2003年日本オペレーションズ・リサーチ学会
春季研究発表会

大規模数理計画問題に対する内点法の並列化とアプリケーション

01307380 数理システム　*田辺隆人 TANABE Takahito
数理システム　尾関貴昭 OZEKI Takaaki

1 概要

大規模数理計画問題の内点法による求解の並列化実装について述べる。確率計画問題、CVaR など一般のリスク尺度による資産配分問題・多期間計画問題 [1]、サポートベクターマシンの学習の際に現れる大規模数理計画問題は変数を P 個のグループに分割した次のような形に定式化することができる。

$$\begin{aligned} \text{変数} \quad & x \in \mathbb{R}^{n_0}, y_k \in \mathbb{R}^{n_k}, k \in \{1,\ldots,P\} \\ \text{最小化} \quad & f(x, y_1, \cdots, y_B) \\ \text{条件} \quad & g^0(x, y_1, \cdots, y_B) = 0, x \geq 0 \\ & g^k(x, y_k) = 0, y_k \geq 0, k \in \{1,\ldots,P\} \end{aligned} \quad (1)$$

ここで問題の変数および制約式の数はそれぞれ n ($\equiv n_0 + \sum_{k=1}^{P} n_k$), m ($\equiv m_0 + \sum_{k=1}^{P} m_k$) とし、目的関数と制約式をそれぞれ $f : \mathbb{R}^n \to \mathbb{R}$, $g^0 : \mathbb{R}^n \to \mathbb{R}^{m_0}$ および $g^k : \mathbb{R}^{n_0+n_k} \to \mathbb{R}^{m_k}$ と定義する。本実装ではこの構造を直接利用することにより高いスケーラビリティを実現している。

本ソフトウエアは MPI パッケージを導入した Windows 環境上で動作可能であり、定式はモデリング言語により記述可能。モデル記述も並列度に依存せず、単一 CPU での実行時のものと共通化可能であるなど、ソフトウエアとしての可用性も高い。

2 アルゴリズム

内点法による求解アルゴリズムの最も主要な部分は制約式や Lagrange 関数の微係数から定義される一次方程式の解法である。この一次方程式を直接法によって解く操作は、変数と制約式について適当な並べ換えを行うと、次のような構造の対称行列の LDL^t 分解と前進消去後退代入に帰着される。

$$\begin{bmatrix} B_1 & 0 & 0 & 0 & 0 & C_1^t \\ 0 & \ddots & 0 & 0 & 0 & \vdots \\ 0 & 0 & B_k & 0 & 0 & C_k^t \\ 0 & 0 & 0 & \ddots & 0 & \vdots \\ 0 & 0 & 0 & 0 & B_P & C_P^t \\ C_1 & \cdots & C_k & \cdots & C_P & \tilde{B}_0 \end{bmatrix} \quad (2)$$

ただし $\tilde{B}_0 \in \mathbb{R}^{(n_0+m_0)\times(n_0+m_0)}, B_k \in \mathbb{R}^{m_k \times m_k}, C_k \in \mathbb{R}^{(n_0+m_0)\times m_k}$, $(k = 1, \cdots, P)$ で、C_k は g^0 の各 y_k についてのヤコビ行列、B_k は $-A_k H_k A_k^t$ に等しい。ただし、$A_k \in \mathbb{R}^{m_k \times n_k}$ は g^k の y_k についてのヤコビ行列である。$H_k \in \mathbb{R}^{n_k \times n_k}$ の実際の内容は内点法のアルゴリズムによって変化するが、Lagrange 関数の y_k に関するヘッセ行列 (もしくはその近似) と主、双対変数値から計算される行列であり、LP および目的関数に変数同士のクロスタームがない QP/NLP の場合には対角行列になる。$\tilde{B}_0$ は g^0, g^k の x についてのヤコビ行列の要素と、Lagrange 関数の x に関するヘッセ行列要素を設定した B_0 から $\tilde{B}_0 \equiv B_0 - \sum_{k=1}^{P} C_k H_k A_k^t$ として計算される。

この行列の設定と LDL^t 分解のアルゴリズムは次のようになる。

1. B_0 の計算
2. H_k, A_k, C_k, $(k = 1, \cdots, P)$ の計算
3. $B_k \equiv -A_k H_k A_k^t, C_k H_k A_k^t$ $(k = 1, \cdots, P)$ の計算
4. B_k の LDL^t 分解
   $B_k \to L_k D_k L_k^t, (k = 1, \cdots, P)$
5. $C_k (D_k L_k)^{-1}, (k = 1, \cdots, P)$ の計算
6. $\hat{B}_0 \equiv B_0 - \sum_{k=1}^{P} (C_k L_k^{-1} D_k^{-1} L_k^{-t} C_k^t + C_k H_k A_k^t)$ の計算
7. $\hat{B}_0$ の LDL^t 分解 ($\hat{B}_0 \to L_0 D_0 L_0^t$)

このうち 2.～5. は各 k について独立に実行できる。ここでは $\tilde{B}_0$ は計算に用いずに B_0 から直接 $\hat{B}_0$ を定義している。正方行列 B_0 の次元が $n_0 + m_0$ であることより、すべての制約式に共通して現れる変数および変数が全体のごく一部分である、すなわち

$$n >> n_0, \quad m >> m_0 \quad (3)$$

であるような状況では、1.,6.,7. の操作の計算時間は無視できるほど小さくなる。したがって、各 m_k, n_k が均等であれば、各 k についての処理をプロセッサ k が行うような並列化を行うことにより、行列の分解演算全体の負荷がほぼ均等に分配されることが期待される。

その場合、6.,7. の操作の実現のためには、各プロセッサ間で $C_k L_k^{-1} D_k^{-1} L_k^{-t} C_k^t + C_k H_k A_k^t$ を集約して $\hat{B}_0$ を計算する必要があるが、その際に必要な通信データ量は $(n_0 + m_0)^2$ に比例するので、(3) が成り立つ状況下では演算全体で扱うデータ量に比べて小さいと言える。

一次方程式の解を求めるためには、得られた LDL^t 分解結果と右辺ベクトルを使って前進消去・後退代入を行う必要があるが、右辺ベクトルを $(b_1, \cdots, b_P, b_0)^t$ と分割しておけば、これについても同様の並列化が可能である。

一次方程式解法の他に内点法アルゴリズムの実現には $f, g^0, g^k, \nabla f, \nabla g^0, \nabla g^k, \nabla^2 f, \nabla^2 g^0, \nabla^2 g^k$ の評価が必要である。これらの式の各コンポーネントは独立に計算可能であり、また特に f, g^0 が、

$$f = \sum_{k=1}^{P} f_k(x, y_k), \quad g^0 = \sum_{k=1}^{P} g_k^0(x, y_k) \quad , k \in \{1, \ldots, P\} \tag{4}$$

のように変数 y_k について分離可能な形をしている場合には、P 個のプロセッサに $f_k, g_k^0, g^k, \nabla f_k, \nabla g_k^0, \nabla g^k, \nabla^2 f_k, \nabla^2 g_k^0, \nabla^2 g^k$ の評価を割り振る並列化を行うことにより計算負荷を分配できる。この場合、最適性条件の評価、直線探索を行う際のステップサイズや信頼領域法の場合の信頼領域サイズの決定、メリット関数の評価を行う際に、各プロセッサ間での集積演算が必要となるが必要な通信量は高々定数か n_0 あるいは m_0 に比例する量であることがわかる。例えば最適性条件の一つである Lagrange 関数の停留条件 $\nabla L \equiv \nabla f - y_0 \cdot \nabla g_0 - \sum_{k=1}^{P} y_k \cdot \nabla g^k$ の評価の際には、$\nabla_x L$ の部分の計算の際に $\nabla_x f^k, \nabla_x g_k^0, \nabla_x g^k$ を各プロセッサ間で集積する必要があり、その際に必要な通信量はこれらのベクトルの長さである n_0 に比例する。

以上のことより、m_k, n_k がほぼ一定で、(3) および (4) が成り立つ状況では、各 k に関する量を別プロセッサが独立して担当する並列化によって、高いスケーラビリティが期待できる。

3 実装

1 で述べた応用例はいずれも (3) および (4) を満たす。これらは不確実性の記述と意思決定を行うものであり、想定される状況を精密に予測するべく結果の精度を上げようとするほど、(3) は強く満たされるようになるので、高いスケーラビリティが期待できる。

前項のアルゴリズムの実装として、各プロセッサ k には (1) を 分割した「部分問題 k」:

$$\begin{array}{rl} \text{変数} & x \in \mathbf{R}^{n_0}, y_k \in \mathbf{R}^{n_k} \\ \text{最小化} & f_k(x, y_k) \\ \text{条件} & g_k^0(x, y_k) = 0, x \geq 0 \\ & g^k(x, y_k) = 0, y_k \geq 0 \end{array} \tag{5}$$

を入力することによって、並列化された内点法が起動、(1) の求解が成されるソフトウエアを開発した。内点法のアルゴリズムと実装は数理計画法パッケージ NUOPT の Windows 版、並列化プラットフォームとして、Windows 上で動作する MPI 環境 (MPI Pro) を用いた。アルゴリズムの実行が終了すると、x^*, y_k^* が各プロセッサに得られる。

問題の記述はモデリング言語 SIMPLE を用いて行うが、前述した応用例では部分問題 k 同士は定式の形は全く同一でデータのみが異なるため、各プロセッサには共通のモデル記述と、各プロセッサに固有のデータを与えることによって問題の入力と求解を行うことができる。ただし、各プロセッサに固有な式や変数を示すためにモデル記述には y_k, g^k に対して「並列化」マークを付けておく仕様とする。

4 計算機実験

まずテストとして CVaR 最小化によるポートフォリオ問題:

$$\begin{array}{rl} \text{変数} & x_j,\ j \in A, \quad u_t, w_t, t \in S, \quad \alpha \\ \text{最小化} & \alpha + \frac{1}{|S|(1-\beta)} \sum_{t \in S} u_t \\ \text{条件} & \sum_j R_{tj} x_j + \alpha + u_t - w_t = 0, t \in S \\ & \sum_j \bar{r}_j x_j >= r_E, \\ & \sum_{j \in A} x_j = 1, \\ & x_j \geq 0, u_t \geq 0, w_t \geq 0 \end{array} \tag{6}$$

について計算機実験を行った。A が投資銘柄の集合、S が収益についての観測点の集合、変数 x_j が各銘柄 j の投資比率、$\bar{r}_j$ が各銘柄 j の期待収益率、r_E が期待収益率の下限、u_t, w_t はある観測点の VaR からの差を示す補助変数、R_{tj} は観測点 t における各銘柄 j の収益率である。観測点の集合の要素数 $|S|$ は各銘柄の収益率の観測に対応しており、CVaR が統計量として意味を持つためには、$|S| >> |A|$(銘柄数) である必要があり、行列 R のサイズは観測点に比例して増大する。サンプルの集合を適当に分割 $S \equiv \cup_{k=1}^{P} S_k$ することによってこの問題は (1) の形で表すことができる。その際、y_k には $u_t, w_t,\ t \in S_k$ が対応し、g^k を構成する制約式の係数行列として $R^k \equiv \{R_{tj} | t \in S_k\}$ が対応する。$S >> |A|$(銘柄数) であることから直接 (3) が満たされていることがわかり、問題が線形であることから、(4) も成り立つこと、相当する部分問題 (5) は元の問題と全く同一の構造を持っていることがわかる。

そのため、この場合には各プロセッサに共通なモデル記述と対応する R^k を与え、モデル記述において u_t, w_t と最初の等式制約に「並列化」マークを付けておくとモデルの記述は完了する。

以下は Celeron 466MHz, 128 M バイトメモリのマシン 3 台 (並列)、で銘柄数 50 銘柄の問題で観測点を変えた問題を解いた結果 (「並列」) を単一のマシンで解いた結果 (「単一」) と計算時間を比較したものである。内点法のアルゴリズムには直線探索法を用いた。

$\|S\|$	並列 (秒)	単一 (秒)	並列化倍率
9000	37	100	2.7
12000	47	135	2.9
15000	79	217	2.8
30000	162	–	–

いずれの観測数の結果からも高いスケーラビリティが実現されていることが観測できる。また、観測点が 30000 のケースでは、単一マシンではメモリ不足により求解が不可能となるが、3 台の並列化実行では求解が可能であった。この実装方式では問題のデータがプロセッサ毎に完全に分離されるため、並列化台数を増大させることにより、求解可能な問題規模の上限を増やすことが可能であることを示している。

謝辞:
並列化プログラムの開発および実験環境の手配と貴重な助言を行っていただきました、日立製作所の二木、直野、高橋の各氏にこの場を借りまして御礼申し上げます。

参考文献

[1] 枇々木規雄, 金融工学と最適化, 朝倉書店, 2001.