPXcreateprob, if necessary ....

  return  ;
 }


main(void)
{
 .... DFEINE VARIABLES AND ALLOCATE MEMEORIES .....
 .... SETUP TIME MESEASUREMENT ....
 .... RECEIVE INITIAL DATA AND FIND THE NUMBER OF ACCUMULATED LINKING VARIABLES AND
CONSTRAINT.....

    UpperBound (LpSub);
}

*******************************************************************************
```

C.3 Processor 3 : Lower-Upper Bound Subproblem

```
double *CompuLY(LpSubProb **LpSub, double *xx)
{
 .... COMPUTE BX AND LY ....
 return(LpSub[cur_block]->LY);
}

double CompuPiB(LpSubProb **LpSub, int *cstat, double *pi, double *dj)
{
 .... COMPUTE PIB ....
 return(LpSub[cur_block]->PiB);
}

double *CompuPiD(LpSubProb **LpSub, double *ppi)
{
 .... COMPUTE PID ....
 return(LpSub[cur_block]->PiD);
}

double *CompuOMeL(LpSubProb **LpSub, double *ppi)
{
 .... COMPUTE OMEL ....
 return(LpSub[cur_block]->OMeL);
}

/* Add initial columns of the linking variables of lower-lower bound subproblem
   of the first iteration */
int LSetaddcols(LpSubProb **LpSub, CPXENVptr env, CPXLPptr lp)
{
  int j, p, k, status=1;

  if(LSPb_n[1] > 0)
   status = CPXnewcols (env,lp, LSPb_n[1], LpSub[1]->Lobjcoef,
                     LpSub[1]->Llo_bnds, LpSub[1]->Lup_bnds,NULL, NULL);

  for(j=0; j < LSPb_n[1] ; j++)
   {
    for(p=LpSub[1]->clpnts[j]; p < LpSub[1]->clpnts[j+1]; p++)
    {
     if(LpSub[1]->rwnmbs[p] >= SPb_m[1]-LOWSPb_m[1]
            && LpSub[1]->rwnmbs[p] < SPb_m[1]-LOWSPb_m[3])
      status = CPXchgcoef(env, lp, LpSub[1]->rwnmbs[p]-SPb_m[1]+LOWSPb_m[1],
         SPb_n[cur_block]+HISPb_m[nb_subs]-LSPb_m[1]+j,LpSub[1]->coeffs[p]);

     else if
       (LpSub[1]->rwnmbs[p] >= SPb_m[1]-LOWSPb_m[3] &&
        LpSub[1]->rwnmbs[p] < SPb_m[1])
      status = CPXchgcoef(env, lp, LpSub[1]->rwnmbs[p]-
            SPb_m[1]+LOWSPb_m[3]+SPb_m[3]-LOWSPb_m[3]-LSPb_m[1]-1,
         SPb_n[cur_block]+HISPb_m[nb_subs]-LSPb_m[1]+j,LpSub[1]->coeffs[p]);
    }
   }
  return(status);
}

/* Add a lamda column for primal information coming from aggregated upper bound
       subproblem */

int AddLamcols(LpSubProb **LpSub, CPXENVptr env, CPXLPptr lp, int iter_count)
```

```
{
 int i, status;

 LpSub[2]->CX = LpSub[2]->CX+ LpSub[4]->CX;
 for(i=0; i<HISPb_m[nb_subs] ; i++)
     LpSub[2]->LY[i] = LpSub[2]->LY[i]+ LpSub[4]->LY[i];

  status=CPXnewcols (env,lp, 1, &LpSub[2]->CX, &Lam_lo, &Lam_up, NULL, NULL);

  for(i=HISPb_m[1]; i <HISPb_m[cur_block] ; i++)
  {
   status = CPXchgcoef (env, lp, i-HISPb_m[1], LpSub[cur_block]->n
     + HISPb_m[nb_subs]-LSPb_m[1]+LSPb_n[1]+iter_count-1, LpSub[2]->LY[i]);
  }

  for(i=HISPb_m[cur_block]; i<HISPb_m[nb_subs]; i++)
  {
   status = CPXchgcoef (env, lp, i-LSPb_m[1]+SPb_m[3]-HISPb_m[nb_subs]-1,
    LpSub[cur_block]->n+HISPb_m[nb_subs]-LSPb_m[1]+LSPb_n[1]+iter_count-1,
                                            LpSub[2]->LY[i]);
  }

  status = CPXchgcoef(env,lp, SPb_m[cur_block]-HISPb_m[1]-1, SPb_n[cur_block]
           + HISPb_m[nb_subs]-LSPb_m[1]+HISPb_n[1]+iter_count-1, Lam_coef);

  return(status);
}

/* Add a cut of lower level iteration with dual information coming from
      lower-lower bound subproblem */

int Addcuts(LpSubProb **LpSub, CPXENVptr env, CPXLPptr lp, double **KOmega, double
**KLY, double **OMeLY)
{
 int status,i, j, k;

 status = CPXnewrows (env, lp, 1, &LpSub[1]->PiB, &cut_sense, NULL, NULL);

 for(j=0; j < LSPb_n[1] ;j++)
 {
   status = CPXchgcoef (env,lp,(int)(SPb_m[cur_block]-HISPb_m[1] +lev2_count-1),
            j+SPb_n[cur_block]+ HISPb_m[nb_subs]-LSPb_m[1],LpSub[1]->PiD[j]) ;
 }

 for(j=0; j<LSPb_n[cur_block]; j++)
 {
   status = CPXchgcoef (env, lp, SPb_m[cur_block]-HISPb_m[1]+lev2_count-1, j,
               LpSub[1]->OMeL[j+HISPb_n[cur_block-1]]);
 }

 status = CPXchgcoef(env,lp, SPb_m[cur_block]-HISPb_m[1]+lev2_count-1,
       LpSub[cur_block]->n+HISPb_m[nb_subs]-LSPb_m[1]+LSPb_n[1], Theta_coef);

 for(i=0; i<HISPb_m[1]; i++)
 {
  if(lev2_count == 1)
   KLY[i][iter_count-1] = LpSub[2]->LY[i];
   KOmega[lev2_count][i] = LpSub[1]->Omega[i];
 }

 for(i=1; i < lev2_count+1; i++)
 {
  for(k=1; k< iter_count; k++)
  {
   OMeLY[i][k] = 0.0;
   for(j=0; j < HISPb_m[1]; j++)
    if(KOmega[i][j] != 0.0 && KLY[j][k] != 0.0)
      OMeLY[i][k] += KOmega[i][j]*KLY[j][k];
   }
 }

 for(i=1; i < lev2_count+1 ;i++)
 {
  for(k=1; k < iter_count; k++)
   status = CPXchgcoef(env, lp, (int) i+LpSub[cur_block]->m-HISPb_m[1]-1,
    LpSub[cur_block]->n+HISPb_m[nb_subs]-LSPb_m[1]+HISPb_n[1]+k, OMeLY[i][k]);
 }

short
LowerBound (LpSubProb **LpSub)
{
 .... DEFINE VARIABLES AND ALLOCATE MEMORIES ....

 .... RECEIVE DATA OF LINKING VARIABLES AND CONSTRAINTS OF OTHER PARTS ....

 .... CONVERT ORIGINAL DATA TO CONFORM CPLEX DATA STRUCTURE SUCH AS SENSE,
      OBJCOEFFS AND MATCNT ....

 .... INITIALIZE THE CPLEX ENVIRONMENT AND COPY SUBPROBLEM DATA TO CPLEX ....

 .... MAKE LOWER-UPPER BOUND SUBPROBLEM OF THE FIRST ITERATION BY ADDING AND
      DELETING ROWS AND COLUMNS ....

 status = LSetaddcols(LpSub, env, lp);

.... OPTIMIZE THE PROBLEM, OBTAIN SOLUTION AND WRITE THE RESULT TO OUTPUT
      FILE...

  objval[1] = 0.0; lev2optUP =0.0;
  objval[2] = objval[1] + 10;

  while (iter_count<150 && (objval[2] - objval[1] > tol))
  {
   iter_count=iter_count+1;

   totlev2C = totlev2C + lev2_count;

   LpSub[cur_block]->PiB = CompuPiB(LpSub, cstat, pi, dj);
   LpSub[cur_block]->OMeL = CompuOMeL(LpSub, pi);
   LpSub[cur_block]->PiD = CompuPiD(LpSub, pi);

   .... SEND AND RECEIVE PROPOSALS ....

   status = AddLamcols(LpSub, env, lp, iter_count);

   nbufid = 0;  lev2_count = 0; lev2optUp = 0;exitlev2 = 0; com_sub = 4;
   objval[3] = objval[1] + 5;

  while(lev2_count < 150 && (objval[3] - objval[1] > 0.0001 || lev2_count <1)
  && (lev2_count < 1 || exitlev2 <= 0 ))
  {
   lev2_count = lev2_count + 1;
  .... COMPUTE CX ....

  LpSub[cur_block]->LY = CompuLY(LpSub, x);

  .... SEND PRIMAL PROPOSALS TO LOWER-LOWER BOUND SUBPROBLEM ....

  status = Addcuts(LpSub, env, lp, KOmega, KLY, OMeLY);

  if(objval[cur_block] - objval[1] < 0.0001 )
   lev2optLo = 1;
  else
   lev2optLo = 0;

  /* If processors 2 and 4 has not reached optimal, send objective value and
     'lev2optLo' to processor 4 and receive objective value and 'lev2optUp*/
  if(lev2optUp <= 0)
  {
   pvm_initsend(PvmDataRaw);
   info = pvm_pkdouble (&objval[cur_block],1 ,1);
   pvm_pkint(&lev2optLo, 1, 1);
   bufid = pvm_send (tid[com_sub],7);

   gettimeofday(&tv1, (struct timezone*)0);
   bufid = pvm_recv (tid[com_sub],7);
   rdt1 = myseconds(rdt1);
   fprintf(fp, " Receiving Set up Data Time : %8f\n", rdt1/1000000);

   info = pvm_upkdouble (&objval[com_sub],1 ,1);
   info = pvm_upkint(&lev2optUp, 1, 1);

   nbufid = 1;
  }
 }

 /* If objective value of this subproblem is less than that of upper-lower
     subproblem, then set the signal exitlev2 1 and send it to lower-lower
     subproblem*/
  if(objval[cur_block] <= objval[com_sub])
   exitlev2 = 1;
  else
   exitlev2 = 0;

   pvm_initsend(PvmDataRaw);
   info = pvm_pkdouble (&objval[cur_block],1 ,1);
   info = pvm_pkint (&exitlev2,1 ,1);
   bufid = pvm_send (tid[1],7);
}

  pvm_initsend(PvmDataRaw);

 info = pvm_pkdouble (&objval[cur_block],1 ,1);
 for(k=1; k <=nb_subs; k++)
  {
   if(k != cur_block)
   bufid = pvm_send (tid[k],8);
  }

 for(k=1; k <=nb_subs; k++)
  {
   if(k != cur_block )
```

```
    {
     gettimeofday(&tvl, (struct timezone*)0);
     bufid = pvm_recv (tid[k],8);
     rdtl = myseconds(rdtl);
     fprintf(fp, " Receiving Set up Data Time : %8f\n", rdtl/1000000);

     info = pvm_upkdouble (&objval[k],1,1);
    }
   }
  }

  TERMINATE:

   /* Free up the problem as allocated by CPXcreateprob, if necessary */

main(void)
{
 .... DEFINE VARIABLES AND ALLOCATE MEMORIES ....
 .... RECEIVE ORIGINAL DATA AND FIND THE NUMBER OF ACCUMULATED LINKING VARIABLES
AND CONSTRAINTS ....
.... MESEAURE PROGRAM TIME ....

 LowerBound (LpSub);
}


*************************************************************************8
```

C.4 Processor 4 : Upper-Lower Bound Subproblem

```
double *CompuLY(LpSubProb **LpSub, double *xx)
{
 .... COMPUTE BX AND LY ....
 return(LpSub[cur_block]->LY);
}

double CompuPiB(LpSubProb **LpSub, int *cstat, double *pi, double *dj)
{
 .... COMPUTE PIB ....
 return(LpSub[cur_block]->PiB);
}

double *CompuPiD(LpSubProb **LpSub, double *ppi)
{
 .... COMPUTER PID ....
 return(LpSub[cur_block]->PiD);
}

double *CompuOMeL(LpSubProb **LpSub, double *ppi)
{
 .... COMPUTE OMEL ....
 return(LpSub[cur_block]->OMeL);
}

int USetaddcols(LpSubProb **LpSub, CPXENVptr env, CPXLPptr lp)
{
 long j, p, k, status;

 for(j=0; j < LSPb_n[1] ; j++)
 {
  for(p=LpSub[1]->clpnts[j]; p < LpSub[1]->clpnts[j+1]; p++)
```

```
   {
    if(LpSub[1]->rwnmbs[p] >= SPb_m[1]-LOWSPb_m[1] && LpSub[1]->rwnmbs[p] <
SPb_m[1]-LOWSPb_m[2])
      status = CPXchgcoef(env, lp, LpSub[1]->rwnmbs[p]-SPb_m[1]+LOWSPb_m[1],
       SPb_n[cur_block]+LSPb_m[cur_block]+LSPb_m[2]+j, LpSub[1]->coeffs[p]);

    if(LpSub[1]->rwnmbs[p] >= SPb_m[1]-LOWSPb_m[3] && LpSub[1]->rwnmbs[p] <
SPb_m[1])
     status=CPXchgcoef(env,lp,LpSub[1]->rwnmbs[p]-SPb_m[1]+LOWSPb_m[3]+LSPb_m[2],
       SPb_n[cur_block]+LSPb_m[cur_block]+LSPb_m[2]+j, LpSub[1]->coeffs[p]);
   }
  }

  for(j=0; j < LSPb_n[3] ; j++)
  {
   for(p=LpSub[3]->clpnts[j]; p < LpSub[3]->clpnts[j+1]; p++)
   {
    if(LpSub[3]->rwnmbs[p] > HISPb_m[1] && LpSub[3]->rwnmbs[p] <= HISPb_m[2])
      status = CPXchgcoef(env, lp, LpSub[3]->rwnmbs[p]-HISPb_m[1]-1,
       SPb_n[cur_block]+LSPb_m[cur_block]+LSPb_m[2]+j+LSPb_n[1],LpSub[3]-
>coeffs[p]);

   if(LpSub[3]->rwnmbs[p] >= SPb_m[3]-LOWSPb_m[3] && LpSub[3]->rwnmbs[p] < SPb_m[3])
     status=CPXchgcoef(env,lp,LpSub[3]->rwnmbs[p]-SPb_m[3]+LOWSPb_m[3]+LSPb_m[2],
       SPb_n[cur_block]+LSPb_m[cur_block]+LSPb_m[2]+j+LSPb_n[1],LpSub[3]->coeffs[
p]);
   }
  }

  return(status);
 }

short AddLamcols(LpSubProb **LpSub, CPXENVptr env, CPXLPptr lp, int iter_count, int
lev2_count, double **KOMeL, double **Kalty, double *alty)
{
  long i, j; int  k, status;

  status=CPXnewcols (env,lp, 1, &LpSub[2]->CX, &Lam_lo, &Lam_up, NULL, NULL);

  for(i=HISPb_m[1]; i <HISPb_m[2] ; i++)
  {
   status = CPXchgcoef (env, lp, i-HISPb_m[1], LpSub[cur_block]->n
     + LSPb_m[4]+LSPb_m[2]+LSPb_n[1]+LSPb_n[3]+lev2_count, LpSub[2]->LY[i]);
  }

  for(i=HISPb_m[3]; i<HISPb_m[4]; i++)
  {
   status = CPXchgcoef (env, lp, i-LSPb_m[1]-LSPb_m[3],
    LpSub[cur_block]->n+LSPb_m[4]+LSPb_m[2]+LSPb_n[1]+LSPb_n[3]+lev2_count,
LpSub[2]->LY[i]);
  }

  status = CPXchgcoef(env,lp, SPb_m[cur_block]-LSPb_m[1]-LSPb_m[3]-1,
SPb_n[cur_block]
           + LSPb_m[4]+LSPb_m[2]+LSPb_n[1]+LSPb_n[3]+lev2_count, Lam_coef);

  for(j=HISPb_n[1]; j<HISPb_n[2]; j++)
   {
    KOMeL[iter_count-1][j] = LpSub[1]->OMeL[j];
    Kalty[j][lev2_count] = alty[j];
   }
  for(k=1; k < iter_count; k++)
  {
    for(i=1; i< lev2_count+1; i++)
    {
     OMeLY[k][i] = 0.0;
     for(j=HISPb_n[1]; j < HISPb_n[2]; j++)
     {
      if(KOMeL[k][j] != 0.0 && Kalty[j][i] != 0.0)
       OMeLY[k][i] = OMeLY[k][i]+KOMeL[k][j]*Kalty[j][i];
     }
    }
   }

    for(i=1; i < iter_count ;i++)
    {
     for(j=1; j < lev2_count+1; j++)
     {
      status = CPXchgcoef(env, lp, (int) i+LpSub[cur_block]->m-LSPb_m[1]-LSPb_m[3]-
1, (int) (LpSub[cur_block]->n+LSPb_m[cur_block]+LSPb_m[2]+LSPb_n[1]+LSPb_n[3]+j),
OMeLY[i][j]);
     }
    }

  return(status);
}

int Addcuts(LpSubProb **LpSub, CPXENVptr env, CPXLPptr lp, int lev2_count, int
iter_count)
{
 int status, k; long i, j;
 status = CPXnewrows (env, lp, 1, &LpSub[1]->PiB, &cut_sense, NULL, NULL);

 for(j=0; j < LSPb_n[1] ;j++)
 {
  status = CPXchgcoef (env, lp,(int) (SPb_m[cur_block]
            -LSPb_m[1]-LSPb_m[3] +iter_count-2), j+SPb_n[cur_block]
            + LSPb_m[2]+LSPb_m[cur_block],LpSub[1]->PiD[j]) ;
 }

 for(j=0; j < LSPb_n[3] ;j++)
 {
  status = CPXchgcoef (env, lp,(int) (SPb_m[cur_block]
            -LSPb_m[1]-LSPb_m[3] +iter_count-2), j+SPb_n[cur_block]
            + LSPb_m[2]+LSPb_m[cur_block]+LSPb_n[1], LpSub[3]->PiD[j]) ;
 }

 for(j=0; j<LSPb_n[cur_block]; j++)
 {
   status = CPXchgcoef (env, lp, SPb_m[4]-LSPb_m[1]-LSPb_m[3]+iter_count-2, j,
   LpSub[1]->OMeL[j+HISPb_n[cur_block-1]]);
 }

 status = CPXchgcoef(env,lp, SPb_m[cur_block]-LSPb_m[1]-LSPb_m[3]+iter_count-2,
          SPb_n[cur_block]+LSPb_m[2]+LSPb_m[cur_block]+LSPb_n[1]+LSPb_n[3],
Theta_coef);

 return(status);

}
```

```
short
UpperBound (LpSubProb **LpSub)
{

 .... RECEIVE NECESSARY OTHER LINKING VARIABLES' DATA .....
 .... DELETE UNNECESSARY ROWS FOR THIS SUBPROBLEM AND ADD COLUMNS FOR ARTIFICIAL
VARIALBES. ALSO, ADD NECESSARY COLUMNS FOR OTHER PART'S LINKING VARIABLES ....

   status = USetaddcols(LpSub, env, lp);

 .... OPTIMIZE THE PROBLEM OF THE FIRST ITERATION, OBTAIN SOLUTIONS AND WRITE THEM
TO OUTPUT FILE ....

  status = CPXnewrows (env, lp, 1, &Lam_rhs, &Lam_sense, NULL, NULL);

  status = CPXnewcols (env,lp, 1, &Theta_coef, &Theta_lo, NULL,NULL, NULL);

  objval[1] = 0.0; objval[2]= 0.0;
  objval[2] = objval[1] + 10;

  while ((iter_count<150) && (objval[2] - objval[1] > tol ))
  {
   iter_count = iter_count + 1;

   .... COMPUTE CX ....

   LpSub[cur_block]->LY = CompuLY(LpSub, x);

 .... SEND PRIMAL INFORMATION TO AND RECEIVE DUAL INFORMATION FROM THE AGGEGATED
 LOWER BOUND SUBPROBLEM (PROCESSOR 1 AND 3) ....

    /* Find proper dual information by adding each information coming from
 aggregated lower bound subproblm (processors 1 and 3) */
    LpSub[1]->PiB = LpSub[1]->PiB + LpSub[3]->PiB ;

    for(j=0; j<LSPb_n[1]; j++)
        LpSub[1]->PiD[j] = LpSub[1]->PiD[j]+LpSub[3]->OMeL[j];

    for(j=0; j<LSPb_n[3]; j++)
        LpSub[3]->PiD[j] = LpSub[3]->PiD[j]+LpSub[1]->OMeL[j+HISPb_n[2]];

    for(i=0; i < HISPb_n[4] ; i++)
        LpSub[1]->CMeL[i] = LpSub[1]->OMeL[i]+ LpSub[3]->OMeL[i];

    /* Delete columns for primal information of previous lower level iteration */
    status = CPXdelcols (env, lp,
 SPb_n[cur_block]+LSPb_m[2]+LSPb_m[4]+LSPb_n[1]+LSPb_n[3]+1, cur_numcols-1);

    status = Addcuts(LpSub, env, lp, lev2_count, iter_count);

    nbufid = 0; lev2optLo = 0; exitlev2 = 0; lev2_count = 0;

  objval[4] = objval[2] - 5;

  while(lev2_count < 150 && (objval[2] - objval[4] > 0.0001 || lev2_count <1)
  && (lev2_count <1 || exitlev2 <= 0  ))
  {
   lev2_count = lev2_count + 1;

   .... SEND DUAL INFORMATION TO AND RECEIVE PRIMAL INFORMATION UPPER-UPPER BOUND
 SUBPROBLEM (PROCESSOR 2) ....
 status = AddLamcols(LpSub, env, lp, iter_count, lev2_count, KOMeL, Kalty,
          alty);

 status = CPXprimopt (env, lp);

.... RECEIVE OBJECTIVE VALUE FROM PROCESSOR 2 .....

if(objval[2] - objval[cur_block] < 0.0001 )
  lev2optUp = 1;
 else
  lev2optUp = 0;

/* If processors 1 and 3 has not reached optimal, send objective value and
    'lev2optUp' to processor 3 and receive objective value and 'lev2optLo*/

 if(lev2optLo <= 0)
 {
  pvm_initsend(PvmDataRaw);
  info = pvm_pkdouble (&objval[cur_block],1 ,1);
  pvm_pkint(&lev2optUp, 1, 1);
  bufid = pvm_send (tid[com_sub],7);

  gettimeofday(&tv1, (struct timezone*)0);
  bufid = pvm_recv (tid[com_sub],7);
  rdt1 = myseconds(rdt1);
  fprintf(fp, " Receiving Set up Data Time : %8f\n", rdt1/1000000);

  info = pvm_upkdouble (&objval[com_sub],1 ,1);
  info = pvm_upkint(&lev2optLo, 1, 1);
  if(info < 0)
    fprintf(fp, "ERROR in receiving objval[%d]\n", com_sub);
 }

  /* If objective value of this subproblem is greater than that of Lower-upper
      bound subproblem, then set the signal exitlev2 1 and send it to upper-
      upper bound subproblem*/

 if(objval[cur_block] >= objval[com_sub] )
  { exitlev2 = 1; }
 else
  { exitlev2 = 0; }

  pvm_initsend(PvmDataRaw);
  info = pvm_pkdouble (&objval[cur_block],1 ,1);
  info = pvm_pkint (&exitlev2,1 ,1);
  bufid = pvm_send (tid[2],7);
 }

pvm_initsend(PvmDataRaw);

info = pvm_pkdouble (&objval[cur_block],1 ,1);
for(k=1; k <=nb_subs; k++)
 {
  if(k != cur_block)
  bufid = pvm_send (tid[k],8);
 }
}

/* Free up the CPLEX environment, if necessary */
```

```
  return  ;
 }

main(void)
{
  .... DEFINE AND DO THE SIMILAR OPERATIONS AS PREVIOUS SUBPROBLEMS ....
   UpperBound (LpSub);
}
```

BIBLIOGRAPHY

Aardal, K. And A. Ari. "On the Resemblance Between the Kornai-Liptak and Cross Decomposition Techniques for Block-Angular Linear Programs", *European Journal of Operational Research*. **46**(1990), 393-398.

Baker, L and B.J. Smith, 1996, "Parallel Programming", McGraw-Hill.

Barr, R.S. and B. Hickman. 1993, "Reporting Computational Experiments with Parallel Algorithms: Issues, Measures, and Experts' Opinions", *ORSA Journal on Computing*, Vol. **5**, No. 1, Winter (1993), pp.2-18.

Benders, J.F., 1962, "Partitioning Procedures for Solving Mixed-variable Programming Problems", *Numerische Mathematik* **4** (1962), pp. 238-252.

Birge, J.R., C.J. Donohue, D.F. Holmes and O.G. Svintsitski, "A Parallel Implementation of the Nested Decomposition Algorithm for Multistage Stochastic Linear Programs", *Mathematical Programming* **75** (1996), pp. 327-352.

Brooke, A., D. Kendrick and A. Meeraus. *GAMS: A User's Guide*, The Scientific Press, Redwood City, California, 1992.

CPLEX Division. *Using the CPLEX Callable Library*. ILOG Inc. Incline Village, NV. 1997.

Dantzig, G. B., and P. Wolfe, "The Decomposition Principle for Linear Programming", *Operations Research* **8** (1960), pp. 101-111.

Dantzig, G.B. 1963. *Linear programming & Extensions*. Princeton University Press.

Dantzig, G.B. 1980. "Time-staged Linear programs". Technical Report SOL 80-28, System

Optimization Laboratory, Department of Operations Research, Stanford University. Stanford. California, USA.

Dantzig, G.B. and P.W. Glynn, "Parallel Processors for Planning Under Uncertainty", *Annals of Operation Research* **22** (1990), pp. 1-21.

Duncan, R. "A Survey of Parallel Computer Architectures", *IEEE Trans. Computers* (1990).

Entriken, R. "Parallel Decomposition: Results for Staircase Linear Programs", *SIAM Jurnal of Optimization*, **4** (1996), pp. 961-977, November.

Flynn, M.J. "Very High Speed Computing Systems", *Proc. IEEE.* **54** (1966), pp. 1901-1909.

Fragniere, E., J. Gondzio, R. Sarkissian. and J. P. Vial, "Structure Exploiting Tool in Algebraic Modeling Languages", *Management Science* **46**, No. 8 (2000), pp. 1145-1158.

Fragniere, E., J. Gondzio, and J. P. Vial. "A Planning Model with one Million Scenarios Solved on an Affordable Parallel Machine", Logilab Technical Report (1998b), Section of Management Studies, University of Geneva, Geneva, Switzerland.

GAMS Development Corporation, The GAMS I/O Library, 1996

Geist, A., A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam, 1994. A PVM: Parallel Virtual Machine – "Users' Guide and Tutorial for Networked Parallel Computing", The MIT Press, Cambridge, MA, USA.

Gnanendran, S.K. and J.K. Ho, "Load Balancing in the Parallel Opimization of Block-Angualar Linear Programs", *Mathematical Programming* **62** (1993), pp.41-67.

Ho, J.K., "Convergence Behavior of Decomposition Algorithms for Linear Programs", *Operations Research Letters* **3** (1984), pp. 91-94.

Ho, J.K., T.C. Lee and R.P. Sundarraj, 1988, "Decomposition of Linear Programs Using Parallel

Computation", *Mathematical Programming* **42** (1988), pp. 391-405.

Ho, J.K. and E. Loute, 1996, "On the degree of Decentralization in Linear Programming", *Informatica* **7**, pp. 337-348.

Holmberg, K. "On the Convergence of Cross Decomposition". *Mathematical Programming*, **47** (1990), pp. 269-296.

Holmberg, K. "Linear Mean Value Cross Decomposition: A Generalization of the Kornai-Liptak Method", *European Journal of Operational Research*, **62** (1992), pp. 55-73.

Kortanek, K. O. and Ji Shan Zhu, 1988, "New purification algorithms for linear programming", *Naval Research. Logistics*, **35**, No. 4, pp. 571—583.

Lan, B. 1993, "A Primal-Dual Decomposition Method for Multi-Stage Linear Programs", Ph.D dissertation, Department of Management Sciences,University of Waterloo, Waterloo, ON. CANADA.

Lan, B. and J.D. Fuller, 1995a, "A Primal-Dual Decomposition Method for Two-Stage Linear Programs", working paper, Department of Management Sciences, University of waterloo, Waterloo, ON. CANADA.

Lan, B. and J.D. Fuller. 1995b, "A Primal-Dual Nested Decomposition Algorithm for Multi-Stage Linear Programs", working paper, Department of Management Sciences, University of Waterloo, Waterloo, ON. CANADA.

Murphy, F. H., and M. V. Mudrageda. "A Decomposition Approach for a Class of Economic Equilibrium Models", *Operations Research*, **46**, *No.* 3 (1998), pp. 368-377.

Nielsen, S. S., and S. A. Zenios. "Scalable parallel Benders decomposition for stochastic linear programming". *Parallel Computing*, **23** (1997), pp. 1069-1088.

Park, H.J. 1996, "A Primal-Dual Decomposition Method for Multi-stage, Convex Nonlinear Programs", master's thesis, Department of Management Sciences, University of Waterloo, Waterloo, ON. CANADA.

Rosen, J.B. and R.S. Maier, "Parallel Solution of Large-scale, Block-angular Linear Programs", *Annals of Operations Research* **22** (1990), pp. 23-41.

Ruszczynski, A. "Parallel Decomposition of Multistage Stochastic Programming Problems", *Mathematical Programming* **58** (1993), pp201-228.

Van Roy, T.J. "Cross Decomposition For Mixed Integer Programming", *Mathematical Programming* **25** (1983), pp. 46-63.

Vladimirou, H. "Computational assessment of distributed decomposition methods for stochastic linear programs", *European Journal of Operational Research* **108** (1998), pp. 653-670.

Zenios, S.A. 1989, "Parallel Numerical Optimization: Current Status and an Annotated Bibliography", ORSA Journal on Computing, Vol.1, No.1. pp20-pp43.