ELSEVIER

Parallel Computing 26 (2000) 3–25

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Selected parallel optimization methods for financial management under uncertainty☆

G.Ch. Pflug*, A. Świętanowski

Institut für Statistik, Operations Research und Computerverfahren, Universität Wien, Universitätsstr. 5, A-1090 Wien, Austria

Received 1 May 1998; received in revised form 1 April 1999

Abstract

A review of some of the most important existing parallel solution algorithms for stochastic dynamic problems arising in financial planning is the main focus of this work. Optimization remains the most difficult, time and resource consuming part of the process of decision support for financial planning under uncertainty. However, other parts of a specialized decision support system (DSS) are also briefly outlined to provide appropriate background.

Finally, financial modeling is but one of the possible application fields of stochastic dynamic optimization. Therefore the same fairly general methods described here are also useful in many other contexts.

Authors hope that the overview of this application field may be of interest to readers concerned with development of parallel programming paradigms, methodology and tools. Therefore special care was taken to ensure that the presentation is easily understandable without much previous knowledge of theory and methods of operations research.

Keywords: Stochastic optimization; Parallel computation; Financial planning

☆ This research was a part of Special Research Program SFB F011 AURORA supported by Austrian *Fonds zur Förderung der wissenschaftlichen Forschung.*

* Corresponding author.

E-mail addresses: georg.pflug@univie.ac.at (G.C. Pflug), swietanowski@bigfoot.com (A. Świętanowski).

0167-8191/00/$ - see front matter
PII: S0167-8191(99)00093-9

1. Introduction

Large-scale optimization methods, especially for structured problems, such as dynamic, stochastic and stochastic dynamic problems, have long been known for their extreme requirements on computer memory and computing power. Each significant increase in available processing power, and especially the advent of parallel computers, was seen as a chance to solve new important and difficult classes of optimization problems. Yet one cannot help noticing the discrepancy between the availability of parallel computers in numerous research centers as well as commercial institutions and the availability of specialized optimization software able to utilize those vast resources.

This is not caused by the lack of appropriate parallel algorithms: those have been proliferating for more than a decade now (not to mention the parallel methods that came before the time of parallel computers). One can enumerate decomposition-based approaches like [2,14,18,24,29,31,33], data parallel algorithms [20,21,32] and even specializations of general optimization methods for solution of a structured problem, like [7,34]. Some generic parallel optimization algorithmic paradigms have also been developed a relatively long time ago (see, e.g., [8,9]). The authors believe that one of the causes for the slow development of practical parallel optimization systems is the difficulty of implementing even a conceptually simple and inherently parallel method using the parallel programming tools of today. In fact, it is immediately apparent to the reader of most of the works listed above, that a parallel implementation was only mentioned as a possible future course of research (e.g., [7]), or that some *sequential* implementation was produced and simulations of parallel execution were performed (e.g., [21,31]). Eventually, after years of hard work new publications appear in which successful truly parallel implementations are described (e.g., [1,4,13,35]). Sometimes the parallel implementations fail to materialize at all.

While parallel optimization methods are likely to be the single most important factor in the development of a parallel financial management DSS, there are even more difficult implementation issues that have to be faced when designing an application which is considerably more complex.

In the following we will use our DSS currently under development as part of the authors' on-going research [12,27], as an example of a complex application which consists of many non-trivial, possibly highly parallel components, each working on a structured set of data. The individual structures present at consecutive stages of data processing result from the different (sometimes unrelated) mathematical models and methods adopted. Each structure is best defined and operated on using a distinct collection of symbols and representations typical for the mathematical method. The transition from one form to the other can be seen as one of the major sources of difficulty of parallel implementation.

We believe that such an overview of the field may be of interest to everyone concerned with development of parallel programming paradigms, methodology and tools.

In Section 2 the financial management optimization problem will be outlined, while in Section 3 we shall provide a brief description of the structure of the DSS.

The main part of this paper, Section 4 is devoted to the discussion of parallel solution methods for stochastic optimization problems arising in financial management, as well as in many other contexts. The conclusions from our survey are presented in Section 5.

2. The decision problem

The basic problem in financial management is to decide about the portfolio structure, i.e., how much should be invested in bonds, stocks and other equities, how much should be kept in cash and how much of loans to give and credits to take.

This basic decision problem is characterized by the following features:

- the prices of the possible financial instruments (bonds, stocks, loans, forwards, options, etc.) are known now, but uncertain in the future,
- the returns on investment are therefore also unknown,
- the portfolio may be restructured in the future, in reaction to changed market conditions.

The financial management problem turns out to be a stochastic, dynamic decision problem.

A decision problem is called *dynamic*, if a sequence of decisions $(x_1, x_2, \ldots, x_d)$ has to be made one after the other in time (see Fig. 1). The irreversibility of time causes a natural ordering of the decisions: the decision x_{n+1} has to be taken under the constraints implied by the previous decision x_n.

A decision problem is called *stochastic*, if some parameters needed for the decision are unknown now and only revealed later as an outcome of a random event. In a stochastic *recourse problem* (see Fig. 2), there are two levels of decision: an immediate decision x_1 and a decision after the uncertain parameters have been observed

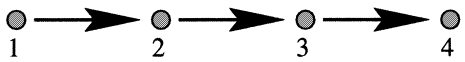

Fig. 1. Discrete time dynamic problem structure. Decisions are taken at the nodes. Arrows indicate the flow of time.

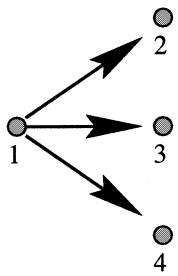

Fig. 2. Recourse problem structure.

(the recourse decision). Assume that there are k possibilities for the uncertain parameters. Then we have to introduce k recourse decisions $x_2, x_3, \ldots, x_{k+1}$. The constraints for these recourse decisions depend on the decision x_1 made immediately and on the actual (single) outcome of the random event.

The stochastic dynamic decision problem is a combination of both a stochastic and a dynamic problem. We have a discrete-time, discrete-state stochastic vector process $\mathbf{X}(t)$; $t = 1, 2, \ldots, T$ as the process of uncertainties. This vector process models all economic time series which are the source of uncertainty and risks for the financial management decision problem, like interest rates, exchange rates, stock market prices, etc.

With the process $\mathbf{X}(t)$ we associate the history process $\mathscr{X}(t) = (\mathbf{X}(1), \mathbf{X}(2), \ldots, \mathbf{X}(t))$. Since the process $\mathbf{X}$ has only finitely many states, one may arrange all states of the history process in a finite tree, called the scenario tree (see Fig. 3). Its root is the starting state $\mathscr{X}_1 = \mathbf{X}(1)$. The nodes of the tree may be numbered $n \in \mathscr{N} = \{1, \ldots, N\}$ such that each node number n corresponds in a one-to-one manner to a history of the process $\mathbf{X}(\cdot)$ up to the time at which this node occurs. We may and will consider the history as a function of the node number of the scenario tree using the notation $\mathscr{X}(n)$. The (unconditional) probability of reaching node n is denoted by p_n.

At each node of the tree a decision is taken. The depth of each node corresponds to the time for this decision. The decision is how much assets (bonds, stocks, forward contracts, futures, options, etc.) and liabilities (credits, loans, pensions, etc.) should be bought and sold. We summarize all these financial instruments under the name of *contracts*. The node number n indicates the history of the economic environment and this in turn determines the buying prices $g_b(n, j)$, the selling prices $g_s(n, j)$ and the induced cash flows $g_c(n, j)$ for each contract j.

The decision vector components are: how much of contract $j \in \mathscr{J}$ is bought $x_b(n, j)$ or sold $x_s(n, j)$ at node n of the scenario tree. Let $c(n)$ denote the cash at node n and n_- the predecessor of node n in the tree. The cash balance equation is

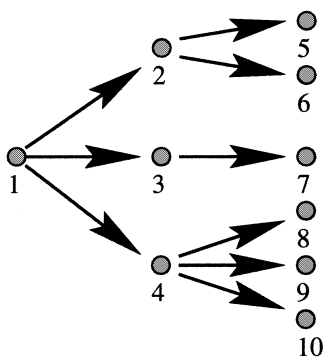

Fig. 3. A tree structured problem.

$$\forall(n \in \mathcal{N}) \quad c(n) = c(n_-) + \sum_{j \in \mathcal{J}} [x_s(n,j)g_s(n,j) - x_b(n,j)g_b(n,j) + h(n,j)g_c(n,j)],$$

where $h(n,j)$ is the amount of contract j held right after the decisions at node n. The bookkeeping equations are

$$\forall(n \in \mathcal{N},\ j \in \mathcal{J}) \quad h(n,j) = h(n_-,j) + x_b(n,j) - x_s(n,j).$$

All decision variables are non-negative, but other (for instance legal) constraints may be added. At each terminal node $t \in \mathcal{T} \subset \mathcal{N}$ the terminal wealth random variable $W(t)$ takes with probability p_t the value

$$\forall(t \in \mathcal{T}) \quad W(t) = c(t) + \sum_{j \in \mathcal{J}} h(t,j)g_s(t,j).$$

The typical objective of the financial management problem is to maximize a risk functional of the terminal wealth. This functional contains expected wealth, but also takes into account the decision maker's risk aversion. We maximize

$$\mathrm{E}(W) - \rho \mathrm{E}|W - \mathrm{E}(W)| = \sum_{t \in \mathcal{T}} p_t W(t) - \rho \sum_{t \in \mathcal{T}} p_t \left| W(t) - \sum_{t \in \mathcal{T}} p_t W(t) \right|, \tag{1}$$

where $\rho \geqslant 0$ is a risk aversion factor to be determined by the decision maker (see [23]). The whole problem is a large scale tree structured linear program.

The terminal wealth $\mathrm{E}(W)$ is a random variable. The objective (1) is only one of many possible statistical characteristics of it. The best way to get insight into this variable is to display its cumulative distribution function and some more statistical characteristics, like lower semi-standard deviation, value at risk (5% quantile), etc. (see Fig. 4).

Please keep in mind that the presentation above is, of necessity, simplified and does not deal with many detailed practical issues. Consequently, the model just developed may only serve as a *framework*. Interested reader should definitely consult more detailed practical model descriptions, such as those described and referenced

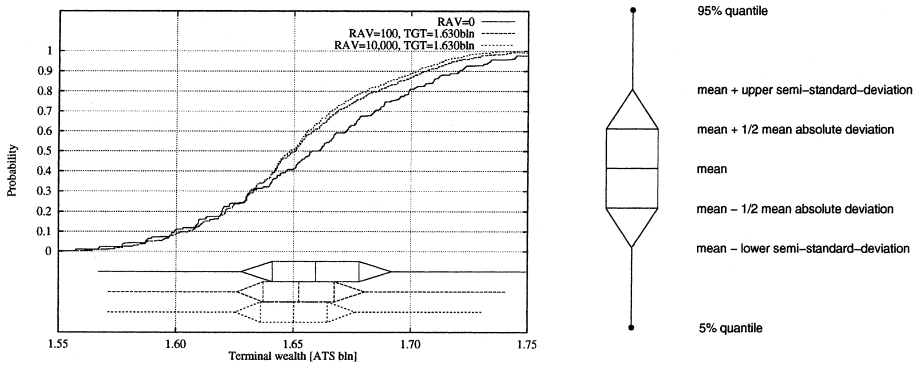

Fig. 4. Example of a comparison of cumulative distributions of wealth.

in, e.g., the recent book [26] devoted entirely to issues of asset-liability management. Most of the issues of financial modeling, which were taken for granted above (e.g., providing a coherent set of prognoses of market prices), need separate complex mathematical models.

3. Financial management DSS structure

The solution of the optimization problem is the core of the DSS. Other modules of the DSS are responsible for data handling, contract pricing and model generation. The typical integrated decision process contains the following computer supported (and often computation intensive) steps:

- The relevant risks (interest rates, exchange rates, stock market prices) are identified and historical time series are input.
- A number of risk factors are extracted by principal component analysis.
- The risk factors are modeled by a discrete-time, discrete-state Markov chain.
- The original risks are represented as functions of state of the Markov chain.
- Expert opinion about long-term trends may be added.
- After the planning horizon has been determined, the scenario tree is generated (see [28] for description of a recently developed optimal scenario tree generation procedure).
- The prices and cash-flows of the different contracts at all nodes of the scenario tree are calculated. This pricing is often based on an extensive simulation, and can be done independently, possibly in parallel, for each node of the tree (see [11] for a description of a recent parallel implementation study).
- The optimization problem is solved and the first stage decisions (the decisions to be made immediately) together with a graphical representation of the terminal wealth random variable are shown to the decision maker.
- The decision maker may now decide to change some of the model parameters and repeat relevant parts of the process.

To give the reader the idea of problem sizes, let us assume that we have five independent risk factors, each described by a binary lattice. Then at each node of the tree there are 2^5 branches. A T-period tree will thus have $(2^5)^{T-1}$ nodes. This explosive growth of the tree is the major source of difficulty. Let us add that the number of risk factors assumed above is rather moderate and the branching factor of 2 per risk factor is clearly the smallest possible. The number of periods T considered would typically be 5 or more.

We stress that most if not all of those tasks within a DSS which do not involve user interaction may have to be performed in parallel (e.g., when memory resources do not allow the computation to take place in just one computational node). Each of those tasks belongs to a different world: it works with different data structures (e.g., dense and sparse matrices vs. graphs), uses different terminology (statistical, econometric, mathematical programming, etc.) and different computational methods. That implies that a complex system like the one outlined above cannot be well

supported by just one parallel programming paradigm, as, at least up until now, no such paradigm is versatile enough to cater for all the needs.

In the following we shall present in more detail a number of optimization methods used in stochastic optimization. The variety of parallel processing structures available in just this single part of the DSS is representative of most of the problem structures present in the whole system.

4. Selected parallel optimization methods

A general form of a linearly constrained optimization problem is

$$\begin{aligned} &\min f(x), \\ &Ax = b, \\ &x \in X, \end{aligned} \tag{2}$$

where $f(\cdot)$ may be linear, quadratic or generally convex, making (2) a linear, quadratic or convex program. When it comes to large-scale optimization problems (i.e., ones with 10^6, 10^8 or more constraints or variables) it is more than likely that behind a general formulation (2) a highly structured matrix A, vectors b, c and set X are hidden. Understanding and careful exploitation of this structure may make a difference between being able to solve a problem in a reasonable amount of time and being unable to even store it in the computer's memory. The presence of structure together with the problem size immediately suggests using parallel processing in solution procedures.

In this section, we shall attempt to provide a simple, unified description of the principles behind some of the representative decomposition methods for multistage stochastic problems. We shall only focus on issues important for parallel processing. We are concerned with *how* the methods work and progress through the iterations and not *why* they work. Thus convergence proofs, starting point calculation and many other mathematical issues will be omitted. Only the most fundamental mathematical ideas are going to be presented. We shall mainly talk about problem structures and reformulations, data flow, synchronization and communication.

Historical development of parallel optimization algorithms followed a few different routes:

1. Parallelism was found and exploited in some of the existing (sequential) optimization methods.
   - 1.1. Typically it was discovered in the algebraic operations inside the algorithms which led to either purely data-parallel linear algebra methods or a mix of data parallelism and involved parallel algorithms for, e.g., matrix factorization (see, e.g., [7,15]). The methods were either assuming a special structure or were trying to identify one in an unstructured problem.
   - 1.2. Other attempts include such reformulations of existing sequential algorithms, which allow exploiting more parallelism (see, e.g., [16]).

2. New inherently parallel algorithms were designed.
   2.1. Some are based on a rather coarse grain parallel structure closely related to the problem domain. Node decomposition on a tree [5,31] or scenario decomposition [24,33] may serve as examples.
   2.2. Others exploit only the fine grain structure of the constraint matrix of problem (2) and are thus well suited to solve both structured and unstructured problems [20,32].

It is possible that the run time of some of these algorithms is longer on sequential hardware than that of a classical sequential algorithm, but improved scalability makes them more efficient on truly parallel machines or distributed platforms. On the other hand, there exist specialized decomposition algorithms which are the most efficient both on sequential and parallel computers (see, e.g., [29]).

In the following, we shall be concerned with the description of some of the known methods belonging in the last two groups.

4.1. The principle of decomposition

4.1.1. A tree-structured stochastic optimization problem

The stochastic dynamic problem is an example of a structured optimization problem. Recall its structure from Fig. 3.

Define a rooted tree $\mathbf{T} = (\mathcal{N}, \mathcal{A})$ where $\mathcal{N} = \{1, \ldots, N\}$ is a set of nodes and $\mathcal{A}$ a set of arcs. Denote the root of the tree with $r \in \mathcal{N}$, depth of the tree (identical with the number of time stages) with T, predecessor of node i with i_- and the set of terminal nodes with $\mathcal{T} \subset \mathcal{N}$. Let i_{-k} denote the kth predecessor of i and define a set of siblings of node $n \in \mathcal{N}$ as $S(n) = \{k : k_- = n\}$. Further assume that all terminal nodes are at level T.

It is typical in the field of mathematical programming to formulate all problems as minimization rather than maximization. Hereafter we shall adhere to this custom. Clearly, only a change in sign of the objective is needed to switch from one form to the other. The linearly constrained stochastic dynamic optimization problem can be expressed most directly using the tree structure introduced above:

$$\begin{aligned} &\min \sum_{n\in\mathcal{N}} f_n(x_n), \\ \forall(n \in \mathcal{N}) \quad &\begin{cases} T_n x_{p(n)} + A_n x_n = b_n, \\ x_n \in X_n, \end{cases} \end{aligned} \tag{3}$$

where $\forall(n \in \mathcal{N})\ A_n \in \mathbb{R}^{m_n \times n_n}$, $T_n \in \mathbb{R}^{m_n \times n_{n-}}$. To avoid treating the root node as a special case, we define $\mathbb{R} \supset X_{r-} = \{0\}$, $x_{r-} = 0$ and $T_r \in \mathbb{R}^{m_r \times 1}$, $T_r = [0]$.

If functions f_n are linear, i.e., $f_n(x_n) = c_n^T x_n$, and the decomposable constraints $x_n \in X_n$ have a simple form $x_n \geqslant 0$, then problem (3) becomes a large-scale linear problem with a dual

$$\begin{aligned} &\max \sum_{n\in\mathcal{N}} b_n^T y_n, \\ \forall(n \in \mathcal{N}) \quad &\begin{cases} A_n^T y_n + \sum\limits_{k\in S(n)} \left[T_k^T y_k\right] + z_n = c_n, \\ z_n \geqslant 0. \end{cases} \end{aligned} \tag{4}$$

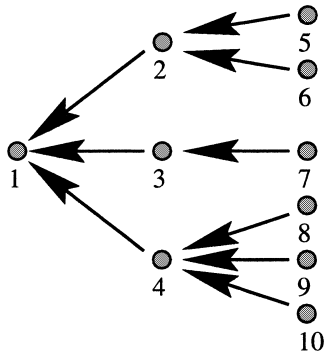

Fig. 5. Information flow in a dual problem.

The dual problem (4) is also a tree structured problem but the primal information flow (i.e., the flow of decisions) is reversed, as illustrated in Fig. 5.

It is well known that a linear problem may be solved either in its primal or dual form. One of the decomposition methods described below, namely the Dantzig–Wolfe decomposition, was actually designed to handle directly the dual problem.

A working example of a scenario tree to be used in the remaining part of this section was shown in Fig. 3. It represents a three stage planning problem with the number of second stage random events depending on the outcome of the first stage random event.

4.1.2. How decomposition works

If the matrices T_n were all zero, the problem (3) would trivially decompose into N completely independent problems, which could be solved in parallel

$$\forall(n \in \mathcal{N}) \quad \begin{cases} \min f_n(x_n), \\ A_n x_n = b_n, \\ x_n \in X_n. \end{cases} \tag{5}$$

The main idea in some methods for parallel optimization is to coordinate the inner optimization problems (5) in an iterative manner. To this end, a coordination function $Q_n^{(v)}(x_n)$ is introduced and updated in an outer loop:

Algorithm 1. General decomposition algorithm

Step 0. $v := 1$

Loop begin

Step 1. $\forall(n \in \mathcal{N})$ update the coordination functions $Q_n^{(v)}(x_n)$.
Step 2. In parallel $\forall(n \in \mathcal{N})$ solve $\min_{\{x_n \in X_n, A_n x_n = b_n\}} f_n(x_n) + Q_n^{(v)}(x_n)$.
Step 3. Stop if termination criteria met.
Step 4. $v := v + 1$

Loop end

The coordination function $Q_n^{(v)}(x_n)$ must in some way approximate the function

$$x_n \to \min_{x_j, j\neq n} \left\{ \sum_{\mathcal{N}\ni j\neq n} f_j(x_j) \text{ s.t. } \forall (j\in\mathcal{N}) A_j x_j + T_j x_{j_-} = b_j, x_j \in X_j \right\}. \tag{6}$$

In other words, the coordination function should express the dependence of non-local part of the global objective $\sum_{n\in\mathcal{N}} f_n(x_n)$ on the value of the local variable x_n.

The update of the local coordination function may be either centralized or decentralized. The local optimizations may be performed with more or less synchronization. The original problem may be restructured according to its different interpretations, and thus have more than one parallel formulation treatable by more than one method. Problem reformulations include also the choice of the right granularity of the subproblems, where by the right granularity we mean such that will result in highest possible solution efficiency. On the one end of the spectrum of possible granularities, the subproblems may be further decomposed. On the other end, groups of subproblems may be amalgamated into larger, higher level form of subproblems. Finally, it is typical (especially in the master–servant algorithms) that the *types* of $Q_n^{(v)}(x_n)$ functions will be different at different levels, or more generally, nodes of the tree, e.g., the master might try to build a global representation of (6) while the slaves would only hold a local description, valid around the current iterate.

All these issues will be discussed in the presentations of the particular methods below.

4.2. The selected methods

4.2.1. The Dantzig–Wolfe and Benders decomposition methods

A number of methods, including some of the oldest and best known, were based on the famous decomposition principle of Dantzig and Wolfe [10,14]. Although the so-called Benders decomposition [2] is typically seen as dual to Dantzig–Wolfe approach, we shall try to express them both using the same terminology.

Let us first recall the notion of a subgradient of convex function. The subdifferential (a set of subgradients) of convex function f at $x^{(0)}$ is defined as

$$\partial f(x^{(0)}) = \{a : \forall x\, f(x) \geqslant f(x^{(0)}) + \langle a, x - x^{(0)}\rangle\}.$$

A list of arguments $(x^{(i)})_{\{i=1,\ldots,v\}}$ pertaining function values $f(x^{(i)})_{\{i=1,\ldots,v\}}$ and subgradients $(a^{(i)})_{\{i=1,\ldots,v\}}$, $a^{(i)} \in \partial f(x^{(i)})$ is called *dual information* on f. If we have dual information on f we may bound f from below by the *subgradient approximation*

$$f(x) \geqslant \max_{i=1,\ldots,v} \{f(x^{(i)}) + \langle a^{(i)}, x - x^{(i)}\rangle\}.$$

If we know only the arguments $x^{(i)}$ and pertaining function values $f(x^{(i)})$, we call this *primal information* on f. Having primal information we may bound f from above by the *primal approximation*

$$f(x) \leqslant \sum_{i=1}^{v} \lambda_i f(x^{(i)}) \text{ if } x = \sum_{i=1}^{v} \lambda_i x^{(i)}, \sum_{i=1}^{v} \lambda_i = 1 \text{ and } \forall i\ \lambda_i \geqslant 0.$$

For illustration we consider the following simple problem, where x_1 denotes the first stage and x_2 the second stage decisions:

$$\begin{aligned} &\min c_1^T x_1 + c_2^T x_2, \\ &A_1 x_1 + A_2 x_2 = b, \\ &x_1 \in X_1, x_2 \in X_2. \end{aligned}$$

The first stage subproblem is concerned with finding the optimal x_1 and the second stage subproblem is concerned with finding the optimal x_2. Benders decomposition and Dantzig–Wolfe decomposition differ in the way how the problem is divided between stages and how the relevant information is passed between the first and the second stage.

- In Benders decomposition, primal information (namely the proposed value of x_1) is passed from the first stage to the second stage. In turn, the second stage problem finds the best x_2 given x_1 is fixed and passes dual information back to the first stage. The first stage problem constructs a subgradient approximation of the second stage function, which is minimized and the solution passed again to the second stage problem. For illustration see Fig. 6.

Algorithm 2. Benders decomposition

Step 1. Solve the *master problem* (*first stage*): $x_1^{(v)} \in \text{Arg}\min_{x_1 \in X_1} c_1^T x_1 + Q_1^{(v)}(x_1)$.

Step 2. Solve the *subproblem* (*second stage*): $f_2(x_1^{(v)}) := \min c_2^T x_2$ subject to constraints $x_2 \in X_2, A_2 x_2 = b - A_1 x_1^{(v)}$.

Step 3. From the solution to the second stage problem obtain a subgradient $y_2^{(v)}$ and pass it back to the first stage.

Step 4. Use $f_2(x_1^{(v)})$, $y_2^{(v)}$ and $Q_1^{(v)}(x_1)$ to calculate the refined approximation $Q_1^{(v+1)}(x_1)$.

Step 5. Set $v := v + 1$. If not optimal, go to *Step* 1.

- In the Dantzig–Wolfe decomposition, dual information, namely a subgradient is passed from the first stage to the second stage problem. In turn, the second stage problem passes primal information back to the first stage. The first stage problem

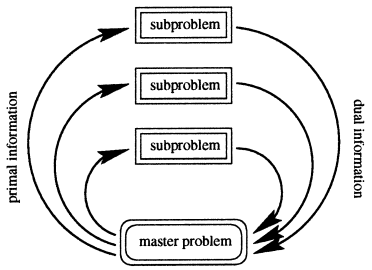

Fig. 6. Information flow in Benders decomposition scheme. Primal information passes in the direction indicated by arrows. Dual information is passed in the opposite direction.

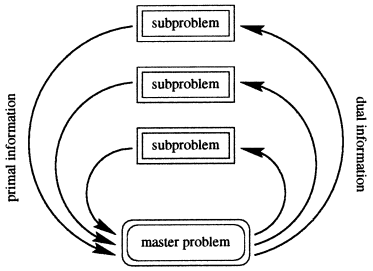

Fig. 7. Information flow in Dantzig–Wolfe decomposition scheme. Primal information passes in the direction indicated by arrows. Dual information is passed in the opposite direction.

builds an approximation of the second stage objective, which is minimized together with first stage objective and then dual information is passed again to the second stage. See also Fig. 7.

Algorithm 3. Dantzig–Wolfe decomposition

Step 1. Solve the *master problem* (*first stage*): $\min c_1^T x_1 + Q_1^{(v)}(x_2)$ subject to constraints $x_1 \in X_1$, $A_1 x_1 + A_2 x_2 = b$, $x_2 = \sum_{j=1}^{v} \lambda_j x_2^{(j)}$, $\sum_{j=1}^{v} \lambda_j = 1$ and $\forall (j = 1, \ldots, v) \lambda_j \geqslant 0$.

Step 2. From the solution obtain the dual variables $y_1^{(v)}$ and pass them forward to the second stage.

Step 3. Solve the *subproblem* (*second stage*): $x_2^{(v)} \in \operatorname{Arg\,min}_{x_2 \in X_2}(c_2 - A_2^T y_1^{(v)})^T x_2$.

Step 4. Pass the $x_2^{(v)}$ and $f_2(x_2^{(v)})$ back to the first stage.

Step 5. In the first stage refine approximation $Q_1^{(v+1)}(x_2) := \sum_{j=1}^{v} \lambda_j f_2(x_2^{(j)}) x_2^{(j)}$.

Step 6. Set $v := v + 1$. If not optimal, go to *Step* 1.

Both algorithms above have been first introduced as two level methods, i.e., with just one master problem and a number of slaves (subproblems). A stochastic recourse problem (see Fig. 2) has several second stage problems, i.e. several subproblems. These subproblems may be solved in parallel.

In a tree-structured problem, all interior nodes (i.e., other than root and leaves) are both masters and subproblems. The pertaining control structure is called *nested*. Every subproblem is at the same time a master problem for the next level. Certain amount of parallelism is present in the processing performed in all non-terminal nodes. In a synchronous implementation, the information flows first in one direction along the tree, then in the other, possibly across all levels. During an iteration one such wave rolls forth and then back. The process repeats until stopping criteria are met.

Both the method of Benders and of Dantzig–Wolfe can be implemented in a nested fashion (see, e.g., [5] and Fig. 8 for illustration). Then they can handle directly a tree structured problem (3) .

From the parallel computation perspective both methods can be characterized in the same way:

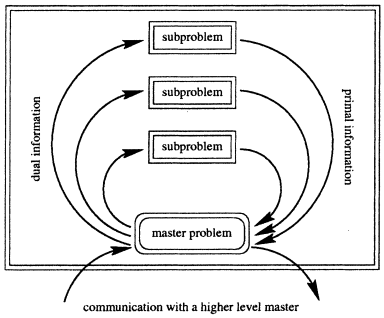

Fig. 8. Nested decomposition: a master (with all its subproblems) at one tree level is seen as a single subproblem at a higher level.

- *Node decomposition*: Methods work using directly the tree structure of the problem, computing at the nodes and communicating along the arcs.
- *Coarse grain parallel*: The subproblems solved in parallel in both methods require significant computational effort; the communicated solution results take the form of single vectors $x_n^{(v)}$ and $y_n^{(v)}$.
- *Synchronized by the master*: First all slaves have to solve their problems before the master starts gathering the data, then the master has to finish the solution before it passes the result to slaves, which have been waiting idle.
- *Asymmetric coordination*: The master maintains a non-smooth representation of the epigraph of the sum of all subproblems' objective functions, thus it holds an approximation of the global objective function. In contrast, subproblems are only given local representation of master's objective. In the Dantzig–Wolfe method subproblems have an affine representation of the master: a price on the linking constraints. In Benders method, subproblems are given the values of the linking variables and produce the price on their perturbation.

4.2.2. Asynchronous nested regularized decomposition

Nested regularized decomposition [31] develops ideas of nested Benders decomposition [5] and two stage regularized decomposition [30,29]. Unlike its predecessors, it allows asynchronous parallel execution of both master and slave problems at all nodes of the tree, thus greatly diminishing the scalability concerns caused by the existence of a serial bottleneck – synchronous master.

A precise discussion of this method should involve consideration of so-called regularization at non-terminal nodes, iterative updates of the regularizing parameter as well as many other complications. We believe those issues are not crucial to understanding of the asynchronous coordination mechanism. Since the regularization changes the objective, and consequently the coordination functions $Q_n^{(v)}$, the derivation of those will be omitted.

Asynchronous coordination is made possible by introducing communication buffers between processing nodes. As seen in Fig. 9 there is one output buffer associated with each non-terminal node. The buffer stores the latest solution (primal

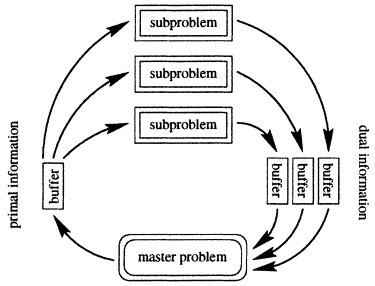

Fig. 9. Nested regularized decomposition information flow (data buffers shown).

information) of this node's optimization problem, or is empty. It can be written to only by its owner node and read by each of its siblings no more than once after it has been filled with a value. Each problem other than the root stores the dual information in an output buffer. The output buffer has the capacity to hold at least one unit of dual information.

Each node problem operates using rather simple principles:

- each, except the root, begins its existence in an idle state,
- when not idle, it retrieves all the values which are stored in its incoming buffers, forms $Q_n^{(v)}$ and solves the resulting optimization problem,
  - when problem is infeasible, it stores the dual information in the buffer connecting it to the lower level of the tree,
  - when problem was solved to optimality and the solution is different from the previous one obtained at this node, it stores the dual information in the buffer connecting it to the lower level of the tree and the primal information in the buffer connecting it to the higher level (if any),
  - then it retires to the idle state,
- idle problem waits until at least one of its inputs contains new information, then it switches to the active state.

When all problems are idle, the status of the last solution of the root problem (optimal or infeasible) is the status of the whole problem and the optimization ends.

The characteristics relevant to parallel processing remain largely the same as in the case of nested Benders and Dantzig–Wolfe methods. One notable exception is the improved scalability resulting from asynchronous coordination.

4.2.3. Augmented Lagrangian decomposition principles

The augmented Lagrangian decomposition methods discussed in this and the two following sections are based on the following principles:

- the idea of relaxing inconvenient constraints (in our case – those linking decomposable subproblems) and instead introducing a form of penalty for their violation,
- Lagrangian augmentation which enables use of a simple iterative method, the so-called multiplier algorithm, for coordination by means of adjustments of the penalties (see [3]),

- use of a decomposable approximation of the augmented Lagrangian.

We shall demonstrate those ideas on a simplified example again

$$\begin{aligned} &\min f_1(x_1) + f_2(x_2), \\ &A_1x_1 + A_2x_2 = b, \\ &x_1 \in X_1, x_2 \in X_2. \end{aligned} \tag{7}$$

Problem (7) would be decomposable, were it not for the linking constraint $A_1x_1 + A_2x_2 = b$. The augmented Lagrangian function for (7) with the linking constraint relaxed is

$$\Lambda(x,y) = f_1(x_1) + f_2(x_2) + \langle y, b - A_1x_1 - A_2x_2\rangle + \frac{\rho}{2}\|b - A_1x_1 - A_2x_2\|^2,$$

where $\rho > 0$, $x^T = [x_1^T\, x_2^T]$. Define $X = X_1 \times X_2$. Now the optimization problem (7) may be solved by the iterative procedure known as the multiplier algorithm [3]:

Algorithm 4. The multiplier algorithm

Step 0. Set $v = 1$. Choose arbitrary initial value for $y^{(v)}$, required accuracy $\varepsilon \geqslant 0$ and fixed penalty $\rho > 0$.

Step 1. For fixed $y^{(v)}$ solve $x^{(v)} \in \operatorname{Arg\,min}_{x\in X} \Lambda(x, y^{(v)})$.

Step 2. If $\left\|Ax^{(v)} - b\right\| < \varepsilon$ then stop (optimal solution found).

Step 3. Update the multiplier vector: $y^{(v+1)} := y^{(v)} + \rho(b - Ax^{(v)})$.

Step 4. Set $v := v + 1$, go to *Step* 1.

For further reference note, that the update of vector $y^{(v)}$ in *Step* 3 provides the coordination.

For simplicity, the index (v) will be omitted below. The main computational effort in Algorithm 4 is to optimize the augmented Lagrangian function in *Step* 1. However, this function is itself only *partly* decomposable with respect to x_i:

$$\Lambda(x,y) = \langle y + \frac{\rho}{2}b, b\rangle + \sum_{i=1,2}\left[f_i(x_i) - \langle y + \rho b, A_ix_i\rangle + \frac{\rho}{2}\|A_ix_i\|^2\right] + \rho\langle A_1x_1, A_2x_2\rangle.$$

Decomposition is obtained by replacing $\Lambda(x,y)$ with its approximation

$$\tilde{\Lambda}(x,\tilde{x},y) = \langle y + \frac{\rho}{2}b, b\rangle + \sum_{i=1,2}\tilde{\Lambda}_i(x_i,\tilde{x},y), \tag{8}$$

where $\tilde{\Lambda}_i(x_i,\tilde{x},y)$ is defined as

$$\tilde{\Lambda}_i(x_i,\tilde{x},y) = f_i(x_i) - \langle y + \rho(b - A_j\tilde{x}_j), A_ix_i\rangle + \tfrac{\rho}{2}\|A_ix_i\|^2, \quad j \neq i \tag{9}$$

and $\tilde{x}^T = [\tilde{x}_1^T\ \tilde{x}_2^T]$ is an additional parameter.

Minimization in *Step* 1 of Algorithm 4 above is done in an inner loop where the separate subproblems (9) are solved in parallel by means of a so-called Diagonal Quadratic Approximation method (DQA).

Algorithm 5. The DQA method.

Step 0. Choose step length factor $0 < \tau < 1$, set $\tilde{x} := x$.

Step 1. For $i = 1, 2$ solve in parallel $x_i \in \operatorname{Arg\,min}_{x_i \in X_i} \tilde{\Lambda}_i(x_i, \tilde{x}, y)$.
Step 2. If $\|A_i(x_i - \tilde{x}_i)\| < \varepsilon$ for $i = 1, 2$ then stop (optimal solution found).
Step 3. For $i = 1, 2$ set $\tilde{x}_i := (1 - \tau)\tilde{x}_i + \tau x_i$.
Step 4. Go to *Step* 1.

In [32] conditions on τ are given, under which convergence of the algorithm above is guaranteed.

The coordination function updated in each inner iteration (in *Step* 3 of both Algorithms 4 and 5 is now $Q^{(\nu)}(x^{(\nu)}) = -\langle y + \rho(b - A_j\tilde{x}_j), A_i x_i\rangle + \frac{\rho}{2}\|A_i x_i\|^2$.

As it is clearly seen, the computational characteristics of this family of methods are quite different from those of nested decompositions of Sections 4.1.1 and 4.2.2:

- *Coarse grain parallelism of subproblem solution*: The *Step* 1 of Algorithm 5 is fully distributed. Minimization of each $\tilde{\Lambda}_i(x_i, \tilde{x}, y)$ is assumed to be a time consuming task.
- *Fine grain parallelism of coordination and termination*: The coordination steps in both nested loops are just elementwise vector operations. Additionally, since in real life problems each block x_i will be connected by linking constraints with only a few other blocks, the coordination may exploit the resulting structure to save most of the communication.
- *Fully synchronized algorithm*: the order of execution of steps of both Algorithms 4 and 5 above is entirely deterministic.

4.2.4. Node-oriented augmented Lagrangian decomposition

Since we already stated three methods which see tree nodes as separate units, we shall now show the augmented Lagrangian method operating on the same principles.

The tree structured problem (3) is seen here as a collection of node problems with local constraints $x_n \in X_n$ linked by the transfer of decisions x_n along the branches

$$\begin{aligned} &\min \sum_{n \in \mathcal{N}} f_n(x_n), \\ \forall (n \in \mathcal{N}) \quad &\begin{cases} T_n x_{n_-} + A_n x_n = b_n, \\ x_n \in X_n. \end{cases} \end{aligned} \tag{10}$$

The constraints $T_n x_{n_-} + A_n x_n = b_n$ define the dynamics of the decision process and link the subproblems. They are relaxed by placing them in the augmented Lagrangian. Let y_n denote the Lagrange multipliers related to those linking constraints. The augmented Lagrangian is defined as

$$\Lambda(x, y) = \sum_{n \in \mathcal{N}} \left[f_n(x_n) + \langle y_n, b_n - T_n x_{n_-} + A_n x_n \rangle + \frac{\rho}{2}\|b_n - T_n x_{n_-} + A_n x_n\|^2 \right].$$

It is not decomposable due to non-locality of predecessor mapping: the references to variables x_{n_-} at node n will require communication. Clearly, substituting $\tilde{x}_{n_-}$ in place of x_{n_-} will render the augmented Lagrangian above decomposable, as it was shown in (8) and (9). Distributed updates of Lagrange multipliers y_n and parameters $\tilde{x}_{n_-}$ are straightforward (see Fig. 10).

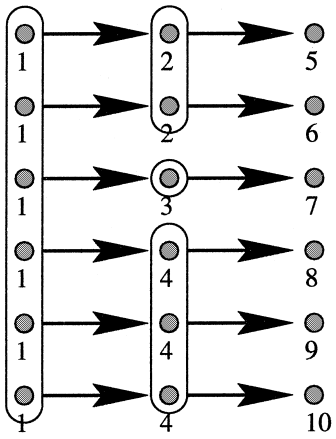

Fig. 10. Scenario tree example from Fig. 3 expanded into separate scenarios linked by non-anticipativity constraints. Illustrates problem formulation (11).

Parallel solution of the overall problem remains a mix of coarse grain parallelism and elementwise vector operations in coordination functions' updates.

4.2.5. Scenario oriented augmented Lagrangian decomposition

The stochastic nature of the modeled phenomena provides yet another way of seeing the optimization problem (3). Each time-ordered sequence of random events (corresponding to a unique path from the root of the tree to a terminal node) is called a scenario. Thus the stochastic problem can be defined as optimization of decisions for separate scenarios $s \in \mathscr{T}$ with an additional constraint stating that a decision at any node n may only depend on the random events that took place on a path from the root to that node, inclusive. This constraint is commonly referred to as non-anticipativity condition, as it prevents the decisions from depending on the outcomes of future random events.

The notation is somewhat more involved than in the previous cases but the underlying concept is simple. Let us first add some new symbols: for $s \in \mathscr{T}$ define $n(s,i) = s_{-(T-i)}$, which is the number of the node of the original tree (3) appearing in scenario s at period i. Scenario s includes nodes $H(s) = \{n(s,1), n(s,2), \ldots, n(s,T)\}$ with $n(s,1) \equiv r$ and $n(s,T) = s$ trivially holding. The problem is now stated as

$$\begin{aligned} &\min \sum_{s\in\mathscr{T}} f_s(x_s), \\ \forall(s \in \mathscr{T}) \quad &\begin{cases} A_s x_s = b_s, \\ x_s \in X_s, \\ \forall(t \in \mathscr{T}, t \neq s)\forall\{i : n(s,i) = n(t,i)\} \quad x_{s,i} = x_{t,i}, \end{cases} \end{aligned} \tag{11}$$

where the matrices and vectors above are defined as

$$x_s^T = \left[x_{s,1}^T \cdots x_{s,T}^T\right], \quad X_s = \prod_{i=1}^{T} X_{n(s,i)}, \quad f_s(x) = \sum_{i=1}^{T} f_{n(s,i)}(x_{s,i}),$$

$$b_s = \begin{bmatrix} b_{n(s,1)} \\ \vdots \\ b_{n(s,T)} \end{bmatrix}, \quad A_s = \begin{bmatrix} A_{n(s,1)} & & & \\ T_{n(s,2)} & A_{n(s,2)} & & \\ & \ddots & \ddots & \\ & & T_{n(s,T)} & A_{n(s,T)} \end{bmatrix}.$$

The linking non-anticipativity constraints are relaxed and placed in the augmented Lagrangian

$$\Lambda(x,y) = \sum_{s\in\mathscr{T}} \left[f_s(x_s) + \sum_{\mathscr{T}\ni t\neq s} \sum_{\{i:n(s,i)=n(t,i)\}} \left(\langle y_{s,t,i}, x_{t,i} - x_{s,i}\rangle + \frac{\rho}{2} \|x_{t,i} - x_{s,i}\|^2 \right) \right].$$

Treating the scenario s as the unit of decomposition, we see that references to variables $x_{t,i}$ are non-local and should be replaced by references to $\tilde{x}_{t,i}$. Further details follow the patterns outlined previously.

For parallel implementation of the method we should note that:

- there is a large degree of redundancy, as node data for all non-terminal nodes with more than one successor (direct or not) is replicated,
- the number of subproblems is smaller than in the case of node decomposition but the difference depends on the growth rate of the number of nodes at consecutive stages, e.g., in case of a binary tree the number of scenarios 2^{T-1} is nearly a half of the number of all nodes $N = \sum_{i=1}^{T} 2^{i-1} = 2^T - 1$,
- the subproblems are much larger then in node decomposition schemes because each of them describes all T stages of a decision process,
- unlike in the case of Lagrangian node decomposition, the linking non-anticipativity constraints are very simple, therefore the effort of calculation of their violation is much smaller,
- due to duplication of decisions, more communication is necessary to compare all pairs $(x_{s,i}, x_{t,i})$, to update $\tilde{x}_{t,i}$ and to pass and update all vectors $y_{s,t,i}$ than it was in any case of node decomposition.

4.2.6. Unstructured data parallel methods

As mentioned before, the problem (3) can be stated as a general large-scale optimization problem, which in the context of stochastic programming is called a deterministic equivalent [36]. It then can be solved (at least in principle) by general purpose optimization methods. The deterministic equivalent would be

$$\begin{aligned} &\min f(x), \\ &Ax = b, \\ &x \in X, \end{aligned} \tag{12}$$

where x, X, $f(x)$, A and b are defined as follows:

$$x^T = \left[x_1^T \cdots x_n^T\right], \quad X = \prod_{n\in\mathcal{N}} X_n, \quad f(x) = \sum_{n\in\mathcal{N}} f_n(x_n),$$

$$b = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}, \quad A = \begin{bmatrix} A_1 & & & \\ ? & A_2 & & \\ \vdots & \ddots & \ddots & \\ ? & \cdots & ? & A_n \end{bmatrix}.$$

The "?" characters in matrix A definition denote the fact that the exact location of matrices T_n in A depends on the structure of the tree (3).

In the special case when functions $f_n(\cdot)$ are linear we can use one of the new data parallel methods for linear programming introduced in [20,32] and related works. All those methods share some common characteristics:

- they are based on the use of augmented Lagrangians,
- they are iterative with a varying number of loop levels,
- easily parallelizable elementwise vector–vector, sparse matrix–vector products and some reduction operators (like vector norm or dot product) are the only mathematical operations.

Thus they appear perfect candidates for fine grain data parallel implementation. The simplest of those algorithms is based on the same principles as the other augmented Lagrangian decomposition methods described here. The other methods are constructed from the same building blocks. For all details of those methods, the reader is referred to the publications mentioned above and further references therein.

In the linear case we define $f(x) = \sum_{n\in\mathcal{N}} f_n(x_n) = \sum_{n\in\mathcal{N}} c_n^T x_n = c^T x$. We also assume that X is the non-negative orthant. In the remainder of this section the unit of decomposition denoted with x_i will be a single variable and not the block of variables corresponding to the decision at some node of the tree. Also, A_i will denote the ith column of matrix A, etc. Keeping in mind this new notation we have the augmented Lagrangian

$$\Lambda(x,y) = c^T x + \langle y, b - Ax\rangle + \frac{\rho}{2}\left\| b - \sum_{i=1}^{L} A_i x_i \right\|^2,$$

which may be approximately decomposed treating each variable x_i as an independent decision

$$\Lambda_i(x_i,\tilde{x},y) = c_i x_i - \langle y, A_i x_i\rangle + \frac{\rho}{2}\left\| b - A_i x_i - \sum_{j\neq i} A_j \tilde{x}_j \right\|^2, \tag{13}$$

where $\tilde{x}$ is fixed. The problem of minimization of $\Lambda_i(x_i,\tilde{x},y)$ subject to $x_i \geqslant 0$ has a closed form solution. With additional definitions of $z = c - A^T y$, $r = b - Ax$ and $D_A = \operatorname{diag}(\|A_1\|^2, \ldots, \|A_n\|^2)$ we may write down in matrix notation a solution of $\min_{x_i\geqslant 0} \Lambda_i(x_i,\tilde{x},y)$ for all i as

$$\hat{x} = \left[D_A^{-1}\left(A^T r - \frac{1}{\rho} z\right)\right]_+$$

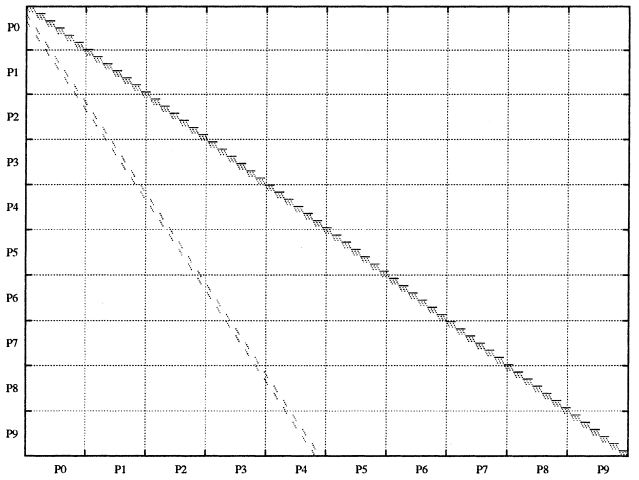

Fig. 11. Sparsity structure of a constraint matrix of a small stochastic dynamic linear program. The grid illustrates the distribution of the matrix data among 10 processors with ith processor storing locally both ith block of rows and ith block of columns.

where $[\cdot]_+$ denotes projection on the non-negative orthant. The updating of $\tilde{x}$ and y is performed in the same way as described before.

Sparse matrix-dense vector products (both of the form Ax and $A^T y$) are the most important operations in the whole algorithm. In case of a general matrix structure (i.e., a random sparsity structure) both those operations would require a large volume of communication, however, the structure present in the constraint matrix of the stochastic problem allows great savings.

Let us first proceed with an abstract description of the parallel computation. Assume that each variable x_i with corresponding matrix column A_i, cost coefficient c_i, etc., are assigned to one virtual processor. Let us denote this set of processors with $\mathscr{P}$. Another processor set $\mathscr{D}$ would hold the sets of values of y_j, b_j, and matrix row $(A^T)_j$.[1]

Elementwise vector operations are then performed entirely in parallel. Interprocessor communication only takes place in matrix–vector products. Then information travels from $\mathscr{P}$ to $\mathscr{D}$ or in the opposite direction. The processor sets $\mathscr{P}$ and $\mathscr{D}$ are most likely going to be mapped on the same real processors. Moreover, the number or both rows and columns of A is likely to be orders of magnitude larger than the number of processors, so that *groups* and not individual rows/columns have to be assigned to each actual processor. See Fig. 11 for an example distribution.

With data distribution shown in Fig. 11, interprocessor communication will result only from references to the non-diagonal blocks of the matrix. Furthermore, since there is only one non-diagonal block in each row and a few consecutive blocks in

[1] This implies duplicate storage of matrix A: both by columns and by rows.

some of the columns, all communication may be performed by sending contiguous blocks of data. This is much easier to handle efficiently than it could ever be in the case of an unstructured sparse matrix.

To summarize the parallel computation characteristics of this and similar methods:

- they are fine grain parallel, with data parallelism providing the best means for algorithm specification,[2]
- they are entirely synchronous,
- communication to computation ratio may never be as favorable as in the case of coarse grain methods described before,
- even a data parallel algorithm which ignores most of the knowledge of the structure, greatly benefits by saving communication.

4.2.7. Other methods

The methods presented above are just a sample of known parallel large-scale optimization methods used in financial planning under uncertainty as well as in many other application areas. It is not possible to even mention all work that was ever done in this field. The reader interested in seeing the "larger picture" is encouraged to consult some of the stochastic programming textbooks, e.g., [6,22], parallel optimization monographs, e.g., [8,9] recent financial modeling collections, e.g., [25,37]. Again, the sources listed above may serve as another level of introduction.

5. Conclusions

The most challenging problems for high performance computing in operations research are large combinatorial problems on one side and large structured linear or convex problems on the other side. For the latter class of problems we have presented principles of organizing the decomposition into more or less coupled subproblems. The data and control flow of these algorithms exhibit new challenges for the design of high level parallel languages and compiler design. While support for data parallel algorithms for dense and regular matrix algebra is well established, there is no easy way of coding nested task parallel asynchronous algorithms, or a mix of data and task parallel processing, as it appears in stochastic dynamic financial management problems.

Future development in language and compiler design should take these and similar large-scale problems as examples of relevant problem structures and actual software engineering requirements of the OR community.

[2] Support for data parallel computations with sparse matrices is less than adequate in current specifications and implementations of data parallel languages known to the authors [19,17].

Acknowledgements

The authors are indebted to J. Merlin from the Vienna Center for Parallel Computing (VCPC), H. Zima, E. Laure and S. Benkner from the Institüt für Softwaretechnik und Parallele Systeme, Universität Wien for many fruitful discussions.

References

[1] J.R. Birge, C.J. Donohue, D.F. Holmes, O.G. Svintsitski, A parallel implementation of the nested decomposition algorithm for multistage stochastic linear programs, Mathematical Programming 75 (2) (1996) 327–352.
[2] J.F. Benders, Partitioning procedures for solving mixed-variable programming problems, Numerische Matematik 4 (1962) 238–252.
[3] D.P. Bertsekas, Constrained Optimization and Lagrange Multiplier Methods, Academic Press, New York, 1982.
[4] J.R. Birge, D.F. Holmes, Efficient solution of two-stage stochastic linear programs using interior point methods, Computational Optimization and Applications 1 (1992) 245–276.
[5] J.R. Birge, Decomposition and partitioning methods for multistage stochastic linear programs, Operations Research 33 (1985) 989–1007.
[6] J.R. Birge, F.V. Louveaux, Introduction to Stochastic Programming, Springer, Berlin, 1997.
[7] J.R. Birge, L. Qi, Computing block-angular Karmarkar projections with applications to stochastic programming, Management Science 34 (12) (1990) 1472–1479.
[8] D.P. Bertsekas, J.N. Tsitsiklis, Parallel and Distributed Computation, Prentice-Hall, Englewood Cliffs, NJ, 1989.
[9] Y. Censor, S.A. Zenios, Parallel Optimization. Theory Algorithms and Applications, Oxford University Press, Oxford, 1997.
[10] G.B. Dantzig, Linear Programming And Extensions, Princeton University Press, Princeton, NJ, 1963.
[11] E. Dockner, H. Moritsch, Pricing constant maturity floaters with embedded options using Monte Carlo simulation, Technical report AURORA TR1999–4, Vienna University, Department of Business, 1999.
[12] E. Dockner, H. Moritsch, G.Ch. Pflug, A. Świętanowski, The AURORA financial management system, Technical report AURORA TR1998–08, Vienna University, 1998.
[13] M.A.H. Dempster, R.T. Thompson, Parallelization and aggregation of nested Benders decomposition, Working paper WP 01/95, The Judge Institute of Management Studies, 1995.
[14] G.B. Dantzig, P. Wolfe, Decomposition principle for linear programs, Operations Research 8 (1960) 101–111.
[15] Y. Ermoliev, R.J.-B. Wets, (Eds.), Numerical Techniques for Stochastic Optimization, Springer, Berlin, 1988.
[16] J.A.J. Hall, K.I.M. McKinnon, Update procedures for the parallel revised simplex method, Technical report MSR 92–13, Departament of Mathematics and Statistics, University of Edinburgh, 1992.
[17] High Performance Fortran Forum, *High Performance Fortran Language Specification, Version 2.0*, 31 January, 1997.
[18] J.L. Higle, S. Sen, Stochastic decomposition: an algorithm for two-stage linear programs with recourse, Mathematics of Operations Research 16 (3) (1991) 650–669.
[19] Ch.H. Koelbel, D.B. Loveman, R.S. Schreiber, G.L. Steele Jr., M.E. Zosel, The High Performance Fortran Handbook, MIT Press, Cambridge, MA, 1994.
[20] M. Kallio, A. Ruszczyński, Parallel solution of linear programs via Nash equilibria, Working paper WP-94-15, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1994.
[21] M. Kallio, A. Ruszczyński, S. Salo, A regularized Jacobi method for large-scale linear programming, Working paper WP-93-61, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1993.

[22] P. Kall, S.W. Wallace, Stochastic Programming, Wiley, New York, 1994.
[23] H. Konno, H. Yamazaki, Mean absolute deviation portfolio optimization model and its applications to tokyo stock market, Management Science 37 (1991) 519–531.
[24] J.M. Mulvey, A. Ruszczyński, A new scenario decomposition method for large-scale stochastic optimization, Operations Research 43 (1995) 477–490.
[25] J.M. Mulvey, W.T. Ziemba, Asset and liability management systems for long-term investors: discussion of the issues, in: J.M. Mulvey, W.T. Ziemba (Eds.), Worldwide Asset Liability Management, Cambridge University Press, Cambridge, MA, 1998.
[26] J.M. Mulvey, W.T. Ziemba (Eds.), Worldwide Asset Liability Management, Cambridge University Press, Cambridge, MA, 1998.
[27] G.Ch. Pflug, A. Świętanowski, Dynamic asset allocation under uncertainty for pension fund management, Technical report AURORA TR1998-15, Vienna University, 1998, Control and Cybernetics (to appear) .
[28] G.Ch. Pflug, A. Świętanowski, Optimal scenario tree generation for multiperiod financial optimization, Technical report AURORA TR1998-22, Vienna University, 1998.
[29] A. Ruszczyński, A. Świętanowski, Accelerating the regularized decomposition method for two stage stochastic linear problems, European Journal of Operations Research 101 (2) (1997) 328–342.
[30] A. Ruszczyński, A regularized decomposition method for minimizing a sum of polyhedral functions, Mathematical Programming 35 (1986) 309–333.
[31] A. Ruszczyński, Parallel decomposition of multistage stochastic programming problems, Mathematical Programming 58 (1993) 201–228.
[32] A. Ruszczyński, On convergence of an augmented Lagrangian decomposition method for sparse convex optimization, Mathematics of Operations Research 20 (3) (1995) 634–656.
[33] R.T. Rockafellar, R.J.-B. Wets, Scenarios and policy aggregation in optimization under uncertainty, Mathematics of Operations Research 16 (1) (1991) 119–147.
[34] B. Strazicky, Some results concerning an algorithm for the discrete recourse problem, in: M.A.H. Dempster (Ed.), Stochastic Programming, Academic Press, New York, 1980, pp. 263–274.
[35] H. Vladimirou, S.A. Zenios, Scalable parallel computations for large-scale stochastic programming, Annals of Operations Research 90 (1999) 87–129.
[36] R.J.-B. Wets, Stochastic programs with fixed recourse: the equivalent deterministic program, SIAM Review 16 (1974) 309–339.
[37] S. Zenios, R.L. D'Ecclesia (Eds.), Operations Research Models in Quantitative Finance, Springer, Berlin, 1994.