Massively Parallel Asset and Liability Management

Andreas Grothey

University of Edinburgh, School of Mathematics, Edinburgh, UK, EH9 3JZ
A.Grothey@ed.ac.uk

Abstract. Multistage Stochastic Programming is a popular method to solve financial planning problems such as Asset and Liability Management (ALM). The desirability to have future scenarios match static and dynamic correlations between assets leads to problems of truly enormous sizes (often reaching tens of millions of unknowns or more). Clearly parallel processing becomes mandatory to deal with such problems.

Solution approaches for these problems include nested Decomposition and Interior Point Methods. The latter class in particular is appealing due to its flexibility with regard to model formulation and its amenability to parallelisation on massively parallel architectures. We review some of the results and challenges in this approach, demonstrate how popular risk measures can be integrated into the framework and address the issue of modelling for High Performance Computing.

1 Introduction

Asset and Liability Management (ALM) is one of the most important applications of portfolio optimization. The basic setup is that a large long-term investor (such as a pension fund or an insurance) has to decide how to invest available capital into a choice of assets (which may be stocks, bonds, real estate, etc) over several time periods. In every time period the investor faces liability payments, but may also receive cash contributions. The problem is to find an optimal investment strategy that maximised net return, while controlling the risk of defaulting on the liability payments. Parameters such as asset returns, liability payments and cash contributions are uncertain, however it is assumed that some information about their (joint) distribution is available.

The simplest model of this kind is the Markowitz Mean-Variance model in which only one period of investment is considered, and all uncertain parameters are assumed to follow a multivariate normal distribution with known expectations and covariances. There has been widespread criticism of this model, most importantly concerning its static nature (which does not take into account effects due to rebalancing or transaction costs), the implicit assumption of normal distributions and the use of variance as a risk measure.

In response a multitude of alternative risk models have been suggested, from simple linear Mean Absolute Deviation through to Stochastic Dominance Constraints. As a result research emphasis as shifted to stochastic dynamic models

M.R. Guarracino et al. (Eds.): Euro-Par 2010 Workshops, LNCS 6586, pp. 423–430, 2011.

that (in principle) allow the use of any return distribution and can be adapted to a variety of risk measures.

Realistic models need to account for long planning horizons and adequate capture of the joint distributions of all future events that can influence the return of the portfolio over the planning horizon. These requirements quickly result in astronomical problem sizes. On the other hand progress on solution algorithms (in particular Interior Point Methods) and the use of High Performance Computing techniques have made problems of unprecedented sizes tractable.

The aim of this paper is to present risk-averse ALM models, the challenges inherent in building and solving them and to show how high-performance computing can be leveraged to overcome these challenges. In the following section we discuss the features of stochastic programming models for ALM with various risk measures. Section 3 will give details of solution challenges and HPC approaches to them, while the final Section 4 will summarise some numerical results.

2 The Stochastic Programming Approach to ALM

Let $\mathcal{A}$ be the set of possible investments available over time periods $t = 1, .., T$. With each investment $j \in \mathcal{A}$ and each time period t we associate an (unknown) return $r_{t,j}$. Further there are (again uncertain) cash contributions C_t and liability payments L_t. Denote by $\xi_t = (C_t, L_t, \{r_{t,j}\}_{j\in\mathcal{A}})$ the uncertain data in each time period. The state of the portfolio is denoted by the vector $x_t^h = \{x_{t,j}^h\}_{j\in\mathcal{A}}$, where $x_{t,j}^h$ is the current position in asset j (i.e. the total capital currently invested in this asset). Decisions are x_t^b, x_t^s, the amount to buy and sell of each asset in each time period.

The task is to find optimal investment decisions $x_t = (x_t^h, x_t^b, x_t^s)$ that maximize expected surplus return at the final time stage subject to cash balance and inventory constraint. The discrete-time ALM problem is thus

$$\begin{array}{ll} \max I\!E[X_T], & X_T = (1-\gamma)\sum_{j\in\mathcal{A}} x_{T,j}^h \\ \text{s.t. } x_{t,j}^h = (1+r_{t-1,j})x_{t-1,j}^h - x_{t,j}^s + x_{t,j}^b & \text{(inventory)} \\ L_t + (1+\gamma)\sum_j x_{t,j}^b = C_t + (1-\gamma)\sum_j x_{t,j}^s, \quad t \neq 1 & \\ L_1 + (1+\gamma)\sum_j x_{1,j}^b = b_0 & \text{(cash balance)} \end{array} \tag{1}$$

where γ are the (assumed proportional) transaction costs and X_T is the value of the portfolio at the final time period when converted into cash.

The model describes a multi-stage decision process in which information ξ_t becomes available at certain discrete points in time, and decisions x_t at time t can be based only on information available at that point in time (non-anticipativity):

$$x_1 \to \xi_2 \to x_2(\xi_2) \to \xi_3 \to x_3(\xi_2, \xi_3) \to \cdots \xi_T \to x_T(\xi_1, \ldots \xi_T).$$

Note that in this setting both the data ξ_t as well as the decisions x_t are described by discrete-time stochastic process, hence problem (1) is a semi-infinite optimization problem and as such very difficult to solve.

The stochastic programming approach to ALM overcomes this problem by replacing the data process ξ_t by a discrete approximation $\tilde{\xi}_t$. That is at every stage there are finitely many different outcomes $\tilde{\xi}_{t,i}$. The resulting process $\tilde{\xi}$ can be visualised by a tree giving the evolution of future data realisation: this is known as the scenario tree. Let $\mathcal{L}_t$ denote the set of all nodes at stage t in the tree, $\mathcal{L} = \bigcup_t \mathcal{L}_t$ the whole tree and for a given node $i \in \mathcal{L}_t$ let $a(i) \in \mathcal{L}_{t-1}$ be the ancestor node. Then the discretised version of (1) reads

$$\max_x \sum_{i \in \mathcal{L}_T} y, \tag{2a}$$

$$\begin{aligned} \text{s.t.} \quad & x^h_{i,j} = (1 + r_{a(i),j}) x^h_{a(i),j} - x^s_{i,j} + x^b_{i,j}, \qquad \forall i \in \mathcal{L} \setminus \{0\}, j \in \mathcal{A} \\ & L_i + \sum_{j \in \mathcal{A}} (1+\gamma) x^b_{i,j} = C_t + \sum_{j \in \mathcal{A}} (1-\gamma) x^s_{i,j}, \forall i \in \mathcal{L} \setminus \{0\} \\ & L_0 + \sum_{j \in \mathcal{A}} (1+\gamma) x^b_{0,j} = b_0 \\ & y = (1-\gamma) \textstyle\sum_{i \in \mathcal{L}_T} \pi_i \sum_{j \in \mathcal{A}} x^h_{T,j,i}. \end{aligned} \tag{2b}$$

which is a standard (if large) linear programming problem. With appropriately chosen matrices W_i, T_i, d_i (see [6]) the Jacobian for the scenario tree given in Figure 1 has the given nested bordered block-diagonal form. In particular note the final constraint in (2b) which stretches over all scenarios and corresponds to the linking row in the constraint matrix. While it would be possible to substitute for y in the objective function, thereby eliminating this row, the presence of y as an explicit variable results in more modelling flexibility and sparser formulations (cf [6] for details).

The question of scenario generation, that is how to construct a discrete approximation $\hat{\xi}_t$ to ξ_t in some optimal sense is an active field of research. An overview can be found for example in [9]. For our purposes it suffices to say that for every node in the scenario tree, the scenarios described by the successor nodes need to capture means, variances and correlations (conditional on the current state) between the $|\mathcal{A}|$ different assets. For a realistic model description the size of the scenario tree quickly reaches astronomical sizes. For a tree with $T = 5$ stages and a branching factor of 30 at each node (barely enough to capture the correlation between say, 60 considered random variables describing the evolution of investments and liabilities), the resulting tree has 24 million scenarios.

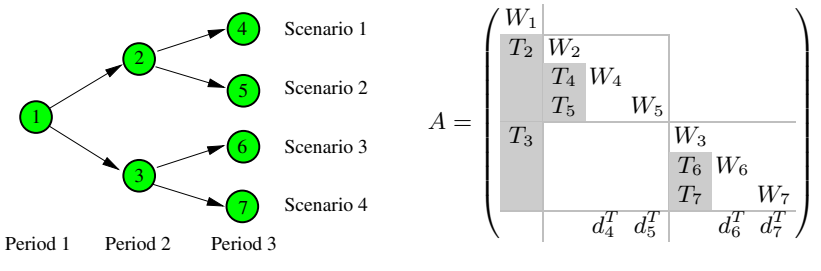

Fig. 1. Scenario tree and resulting structured constraint Jacobian

Assuming 20 different assets, model (2) would have 1.5×10^9 variables and 5.3×10^8 constraints.

2.1 Risk Averse ALM Modelling

The main deficiency of the prototype ALM model in the previous section is that it only aims to maximise expected excess return, without attempting to control risk. The standard approach to handle risk follows the suggestion of Markowitz[12] and uses the variance of the surplus return as a risk measure. In the Markowitz model the twin contradictory objective of maximizing expected return while minimizing risk are combined into a single objective

$$\max_x I\!E[X_T] - \lambda \mathrm{Var}[X_T], \tag{3}$$

where $\lambda > 0$ is a *risk-aversion parameter*. Using the identity $\mathrm{Var}[X] = I\!E[X^2] - I\!E[X]^2$, the variance of the final wealth can be expressed within model the ALM model as

$$\mathrm{Var}[X_T] = (1-\gamma)^2 \sum_{i \in \mathcal{L}_T} \pi_i (\sum_{j \in \mathcal{A}} v_j x^h_{i,j})^2 - y^2 \tag{4}$$

which can be readily incorporated into the formulation (2) leading to a quadratic programming problem with a block-diagonal Hessian [6].

On the other hand the use of of the variance as a risk measure has been criticised in various places as being to simplistic. Practitioners often recommend the use of a von Neumann-Morgenstern type [15] nonlinear utility formulation

$$I\!E[U(X_T)] \tag{5}$$

where $U : I\!R \to I\!R$ is a *utility function*. Usually U is assumed to be convex and non-decreasing which corresponds to a risk-averse investor. A popular choice for $U(x)$ is $U(x) = -\log x$. Konno et al. [10, 11], suggest the use of skewness (third moment of X_T) in the objective to adequately cover non-symmetrical distribution of return. Pflug [13] suggests the use of lower-semivariance

$$I\!E[X_T] - \lambda \sqrt{I\!E[([X_T - I\!E[X_T]]^-)^2]}$$

for the same reason. As shown in [6] these formulations can be incorporated into the model (2), at the expense of introducing nonlinear constraint and objective terms, while keeping the structure of the problem intact.

Stochastic dominance has been suggested as an alternative for risk modelling by [2]. A random variable X is said to dominate another r.v. y by *second order stochastic dominance* ($X \succeq_{ssd} Y$) if and only if

$$I\!E[u(X)] \geq I\!E[u(Y)]$$

for all non-decreasing concave utility functions; i.e. any rational risk-averse investor would prefer portfolio X to portfolio Y. In order to use stochastic dominance within ALM models one can define a *benchmark portfolio* B (whose return

is a random variable) and only consider investment decisions whose return outperforms the benchmark by s.s.d.:

$$\max_x I\!E[X_T(x)] \text{ s.t. } X_T(x) \succeq_{ssd} B \quad (+ \text{ inventory \& cash balance constraints}) \tag{6}$$

It has been shown in [2] that $X_T \succeq_{ssd} B$ is equivalent to

$$I\!E[(\eta - X_T)_+] \leq I\!E[(\eta - B)_+], \quad \forall \eta, \qquad (\text{here } (x)_+ = \max\{x, 0\}). \tag{7}$$

In the case where the benchmark (which is often sampled from historical data) has only discrete outcomes $\{b_j\}_{j=1,..,m}$ and X_T is given by a discrete distribution (as is the case in stochastic programming), constraints (7) can be modelled by the system

$$\sum_{i \in \mathcal{L}_T} \pi_i s_{i,j} \leq v_j, j = 1, ..., m, \qquad s_{i,j} \geq b_j - X_{T,i}, s_{i,j} \geq 0, \quad i \in \mathcal{L}_T, j = 1, ..., m$$

This leads to m constraints that are linking all final stage scenarios (and are thus of the same form as the final constraint in (2b)).

3 Solution Based on Interior Point Methods

Since their emergence in the 1990's Interior Point Methods (IPM) have proven to be among the most efficient methods for solving large stochastic programming problems and multitude of applications to ALM exist [1, 5, 6, 14]. The reasons for the success of IPMs on these problems include their applicability to a wide range of formulations, spanning linear, quadratic and nonlinear models, their comparative non-sensitivity to large problem sizes (IPMs are in practice observed to converge in $\mathcal{O}(\log n)$ iterations, where n is the problem size), and not least the amenability of the linear algebra operations to parallelisation. In the remainder of this section we concentrate on the computational complexity of IPMs for multistage ALM models. A more thorough description of the algorithm can be found in the above references. For the quadratic programming problem

$$\min \; c^T x + \tfrac{1}{2} x^T Q x, \qquad \text{s.t. } Ax = b, x \geq 0 \tag{8}$$

the main computational work consists of solving the Newton system

$$\Phi \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \xi_d - X^{-1}\xi_\mu \\ \xi_p \end{bmatrix}, \qquad \Phi = \begin{bmatrix} -Q - \Theta^{-1} & A^T \\ A & 0 \end{bmatrix} \tag{9}$$

where (x_k, s_k, y_k) is the current iterate, $\xi_p = b - Ax_k, \xi_d = c - A^T y_k - s_k + Qx_k, \xi_u = \mu e - X_k S_k e$, $X = \text{diag}(x_1, ..., x_n)$ and $\Theta = XS^{-1}$. For an IPM applied to the nonlinear problem

$$\min_x f(x) \text{ s.t. } g(x) = 0, x \geq 0$$

we need to use $A = \nabla g(x), Q = \nabla^2 f(x) + \sum_{i=1}^m y_i \nabla^2 g_i(x)$ in (9). For the ALM model (2) with the Jacobian as in Figure 1, matrix Φ can be reordered into the nested double-bordered block-diagonal form shown in Figure 2.

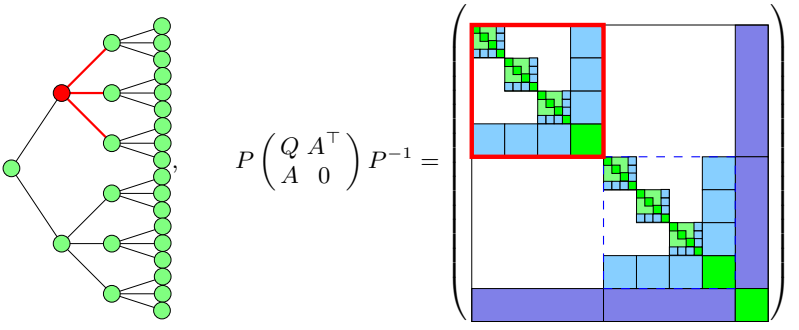

Fig. 2. Scenario tree and resulting nested bordered block-diagonal augmented system matrix Φ: highlighted is a matrix block corresponding to a scenario tree node

3.1 Parallelisation and Modelling

As pointed out earlier, the main computational effort in solving the ALM model (2) by Interior Point Methods consists of the repeated solution of system (9) for an augmented system matrix Φ with the nested structure displayed in Figure 2. Rather than solving this system directly it is more efficient to obtain a Cholesky-like factorisation $\Phi = LDL^T$ and solve system (9) by successive backsolves with L^T, D and L. Naturally parallelisation efforts are primarily targeted at these steps. A appealing property of bordered block-diagonal matrices is that by using Schur complement techniques the factorisation and backsolve operations can be decomposed into corresponding operations on the diagonal sub-blocks. Since these sub-block operations are independent they can be readily parallelised.

To exploit the nested structure in Φ in the parallel IPM solver OOPS the whole matrix is represented by a tree of (simple) bordered block-diagonal matrices, mirroring the scenario tree in Figure 2. Every node of the *matrix tree* is represented by an instance of a *structured matrix* object following object-oriented principles. In this manner linear algebra operations on the root node of the matrix tree (corresponding to the whole matrix) are recursively decomposed into corresponding operations on the leaf node matrices. Parallelism is dealt with in the same manner: to every node in the matrix tree a set of processors $\mathcal{P}_i$ is allocated. Between them the processors in $\mathcal{P}_i$ are responsible for all operations to be performed on this node. This is done by recursively allocating the processors in $\mathcal{P}_i$ to the child nodes of the current node (see Figure 3). The approach has been demonstrated to be scalable up to 1280 processors. Details of the implementation can be found in [7].

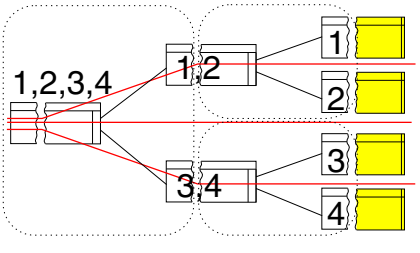

Fig. 3. Allocation of processors to matrix blocks

An important consideration is how to transfer the data needed to describe the problem onto the processors. Two approaches have been used traditionally: Either the model and data is generated in-situ on every processor; while efficient in execution this requires writing carefully handcrafted code for every new model. Alternatively, modelling languages such as AMPL[3] are designed to overcome this problem by providing the user with a descriptive language in which to formulate the model and automating the task of model generation. None of the existing modelling languages, however, support parallelisation of the model generation, creating a major bottleneck. In some of the large ALM problems solved by OOPS, the model description alone requires upwards of 40GB making serial model generation impossible. To overcome this issue OOPS is linked to SPML (Structure-conveying Parallel Modelling Language), a parallel model generator[8]. SPML extends AMPL with keywords to indicate model structure. A preprocessor extracts the model structure, builds processor-sized submodels and distributes them among the available processor, leaving the local model generation to be done on each processor independently.

4 Numerical Results

Numerical results on OOPS and SPML have been reported in various papers[5, 6, 4, 16]. We therefore restrict ourselves to some highlights. OOPS was able to solve an ALM problem with 6 stages, 12.8 million scenarios, and 1.02×10^9 variables on 1280 processors of the 1600-1.7GHz processor machine HPCx in 3020 seconds[4]. OOPS has been applied to various formulations of the ALM problem reported in this paper such as nonlinear utility functions and stochastic dominance constrained problems. In most cases solution times in serial improve (sometimes significantly) on those that can be obtained with the commercial solver CPLEX. On parallel machines a consistent speed-up of $6.22 - 7.64$ on 8 processors has been reported[5] as has been a speedup of 27.5 going from 16 to 512 processors[4].

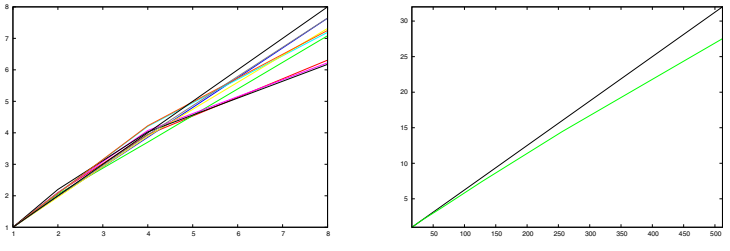

Fig. 4. Speedup for OOPS on 1-8 (a) and 16-512 (b) processors

5 Conclusions

We have presented recent approaches to the parallel solution of multistage portfolio optimization problems using a variety of approaches to model risk. Problems

of many millions of variables can now be routinely solved on moderate parallel hardware with almost linear speed-up, while the use of dedicated massively parallel machines makes the solution of problems with 10^9 variables and more feasible.

References

1. Blomvall, J., Lindberg, P.O.: Backtesting the performance of an actively managed option portfolio at the Swedish stock market, 1990–1999. J. Econ. Dyn. Control 27, 1099–1112 (2003)
2. Dentcheva, D., Ruszczyński, A.: Portfolio optimization with stochastic dominance constraints. J. Bank Financ 30, 433–451 (2006)
3. Fourer, R., Gay, D., Kernighan, B.W.: AMPL: A Modeling Language for Mathematical Programming. The Scientific Press, San Francisco (1993)
4. Gondzio, J., Grothey, A.: Direct solution of linear systems of size 10^9 arising in optimization with interior point methods. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. (eds.) PPAM 2005. LNCS, vol. 3911, pp. 513–525. Springer, Heidelberg (2006)
5. Gondzio, J., Grothey, A.: Parallel interior point solver for structured quadratic programs: Application to financial planning problems. Ann. Oper. Res. 152, 319–339 (2007)
6. Gondzio, J., Grothey, A.: Solving nonlinear portfolio optimization problems with the primal-dual interior point method. Eur. J. Oper. Res. 181, 1019–1029 (2007)
7. Gondzio, J., Grothey, A.: Exploiting structure in parallel implementation of interior point methods for optimization. Comput. Manage Sci. 6, 135–160 (2009)
8. Grothey, A., Hogg, J., Woodsend, K., Colombo, M., Gondzio, J.: A structure-conveying modelling language for mathematical and stochastic programming. Math. Program. Comput. 1, 223–247 (2009)
9. Høyland, K., Wallace, S.W.: Generating scenario trees for multistage decision problems. Manage Sci. 47, 295–307 (2001)
10. Konno, H., Shirakawa, H., Yamazaki, H.: A mean-absolute deviation-skewness portfolio optimization model. Annals of Operational Research 45, 205–220 (1993)
11. Konno, H., Suzuki, K.-I.: A mean-variance-skewness portfolio optimization model. J. Oper. Res. Soc. Jpn 38, 173–187 (1995)
12. Markowitz, H.M.: Portfolio selection. J. Financ, 77–91 (1952)
13. Pflug, G.C.: How to measure risk. In: Modelling and Decisions in Economics, pp. 39–59. Physica-Verlag, Heidelberg (1999)
14. Steinbach, M.: Recursive direct algorithms for multistage stochastic programs in financial engineering. In: Kall, P., Lüthi, H.-J. (eds.) Operations Research Proceedings, Selected Papers of the International Conference on Operations Research Zürich 1998, pp. 241–250. Springer, Heidelberg (1999)
15. von Neumann, J., Morgenstern, O.: Theory of Games and Economic Behaviour. Princeton University Press, Princeton (1953)
16. Yang, X., Gondzio, J., Grothey, A.: Asset-liability management modelling with risk control by stochastic dominance. J. Ass. Manag. 11, 73–93 (2010)