A Parallel Implementation of the Nested Decomposition Algorith for Multistage Stochastic Linear Programs

John R. Birge
Christopher J. Donohue
Derek F. Holmes
Oleg G. Svintsitski
Department of Industrial and Operations Engineering
University of Michigan
Ann Arbor, MI 48109-2117
USA

Tech Report 94-1

A Parallel Implementation of the Nested Decomposition Algorithm for Multistage Stochastic Linear Programs[1]

John R. Birge
Christopher J. Donohue
Derek F. Holmes
Oleg G. Svintsitski
Department of Industrial and Operations Engineering
University of Michigan
Ann Arbor, Michigan 48109-2117
USA
12/22/93

Abstract: Multistage stochastic linear programs can represent a variety of practical decision problems. Solving a multistage stochastic program can be viewed as solving a large tree of linear programs. A common approach for solving these problems is the nested decomposition algorithm, which moves up and down the tree solving nodes and passing information between nodes. The natural independence of subtrees suggests that much of the computational effort of the nested decomposition algorithm can run in parallel across small numbers of fast processors. This paper explores the advantages of such parallel implementations over serial implementations and compares alternative sequencing protocols for parallel processors. Computational experience on a large test set of practical problems with up to 1.5 million constraints and almost 5 million variables suggests that parallel implementations may indeed work well, but they require careful attention to processor load balancing.

[1]Supported in part by the National Science Foundation under Grant DDM-9215921.

1 Introduction

Most sequential decisions must be made in uncertain environments. Multistage stochastic programs may model these decisions by seeking to find decision variable levels that attain an optimal expected value subject to possibly uncertain constraints on available resources. Multistage stochastic programs (MSPs) have been applied to a variety of problems, ranging from forestry management (Gassman, [10]) to portfolio optimization (Mulvey and Vladimirou, [19]) and capacity expansion (Sims, [28]).

A multistage stochastic program assumes that a sequence of periodic decisions incur costs which are affected by a known set of stochastic parameters. After making each decision, an observation of these parameters becomes available. Following each observation, an additional "corrective" (or recourse) decision may be made. The decision process then starts over, with a new decision that must consider the effects of further uncertainty. The objective of the model is to minimize the sum of costs associated with each period's decision subject to model constraints and *nonanticipativity* constraints that require each period's decision to depend only on the information available at the time. To avoid ambiguity in the solution, we require that the constraints hold for all possible data realizations.

We focus on the special case of this decision problem where the objective and constraints are linear functions of the decision variables. This simplifies the mathematical program, but still requires multivariate integration of an implicitly defined cost function. Many characterizations of the random parameters may yield computationally intractable formulations. To reduce the problem to manageable size, the most common simplification is to solve a stochastic program that assumes a finite time horizon and discrete probabilistic representation of the uncertainty. This allows the expectations to be written as finite sums and the constraints to be enforced explicitly for each realization.

The resultant deterministic linear program can still be quite large and difficult to solve, since a linear program must be solved for each event. The number of events generally grows exponentially with the number of periods considered. Approaches for solving these problems generally either take advantage of the problems' structure within the context of a general optimization algorithm (see, e.g. [31] or [5]) or decompose the problem into subproblems.

Decomposition algorithms can define subproblems as linear programs to find an optimal solution for all periods in a scenario (scenario-decomposed methods) or for one period given an event history (period-decomposed methods). Algorithms in the former category include the progressive hedging algorithm [24] and diagonal quadratic approximation ([18], [20]).

Many methods follow the period-decomposed form, which can also be interpreted as a form of dynamic programming. Van Slyke and Wets [29] first applied these methods to two-stage stochastic programs. Multistage extensions appear in Birge [3]. The multistage algorithm is also known as a *nested decomposition* method, since an n stage program has the same form as two-stage program when the first $n-1$ stages are considered as one problem.

Several improvements to the original algorithm appear in the literature, including multicuts [4], varying sequencing protocols [32], and bunching [31]. (These will be discussed in further detail in Section 2.) All of these improvements have been implemented in the implementation by Gassmann, MSLiP ([11]). A variant of the nested decomposition algorithm called the *regularized decomposition* method has also been implemented to solve large problems ([26]). This method adds a quadratic term to the objective of each subproblem to keep successive solutions closer together and hence stabilize the progress of the algorithm.

Nested decomposition procedures are well suited for parallelization, since solutions to subtrees are independent of one another. The efficiency of any parallelization depends heavily on the speed

of the processors available and the speed of communications between them. A goal of this paper to assess this efficiency in a distributed environment with similar processors.

Other studies have considered tightly coupled architectures, such as a Connection Machine, with comparatively little power in each processor, but high communications bandwidths between each. These machines are more suited for fine decompositions that communicate repeatedly with each other (see, e.g. [23]. Implementations by Ariyawansa and Hudson [1] and Entriken [7] showed that speedups on a Sequent/Balance computer with shared memory architecture with up to 24 processors approached a linear function for two-stage problems, but decreased substantially for some multistage problems.

Ruszczyński [25] suggested parallelizing the regularized decomposition method by queuing subproblems for idle processors. His simulations of solving a multistage stochastic inventory problem using a Sequent-like architecture suggested near linear speedups for 4 stage problems with up to 7 processors and up to an increasing number of processors as the number of stages increase.

At the other extreme from tightly coupled architectures are groups of distributed workstations, which have a few powerful processors but may have slow communications. They have a practical advantage over massively parallel architectures, however, due to their availability in many sites. These "machines" are best suited for large tasks that do not communicate often. One example of their use in a multistage stochastic programming methods is Berger, et. al.'s [2] Diagonal Quadratic Approximation (DQA), which showed a maximum speedup of 5.3 for 8 processors for a large asset allocation problem. DQA, introduced in [18], is a scenario-based method.

The implementation of the nested decomposition algorithm discussed here is ideally suited for loosely distributed computing environments. The only communications necessary between processors occurs between subproblems in the first few stages. A more detailed description of the nested decomposition algorithm is in Section 2, followed by a discussion of distributed implementational issues in Section 3. Sections 4 and 5 report numerical results and conclusions based on experience with a large test suite of stochastic programming problems from the literature.

2 The Nested Decompositon Method

The general form of a multistage stochastic linear program with fixed recourse can be expressed as

$$\min_{x_0} \; cx_0 + E_{\boldsymbol{\xi}_1}(\min_{\boldsymbol{x}^1} \boldsymbol{c}^1\boldsymbol{x}^1 + \ldots E_{\boldsymbol{\xi}_H|\boldsymbol{\xi}_1\cdots\boldsymbol{\xi}_{H-1}}(\min_{\boldsymbol{x}^H} \boldsymbol{c}^H\boldsymbol{x}^H)\ldots)$$

$$\begin{aligned}
\text{s.t.} \qquad Ax^0 &= b \\
\boldsymbol{T}^0x^0 + \boldsymbol{W}^1\boldsymbol{x}^1 &= \boldsymbol{h}^1 \;\; a.s. \\
\vdots \qquad &\vdots \\
T^{H-1}\boldsymbol{x}^{H-1} + \boldsymbol{W}^H\boldsymbol{x}^H &= \boldsymbol{h}^H \;\; a.s. \\
l^0 \le x^0 \le u^0, \boldsymbol{l}^t \le \boldsymbol{x}^t \le \boldsymbol{u}^x \;\; & t = 1,\ldots,H \;\; a.s.
\end{aligned}$$

where bold face vectors are (possibly) stochastic. We assume the stochastic elements are defined over a discrete canonical probability space $(\Xi, \sigma(\Xi), P)$, where $\Xi = \Xi_1 \otimes \cdots \otimes \Xi_H$, and the S_t elements of Ξ_t are $\{\xi_s^t = (T_s^t, W_s^t, h_s^t, c_s^t), s = 1,\ldots,S_t\}$. Since the probability space is discrete, we can define a *deterministic equivalent* linear program by replicating the deterministic linear program for each possible event in Ξ, and requiring that each decision not depend on the future.

If a set of decisions is defined for each scenario in Ξ, the expected value can be written as a finite sum of the costs of these decisions. The requirement that each stochastic constraint must

hold almost surely may be enforced by defining the same set of constraints for each realization. The decisions meeting these restrictions correspond to a decision tree where each node at stage t contains a linear program which determines the optimal decision to take given all realized data $(\xi_1, \ldots, \xi_{t-1})$. Using the decision tree characterization, we refer to the stage $t-1$ node (or subproblem) connected to a stage t problem as a *parent* of that subproblem. Other relations, such as *grandparents, siblings* and *cousins*, are defined analogously.

Formally, the deterministic equivalent we consider here can be written as

$$\min \quad cx_0^1 + \sum_{k=1}^{\bar{N}_1} p_k^1 c_k^1 x_k^1 + \cdots + \sum_{k=1}^{\bar{N}_H} p_k^H c_k^H x_k^H$$

$$\begin{aligned} \text{subject to} \qquad Ax_1^0 &= b \\ T_j^t x_{\alpha(j,t)}^{t-1} + W^t x_j^t &= h_j^t \quad j = 1, \ldots, \bar{N}_t, \quad t = 1, \ldots, H \\ l_j^t \le x_j^t &\le u_j^t \quad j = 1, \ldots, \bar{N}_t, \quad t = 1, \ldots, H \end{aligned} \tag{2.1}$$

where

$$\begin{aligned} N_t &= \text{Number of possible outcomes in stage } t \\ \bar{N}_t &= \text{Cumulative number of scenarios through stage } t, \bar{N}_t = N_1 \times \cdots \times N_t \\ x_j^t &= \text{Decision vector to take in stage } t \text{ given outcome } j \\ p_i^t &= \text{Probability that scenario } i \text{ in stage } t \text{ occurs, } i = 1, \ldots, \bar{N}_t \\ (c_i^t, h_i^t, T_i^t) &= \text{Cost and RHS vectors and Technology matrix for scenario } i \text{ in stage } t \\ W^t &= \text{Recourse matrix for stage } t, \ W^t \in \Re^{n^t \times m^t} \\ \alpha(j,t) &= \lfloor (j-1)/N_t \rfloor + 1, \text{ the predecessor of node } j \text{ in stage } t. \end{aligned}$$

For ease of explanation in this formulation, we use the same number of scenarios as descendants in stage t of each scenario in stage $t-1$. Other structures can be used without loss of generality.

2.1 The Nested Decomposition Algorithm

The basic subproblem used by the nested decomposition algorithm (ND) for scenario j in period t is $\min\{c_j^t x_j^t + Q^{t+1}(x_j^t) \mid W^t x_j^t = h_j^t - T_j^{t-1} x_{\alpha(j,t)}^{t-1}, \ l_j^t \le x_j^t \le u_j^t\}$, where $Q^{t+1}(x_j^t)$ is the optimal expected value of stages $t+1$ to H, given an input x_j^t. (Additional constraints, such as $Ax_0^1 = b$, are added where necessary.) The objective can be parameterized by introducing a free variable θ_j^t and solving the equivalent problem

$$\min \left\{ c_j^t x_j^t + \theta_j^t | W_j^t x_j^t = h_j^t - T_j^{t-1} x_{\alpha(j,t)}^{t-1}, l_j^t \le x_j^t \le u_j^t, \theta_j^t \ge Q^{t+1}(x_j^t) \right\}.$$

Since $Q^{t+1}(y_j^t)$ is the solution to a linear program that parametrically depends on right-hand side constraint parameters, it is a polyhedral convex function. The ND algorithm avoids describing the entire function by finding selected linear supports and constraining θ_j^t with them. These pieces, also called *optimality cuts*, are formed from the dual solutions to descendant subproblems.

To see how these cuts are constructed, consider a subproblem in period H. (We drop time superscripts where the context is clear.) This subproblem has the same form as a two-stage stochastic program, where

$$Q^H(x) = \left\{ \min \sum_{k=1}^{K} p_k^H c_k^H x_k^H \mid W_k^H x_k^H = h_k^H - T_j^{H-1} x, l_j^H \le x_j^H \le u_j^H \right\}.$$

The dual to this problem at a given $\hat{x}$ is

$$\begin{array}{ll} \max & \sum_{k=1}^{K} p_k[\pi_k(b_k - T_k\hat{x}) + \lambda_k l_k - \mu_k u_k] \\ \text{s.t.} & \pi_k W + \lambda_k - \mu_k = c_k \qquad k = 1, \ldots, K \\ & \lambda_k, \mu_k \geq 0. \end{array} \tag{2.2}$$

Note that this problem is separable into K subproblems, and that λ_k and μ_k can be chosen to force feasibility in (2.2). The solution $(\pi_k^*, \lambda_k^*, \mu_k^*)$ to each subproblem will be dual feasible for any x. By duality theory,

$$\mathcal{Q}(x) \geq \sum_{k=1}^{K} p_k[\pi_k^*(b_k - T_k x) + \lambda_k^* l_k - \mu_k^* u_k]$$

with equality holding for $\hat{x}$. Let $E_k = \sum_{k=1}^{K} p_k \pi_k^* T_k$ and $e_k = \sum_{k=1}^{K} p_k[\pi_k^* b_k + \lambda_k^* l_k - \mu_k^* u_k]$. Inequality (2.1) can be rewritten in these terms to constrain θ_j^t in the corresponding $H-1$ subproblem as

$$\theta_k^{H-1} \geq e_k - E_k x_k^{H-1}.$$

If the dual problem (2.2) is unbounded for any k, then the primal subproblem must be infeasible. The $\hat{x}$ obtained from the previous period must be rejected. This can be done with a *feasiblity cut*, which is derived from a dual feasible direction of unboundedness, $(\pi_k^*, \lambda_k^*, \mu_k^*)$, which must satisfy $\pi_k^*(b_k - T_k\hat{x}) + \lambda_k^* l_k - \mu_k^* u_k > 0$. Thus, the constraint

$$\pi_k^*(b_k - T_k\hat{y}) + \lambda_k^* l_k - \mu_k^* u_k \leq 0$$

eliminates the infeasible $\hat{x}$. Letting $D_k = -\pi_k^* T_k$ and $d_k = \pi_k^* b_k + \lambda_k^* l_k - \mu_k^* u_k$, a feasibility cut in scenario k in period $H-1$ becomes $D_k^{H-1} x_k^t \geq d_h^{H-1}$.

We can now describe the full subproblem for scenario k in stage t. This problem, designated $NLDS(t,k)$, is

$$\begin{array}{lllll} NLDS(t,k) & \min & c_k^t x_k^t + \theta_k^t & & \\ & \text{s. t.} & W^t x_k^t & = h_k^t - T_k^{t-1} x_{\alpha(k,t)}^{t-1} & \\ & & D_{k,j}^t x_j^t & \geq d_{k,j}^t, & j = 1, \ldots, r_k^t \\ & & E_{k,j}^t x_j^t + \theta_k^t & \geq e_{k,j}^t, & j = 1, \ldots, s_k^t \\ & & l_k^t \leq x_k^t \leq u_k^t & & \end{array} \tag{2.3}$$

with dual solutions $(\pi_k^t, \lambda_k^t, \mu_k^t, \rho_k^t, \sigma_k^t)$.

The basic nested decomposition algorithm is shown in Figure 1. The algorithm assumes that all variables are bounded to avoid complications, but this restriction can be dropped as in [29].

Step 0. Set $t = 1, k = 1, r_k^t = s_k^t = 0$, add the constraint $\theta_k^t = 0$ to $NLDS(t,k)$ for all t and k. Let $DIR = FORW$ and goto 1.
Step 1. Solve the current problem $NLDS(t,k)$. If infeasible and $t = 1$, then stop, since the entire problem is infeasible. If $NLDS(t,k)$ is infeasbile and $t > 1$, then let $r_{\alpha(t,k)}^{t-1} = r_{\alpha(t,k)}^{t-1} + 1$ and let $DIR = BACK$. Find a dual feasible direction of unboundedness $(\pi_k^t, \lambda_k^t, \mu_k^t, \rho_k^t)$ such that $(\pi_k^t W^t + \lambda_k^t - \mu_k^t) + \rho_k^t D_k^t \leq 0$ but $\pi_k^t(h_k^t - T_k^{t-1}x_{\alpha(k,t)}^{t-1}) + \lambda_k^t l_k^t - \mu_k^t u_k^t + \rho_k^t d_k^t \geq 0$. Let $k' = \alpha(k,t)$ and

$$\begin{aligned} D_{k',r_{k'}^{t-1}}^{t-1} &= \pi_k^t T_k^{t-1} \\ d_{k',r_{k'}^{t-1}}^{t-1} &= \pi_k^t h_k^t + \lambda_k^t l_k^t - \mu_k^t u_k^t + \rho_k^t d_k^t \end{aligned}$$

Let $k = \alpha(k,t), t = t-1$ and return to step 1. If $NLDS(t,k)$ is feasible, update the values of x_k^t and θ_k^t and store the basic dual multipliers as $(\pi_k^t, \lambda_k^t, \mu_k^t, \rho_k^t, \sigma_k^t)$. If $k < K_t$, let $k = k+1$ and return to step 1. Otherwise, $k = K^t$, and we are done with this stage. If $DIR = FORW$ and $t < T$, let $t = t+1$ and restart step 1. If $t = T$, let $DIR = BACK$ and go to step 2.
Step 2. For all scenarios $j = 1, \ldots, K^{t-1}$ at $t-1$, compute

$$\begin{aligned} E_j^{t-1} &= \sum_{k \in \mathcal{D}^t(j)} \frac{p_k^t}{p_j^{t-1}} \pi_k^t T_k^{t-1} \\ e_j^{t-1} &= \sum_{k \in \mathcal{D}^t(j)} \frac{p_k^t}{p_j^{t-1}} \left(\pi_k^t h_k^t + \lambda_k^t l_k^t - \mu_k^t u_k^t + \sum_{i=1}^{r_k^t} \rho_{k,i}^t d_{k,i}^t + \sum_{i=1}^{s_k^t} \sigma_{k,i}^t e_{k,i}^t \right) \end{aligned}$$

The current conditional expected value of all scenario problems in $\mathcal{D}^t(j)$ is then $\bar{\theta}_j^{t-1} = e_j^{t-1} - E_j^{t-1}x_j^{t-1}$. If the constraint $\theta_j^{t-1} = 0$ appears in $NLDS(t-1,j)$ then remove it, set $s_j^{t-1} = 1$, and add the optimality cut defined by $E_{k,j}^t x_j^t + \theta_k^t \geq e_{k,j}^t$ to $NLDS(t-1,j)$.
If $\bar{\theta}_j^{t-1} > \theta_j^t$, then increment s_j^{t-1} and add the optimality defined by E_j^{t-1} and e_j^{t-1} to $NLDS(t-1,j)$.
If $t = 2$ and no constraints are added to $NLDS(1,0)$ then stop with x_0^1 optimal. Otherwise, let $t = t-1$ and $k = 1$. If $t = 1$, set $DIR = FORW$. Go to step 1.

Figure 1. **Nested Decomposition Algorithm for Multistage Stochastic Linear Programs**

Communications between nodes in the solution tree are bidirectional. Each stage's right-hand side depends on the previous stage's decision vector via constraints (2.3). Primal solutions must be passed down the decision tree. Dual solutions for future periods are passed back up the tree in the form of cuts on the primal solutions.

Other communications within the tree are possible. For example, a cut placed in any node is valid for every other node in that stage, since it defines a support for the recourse function. Also, a basis which is optimal for a node in stage t may be optimal for other nodes in stage t. Communicating bases between nodes may reduce the effort necessary to solve all nodes in a stage. In a parallel computing environment, however, a trade-off exists between more detailed communications among parts of the tree (reducing the effort to solve the tree) and the (potentially) high time cost of that communiction.

The ND algorithm terminates finitely (Birge, [3]), since each cut can uniquely be assigned to an optimal basis of a subproblem, and there are a finite number of bases. Furthermore, the process terminates with an optimal solution (if one exists) since the algorithm only terminates when $\theta_j^{t-1} = \mathcal{Q}^t(x_j^{t-1})$ or the problem is infeasible or unbounded.

Several improvements to the above algorithm have been suggested in the literature. Three of these are described in the subsections below.

2.2 Sequencing protocol

The first concerns the order in which subproblems are are solved. Scott [27] investigated the issue for deterministic problems and suggested that a particular subproblem within a period need only be solved if it receives new information in the form of a new ancestor solution or cut. The choice of which period to solve (*sequencing protocol*) may vary without disrupting the algorithm's convergence. Once stage t (where $1 < t < H$) is solved, the algorithm can solve stage $t-1$ to recalculate new solutions to the nodes in that stage, or continue on to stage $t+1$ with a new RHS. At stages $t = 1$ and $t = T$, the choice is limited.

Three basic sequencing protocols have been suggested in the literature:

1. *Fast-Forward-Fast-Back* (FFFB). Wittrock [32] suggested that the fastest way to propagate information through the tree would be to change directions as little as possible. Specifically, the FFFB protocol starts going forward through the tree at stage 1, and continues to go in the same direction until a move in that direction is blocked. Blockage can occur when an end of the tree is reached or an infeasibility is discovered. (The FFFB protocol is also described in Figure 1.)

2. Forwards first (FF). Birge [3] first suggested an ND algorithm that only moves to stage $t-1$ when all current solutions for periods t through H are optimal, and no new primal solutions from stage $t-1$ are forthcoming. The effect of this protocol is to spend the most computational effort in the latter stages of the decision tree.

3. Backwards first (BF). Alternatively, the ND algorithm can always move back to stage $t-1$ unless no new cuts to that stage are generated. This was also described in (Gassman, [11]) as the "path of least resistance", since the algorithm returns to stages with fewer nodes whenever possible.

Gassman [11] tested these three protocols in a serial implementation of the ND algorithm. He showed that the FFFB protocol was slightly superior to the FF protocol for larger problems and that both were superior to the BF protocol. Opposite conclusions were drawn for smaller problems.

2.3 Multicuts

Birge and Louveaux [4] suggested that more information from a node's descendants may be gained by disaggregating optimality cuts (2.1). The cuts developed in the previous section summed the (weighted) dual variables from each node. The new set of cuts at each node are

$$\theta_{j,k}^{H-1} \geq e_{j,k} - E_{j,k} x_k^{H-1}$$

where

$$\begin{aligned} E_{j,k}^{t-1} &= \pi_k^t T_k^{t-1} \\ e_{j,k}^{t-1} &= \pi_k^t h_k^t + \lambda_k^t l_k^t - \mu_k^t u_k^t + \sum_{i=1}^{r_k^t} \rho_{k,i}^t d_{k,i}^t + \sum_{i=1}^{s_k^t} \sigma_{k,i}^t e_{k,i}^t. \end{aligned}$$

Multicuts may reduce the number of subproblems which must be solved to find a solution to the tree, but at the expense of a larger (and more complicated) subproblem at each node. Gassman [11] concluded that multicuts reduced serial solution time for large problems, but was not worth the additional overhead for smaller problems. Moreover, he found that multicuts may not be effective for tightly constrainted problems, since infeasibility in any descendant node renders the multicuts obsolete.

2.4 Bunching

When all objective coefficients and within period constraint matrices (W^t) are deterministic, a useful characteristic of the nested decomposition algorithm's subproblems is that they only differ in their right-hand sides if the same cuts have been added to every problem. Two techniques have been proposed in the literature for solving LPs with multiple right hand sides, sifting [9] and bunching [30]. (An overview may be found in Wets [31]). Since sifting assumes independence of the components of the right-hand side vector, we consider only bunching here.

The basic assumption Suppose we are given a list of right-hand sides, $\mathcal{H} = (\chi_1, \ldots, \chi_p)$ where $\chi_i = h_i - Tx$. Bunching takes a basis B which is optimal for some χ_i and tries to identify any other problems in $\mathcal{H}$ which are optimal for that basis. The goal is to minimize the number of full simplex pivots which must be performed to solve linear programs for all vectors in $\mathcal{H}$.

Two variants of bunching are possible. For both, a *paradigm problem* χ_i is chosen from H. The first variant, also called *trickling*, performs (see Gassman, [11]) a simplex pivot on this problem and checks whether that pivot is also valid for other problems in $\mathcal{H} - \{\chi_i\}$. If so, the pivot is taken, and the problem is marked DONE if the problem satisfies optimality conditions. The last basis visited for each problem is stored, along with a pointer to the next-to-last basis. Eventually, the paradigm problem will be finished. The procedure then travels backwards along the chain of (stored) bases until all other problems have been solved.

The second variant (see, e.g. [31]) solves the paradigm problem, and then checks other problems in $\mathcal{H}$ for dual feasibility. These problems are marked as done. A new paradigm problem is then chosen, preferably one only a few pivots away from the previous paradigm. This variant is simpler than the first, but requires more operations since the entire basis must be solved for each unclassified problem.

All of these procedures can be implemented in serial as well as parallel versions. Parallel implementations have special implications, however. The next section reviews the serial implementation used for this study and its adaptation to a distributed computing environment.

3 Serial Implementation

To implement the nested decomposition algorithm, the program ND-UM was written in C for IBM RS/6000 workstations. ND-UM works interactively with IBM's Optimization Subroutine Library (OSL, [17]), which solves each of the linear programs generated during the algorithm.

ND-UM is not restricted by any preset upper bounds on the number of variables, number of stages, or number of right-hand side realizations. The code allocates memory as needed. It is only limited by the memory capacities of the machines on which the code is run. For each stage t in the problem, ND-UM stores the linear program

$$\begin{array}{lll} \min & cx^t & \\ \text{s. t.} & W^t x^t & = h^t \\ & l^t \leq x^t \leq u^t & \end{array} \tag{3.4}$$

in OSL's memory. (The programs (3.4) are input sequentially in a separate file.) Cuts are stored in separate data structures and added to (3.4) as each node is solved.

Due to the sparsity of the T matrices of the problems considered, the T matrix for each stage $t < H$ is input and stored separately in memory. For each node in the scenario tree, ND-UM also stores: stage number, size of corresponding stage's W matrix, the set of right-hand side and interperiod constraint matrices, $(h_i^t, T_i^t; i = 1, \ldots, \bar{N}_t)$, the probability of realization i, and the number of possible descendants. Note that ND-UM currently only considers right-hand side and interperiod stochastic parameters so that bunching is possible.

To complete the description of the solution tree, each node contains pointers to the set of feasibility and optimality cuts and pointers to this node's parent, sibling or cousin, and eldest (first listed) child nodes. The code is capable of handling problems in which the right-hand side realizations in stage $t+1$ are either dependent or independent of stage t realizations. All problems considered here assume independence between successive stage right-hand side realizations.

ND-UM uses several techniques to speed the solution times to sets of nodes. First, nodes are not resolved unless a new (binding) cut from the next stage is generated. Second, the problems can be solved in the same order each time or can start from the last basis obtained for each node. If neither of these options are employed, OSL uses the previously optimal basis, which is a solution to an adjacent node, or, in the case of the first node in each stage, the solution to the last node in that stage. Finally, the last stage can be bunched by completely checking primal feasibility of a paradigm problem's optimal basis for each unsolved problem. Several paradigm selection rules were tried, including finding the problem with the fewest number of primal infeasiblities, and finding the problem closest to the mean of all unsolved problems. The effort to rank problems generally exceeded the benefit of better paradigm selection. The decision rule used here finds the first unsolved problem.

When an optimal solution exists, ND-UM declares optimality when $|\bar{\theta}_j^t - \theta_j^t|/|\theta_j^t + 0.1| \leq \epsilon$ for all nodes j in stage t with ϵ set to 10^{-6}.

4 Parallel Implementation

The parallel version of ND-UM is identical to the serial version, with additional coding for communications between processors. Communications tasks are performed using Parallel Virtual Machine version 3.1 (PVM, [13]), which allows for arbitrary connections between heterogeneous computers.

Instead of running a single computation task, the parallel implementation has two parts communicating between each other. The "Master" subprogram is responsible for the root of the problem tree and at least one path from the root to the last stage. The master starts all computations and initiates an arbitrary number of "slaves" to solve other subtrees. Once initiated, the slaves read in their data from disk and wait for the master to start the calculations.

All of the processors read input data from the same files used for the serial implementation. To minimize communication coding and minimize communications delays, each slave task reads in its own copy of the input data. An extra input file read by the master specifies all other configuration information, including nodes to split and processors to use.

The calculations are parallelized by splitting the original problem tree "horizontally," i.e. by cutting out an entire subtree along some path between two adjacent nodes in any two stages t and $t+1$, $(0 \leq t < T-2.)$ The node at stage $t+1$ acts as a root node for the slave processor and is called a *split node* (denoted $S_1, \ldots, S_p$). Each split node is in a *split stage*, and has a parent $\alpha(S_i, t+1)$. The portion of the tree from the root up to the parent of a split node is solved by the master task. The remaining subtree rooted at the split node is solved by a slave task.

The parallel algorithm proceeds from stage t to stage $t+1$ when the master solves the parent of a split node S_i, and passes it to each child, including S_i. Each child may be assigned either to the master or to a slave. If the child is a split node, the primal solution is passed via PVM. Otherwise, it is placed in ND-UM's internal data structures. Slave i then starts solving the multistage program rooted at S_i, while the master continues solving any children not assigned to slaves. Thus, the master and slave run independently after passing the communication phase.

After a slave finds a dual solution or a dual direction (in the case of infeasibility) to its root (split node), the solution is sent back to the master. The slave then waits for the master to send either a continuation signal (a new primal solution to the parent) or a termination signal. The termination signal is sent when the master determines that the entire problem is either solved or infeasible.

The master can either wait for a return signal from each slave only when it comes back to the split stage, or it can periodically check for a signal between moves to a new node or to a new stage (i.e. between steps in Figure 1). When the return signal is an optimal dual solution, the master uses this information to generate an optimality cut and proceeds normally.

When the return signal is a dual direction, if the master is currently solving the split stage, a feasibility cut is placed in the parent. If the master is not solving the split stage, and has been set to check for signals between tree moves, any resulting calculations are of no use. They implicitly assume an infeasible solution to an ancestor. In this case, the master immediately jumps back to the split stage and starts there with the feasibility cut derived from the slave.

Once the feasiblity cut has been placed in a parent node, the node must be resolved to get a new primal solution. Since the slaves may still be working using a previous (infeasible) solution, they can be interrupted to start at the split node $NLDS(S_i, t+1)$. Checking for interrupts requires a call to a PVM routine each time the algorithm changes stages or nodes. Since these checks might take a great deal of time, we tried the algorithm with interruptions and without interruptions. We generally found that checking for interruptions was worth the effort.

A variety of ways may be found to split a problem tree for parallelized calculations. A reasonable principle here is to assign as large a subtree as possible to each slave, thereby decreasing the size of the tree left for the master task. The obvious result then is spreading equal portions of the original problem tree among processors available for calculations (assuming the processors have similar speeds and abilities). To maximize the load on the slave, the tree should be cut as close to the tree root node as possible, i.e. between stages 1 and 2. In this case, each slave processor is responsible for the longest subtree possible.

Our parallel implementation allows an arbitrary number of processors to be used. Generally, the problem tree can be split in any place. The only constraints are that each processor, including the master, must have only one root (making the subproblems identical to one multistage program) and at least one path leading to the last stage of the tree. A slave task can not be started from another slave task, i.e. subtrees assigned to processors cannot intersect and are"parallel." Also, the minimal number of stages for each slave task is two (although this requirement can be removed easily). All the problems solved by the parallel code were decomposed using the above observations and rules. Each tree was split between $NLDS(1,1)$ and its children, and the slave processes were split evenly among the available workstations.

While the serial implementation of the algorithm always keeps the same sequencing protocol while solving a problem, some variations are possible in the parallel implementations. Within a slave, the fastest sequencing protocol is generally FFFB (as shown in [11] and confirmed in Section 5). In stages where the problem is split among processors, a different protocol is possible. To minimize communications between the master and slaves, we tried a *hybrid protocol.* In the hybrid protocol, each slave uses FFFB and the master uses FFFB for all stages after a split stage. For the

split stage itself (i.e. between stages 1 and 2), the master uses a forward first protocol.

Using a forward first protocol forces each slave to solve its subtree to full optimality, thereby increasing the chances that a better cut will be formed. If a better cut can be formed, fewer *restarts* (entrances to step 1 in Figure 1 with $t = 1$) may be necessary and overall solution (and, for our concern, communication) time may decrease.

A disadvantage to the hybrid protocol is that the master must wait until all slaves report back with optimal solutions to their subtrees. If the slaves are not well balanced, i.e. they do not solve their subtrees in roughly the same times, every processor except the slowest may be idle for a significant time. (We term these *coordination delays.*) We would expect the hybrid protocol to be sensitive to load balancing and that the portion of the tree assigned to the master should be larger or harder than the subtrees, which should be evenly balanced.

The hybrid protocol will also be appropriate for a heterogeneous network of processors. If one machine is much faster than the others, more of the tree can be assigned to the master (running on the fast machine) and fewer nodes can be assigned to slaves. A theoretical analysis is impossible without knowing in advance how the algorithm will proceed. Empirical tests are, however, possible and are reported in Section 5.

While the parallel code implements an algorithm identical to the one in the serial code, a parallelized solution path does not necessarily repeat the path that would be obtained from the serial implementation. For instance, we can think of a situation when several slave tasks report feasibility cuts to the same node. In the serial implementation, the sequence of actions in the analogous situation resolves this node until getting feasibility for the first descendant, then possibly getting a feasibility cut from the next descendant, resolving the node, etc. In the parallel implementation all these feasibility cuts can be received at once (or in different orders) if they come from different slave tasks. Other examples of different behavior can be found. Evidence of these differences may be found in Section 5, where one problem experienced superlinear speedup.

5 Computational Results

The implementation described in the previous section was tested on a suite of large stochastic programs from the literature. The `nd` program was tested against MSLiP and solution of the deterministic equivalent using OSL. We chose MSLiP because it is a very similar implementation to ours but has the additional feature of Gassmann's "trickling" form of bunching. We used comparisons with it to see the effect of this trickling. OSL is used because of its wide availability. We first review in some detail the problems in the test suite and then discuss our serial and parallel computational experience.

5.1 Problem Descriptions

The suite of seven problems, described completely in [15], includes five of six problems used in (Gassman [11]), a capacity expansion problem, and a network planning problem. Every problem except STOCHFOR and SCFXM is scalable to an arbitrary number of stages and scenarios. (SCFXM is scalable to an arbitrary number of scenarios only.) The problems were generated in a standard format proposed by [6] and in a format required by `nd` using software described in [16].

The problem types and their references are shown in Table 1.

Problem	*Description*
SCAGR7	A multiperiod dairy farm expansion model. [14]
SC205	A dynamic multisector development planning model. [14]
SCSD8	A model to find the minimal design of a multistage truss. [14]
SCFXM1	A real production scheduling problem of unknown origin. [14]
STOCHFOR	A forestry management problem. [10]
PLTEXP	A linear relaxation of a discrete capacity expansion problem [28].
STORM	A two period freight scheduling problem. [18]

Table 1. Problem descriptions.

Table 2 lists the number of stages, total scenarios, and number of scenarios in the final period for each problem. SCAGR7, SCSD8, SC205, and PLTEXP are scalable to any size. Instantiations for these problems were formed by assuming a certain number of scenarios (n) per stage (up to H). (The problem names in Table 2 are of the form Pn SH.) STORM was scaled only within 2 stages, STOCHFOR was taken directly from [11], and SCFXM was enlarged slightly from the versions used in [11]. The sizes (rows by columns) of the deterministic equivalent problems are also shown.

SCAGR7				SC205			
Problem	H	N_H	Deterministic Equivalent	Problem	H	N_H	Deterministic Equivalent
P4S2	4	8	(2.80E+02 , 3.00E+02)	P4S2	4	8	(3.20E+02 , 3.20E+02)
P4S4	4	64	(1.60E+03 , 1.70E+03)	P4S4	4	64	(1.90E+03 , 1.90E+03)
P4S8	4	512	(1.10E+04 , 1.20E+04)	P4S8	4	512	(1.30E+04 , 1.30E+04)
P4S16	4	4096	(8.30E+04 , 8.70E+04)	P4S16	4	4096	(9.60E+04 , 9.60E+04)
P4S32	4	32768	(6.40E+05 , 6.80E+05)	P4S32	4	32768	(7.40E+05 , 7.40E+05)
P5S2	5	16	(5.90E+02 , 6.20E+02)	P5S2	5	16	(6.70E+02 , 6.70E+02)
P5S4	5	256	(6.50E+03 , 6.80E+03)	P5S4	5	256	(7.50E+03 , 7.50E+03)
P5S8	5	4096	(8.90E+04 , 9.40E+04)	P5S8	5	4096	(1.00E+05 , 1.00E+05)
P5S16	5	65536	(1.30E+06 , 1.40E+06)	P5S16	5	65536	(1.50E+06 , 1.50E+06)
P6S2	6	32	(1.20E+03 , 1.30E+03)	P6S2	6	32	(1.40E+03 , 1.40E+03)
P6S4	6	1024	(2.60E+04 , 2.70E+04)	P6S4	6	1024	(3.00E+04 , 3.00E+04)
P6S8	6	32768	(7.10E+05 , 7.50E+05)	P6S8	6	32768	(8.20E+05 , 8.20E+05)
P7S2	7	64	(2.40E+03 , 2.50E+03)	P7S2	7	64	(2.80E+03 , 2.80E+03)
P7S4	7	4096	(1.00E+05 , 1.10E+05)	P7S4	7	4096	(1.20E+05 , 1.20E+05)

Table 2a. Problem Descriptions

SCSD8				SCFXM1			
Problem	H	N_H	Deterministic Equivalent	Problem	H	N_H	Deterministic Equivalent
P4S2	4	8	(1.50E+02 , 1.10E+03)	s1	3	4	(5.20E+02 , 8.20E+02)
P4S4	4	64	(8.50E+02 , 6.00E+03)	s2	3	8	(7.80E+02 , 1.30E+03)
P4S8	4	512	(5.90E+03 , 4.10E+04)	s3	3	32	(2.40E+03 , 4.30E+03)
P4S16	4	4096	(4.40E+04 , 3.10E+05)	s4	3	128	(4.50E+03 , 8.40E+03)
P4S32	4	32768	(3.40E+05 , 2.40E+06)	s5	3	512	(4.70E+04 , 9.00E+04)
P5S2	5	16	(3.10E+02 , 2.20E+03)	STOCHFOR			
P5S4	5	256	(3.40E+03 , 2.40E+04)	s1	7	8	(6.60E+02 , 6.20E+02)
P5S8	5	4096	(4.70E+04 , 3.30E+05)	s2	7	64	(2.20E+03 , 2.00E+03)
P5S16	5	65536	(7.00E+05 , 4.90E+06)	s3	7	144	(4.80E+03 , 4.50E+03)
P6S2	6	32	(6.30E+02 , 4.40E+03)	s4	7	288	(9.50E+03 , 8.90E+03)
P6S4	6	1024	(1.40E+04 , 9.60E+04)	s5	7	384	(1.30E+04 , 1.20E+04)
PLTEXP				s6	7	512	(1.70E+04 , 1.60E+04)
P2S4	2	4	(4.80E+02 , 9.70E+02)	STORM			
P2S8	2	8	(8.90E+02 , 1.80E+03)	s1	2	1	(585 , 1380)
P2S16	2	16	(1.70E+03 , 3.50E+03)	s2	2	10	(1647 , 12711)
P3S6	3	36	(4.40E+03 , 9.00E+03)	s3	2	50	(6367 , 63071)
P3S16	3	144	(1.60E+04 , 3.30E+04)	s4	2	100	(1.20E+04 , 1.30E+05)
P4S16	4	4096	(4.50E+05 , 9.20E+05)	s5	2	1000	(1.20E+05 , 1.30E+06)

Table 2b. Problem Descriptions

5.2 Serial Results

To ensure that the ND-UM implementation was competitive with other serial algorithms, a series of single processor tests were undertaken. Each problem in the test set was run on an IBM RS/6000 model 320H using three programs. MSLiP was run using both single cuts/no bunching and multi-cuts/bunching. OSL's primal simplex algorithm was used directly on the deterministic equivalent problems. ND-UM was used using single cuts. With one exception, each program was compiled using default optimizations and run with default parameters. MSLiP was modified to have the same stopping criterion as ND-UM. Its data structures were enlarged to solve larger problems. One key difference between MSLiP and ND-UM is that MSLiP always uses its trickling form of bunching in the last stage.

Figure 2 shows serial running times for SCAGR7, which generally has few distinct optimal bases for each stage. Due to the nature of the problem, sharing bases between nodes was turned off. The bunching routine was less efficient than the default and was turned off. Comparing the vertical positions of the curves, we see that ND-UM is generally faster than MSLiP-SC. Since the methods are quite similar, the differences in solution speed are mainly due to the efficiency of the linear programming solver. Profiling tests performed on ND-UM support this contention. Figure 1 also shows that ND-UM's dynamic memory allocation allows substantially larger problems to be solved than the other codes. MSLiP and OSL, are FORTRAN-based, and cannot manage memory as effectively.

Figures 3 and 4 present the serial solution times for the other problems in the test set. STORM is a large problem with many random RHSs, almost all of which have unique optimal bases. The nested decomposition algorithm cannot benefit from basis sharing or bunching and, when a direct OSL solution is available, the two methods are roughly equivalent in solution time.

SCFXM1 is a "fat" 3-stage problem with many scenarios in each stage, while STOCHFOR is a "thin" 7-stage problem with a few scenarios in each stage. Comparing results for the two indicates ND-UM is better suited to the "fatter" problems than "thinner," again since most time is spent solving nodes.

Solution times for STOCHFOR indicate that multicuts slightly improves performance and that the effects of bunching the last stage can be substantial. Results for the PLTEXP problem, which is both fat and long, support this observation. However, the four-stage problem suggests that large instances of the problem require a very efficient bunching algorithm. It appears that MSLiP's routine has substantial advantages in this circumstance.

The results hold for SCSD and SC205, shown in Figure 4, are consistent with the other test problems. We could only get MSLiP to solve four-stage SCSD problems, so full comparisons for that problem cannot be made. SC205 is a mainly a problem to find a feasible solution. It has very few optimality cuts. The performance of ND-UM suggests that there is no computational difference between problems predominantly solved with feasibility cuts and those solved with optimality cuts.

Figure 5 shows the effects of sequencing protocol for a few representative problems. As conjectured by Wittrock [32], and confirmed by Gassman [11], the Fast-Forward-Fast-Back protocol is the fastest on these problems for serial solution. Since forward first is considerably slower than the others, the necessity of obtaining quick solutions close to optimal for early stages is evident for serial algorithms.

Taken together, Figures 2 though 5 confirm that the ND-UM implementation is indeed comparable with MSLiP and has some advantage with larger subproblems. Both decomposition implementations (as expected) are considerably better than solving the deterministic equivalent for medium and large problems. For smaller problems, solving the deterministic equivalent with OSL is generally faster.

5.3 Parallel Results

To investigate the efficacy of a parallel implementation of the nested decomposition algorithm, a subset of problems discussed in the previous section were run on a network of RS/6000 model 320H workstations, connected by a local ethernet. To make the subtrees assigned to nodes as large as possible, problems were split in the second stage and evenly distributed across up to 8 processors. Nodes within each tree did not share bases. Bunching was not included. (The potential effects of these options are currently under study.) Each instance was run several times to minimize the effects of network traffic, and the best solution times were recorded.

Parallel solution times for larger instances of 6 of the 7 test problems are shown in Table 3, and are plotted on Figures 2, 3, and 4. Since a primary goal is to indicate preferred sequencing protocols in multistage implementations, we only tested problems with three or more stages. The two-stage problem, STORM, was not tested.

Table 3 also shows speedups and efficiencies for the test problems. The speedups shown are the ratios between the serial wall clock times and parallel wall clock times. The efficiency measure is defined as the speedup divided by the number of processors used. (An efficiency of 1.0 indicates linear speedup.)

Problem		Serial	P	Wall Clock Time		Speedup		Efficiency	
				Hybrid	FFFB	Hybrid	FFFB	Hybrid	FFFB
SCAGR7	P4S4	6	4	14	12	0.43	0.50	0.11	0.13
	P4S16	207	8	99	51	2.09	4.06	0.26	0.51
	P5S16	7361	8	1711	1218	4.30	6.04	0.54	0.76
	P6S8	NA	8	927	280	NA	NA	NA	NA
SC205	P5S8	47	8	24	21	1.96	2.24	0.24	0.28
SCFXM1	s3	33	2	45	28	0.73	1.18	0.37	0.59
	s4	44	2	62	33	0.71	1.33	0.35	0.67
	s5	63	2	64	62	0.98	1.02	0.49	0.51
STOCHFOR	s4	190	4	220	82	0.86	2.32	0.22	0.58
	s5	288	4	302	88	0.95	3.27	0.24	0.82
	s6	373	4	459	105	0.81	3.55	0.20	0.89
SCSD	P4S8	90	8	74	36	1.22	2.50	0.15	0.31
	P4S16	536	8	343	125	1.56	4.29	0.20	0.54
	P4S32	3146	8	2756	1221	1.14	2.58	0.14	0.32
	P5S8	835	8	560	201	1.49	4.15	0.19	0.52
	P5S16	NA	8	9139	6957	NA	NA	NA	NA
PLTEXP	P3S6	35	6	9	7	3.89	5.00	0.65	0.83
	P3S16	225	8	22	18	10.23	12.50	1.28	1.56

Table 3. Parallel Solution Times: Even process balancing

The results in Table 3 indicate that parallelization generally works well, but success is relatively problem and sequencing protocol dependent. In some cases, the parallel algorithm takes a much faster solution path than the serial code, and can give near-linear or even superlinear performance. For PLTEXP, we found that the parallel implementation forces subproblem solutions to be near each other, analagous to bunching. Close solutions in later stages seem to speed convergence by making closer cuts in early stages.

Conversely, there are two cases where the parallelization loses effectiveness. Problems (such as SCSD P4S32) with many children per parent benefit from bunching or more global basis sharing than is possible on multiple processors. Also, problems (such as SC205 and SCFXM) with many feasibility cuts do not parallelize well because a single bottleneck scenario determines the feasibility cut. The empirical result of poor speedup is to be expected, since most of the work lies in creating feasibility cuts in the earliest stages of the problem. Comparatively little work can be distributed to independent subtasks.

Several other observations can be made from Table 3. First, the parallel implementation is not as effective for small and mid-sized problems as it is for the larger problems. This seems natural, since the time overhead imposed by communication and coordination delays outweighs the benefits of parallelization for smaller problems. For large problems that do not have substantial number of children per parent, significant speedups are obtained with the FFFB sequencing protocol.

Another advantage of the parallel implementation is to allow much larger problems to be solved. This is a natural consequence of splitting the tree among multiple processors (and hence multiple memories). The largest problems studied for this study were SCAGR P6S8, SCSD P4S32 and P5S16. The latter problem's deterministic equivalent has approximately 5 million columns and 700,000 rows. It was solved (with relative tolerances in each subproblem of 10^{-6}) in just over 2.5 hours. The only limitation to solving larger problems is availability of processors and memory in each processor.

The results indicate that the FFFB sequencing protocol is superior to the hybrid sequencing protocol for every problem. Waiting for a high-quality dual solution to improve optimality cuts requires waiting for each slave task to solve its subproblem completely. (The sole exception is

when a slave is interrupted with a new primal solution resulting from an infeasibility elsewhere in the problem. Since most problems obtain feasibility quickly, these interruptions do not occur frequently.)

Figures 6 and 7 show these results in more detail. The bars in each figure represent the ranges of node process utilizations, which were measured by the ratio of wall clock time spent processing over total wall clock time. The diamonds are the utilization percentages for the host. Ideally, each process will have a high utilization. The slaves will be equally utilized (i.e. have a "short" bar), and the master and slaves will have roughly equal utilizations (i.e. the diamonds and bars will be close to one another.)

Coordination delays can occur either among slaves or between the slaves as a group and the master. Comparing the load balances between the FFFB and hybrid balancing profiles shows that the biggest delays result from the latter. For every problem except SCFXM, the hybrid protocol generally loads too much work on the slaves and not enough on the master. A major reason for this is that the master is able to share bases among adjacent nodes. The slaves are unable to do so, and require more work to obtain the same solutions. The slaves evidently are able to solve their subtrees in roughly the same times, since the ranges of processor utilizations are generally tighter for the hybrid algorithm. These observations suggest that the hybrid protocol may give the best load balancings *provided* the work is distributed more evenly between master and slaves.

We investigated the effects of uneven load balancings on SCSD, the problem with the poorest mismatch in terms of master-slave utililizations. Table 4 shows the solution times when more nodes are added to the master and fewer nodes are assigned to slaves. Reassigning nodes to the host makes better use of the available processors, but still gives longer solution times, since the theoretically best possible speedup decreases substantially. As can be seen in the last two columns of Table 4 and in the lower numbered columns on the left of Figure 8, processor utilization increases, and the fraction of best obtainable speedup increases, but the problems still take longer than with FFFB. Also, P4S32 (2 nodes per processor) shows that further efficiency gains are difficult to obtain after some point.

SCSD		Nodes per	Wall Clock Time		Speedup		Theoretical Best speedup		Percentage of Best Possible	
Problem	Serial	Slave	Hybrid	FFFB	Hybrid	FFFB	Hybrid	FFB	Hybrid	FFFB
P4S16	536	2	343	125	1.56	4.29	8	8	20%	54%
	536	1	532	478	1.01	1.12	2	2	50%	56%
P4S32	3146	4	2972	1187	1.06	2.65	8	8	13%	33%
	3146	3	2551	2566	1.23	1.23	2.91	2.91	42%	42%
	3146	2	4771	4314	0.66	0.73	1.78	1.78	37%	41%

Table 4. Parallel results with uneven node partitionings

As discussed in Section 3, the hybrid protocol on a serial machine will minimize the number of restarts needed to solve the entire tree. (The number of restarts is half the number of messages passed between master and slave.) When the solution path varies due to communications delays, this effect cannot be guaranteed. We measured the number of restarts for each problem and found that the differences in restarts between the two protocols can be substantial.

Figure 9 plots the number restarts for each problem. The two problems with the most dramatic differences are SCSD and STOCHFOR. The plots for SCSD suggest that adding more scenarios per period does not substantially change the number of restarts, but adding a stage adds many more restarts to a problem solved with the FFFB protocol. All of the STOCHFOR problems have the same number of stages and differ mainly in the number of nodes in the last few stages. In light of the benefits of bunching for this problem, the decrease in the number of restarts suggests that adding scenarios past a certain level of detail does not substantially change the time necessary to

solve the problem. Problems with very few optimal bases (such as SCAGR7) or very many optimal bases (such as PLTEXP) appear to be insensitive to the choice of protocol.

The last study was undertaken to investigate the effects of different numbers of machines on parallel performance. Two problems were split evenly across subsets of the eight IBM RS/6000 workstations. Our results, shown in Table 5, show that efficiencies are higher for smaller numbers of processors.

Problem	Serial Time	Machines	Parallel Time	Speedup	Efficiency
SCAGR7 - P5S16	7361	4	2005	3.67	92%
	7361	5	1824	4.04	81%
	7361	8	1260	5.84	73%
SCSD - P4S16	536	2	420	1.28	64%
	536	3	307	1.75	58%
	536	4	216	2.48	62%
	536	8	132	4.06	51%

Table 5. Different Numbers of Processors

As might be expected, less communication delay and better coordination occurs with fewer processors. Also, the master and slaves spend less time idle. The far right portion of Figure 8 confirms that load imbalances are reduced for smaller sets of processors. These figures suggest that the efficiency of the nested decomposition algorithm is robust for loads down to a single node in the second stage per processor, as long as the work is evenly balanced.

6 Conclusion

This study has shown that the nested decomposition algorithm for solving multistage stochastic linear programs can effectively be parallelized without substantial modification. The parallel implementation takes advantage of the independence of subtrees within the solution tree to distribute the solution work among different slave tasks. Each slave task requires only a minimum of communication with a master task to solve the entire problem. As a result, the nested decomposition is well-suited to networks with comparatively slow communication times.

Several design choices are possible when implementing the nested decomposition algorithm. We focussed on the sequencing protocol for moving between stages, which in previous papers [11] has been shown to be a major determinant in solution speed. A fast-forward, fastback and a hybrid protocol were implemented and compared on several practical problems.

Computational tests suggest that the FFFB protocol is fastest both for serial and parallel environments. The hybrid protocol appears better suited for parallel environments with excessively slow communications or with one machine which dominates the others in computational power. If the FFFB protocol is used, good speedups are possible when the problems do not have many children per parent or require many feasibility cuts. In some cases, near linear or even superlinear speedups were observed. The parallel implementation also enables the solution of extremely large-scale problems ranging up to almost 5 million variables and 1.5 million constraints.

References

[1] Ariyawansa, K.A. and D.D. Hudson, 1991. "Performance of a Benchmark Parallel Implementation of the Van Slyke and Wets Algorithm for Two-Stage Stochastic Programs on the Sequent/Balance," *Concurrency: Practive and Experience*, Vol. 3(2), pp. 109-128.

[2] A. J. Berger, J. M. Mulvey, and A. Ruszczyński, 1993. "A distributed Scenario Decomposition Algorithm for Large Stochastic Programs," Technical Report SOR-93-2, Department of Civil Engineering and Operations Research, Princeton University, Princeton, NJ, 08544.

[3] J. R. Birge, 1985. "Decomposition and Partitioning Methods for Stochastic Linear Programs, " *Operations Research, 33*, 989-1007.

[4] J. R. Birge and F. V. Louveaux, 1988. "A multicut algorithm for two-stage stochastic lienar programs," *European JOurnal of Operations Research*, 34, pp. 384-392.

[5] J. R. Birge, and D. Holmes, 1992. "Efficient solution of two-stage stochastic linear programs using interior point methods," *Computational Optimization and Applications*, 1, pp. 245-276.

[6] J. Edwards, 1988. "A Proposed Standard Input Format for Computer Codes which solve Stochastic Programs with Recourse," **Numerical Techniques for Stochastic Optimization**, Yu. Ermoliev and R.J.-B. Wets (eds).

[7] E. Entriken, 1988. "A Parallel Decomposition Algorithm for Staircase Linear Programs," Technical report SOL 88-21, Department of Operations Research, Stanford University, Stanford, CA 94305.

[8] R. Fourer, D. Gay, and B. Kernighan, 1987. "AMPL: A Mathematical Programming Language," Computing Science Technical Report No. 133.

[9] S. Gartska and D. Rutenberg, 1973. "Computation in discrete stochastic programs with recourse," *Operations Research* 21, pp. 112-122.

[10] H. I. Gassman, 1989. "Optimal harvest of a forest in the presence of uncertainty," *Canadian Journal of Forest Research*, Vol 19, pp. 1267-1274.

[11] Gassman, H.I., 1990. "MSLiP: A Computer Code for the Multistage Stochastic Linear Programming Problem, " *Mathematical Programming 47*, 407-423.

[12] Gassman, H.I., 1990. " MSLip User's Guide, " Working Paper WP-90-4, School of Business Administration, Dalhousie University, Halifax, Canada.

[13] A. Geist, A. Beguelin, J. Dongarra, W. Jang, R. Mancheck, and V. Sunderam, 1993. "PVM 3 User's Guide and Reference Manual" Report ORNL/TM-12187, Oak Ridge National Laboratory, Deparment of Energy, Oak Ridge, Tennessee.

[14] J.K. Ho and E. Loute, 1981. "A Set of Linear Programming Test Problems," *Mathematical Programming 20* 245-250.

[15] D. Holmes, 1993. "A Collection of Stochastic Programming Problems," Technical Report (in preparation), Industrial and Operations Engineering Department, University of Michigan, 48109-2117.

[16] D. Holmes, 1993. "Selected *Mathematica* Routines for use in mathematical programming." Technical Report 93-26, Industrial and Operations Engineering Department, University of Michigan, 48109-2117.

[17] International Business Machines Corp., 1991. "Optimization Subroutine Libary Guide and Reference, Release 2," document SC23-0519-02, International Business Machines Corp.

[18] J. M. Mulvey, and A. Ruszczy/'nski, 1992. "A New Scenario Decomposition Method for Large Scale Stochastic Optimization," Technical Report SOR-91-19, Dept. of Civil Engineering and Operations Research, Princeton Univ. Princeton, N.J. 08544

[19] J. M. Mulvey and H. Vladimirou, 1989. "Stochastic network optimization model for investment planning," *Annals of Operations Research.* Vol. 20, p. 187.

[20] J. M. Mulvey and H. Vladimirou, 1992. "A new scenario decomposition method for large-scale stochastic optimization," Technical Report SOR 91-19, Dept. of CEOR, Princeton University, Princeton, NJ.

[21] J. M. Mulvey and H. Vladimirou, 1992. "A diagonal quadratic approximation method for large scale linear programs," *Operations Research Letters* 12, pp. 205-215.

[22] B.A. Murtaugh and M.A. Saunders, 1983. "MINOS 5.1 User's Guide," Technical Report SOL-83-20R, Systems Optimization Laboratory, Stanford University, Stanford, California.

[23] S. Nielsen and S. A. Zenios, 1990. "A Massively Parallel Algorithm for Nonlinear Stochastic Network Problems," Technical Report 90-09-08, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104.

[24] R. T. Rockafellar and R. J.-B. Wets, 1991. "Scenarios and policy aggregation in optimization under uncertainty," *Mathematics of Operations Research,* 16, pp. 119-147.

[25] A. Ruszczyński, 1993. "Parallel Decomposition of Multistage Stochastic Programing Problems, " *Mathematical Programming*, 58, pp. 201-228.

[26] A. Ruszczy ński, 1986. "A regularized decomposition tistage method for minimizing a sum of polyhedral functions, " *Mathematical Programming*, 35, pp. 309-333.

[27] D. M. Scott, 1985. " A dynamic programming approach to time staged convex programs," Technical report SOL 85-3, Systems Optimization Laboratory, Stanford University, Stanford, CA.

[28] M. J. Sims, 1992. "Use of a stochastic capacity planning model to find the optimal level of flexibility for a manufacturing system," Senior Design Project, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI 48109.

[29] R. Van Slyke and R. J.-B. Wets, 1969. "L-Shaped linear programs with application to optimal control and stochastic optimization," *SIAM Journal on Applied Mathematics* 17, 625-638.

[30] D. Walkup and R. J.-B. Wets, 1969. "Lifting projections of convex polyhedra," *Pacific Journal of Mathematics*, 28, pp. 365-475.

[31] Wets, R. J.-B., 1988. "Large scale linear programming techniques," **Numerical Techniques for Stochastic Optimization**, Yu. Ermoliev and R.J.-B. Wets (eds), Springer- Verlag, Berlin.

[32] R. Wittrock, 1985. "Dual nested decomposition of staircase linear programs," *Mathematical Programming Study* 24, pp. 65-86.

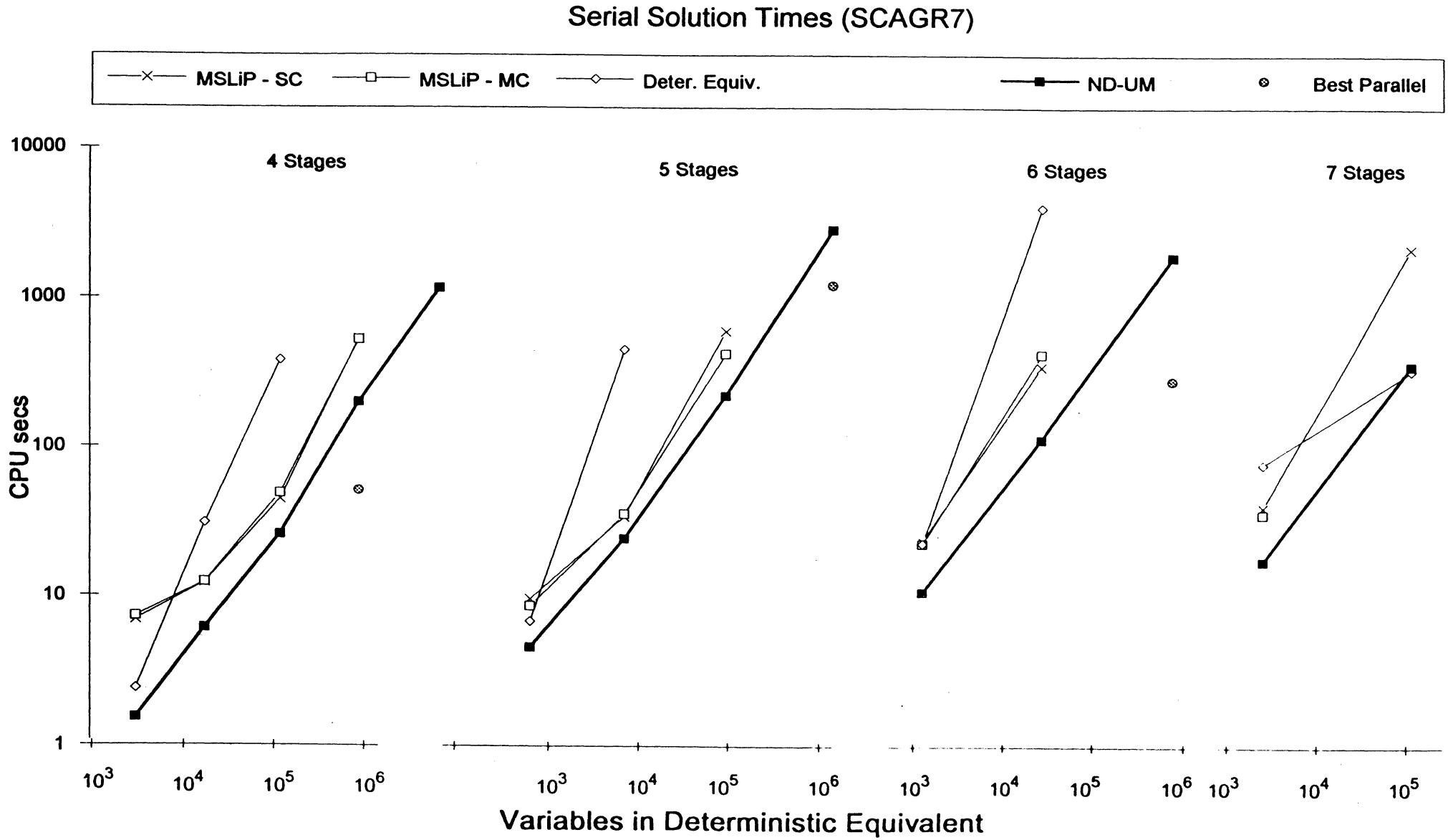

Figure 2. Comparison of solution times on SCAGR7.

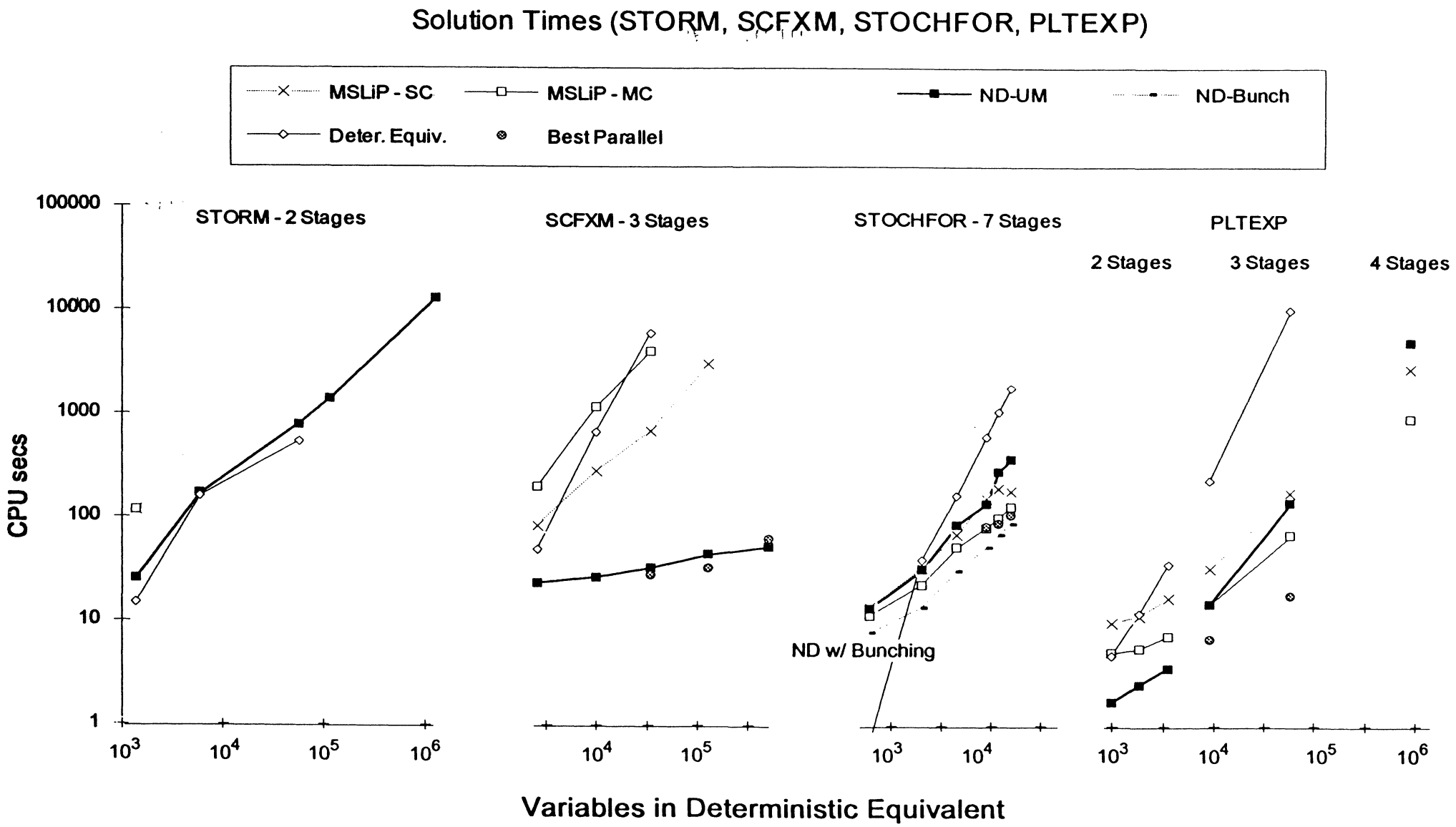

Figure 3. Comparison of solution times on STORM, SCFXM, STOCHFOR, and PLTEXP.

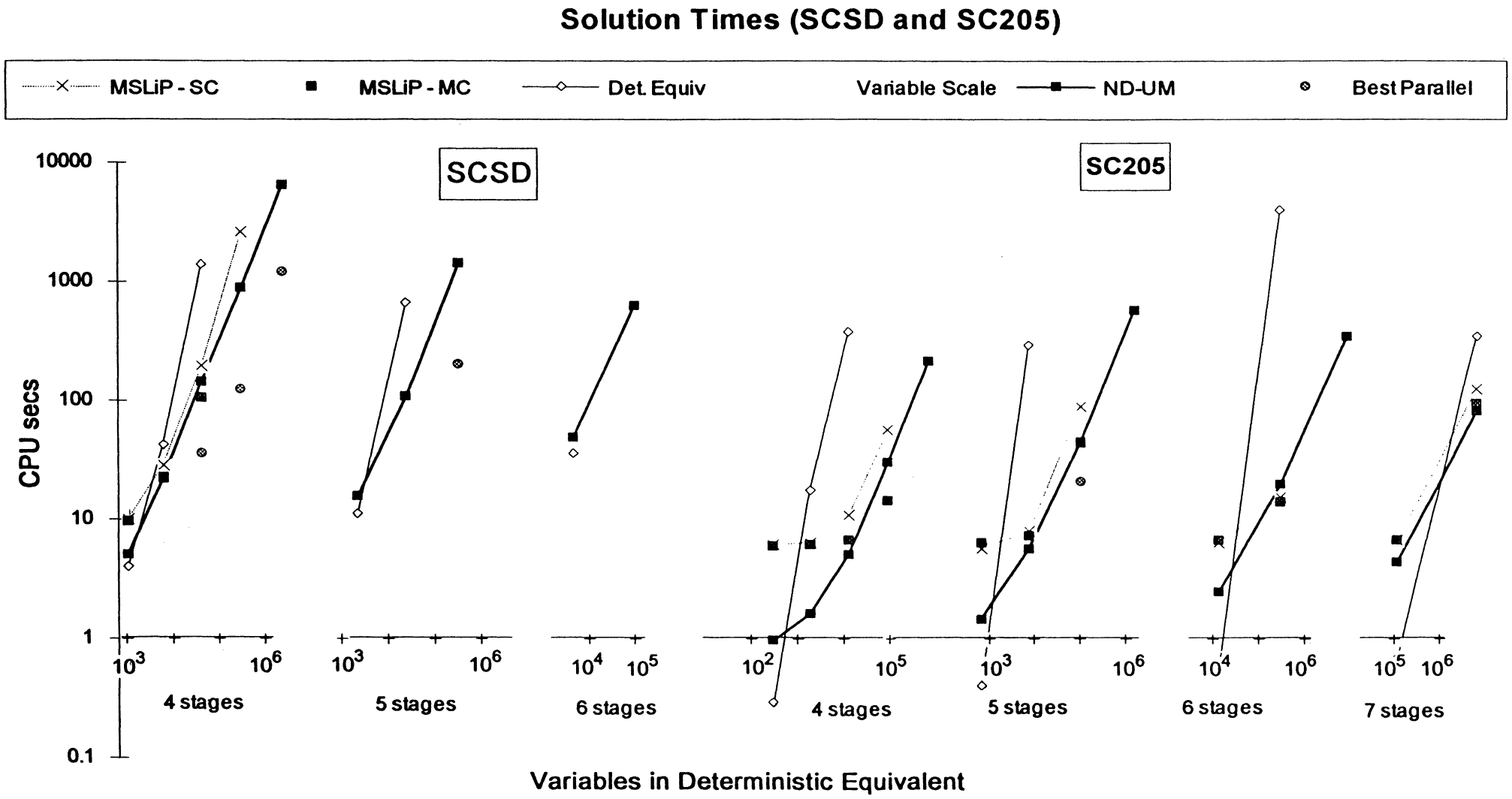

Figure 4. Comparison of solution times on SCSD8 and SC205.

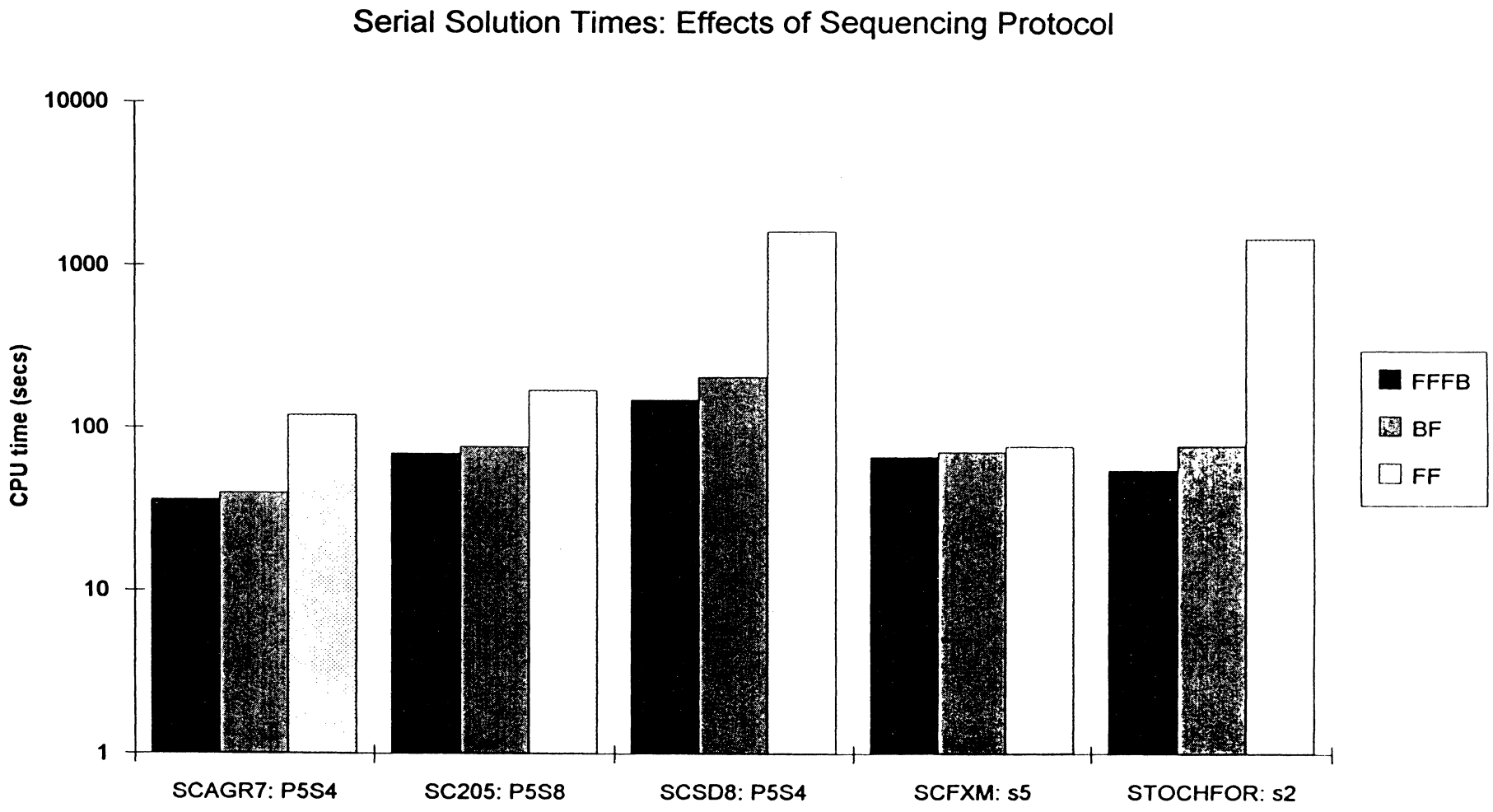

Figure 5. Effects of sequencing protocol.

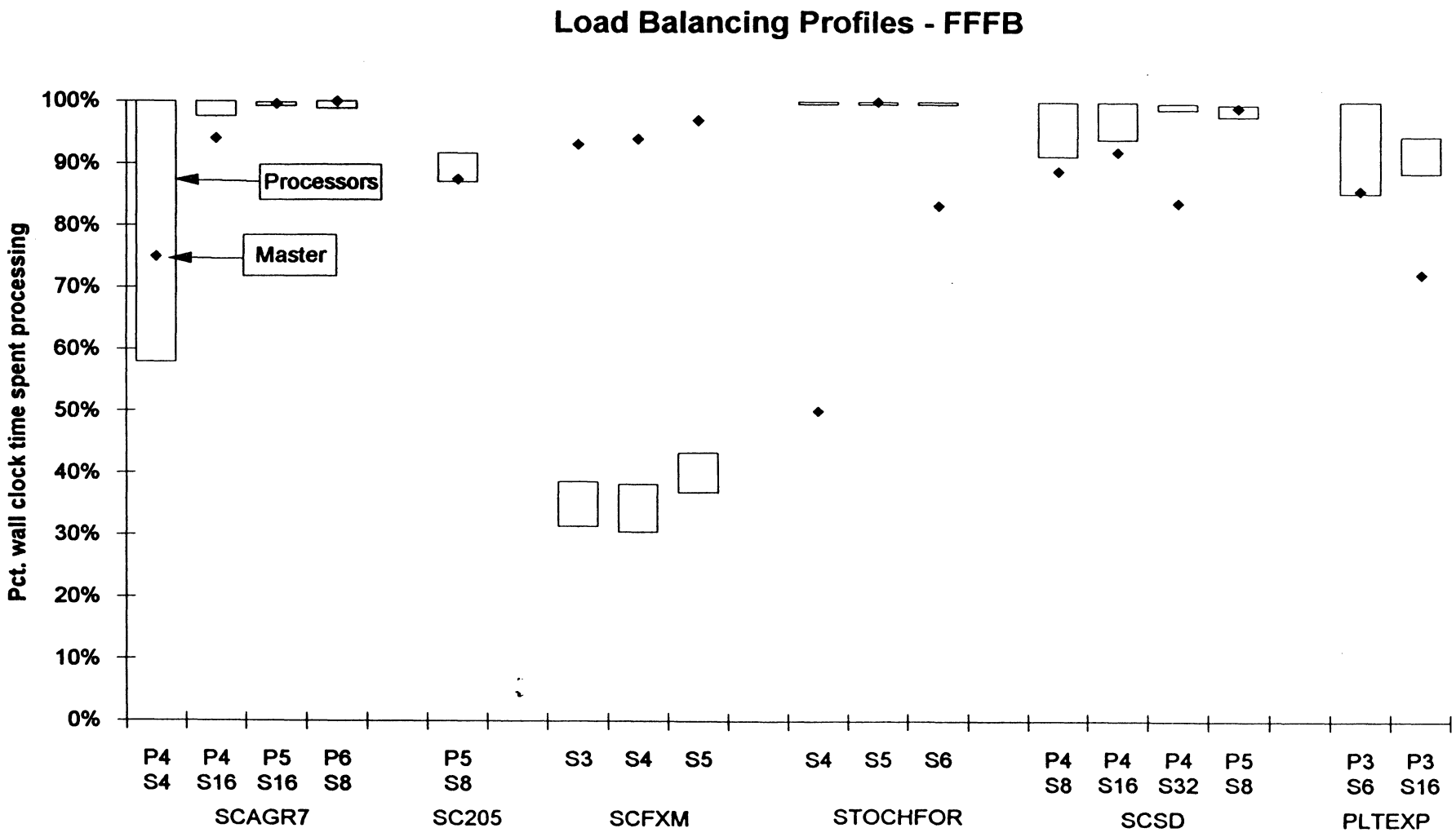

Figure 6. Load balancing profiles with FFFB.

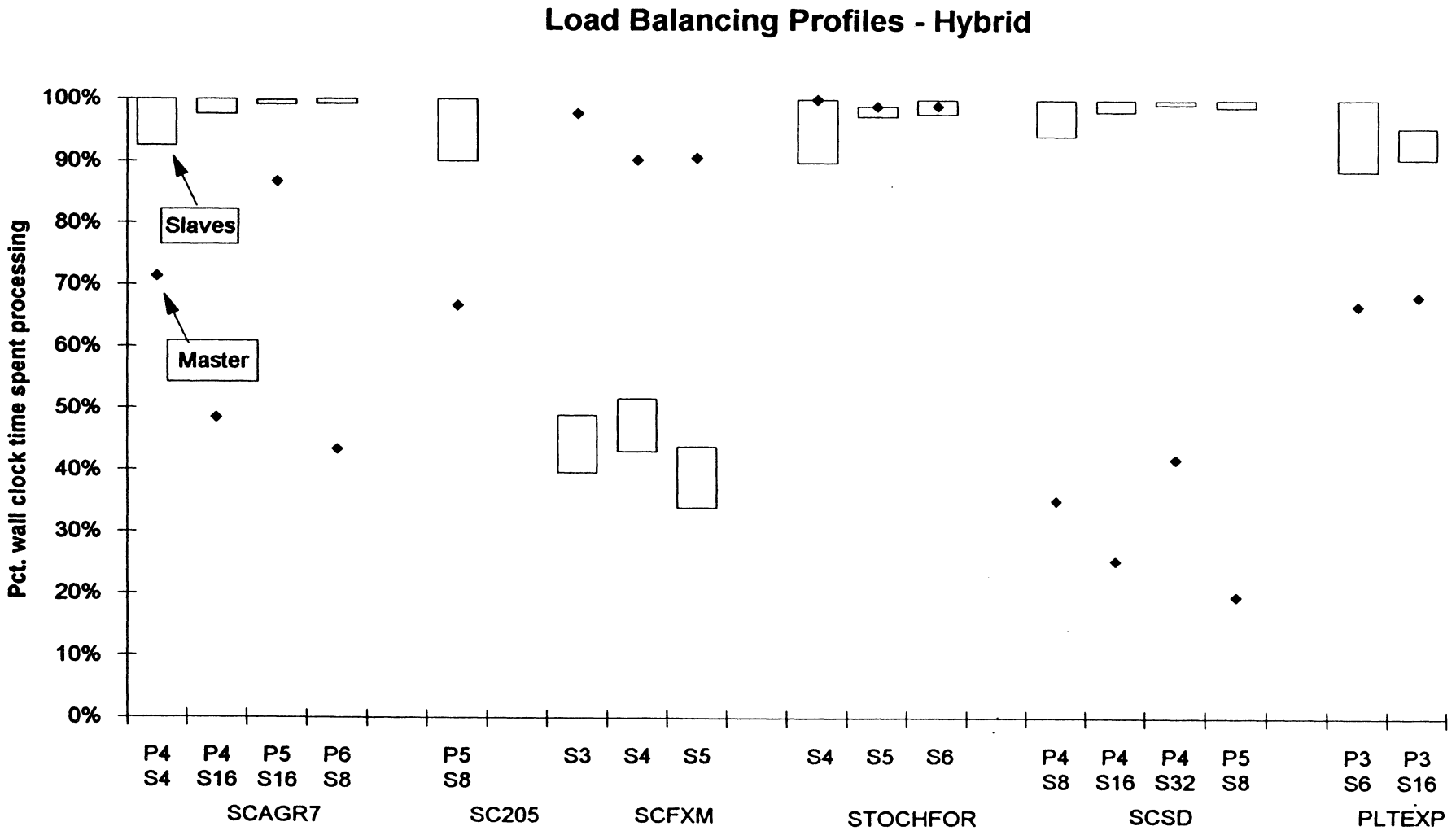

Figure 7. Load balancing profiles with Hybrid.

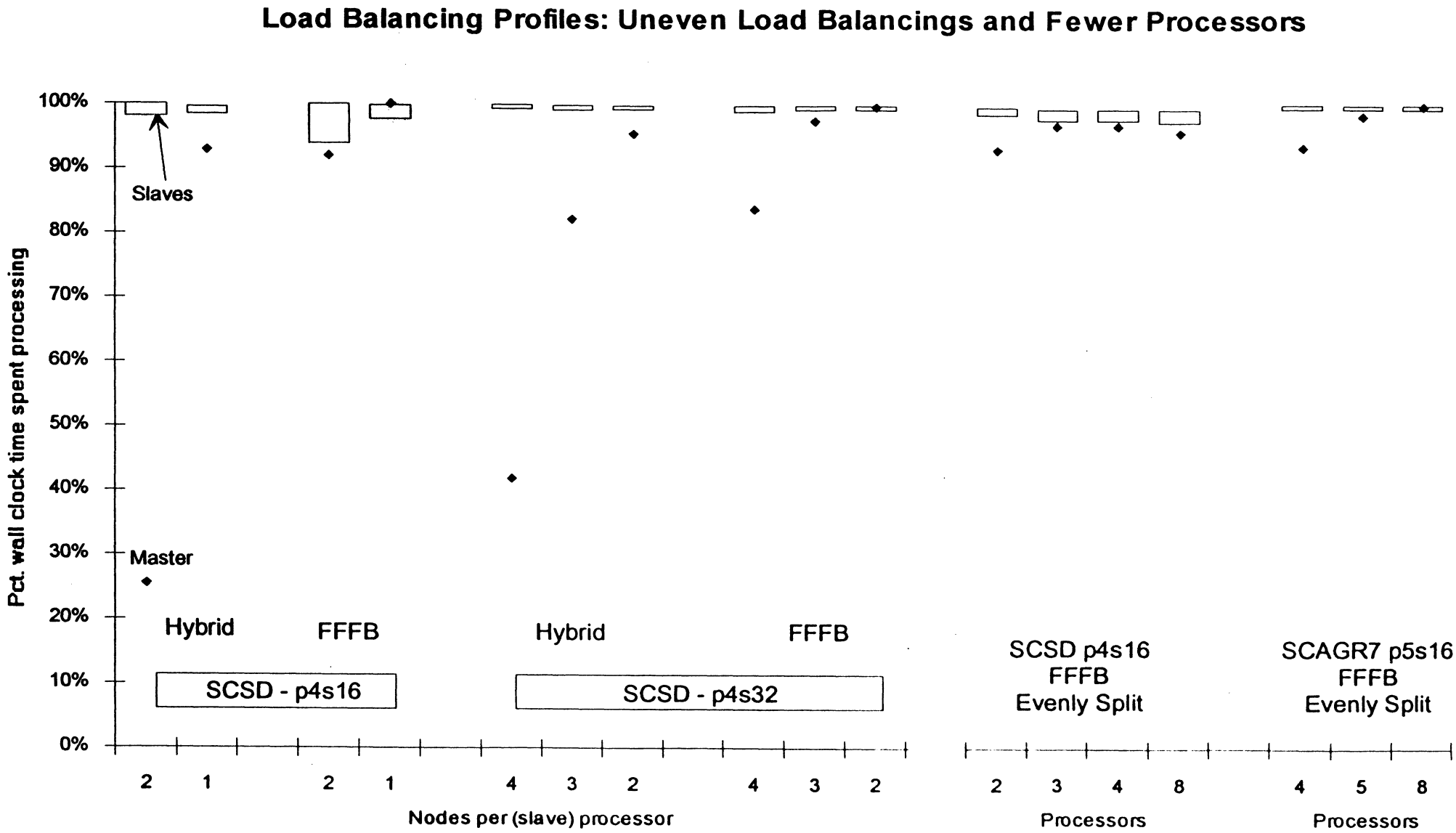

Figure 8. Load balancing profiles with uneven load balancing.

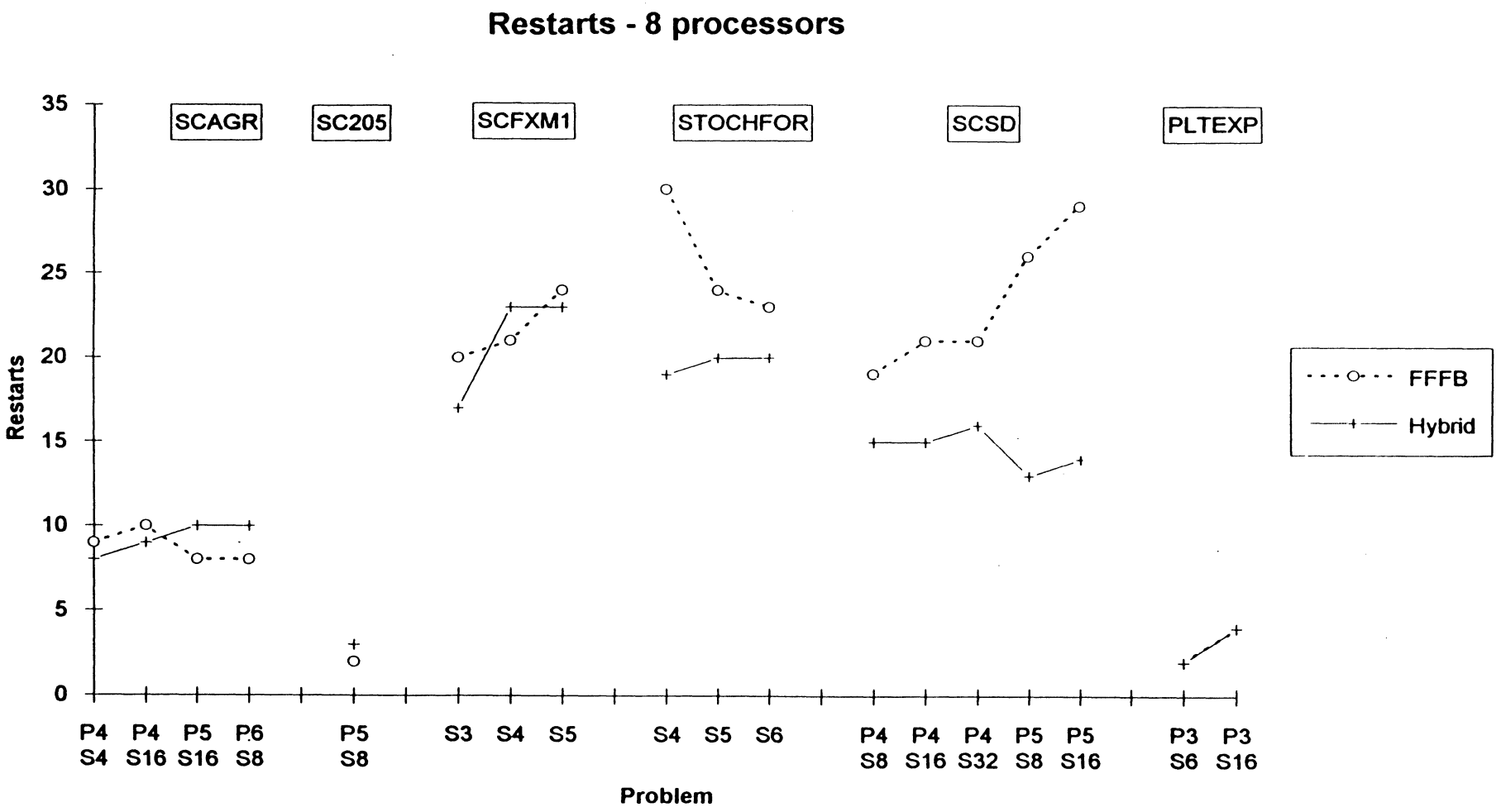

Figure 9. Restarts with FFFB and Hybrid.