高松大学紀要，30．51～76

2次輸送問題に対する交互方向乗数法のベクトル並列計算機における実行

山　川　栄　樹

1．はじめに

実用的な並列計算機の開発に伴って，数理計画問題に対してもさまざま並列アルゴリズムが提案されている［Bertsekas and Tsitsiklis（1989），Censor and Zenios（1997），福島（1993），山川（1995）］。これらのアルゴリズムは，データ並列型のプログラミング環境において実行できる粒度の細かい並列アルゴリズムと，コントロール並列型のプログラミング環境を必要とする粒度の粗い並列アルゴリズムの2種類に大別される。とくに，前者に分類される並列アルゴリズムは，輸送問題をはじめとするネットワーク構造をもつ数理計画問題に適用すると，コネクションマシンCM-2などの大規模並列計算機上で非常に効率良く実行できることが知られている［Eckstein（1993），Zenios and Censor（1991）］。

ところが，社会の複雑化に伴って解くべき問題の規模はますます増大し，大規模並列計算機に実装できるプロセッサの数をはるかに上回る値となっている。そこで，CM-5やVPP500などの最近の並列計算機においては，プロセッサの数を比較的少数に抑え，各プロセッサに極めて高性能のベクトル演算装置を装備するものが多い。このようなベクトル並列計算機を効率よく動作させるためには，並列アルゴリズムの各ステップを構成する手続きを，複数のプロセッサを用いる並列処理に適した部分と，それぞれのプロセッサでベクトル処理を行う方が望ましい部分に切り分ける必要がある。また，各プロセッサのローカルメモリへのデータの割当てやプロセッサ間のデータ転送を，利用者が適切に指示しなければならない。しかしながら，数理計画問題問題に対する並列アルゴリズムの数値実験にしばしば利用されてきたCM-5においては，データ並列型のプログラミング環境が配列要素を各プロセッサに均等配分することを前提としている一方で，コントロール並列型のプログラミング環境を用いる場合はメッセージ受渡しの考え方に基づいてデータの転送を指示する複雑なコーディングを行うことが要求される。これに対してVPP500においては，コンパイラに対する指示を通常のFORTRAN 77プログラムに挿入するだけで，データと処理の分割やデータの転送を容易に実現することができる。

そこで，本稿では，分離可能な2次のコストをもつ輸送問題に対して交互方向乗数法を

適用した場合に得られる粒度の細かい並列アルゴリズム［Eckstein and Fukushima（1994）］を，ベクトル並列計算機 VPP500 上で実行する方法を提案し，大規模なテスト問題に対する性能を数値実験により検証する。本稿の構成は，次の通りである。まず 2 節において交互方向乗数法のアルゴリズムを説明する。つぎに，3 節において VPP500 上での実行方法を詳細に述べる。数値実験の結果は，4 節で報告する。最後に，5 節において簡単なまとめを行う。

以下では，ベクトルはすべて列ベクトルであると仮定し，ベクトルおよび行列の転置を上つき添字 $^{\top}$ により表す。また，微分可能な実数値関数 f の勾配ベクトルおよびヘッセ行列を，それぞれ ∇f, $\nabla^2 f$ と記述する。一方，h が $\Re^n$ から $\Re^m$ へのベクトル関数であるとき，∇h は $n \times m$ ヤコビ行列を表すものとする。なお，偏微分する変数を指定する必要がある場合には，$\nabla_x f$ のようにその変数の名前を下つき添字として明記する。さらに，ベクトルのノルムとしてはユークリッドノルムを用い，記号 $\|\cdot\|$ により表す。

2．アルゴリズム

数理計画問題に対する並列アルゴリズムには，行列や関数を分割し，もとの問題を二つの取扱いやすい問題に分解することによって得られるものが少なくない［福島（1993），山川（1995）］。本稿で考える交互方向乗数法も，そのような並列アルゴリズムの一つである。ここでは，まず，一般の非線形計画問題に対する乗数法について簡単に述べる。つぎに，二つの凸関数の和の形に分解された凸計画問題に乗数法を適用することにより，交互方向乗数法を導く。さらに，2 次輸送問題に対して交互方向乗数法を適用すると，高度な並列アルゴリズムが得られることを示す。

2.1 乗数法

ここでは，つぎのような等式制約つき最適化問題について考える。

$$\begin{aligned} &\text{目的関数：}\ f(x) \qquad \rightarrow \ \text{最小} \\ &\text{制約条件：}\ h(x) = 0. \end{aligned} \tag{2.1}$$

ただし，$f : \Re^n \rightarrow \Re$ と $h : \Re^n \rightarrow \Re^m$ は 2 回連続的微分可能な非線形関数とする。集合

$$S = \{ x \mid h(x) = 0 \}$$

の点 x^* に対して，x^* を含む $\Re^n$ の開部分集合 $U(x^*)$ が存在し，条件

$$f(x^*) \leq f(x), \qquad {}^\forall x \in S \cup U(x^*),$$

がなりたつとき，x^* を問題（2.1）の局所最適解と呼ぶ。とくに，x^* が条件

$$f(x^*) < f(x), \qquad {}^\forall x \in S \cup U(x^*), \quad x \neq x^*,$$

を満たす場合には，孤立局所最適解と呼ばれる。よく知られているように，点 $x^* \in \Re^n$ に対してベクトル $v^* \in \Re^m$ が存在し，2次の最適性条件

$$\begin{aligned} &\nabla_x l(x^*, v^*) = 0, \\ &h(x^*) = 0, \\ &y^\top \nabla_x^2 l(x^*, v^*)\, y > 0, \quad {}^\forall y \in \{\, y \mid \nabla h(x^*)^\top y = 0,\ y \neq 0 \,\}, \end{aligned} \tag{2.2}$$

がなりたつならば，x^* は問題（2.1）の孤立局所最適解である。ただし，

$$l(x, v) = f(x) + v^\top h(x)$$

は問題（2.1）に対するラグランジュ関数，v はラグランジュ乗数ベクトルである。

乗数法は，制約なし最適化問題を繰返し解くことによって制約つき最適化問題の解を得ようとする変換法の一つである。問題（2.1）に対して，拡張ラグランジュ関数を

$$l_t(x, v) = f(x) + v^\top h(x) + \frac{t}{2} \|h(x)\|^2$$

により定義する。ただし，t は正の値をとるパラメータである。乗数法の第 k 反復においては，$v^{(k)} \in \Re^m$ と十分大きな正の数 t に対して，制約なし最適化問題

$$\text{目的関数：}\ l_t(x, v^{(k)}) \quad \rightarrow \quad \text{最小} \tag{2.3}$$

の局所最適解 $x^{(k+1)}$ を求め，ラグランジュ乗数ベクトルを

$$v^{(k+1)} := v^{(k)} + t\, h(x^{(k+1)}) \tag{2.4}$$

により更新する。容易に確かめられるように，条件（2.2）を満たす問題（2.1）の孤立局所最適解は，$v^{(k)} = v^*$ であれば，十分大きな正の数 t に対して問題（2.3）の孤立局所最適解となる。一方，$x^{(k+1)}$ は問題（2.3）の局所最適解であるから，

$$\nabla_x l_t(x^{(k+1)}, v^{(k)}) = \nabla f(x^{(k+1)}) + \nabla h(x^{(k+1)})v^{(k)} + t\nabla h(x^{(k+1)})h(x^{(k+1)}) = 0$$

がなりたつ。このとき，式（2.4）により$v^{(k)}$を更新すれば，

$$\nabla_x l(x^{(k+1)}, v^{(k+1)}) = \nabla f(x^{(k+1)}) + \nabla h(x^{(k+1)})v^{(k+1)} = 0$$

である。よって，条件（2.2）の第1式より，$v^{(k+1)}$はv^*のよい近似となることが期待される。さらに，この手続きによって生成される点列$\{x^{(k)}\}$, $\{v^{(k)}\}$がそれぞれある値に収束するならば，(2.4) より$\{h(x^{(k)})\}$は0に収束する。ところが，問題（2.3）の局所最適解において$h(x)=0$ならば，それはもとの問題（2.1）の局所最適解となる。実際，適当な仮定のもとで，点列$\{x^{(k)}\}$と$\{v^{(k)}\}$はそれぞれ問題（2.1）の局所最適解と対応するラグランジュ乗数ベクトルに収束することが証明される［Bertsekas（1982），今野・山下（1978)]。

2.2 交互方向乗数法

ここでは，つぎのような一般的な凸計画問題のクラスについて考える。

$$\text{目的関数：}\ F(x) + G(Ax) \quad \rightarrow \quad \text{最小.} \tag{2.5}$$

ただし，$F : \mathfrak{R}^n \rightarrow (-\infty, +\infty]$ と $G : \mathfrak{R}^m \rightarrow (-\infty, +\infty]$ は閉真凸関数［Rockafellar(1970)］，Aは$m \times n$行列で$\mathrm{rank}\, A = m$とする。問題（2.5）は，さまざまな凸計画問題を包含している。たとえば，通常の凸計画問題は，有限値をとる凸関数$f : \mathfrak{R}^n \rightarrow \mathfrak{R}$と凸集合$C \subset \mathfrak{R}^n$を用いて

$$\begin{aligned} &\text{目的関数：} f(x) \quad \rightarrow \quad \text{最小} \\ &\text{制約条件：} x \in C, \end{aligned}$$

と記述されるが，$m = n$としてAをn次元の単位行列に選び，

$$\begin{aligned} F(x) &= f(x), \quad x \in \mathfrak{R}^n, \\ G(x) &= \begin{cases} 0, & x \in C, \\ +\infty, & x \notin C, \end{cases} \end{aligned}$$

とおくことにより，問題（2.5）に帰着される。また，問題（2.5）のような定式化は，閉真凸関数$F+G$を全体として最小化するよりもFとGを個別に最小化する方が容易である

問題に対して，分割型の解法を構築する場合にしばしば用いられる。

問題 (2.5) は，つぎのような制約つき最適化問題とみなすことができる。

$$\begin{aligned} &\text{目的関数：} F(x)+G(Ay) \quad \rightarrow \quad \text{最小} \\ &\text{制約条件：} y-x=0. \end{aligned} \tag{2.6}$$

問題 (2.6) に対する拡張ラグランジュ関数は，

$$L_t(x,y,v)=F(x)+G(Ay)+v^{\top}(y-x)+\frac{t}{2}\|y-x\|^2$$

より定義されるが，任意の $t>0$ に対して L_t は x, y に関する閉真凸関数である。そこで，$\mathfrak{R}^n$ の点 $x^{(k)}, v^{(k)}$ と適当な正の数 t に対して，変数 y に関する制約なし最適化問題

$$\text{目的関数：} L_t(x^{(k)},y,v^{(k)}) \quad \rightarrow \quad \text{最小} \tag{2.7}$$

の最適解 $y^{(k+1)}$ と，変数 x に関する制約なし最適化問題

$$\text{目的関数：} L_t(x,y^{(k+1)},v^{(k)}) \quad \rightarrow \quad \text{最小}$$

の最適解 $x^{(k+1)}$ を順次求め，ラグランジュ乗数ベクトルを

$$v^{(k+1)}:=v^{(k)}+t\,(y^{(k+1)}-x^{(k+1)})$$

により更新するような反復解法を考える。この手続きは，拡張ラグランジュ関数をもとの問題 (2.6) の変数 y, x について交互に最小化することから交互方向乗数法と呼ばれており，与えられた凸計画問題の特殊な構造を利用して並列アルゴリズムを構成する有力な手法の一つである[Bertsekas and Tsitsiklis(1989), Eckstein(1993), Fukushima(1992)]。なお実用上は，問題(2.7)の最適解 $y^{(k+1)}$ を，パラメータ $\rho\in(0,2)$ を用いて $\rho y^{(k+1)}+(1-\rho)x^{(k)}$ のように緩和し，変数 x に関する制約なし最適化問題

$$\text{目的関数：} L_t(x,\rho y^{(k+1)}+(1-\rho)x^{(k)},v^{(k)}) \quad \rightarrow \quad \text{最小} \tag{2.8}$$

の最適解 $x^{(k+1)}$ を用いて，ラグランジュ乗数ベクトルを

$$v^{(k+1)}:=v^{(k)}+t\{\rho y^{(k+1)}+(1-\rho)x^{(k)}-x^{(k+1)}\} \tag{2.9}$$

と更新することにより，収束を加速できることが知られている。

問題（2.7）の目的関数は，行列Aによる線形変換と閉真凸関数Gの合成関数を含んでいるが，問題（2.7）に対する最適性条件を考慮することにより，このような合成関数を直接取り扱うことは回避できる[Eckstein and Fukushima（1994）]。以下では，このことを簡単に説明しよう。問題（2.7）の最適解$y^{(k+1)}$に対して，ある$\pi^{(k+1)} \in \partial G(A y^{(k+1)})$が存在し，

$$A^{\top}\pi^{(k+1)} + v^{(k)} + t\,(y^{(k+1)} - x^{(k)}) = 0 \tag{2.10}$$

がなりたつ。ただし，∂GはGの劣微分［Rockafellar（1970）］である。ここで$z^{(k+1)} = A y^{(k+1)}$とおき，$\mathbf{rank}\,A = m$に注意すると，（2.10）より

$$\pi^{(k+1)} = (A\,A^{\top})^{-1}\{t\,(A\,x^{(k)} - z^{(k+1)}) - A\,v^{(k)}\} \tag{2.11}$$

となるが，$\pi^{(k+1)} \in \partial G(z^{(k+1)})$より$z^{(k+1)}$は制約なし最適化問題

$$\text{目的関数：}\ G(z) + z^{\top}(A\,A^{\top})^{-1}A(v^{(k)} - t\,x^{(k)}) + \frac{t}{2}\,z^{\top}(A\,A^{\top})^{-1}z \quad \rightarrow \quad \text{最小} \tag{2.12}$$

の最適解であることがわかる。さらに，（2.10）より

$$y^{(k+1)} = x^{(k)} - \frac{1}{t}\,(A^{\top}\pi^{(k+1)} + v^{(k)})$$

であるから，

$$w^{(k+1)} = (1-\rho)v^{(k)} - \rho\,A^{\top}\pi^{(k+1)}$$

とおけば，問題（2.8）は制約なし最適化問題

$$\text{目的関数：}\quad F(x) - x^{\top}w^{(k+1)} + \frac{t}{2}\,\|x - x^{(k)}\|^2 \quad \rightarrow \quad \text{最小} \tag{2.13}$$

と等価であり，その最適解$x^{(k+1)}$を用いると，更新公式（2.9）は

$$v^{(k+1)} := w^{(k+1)} - t\,(x^{(k+1)} - x^{(k)})$$

と書き直すことができる。

結局，問題（2.5）に対する交互方向乗数法は，つぎのようにまとめられる。

アルゴリズム　1.

ステップ 0: パラメータ $t>0$ と $0<\rho<2$ を定める。初期探索点 $x^{(0)}, v^{(0)} \in \mathfrak{R}^n$ を選び，

$k:=0$ とおく。

ステップ 1: 部分問題

$$\text{目的関数：}\ G(z)+z^{\top}(AA^{\top})^{-1}A(v^{(k)}-tx^{(k)})+\frac{t}{2}z^{\top}(AA^{\top})^{-1}z \ \rightarrow\ \text{最小}$$

の最適解 $z^{(k+1)}$ を求め，

$$\pi^{(k+1)}:=(AA^{\top})^{-1}\{t(Ax^{(k)}-z^{(k+1)})-Av^{(k)}\},$$
$$w^{(k+1)}:=(1-\rho)v^{(k)}-\rho A^{\top}\pi^{(k+1)}$$

とおく。

ステップ 2: 部分問題

$$\text{目的関数：}\ F(x)-x^{\top}w^{(k+1)}+\frac{t}{2}\|x-x^{(k)}\|^2 \ \rightarrow\ \text{最小}$$

の最適解 $x^{(k+1)}$ を求め，

$$v^{(k+1)}:=w^{(k+1)}-t(x^{(k+1)}-x^{(k)})$$

とおく。

ステップ 3: $x^{(k+1)}$，$v^{(k+1)}$ が適当な収束判定条件を満たせば終了する。さもなければ，

$k:=k+1$ としてステップ 1 へ戻る。　□

問題（2.12）の目的関数と式（2.11）には，行列 $(AA^{\top})^{-1}$ の計算が含まれている。しかし，グラフの接続行列のように行列 A が特殊な構造をもつ場合には，その値を効率的に求めることができる。

2.3　2次輸送問題への適用

ここでは，2部グラフ $\mathcal{G}=(\mathcal{V}_1,\mathcal{V}_2,\mathcal{E})$ 上で定義される分離可能な2次輸送問題

$$\begin{aligned}
&\text{目的関数：} \sum_{(i,j)\in\mathcal{E}}\left\{\frac{d_{ij}}{2}(x_{ij})^2+c_{ij}x_{ij}\right\} \quad \rightarrow \quad \text{最小}\\
&\text{制約条件：} \sum_{j:(i,j)\in\mathcal{E}} x_{ij}=\alpha_i, \qquad i\in\mathcal{V}_1,\\
&\qquad\qquad \sum_{i:(i,j)\in\mathcal{E}} x_{ij}=\beta_j, \qquad j\in\mathcal{V}_2,\\
&\qquad\qquad x_{ij}\geq 0, \quad (i,j)\in\mathcal{E},
\end{aligned} \tag{2.14}$$

に交互方向乗数法を適用することを考える。ただし，$\sum_{i\in\mathcal{V}_1}\alpha_i=\sum_{j\in\mathcal{V}_2}\beta_j$, $d_{ij}\geq 0,(i,j)\in\mathcal{E}$ である。いま，$m=|\mathcal{V}_2|, n=|\mathcal{E}|$ とおき，グラフ $\mathcal{G}$ の接続行列の行のうち節点集合 $\mathcal{V}_2$ に対応する部分のみから構成される $m\times n$ 行列を A とする。このとき，問題（2.14）の $j\in\mathcal{V}_2$ に対する等式制約条件は $(Ax)_j=\beta_j$ のように書ける。また，節点 $j\in\mathcal{V}_2$ の次数を ξ_j とすれば，$AA^{\mathsf{T}}=\mathrm{diag}\{\xi_j\}_{j\in\mathcal{V}_2}$ である。さらに，

$$F(x)=\begin{cases}\displaystyle\sum_{(i,j)\in\mathcal{E}}\left\{\frac{d_{ij}}{2}(x_{ij})^2+c_{ij}x_{ij}\right\}, & x\in X, \quad (i,j)\in\mathcal{E},\\ +\infty, & \text{上記以外,}\end{cases}$$

$$G(z)=\begin{cases}0, & z_j=\beta_j, \quad j\in\mathcal{V}_2,\\ +\infty, & \text{上記以外,}\end{cases}$$

とおくと，問題（2.14）は問題（2.5）の形に表される。ただし

$$X=\{x \mid \textstyle\sum_{j:(i,j)\in\mathcal{E}} x_{ij}=\alpha_i,\ i\in\mathcal{V}_1, \quad x_{ij}\geq 0,\ (i,j)\in\mathcal{E}\}$$

である。このとき，関数 G の定義より，制約なし最適化問題（2.12）は自明な最適解 $z_j^{(k+1)}=\beta_j, j\in\mathcal{V}_2$，をもつ。一方，関数 F と集合 X の定義から，制約なし最適化問題（2.13）は，各 $i\in\mathcal{N}_1$ について独立に解けることが確かめられる。結局，問題（2.14）に対する交互方向乗数法は，つぎのように記述される［Eckstein and Fukushima（1994）］。

アルゴリズム 2.

ステップ 0: パラメータ $t>0$ と $0<\rho<2$ を定める。初期探索点 $x^{(0)}, v^{(0)}\in\Re^n$ を選び，$k:=0$ とおく。

ステップ 1: すべての $j \in \mathcal{V}_2$ に対して，次式の計算を行う。

$$\pi_j^{(k+1)} := \frac{1}{\xi_j}\left\{t\left(\sum_{i:(i,j)\in\mathcal{E}} x_{ij}^{(k)} - \beta_j\right) - \sum_{i:(i,j)\in\mathcal{E}} v_{ij}^{(k)}\right\}.$$

ステップ 2: すべての $(i,j) \in \mathcal{E}$ に対して，次式の計算を行う。

$$w_{ij}^{(k+1)} := (1-\rho)v_{ij}^{(k)} - \rho\,\pi_j^{(k+1)}.$$

ステップ 3: すべての $i \in \mathcal{V}_1$ に対して，部分問題

$$\text{目的関数：} \sum_{j:(i,j)\in\mathcal{E}} \left\{\frac{d_{ij}+t}{2}(x_{ij})^2 + (c_{ij} - w_{ij}^{(k+1)} - t\,x_{ij}^{(k)})\,x_{ij}\right\} \quad \rightarrow \quad \text{最小}$$

$$\text{制約条件：} \sum_{j:(i,j)\in\mathcal{E}} x_{ij} = \alpha_i,$$

$$x_{ij} \geq 0, \quad j:(i,j)\in\mathcal{E},$$

の最適解 $x_{ij}^{(k+1)}$, $j:(i,j)\in\mathcal{E}$, を求める。

ステップ 4: すべての $(i,j) \in \mathcal{E}$ に対して，次式の計算を行う。

$$v_{ij}^{(k+1)} := w_{ij}^{(k+1)} - t\,(x_{ij}^{(k+1)} - x_{ij}^{(k)}).$$

ステップ 5: $x^{(k+1)}$, $v^{(k+1)}$ が適当な収束判定条件を満たせば終了する。さもなければ，

$k := k+1$ としてステップ 1 へ戻る。 □

ステップ 1 はすべての $j \in \mathcal{V}_2$ について，ステップ 3 はすべての $i \in \mathcal{V}_1$ について，さらに，ステップ 2 と 4 はすべての $(i,j) \in \mathcal{E}$ について並列的に処理できるため，アルゴリズム 2 は並列計算機を用いて効率的に実行できるものと期待される。

ステップ 3 の部分問題は連続型の単純制約資源配分問題であり，効率的な解法が存在する［一森(1994)］。まず，節点 $i \in \mathcal{V}_1$ の次数を ζ_i とし，$\bar{c}_{ij}^{(k+1)} = c_{ij} - w_{ij}^{(k+1)} - t\,x_{ij}^{(k)}$, $\bar{d}_{ij} = d_{ij} + t$ とおく。また，$\bar{c}_{ij}^{(k+1)}$, $\bar{d}_{ij}$, x_{ij} の第 2 添字を適当につけ直して，$\bar{c}_{i1} \leq \bar{c}_{i2} \leq \cdots \leq \bar{c}_{i\zeta_i}$ となるようにする。このとき，各部分問題は

$$\text{目的関数：} \sum_{h=1}^{\zeta_i} \left\{ \frac{\bar{d}_{ih}}{2} (x_{ih})^2 + \bar{c}_{ih}^{(k+1)} x_{ih} \right\} \quad \rightarrow \quad \text{最小}$$

$$\text{制約条件：} \sum_{h=1}^{\zeta_i} x_{ih} = \alpha_i, \tag{2.15}$$

$$x_{ih} \geq 0, \quad h = 1, \ldots, \zeta_i,$$

と書きかえることができる。さらに，問題（2.15）の等式制約に対する最適ラグランジュ乗数を $u_i^{(k+1)}$ とおくと，1次の最適性条件より，ある $\hat{h}_i \in [1, \zeta_i]$ が存在して

$$x_{ih}^{(k+1)} = \begin{cases} \dfrac{u_i^{(k+1)} - \bar{c}_{ih}^{(k+1)}}{\bar{d}_{ih}}, & h = 1, \ldots, \hat{h}_i, \\ 0, & h = \hat{h}_i + 1, \ldots, \zeta_i, \end{cases} \tag{2.16}$$

がなりたつ。ところが，問題（2.15）の等式制約条件より，

$$u_i^{(k+1)} = \frac{\alpha_i + \sum_{h=1}^{\hat{h}_i} \dfrac{\bar{c}_{ih}^{(k+1)}}{\bar{d}_{ih}}}{\sum_{h=1}^{\hat{h}_i} \dfrac{1}{\bar{d}_{ih}}} \tag{2.17}$$

を得る。また，$(u_i - \bar{c}_{ih}^{(k+1)})/\bar{d}_{ih}$, $h = 1, \ldots, \hat{h}_i$, は正の値をとる u_i の増加関数であり，その和は $\hat{h}_i$ の値に関係なく一定であるから，$\hat{h}_i$ が大きいほど $u_i^{(k+1)}$ の値は小さくなる。したがって，式（2.17）により計算した $u_i^{(k+1)}$ が $u_i^{(k+1)} > \bar{c}_{i\hat{h}_i}^{(k+1)}$ となる最大の $\hat{h}_i$ を見つけ，（2.16）により $x_{ih}^{(k+1)}$, $h = 1, \ldots, \zeta_i$, を定めればよい。

実際の計算においては，必ずしも $\bar{c}_{ij}^{(k+1)}$, $j : (i,j) \in \mathcal{E}$, を完全に整列する必要はない。例えば，次数 ζ_i が大きい場合には，クイックソート［茨木（1989）］におけるグループ分割の考え方を組込んだつぎのような手続きを用いて，部分問題（2.15）を効率的に解くことができる。

手続き 3.

ステップ 0: 添字集合 $J_i := \{ j \mid (i,j) \in \mathcal{E} \}$ をセットし，

$$u_i := \frac{\alpha_i + \sum_{j \in J_i} \dfrac{\bar{c}_{ij}^{(k+1)}}{\bar{d}_{ij}}}{\sum_{j \in J_i} \dfrac{1}{\bar{d}_{ij}}}$$

とおく。もし，J_i のすべての要素 j に対して $u_i > \bar{c}_{ij}^{(k+1)}$ ならば，ステップ2へ進む。さもなければ，$\bar{J}_i := \emptyset$ としてステップ1へ進む。

ステップ 1: 添字集合 J_i の要素数が1であるか，すべての要素 j について $\bar{c}_{ij}^{(k+1)}$ が同じ値である場合には，ステップ2へ進む。さもなければ，J_i から適当な二つの要素 j_1, j_2 を選び，$\bar{c}_{ij_1}^{(k+1)} < \bar{c}_{ij_2}^{(k+1)}$ ならば j_1 を，$\bar{c}_{ij_1}^{(k+1)} > \bar{c}_{ij_2}^{(k+1)}$ ならば j_2 を $\hat{j}$ とおいてステップ3へ進む。

ステップ 2: ラグランジュ乗数の値 u_i をもとに，最適解を

$$x_{ij}^{(k+1)} := \max\left\{0, \frac{u_i - \bar{c}_{ij}^{(k+1)}}{\bar{d}_{ij}}\right\}, \quad j : (i,j) \in \mathcal{E},$$

により計算し，処理を終了する。

ステップ 3: 添字集合

$$\hat{J}_i := \{ j \mid \bar{c}_{ij}^{(k+1)} \le \bar{c}_{i\hat{j}}^{(k+1)},\ j \in J_i \}$$

を求め，ラグランジュ乗数の推定値を次式により計算する。

$$u_i := \frac{\alpha_i + \sum_{j \in \bar{J}_i \cup \hat{J}_i} \dfrac{\bar{c}_{ij}^{(k+1)}}{\bar{d}_{ij}}}{\sum_{j \in \bar{J}_i \cup \hat{J}_i} \dfrac{1}{\bar{d}_{ij}}}.$$

ステップ 4: もし $u_i > \bar{c}_{i\hat{j}}^{(k+1)}$ ならば，$\bar{J}_i := \bar{J}_i \cup \hat{J}_i$, $J_i := J_i \setminus \hat{J}_i$ とおく。さもなければ $J_i := \hat{J}_i$ とおく。ステップ1へ戻る。 □

ステップ3で求めるべき u_i の分母および分子に現れる総和計算のうち，$j \in \bar{J}_i$ に対する総和はそれ以前の反復において既に計算済である。そこで，ステップ3においては，$j \in \hat{J}_i$ に対する総和のみを新たに計算し直せばよい。なお，ステップ2は $j : (i,j) \in \mathcal{E}$ について並列処理可能である。

一方，次数 ζ_i が小さい場合には，つぎのような単純な手続きを適用すれば十分である。

手続き 4.

ステップ 0: 添字集合 J_i および $\bar{J}_i$ の初期値を，それぞれ $J_i := \{ j \mid (i,j) \in \mathcal{E} \}$, $\bar{J}_i := \emptyset$ に

セットする。

ステップ 1: 添字集合 J_i のすべての要素 j に対して $\bar{c}_{i\hat{j}}^{(k+1)} \leq \bar{c}_{ij}^{(k+1)}$ となる $\hat{j} \in J_i$ を一つ見つける。

ステップ 2: ラグランジュ乗数の推定値を，次式により計算する。

$$u_i := \frac{\alpha_i + \sum_{j \in \tilde{J}_i \cup \{\hat{j}\}} \dfrac{\bar{c}_{ij}^{(k+1)}}{\bar{d}_{ij}}}{\sum_{j \in \tilde{J}_i \cup \{\hat{j}\}} \dfrac{1}{\bar{d}_{ij}}}.$$

ステップ 3: もし $u_i > \bar{c}_{i\hat{j}}^{(k+1)}$ ならば，$\tilde{J}_i := \tilde{J}_i \cup \{\hat{j}\}$, $J_i := J_i \setminus \{\hat{j}\}$ として，ステップ 1 へ戻る。さもなければ，ステップ 4 へ進む。

ステップ 4: ラグランジュ乗数の値を

$$u_i := \frac{\alpha_i + \sum_{j \in \tilde{J}_i} \dfrac{\bar{c}_{ij}^{(k+1)}}{\bar{d}_{ij}}}{\sum_{j \in \tilde{J}_i} \dfrac{1}{\bar{d}_{ij}}}$$

により計算し，最適解を

$$x_{ij}^{(k+1)} := \max\left\{0, \frac{u_i - \bar{c}_{ij}^{(k+1)}}{\bar{d}_{ij}}\right\}, \quad j : (i,j) \in \mathcal{E},$$

として処理を終了する。□

ステップ 1 で添字 $\hat{j}$ を見つけるためには，バブルソート［茨木（1989)］などの基本的な整列法と同じように，J_i の隣り合う要素について対応する $\bar{c}_{ij}^{(k+1)}$ の値を順次比較し，必要に応じてそれらの格納場所を交換すればよい。また，ステップ 2 で求めるべき u_i の分母および分子に現れる総和計算のうち，$j \in \tilde{J}_i$ に対する総和は，前反復の同じステップにおいて既に計算済である。そこで，ステップ 2 においては，添字が $\hat{j}$ の項を新たに加算するだけでよい。一方，ステップ 4 で必要となる u_i の値は，前反復のステップ 2 で求めた u_i の値にほかならない。なお，ステップ 4 における $x_{ij}^{(k+1)}$ の計算は，$j : (i,j) \in \mathcal{E}$ について並列処理可能である。

3．実行方法

ここでは，アルゴリズム2をベクトル並列計算機VPP500上で実行する方法について述べる。まず，VPP500のハードウェアおよびソフトウェアの特徴を説明する。つぎに，ベクトル化により処理の高速化を図るためのコーディング上の工夫について述べる。最後に，複数のプロセッサを用いた処理の並列化について考える。

3.1　ベクトル並列計算機

ベクトル並列計算機VPP500は，クロスバーネットワークにより相互接続された複数のプロセッサから構成されるスーパーコンピュータである。各プロセッサには，演算能力が200 MFLOPSのスカラーユニットと，1.6 GFLOPSの性能をもつベクトル演算装置が装備されている［平野(1995)］。ベクトル演算装置は，加減算／論理，乗算，除算，マスク，ロード，ストアの各パイプラインをもち，これらは並列的に動作できる［富士通（1997a)］。また，各プロセッサは256 Mbytesのローカルメモリをもち，大規模なデータに対する効率的なアクセスを実現している。クロスバーネットワークは，プロセッサ間のデータ転送と同期制御を行い，データ転送速度は400 Mbpsである。

並列アルゴリズムのコーディングには，専用のプログラミング言語VPP FORTRAN 77を用いる。VPP FORTRAN 77では，通常のFORTRAN 77の文に加えて，コンパイラにベクトル化または並列化に関する指示を与える特別な文を記述することができる。なお，とくに指示をしなくてもコンパイラはある基準のもとで自動的にベクトル化を行うが，並列化については利用者がその方法をコンパイラに指示しなければならない。

コンパイラが自動的に行うベクトル化には，つぎのようなものがある［富士通(1997a)］。繰返しに伴う依存関係のないDOループは，一般の条件文や最大（小）値探索，総和演算が含まれている場合でも自動的にベクトル化される。さらに，飛出しがあるDOループも，一定の条件を満たせばベクトル化される。多重DOループは，対象データに依存関係がなければ，原則として最も内側のループが自動的にベクトル化される。なお，対象データがメモリの連続した領域に格納されている場合には，ベクトル長を長くする目的でループの一重化が施されることがある。一方，一重化されなかった多重DOループにおいては，ベクトル化対象ループのすぐ外側のDOループの回転数を半分にし，代わりにベクトル化対象ループの実行文を二重に展開するベクトルアンローリングとよばれる最適化が自動的に行われる。これによって，外側ループではループ制御命令の実行回数が減少し，内側ルー

プではベクトル演算の密度が高くなるため，処理が高速化される。したがって，内側ループのベクトル長が十分長い場合には，自動的に行われる一重化を抑止し，ベクトルアンローリングを施す方が効果的である。このような場合には，適当なベクトル化指示行を挿入するか，予め外側ループをアンローリングするコーディングをしておけばよい。

一方，並列化指示行は !XOCL で始まる制御文で，つぎのようなものがある[富士通(1997b)]。

!XOCL PROCESSOR プロセッサグループ名（使用プロセッサ数）

!XOCL PARALLEL REGION

!XOCL SPREAD REGION /プロセッサグループ名(プロセッサ番号)

!XOCL REGION /プロセッサグループ名(プロセッサ番号)

!XOCL SPREAD DO

!XOCL END SPREAD

!XOCL UNIFY(配列名(/プロセッサグループ名))

!XOCL END PARALLEL

プログラムの先頭には PROCESSOR 文を置き，並列処理を行う際に使用するプロセッサの数を宣言する。並列処理を行う範囲は，プログラムの対応する部分を PARALLEL REGION 文と END PARALLEL 文で囲むことによって指示する。これらの文の外側では，プログラムは一つのプロセッサ上で実行される。ところが，PARALLEL REGION 文に出会うと，実行中のプログラムとデータが利用可能なすべてのプロセッサに複写され，各プロセッサはそれぞれのローカルメモリに格納されたデータに対して同一の処理を独立に実行し始める。プロセッサごとに別々の処理を実行させるためには，SPREAD REGION 文と END REGION 文の間に必要に応じて REGION 文を挿入し，FORTRAN 77 のブロック IF 文と同じような形式で各プロセッサの実行文を記述すればよい。一方，DO ループを SPREAD DO 文と END SPREAD 文で囲めば，繰返しの処理が利用可能なプロセッサにおいて分割実行される。ただし，配列要素に対する演算を分割実行すると，実行結果を格納する配列要素の値を他のすべてのプロセッサに転送しなければならない場合が多い。このようなデータ転送を指示するのが，UNIFY 文である。最後に，END PARALLEL 文に出会うと，ひとつのプロセッサを残してデータとプログラムが消滅する。

なお，PARALLEL REGION 文の実行には負荷がかかるため，アルゴリズムの形態に関係なくプログラムの最初に一度だけ実行することが望ましい。もし，反復解法のなかに一つ

のプロセッサで実行できるステップが含まれるなら，すべてのプロセッサで同じ処理を実行させるようにする。また，UNIFY 文によるデータ転送も，同じ反復のなかで繰返し使用すると，通信の負荷が増大して処理効率の低下を招くので，注意が必要である。

3.2 ベクトル化

VPP500 の構造的特徴に注意すると，互いに独立な手続きを複数のプロセッサを用いて並列的に実行するとともに，依存関係のない大量のデータに対する演算をそれぞれのプロセッサにおいてベクトル処理することによって，並列アルゴリズムを効率的に実行できることがわかる。アルゴリズム 2 の場合，ステップ 3 が添字 $i \in \mathcal{V}_1$ について独立な手続きであるのに対して，ステップ 1 は $j \in \mathcal{V}_2$ を，ステップ 2 と 4 は $(i,j) \in \mathcal{E}$ を添字とする大量のデータに対する同一の演算である。そこで，まず，ステップ 1，2 および 4 の演算をベクトル化することを考える。

ステップ 1 は，添字 $j \in \mathcal{V}_2$ について独立な演算である。しかも，大規模な問題においては $|\mathcal{V}_2|$ の値が十分に大きいと考えられるため，添字 j に関する繰返し処理のベクトル化は非常に効果的である。なお，ベクトル化の対象データがメモリの連続した領域に存在すれば，ベクトル処理はより高速に実行される。そこで，枝 $(i,j) \in \mathcal{E}$ に関する情報 $x_{ij}^{(k)}$ と $v_{ij}^{(k)}$ は，j を第 1 添字とする 2 次元配列に格納する。ただし，現実の大規模問題においてグラフ $\mathcal{G}$ は構造的に疎であると考えられるため，配列のサイズは $|\mathcal{V}_2| \times \max_{j \in \mathcal{V}_2} \xi_j$ とし，その第 2 添字は節点 $j \in \mathcal{V}_2$ に接続される枝に付した連番とする。なお，節点 $j \in \mathcal{V}_2$ に接続される枝の数 ξ_j は j ごとに異なるが，第 2 添字の値が ξ_j を越える要素には常に値 0.0 をセットしておくことにする。一方，節点 $j \in \mathcal{V}_2$ に関する情報 β_j, ξ_j および $\pi_j^{(k+1)}$ は，サイズ $|\mathcal{V}_2|$ の 1 次元配列に格納する。このとき，j に関する繰返し処理を内側，総和演算を定義する繰返し処理を外側に置く多重 DO ループによりコーディングすれば，ステップ 1 の演算は $j \in \mathcal{V}_2$ について効率的にベクトル実行される。

これに伴って，$w_{ij}^{(k+1)}$ および $x_{ij}^{(k+1)}$ も $x_{ij}^{(k)}$ や $v_{ij}^{(k)}$ と同じような形式で $|\mathcal{V}_2| \times \max_{j \in \mathcal{V}_2} \xi_j$ の 2 次元配列に格納し，ステップ 2 および 4 の演算を $j \in \mathcal{V}_2$ についてベクトル実行させることにする。ただし，ステップ 2 においては，$w_{ij}^{(k+1)}$ を格納する 2 次元配列の第 2 添字が ξ_j を越える要素に $-\rho\pi_j^{(k+1)}$ がセットされないようにコーディングする必要がある。そこで，その配列要素が演算対象であるかどうかを示す同じサイズの論理配列を用意して，対応する要素の真偽を条件とする IF 文を DO ループの実行文に付加する。このようなマ

スク配列の評価は，比較的高速に実行される。また，既に述べたように，IF 文が含まれるDO ループもコンパイラによって自動的にベクトル化される。

つぎに，**ステップ 3** について考える。**ステップ 3** は並列化の対象であるが，独立に処理できる部分問題の数 $|\mathcal{V}_1|$ に比べて，VPP500 において使用できるプロセッサの数は極めて少ない。よって，各プロセッサが担当する部分問題に対する処理を効率的にベクトル化しない限り，十分な性能が得られないと考えられる。ところが，グラフ $\mathcal{G}$ が構造的に疎である場合には，ひとつひとつの部分問題の規模は必ずしも大きくない。それゆえ，部分問題を解く手続きの各ステップに現れる演算を変数の添字 j についてベクトル化しても，処理の高速化は期待できないことがわかる。

そこで，部分問題を解く手続きそのものの，節点 $i \in \mathcal{V}_1$ に対する繰返し処理をベクトル化することを考える。そのためには，手続き内部の繰返し回数を，すべての $i \in \mathcal{V}_1$ について同じ値にそろえる必要がある。そこで，i を添字とする 1 次元の論理配列を用意して，初期値としてすべての要素に“真”を与える。そして，節点 i に対する部分問題の最適解が見つかれば，対応する要素を“偽”に変更する。それぞれの部分問題を解く手続きに含まれるすべての実行文を，このような論理配列の対応する要素を条件とするブロック IF 文の中に置くならば，手続き内部の繰返し回数を部分問題の変数の数が $\max_{i \in \mathcal{V}_1} \zeta_i$ である場合に想定される最悪の繰返し回数にそろえることができる。このとき，部分問題を解く手続きの各ステップを構成する実行文を，それぞれ i を制御変数とする DO ループで囲むことにより，**アルゴリズム 2 のステップ 3** が添字 $i \in \mathcal{V}_1$ についてベクトル化されることになる。とくに，**手続き 4** は繰返し回数の定まった簡単な二重 DO ループで記述できるため，ベクトル化のためのコーディングの変更は容易である。

ベクトル処理をより効率的に実行するために，部分問題を解く際に必要な枝 $(i,j) \in \mathcal{E}$ に関する情報 $\bar{c}_{ij}^{(k+1)}$ と $\bar{d}_{ij}$ は，i を第 1 添字とする 2 次元配列に格納する。ただし，配列のサイズは $|\mathcal{V}_1| \times \max_{i \in \mathcal{V}_1} \zeta_i$ とし，その第 2 添字は節点 $i \in \mathcal{V}_1$ に接続される枝に付した連番とする。これに伴って，部分問題の解 $x_{ij}^{(k+1)}$ も同じような形式で $|\mathcal{V}_1| \times \max_{i \in \mathcal{V}_1} \zeta_i$ の 2 次元配列に格納されるが $x_{ij}^{(k+1)}$ は**ステップ 4** において j を第 1 添字とする $|\mathcal{V}_2| \times \max_{j \in \mathcal{V}_2} \xi_j$ の 2 次元配列に保持していなければならない。そこで，これら 2 種類の 2 次元配列の対応する要素を指示するポインタをあらかじめ用意しておき，**ステップ 4** の実行前に $x_{ij}^{(k+1)}$ の値を転送することにする。一方，**ステップ 3** の実行前には，j を第 1 添字とする $|\mathcal{V}_2| \times \max_{j \in \mathcal{V}_2} \xi_j$ の 2 次元配列に保持された $x_{ij}^{(k)}$ と $w_{ij}^{(k+1)}$ から $\bar{c}_{ij}^{(k+1)}$ を計算し，同じポインタを

利用して i を第1添字とする $|\mathcal{V}_1| \times \max_{i \in \mathcal{V}_1} \zeta_i$ の2次元配列に転送することにする。

これによって，アルゴリズム2の各ステップを構成するすべての実行文が，添字 $i \in \mathcal{V}_1$ または $j \in \mathcal{V}_2$ のいずれかについてベクトル化される。なお，VPP500 のコンパイラは，ステップ2や4に現れる $(i,j) \in \mathcal{E}$ について独立な演算を記述する二重 DO ループを，ベクトル長を長くする目的で強制的に一重化してしまう。しかし，現実の大規模問題においては $|\mathcal{V}_2|$ のみで十分大きな値となるため，予め外側ループをアンローリングするコーディングを行い，ループの一重化を抑止している。その他の多重 DO ループにおいては，ベクトル化される最内ループのすぐ外側の DO ループをアンローリングする最適化がコンパイラによって自動的に行われる。

3.3 並列化

アルゴリズム2において処理を並列化するのはステップ3だけであるため，その前後にそれぞれ並列化指示行 !XOCL PARALLEL REGION と !XOCL END PARALLEL を置くならば，並列処理に使用するプロセッサの起動・停止と必要なデータの転送を自動的に行うことができる。しかしながら，負荷の大きい PARALLEL REGION 文を反復ごとに実行すると，処理時間の著しい増加を招いてしまう。そこで，アルゴリズム全体を PARALLEL REGION 内に置き，ステップ3以外のステップにおいては，すべてのプロセッサ上で同じ演算を実行することにした。そのためには，演算に必要なデータを各プロセッサのローカルメモリに重複して保持しなければならないが，通常の FORTRAN 77 と全く同じように変数の宣言を行うだけでこれを実現することができる。また，PARALLEL REGION 内に並列化指示行を付加せずに記述した文はすべてのプロセッサ上で同時に実行されるため，ステップ3以外のコーディングは全く変更する必要がない。

一方，ステップ3は節点 $i \in \mathcal{V}_1$ について並列的に処理できるため，集合 $\mathcal{V}_1$ の要素を各プロセッサに分配することにより並列化を実現する。いま，$\mathcal{V}_{1\ell}, \ell = 1, \ldots, p$, を条件

$$\bigcup_{\ell=1}^{p} \mathcal{V}_{1\ell} = \mathcal{V}_1 \quad \text{かつ} \quad \mathcal{V}_{1\ell} \cap \mathcal{V}_{1\ell'} = \emptyset, \quad \ell \neq \ell',$$

を満たす集合 $\mathcal{V}_1$ の分割とする。このとき，ℓ を制御変数とする DO ループの実行文として $\mathcal{V}_{1\ell}$ のすべての要素に対する部分問題を解く手続きを記述し，この DO ループの前後にそれぞれ並列化指示行 !XOCL SPREAD DO および !XOCL END SPREAD を付加すれば，ステップ3を p 個のプロセッサを用いて並列的に処理することができる。なお，3.2節で述

べたステップ3の部分問題を解く手続きのベクトル化方法は，対象となる節点集合 $\mathcal{V}_1$ を $\mathcal{V}_{1\ell}$ に置き換えても全く同様である。また，$|\mathcal{V}_{11}| \cong |\mathcal{V}_{12}| \cong \cdots \cong |\mathcal{V}_{1p}|$ となるように分割を選べば，各プロセッサの負荷はほぼ均等となる。このとき，プロセッサ ℓ におけるベクトル化対象データの数は $|\mathcal{V}_1|$ から $|\mathcal{V}_{1\ell}| \cong |\mathcal{V}_1|/p$ に減少するが，VPP500 において使用可能なプロセッサの数はたかだか10程度であるため，もとの問題の規模が十分大きい場合には，ベクトル長の減少による処理効率の低下はほとんどないと考えられる。

ステップ3の並列化実行が終了したとき，プロセッサ ℓ は解 $x_{ij}^{(k+1)}$, $(i,j) \in \mathcal{E}$ ，を格納すべき2次元配列の添字 $i \in \mathcal{V}_{1\ell}$ に対する要素にのみ正しい値を保持している。それゆえ，すべてのプロセッサにおいてステップ4を同時に実行するためには，各要素の正しい値を保持しているプロセッサが他のプロセッサにその値を転送しなければならない。そこで，ステップ4の前に $x_{ij}^{(k+1)}$, $(i,j) \in \mathcal{E}$, を格納する2次元配列を引数とする並列化指示行 !XOCL UNIFY を置く。実際のコーディングでは，データの転送時間を削減するため，部分問題の最適ラグランジュ乗数 $u_i^{(k+1)}$ を格納する1次元配列を UNIFY 文により転送し，すべてのプロセッサが式（2.16）にもとづいて最適解 $x_{ij}^{(k+1)}$ を計算している。

結局，ベクトル化を施したコードにおいて，ステップ3の部分問題を解く手続きそのものに対する繰返し処理を定義する DO ループを二重化し，並列化指示行を7行追加するだけで，並列処理を実行するコードを得ることができる。

4．数値実験

2次輸送問題に対する交互方向乗数法の性能を検証するために，京都大学大型計算機センターに導入されているベクトル並列計算機 VPP500 を用いてアルゴリズム2の数値実験を行った。ここではまず，数値実験に用いたテスト問題の特徴を説明し，つぎに実験の結果を述べることにする。

4.1 テスト問題

数値実験は，ランダムに生成された分離可能な2次輸送問題（2.14）に対して行った。大規模な問題に対する数値実験を効率よく進めるために，文献［Eckstein and Fukushima (1994)］と同様の方法により，テスト問題そのものを VPP500 上でベクトル処理を用いて生成している。まず，2部グラフ $\mathcal{G}$ については，$2\max\{|\mathcal{V}_1|, |\mathcal{V}_2|\}$ 本の枝を規則的に生成した後，枝の総数が $|\mathcal{E}|$ 本になるまで枝をランダムに追加する。具体的には，始点と終点

をそれぞれ節点集合$\mathcal{V}_1, \mathcal{V}_2$から順番に（要素数の少ない集合については要素番号を循環させて）選ぶことによって$\max\{|\mathcal{V}_1|, |\mathcal{V}_2|\}$本の枝を生成する。そして，始点と終点の組合せを少しずらせて，さらに$\max\{|\mathcal{V}_1|, |\mathcal{V}_2|\}$本の枝を同様に生成する。残りの枝は，始点と終点をそれぞれ節点集合$\mathcal{V}_1, \mathcal{V}_2$からランダムに選ぶことによって生成するが，$\max\{|\mathcal{V}_1|, |\mathcal{V}_2|\}$本の枝を生成するごとに，同じ始点と終点の組合せをもつ枝を探索して削除するようにしている。なお，生成した枝の情報は，始点の番号ごとに固めて格納する。このとき，始点番号の小さいものから順に枝を取出して終点番号ごとに整理しなおす処理を実行すれば，重複して生成された枝を容易に検出することができる。

一方，目的関数の係数c_{ij}およびd_{ij}は，それぞれ区間$[0, 100]$，$[0.1, 1.0]$から一様に選んだ。さらに，変数x_{ij}の値を区間$[0, 100]$からランダムに選び，これらの値に対してすべての制約条件がなりたつように，制約条件の右辺定数α_iおよびβ_jの値を決定した。

すべてのテスト問題において$|\mathcal{V}_1| = |\mathcal{V}_2|$とし，節点の平均次数は8および16の2種類に限定した。そして，それぞれのタイプについて，枝の本数$|\mathcal{E}|$を16384から524288まで6段階に変化させてテスト問題を生成し，実験を行っている。なお，使用した計算機におけるメモリの制約から，枝の本数をこれ以上増やすことはできなかった。ところで，枝の本数は，問題（2.14）における変数の数に対応する。一方，等式制約条件の数は$|\mathcal{V}_1| + |\mathcal{V}_2|$であるが，節点の平均次数が8の問題で変数の数の1/4，次数が16の問題で変数の数の1/8となる。数値実験においては，それぞれのテスト問題について乱数の種類を変えて5題の問題例を生成し，それらの平均をもって実験結果とした。

4.2 実験結果

数値実験は，すべて倍精度で行った。また，文献[Eckstein and Fukushima (1994)]に従って，パラメータρの値を1.6，tの値を

$$t = \left(\frac{1}{50}\right)\left(\frac{(|\mathcal{V}_1| + |\mathcal{V}_2|)\max_{(i,j)\in\mathcal{E}}\{c_{ij}\}}{|\mathcal{E}|}\right)$$

に選んだ。係数c_{ij}の選び方により，節点の平均次数が8の問題で$t \cong 0.5$，次数が16の問題で$t \cong 0.25$程度になる。一般の非線形計画問題に対する乗数法では，拡張ラグランジュ関数を局所凸化するためにパラメータtの値を十分大きくとらなければならないが，凸計画問題に対する交互方向乗数法においては拡張ラグランジュ関数がもともと凸であり，t

の値を大きくとる必要はない。実際，パラメータ t の値が大きすぎたり小さすぎたりした場合には，交互方向乗数法の収束が非常に遅くなることが示されている［Fukushima (1992)］。

アルゴリズム2のステップ1において，初期探索点は $x^{(0)}=0$, $v^{(0)}=0$ に選んだ。テスト問題として生成される2部グラフの各節点の平均次数は8または16であるため，ステップ3で解くべき各部分問題の変数の数は，枝の生成に極端な偏りがない限り十分小さい値にとどまる。そこで，部分問題の解法として手続き4を用いた。一方，ステップ5における収束判定条件は，

$$|x_{ij}^{(k+1)}-x_{ij}^{(k)}|\leq 10^{-6}, \qquad (i,j)\in\mathcal{E}, \tag{4.1}$$

$$|\sum_{i:(i,j)\in\mathcal{E}} x_{ij}^{(k+1)}-\beta_j|\leq 10^{-6}\max_{j\in\mathcal{V}_2}\beta_j, \quad j\in\mathcal{V}_2, \tag{4.2}$$

とした。ステップ3の部分問題の定義より，$x^{(k+1)}$ はもとの問題（2.14）の $i\in\mathcal{V}_1$ に関する等式制約条件と非負条件を常に満たすから，$x^{(k+1)}$ 実行可能性の判定は条件（4.2）だけで十分である。なお，条件（4.2）の左辺絶対値内部の値は，次の反復のステップ1において再度利用することができる。ただし，収束判定による計算負荷を削減するため，条件（4.1）が成立した場合にのみ条件（4.2）の判定を行うことにしている。

さまざまなサイズの2次輸送問題に対してアルゴリズム2の性能を測定した結果を，表1に示す。表の“逐次処理”の欄は，VPP500のスカラー演算装置を一つだけ用いて，アルゴリズム2のすべてのステップを逐次的に処理した場合の結果である。一方，“ベクトル処理”の欄は，VPP500のベクトル演算装置を一つだけ用いて，アルゴリズム2の各ステップにおいて独立に実行可能な計算を3.2節に示した方法でベクトル処理した場合の結果である。これに対して，“ベクトル並列処理”の欄は，3.3節に示した方法により，VPP500の p 個のプロセッサにステップ3で解くべき $|\mathcal{V}_1|$ 個の部分問題を均等に分配して並列的に処理した場合の結果である。なお，各プロセッサは，“ベクトル処理”の場合と同じような方針で，独立に実行可能な計算をベクトル処理している。

並列アルゴリズムの計算効率を表す指標としてよく用いられるものに，つぎのように定義される加速率 S_p と効率 e_p がある［Bertsekas and Tsitsiklis (1989)］。

$$S_p=\frac{\text{1個のプロセッサによる計算時間}}{\text{}p\text{個のプロセッサによる計算時間}}, \qquad e_p=\frac{S_p}{p}, \qquad p\geq 2.$$

表１：２次輸送問題に対する交互方向乗数法の数値実験結果

問題のサイズ			反復回数	計算時間［秒］				
				逐次処理	ベクトル処理†	ベクトル並列処理		
$\|\mathcal{V}_1\|$	$\|\mathcal{V}_2\|$	$\|\mathcal{E}\|$				$p=2$	$p=4$	$p=8$
2048	2048	16384	53	18.675	0.480(0.136)	0.342	0.280	0.252
4096	4096	32768	59	59.768	1.196(0.332)	0.817	0.664	0.590
8192	8192	65536	66	191.782	2.767(0.764)	1.951	1.500	1.325
16384	16384	131072	67	449.131	5.738(1.586)	4.066	3.179	2.712
32768	32768	262144	74	1389.712	13.914(3.629)	9.734	7.490	6.376
65536	65536	524288	110	4182.433	41.567(10.643)	29.021	22.278	18.912
1024	1024	16384	52	19.893	0.563(0.124)	0.386	0.303	0.262
2048	2048	32768	55	54.526	1.271(0.258)	0.808	0.614	0.520
4096	4096	65536	56	148.537	2.751(0.576)	1.704	1.290	1.088
8192	8192	131072	61	459.121	6.476(1.236)	4.162	2.881	2.388
16384	16384	262144	71	1227.056	15.926(2.982)	10.248	7.237	5.641
32768	32768	524288	85	3809.895	37.371(7.196)	24.274	17.275	13.772

† ( )内は，アルゴリズム２のステップ３以外の部分の実行に要した計算時間である。

しかしながら，ベクトル並列計算機 VPP500 においては，１個のプロセッサのみで実行しても，ベクトル演算装置を利用すれば独立に実行可能な計算がパイプラインを用いて非常に高速に処理される。そこで，速度向上率

$$\widehat{S}_p = \begin{cases} \dfrac{\text{逐次処理による計算時間}}{\text{ベクトル処理による計算時間}}, & p=1, \\ \dfrac{\text{逐次処理による計算時間}}{p\text{個のプロセッサを用いたベクトル並列処理による計算時間}}, & p \geq 2, \end{cases}$$

をアルゴリズム２の性能指標と考えることにする。表１の結果をもとに速度向上率を計算した結果を，表２に示す。表２より，ベクトル化によって顕著な速度向上が得られていることがわかる。また，問題サイズが大きくなるにつれて，速度向上率はさらに増大している。これは，交互方向乗数法のアルゴリズムが本質的に粒度の細かい並列アルゴリズムであり，ベクトル化になじみやすいものであることを示している。

使用するプロセッサ数 p を増やした場合も速度向上率は増加しているが，その増加率は必ずしも十分とは言えない。これは，並列処理の対象とした部分が，アルゴリズム２のステップ３のみであったことと，複数のプロセッサ上で分担して処理した結果を UNIFY 文

表２：２次輸送問題に対する交互方向乗数法の速度向上率

問題のサイズ			計算時間［秒］			
			ベクトル処理	ベクトル並列処理		
$\|\mathcal{V}_1\|$	$\|\mathcal{V}_2\|$	$\|\mathcal{E}\|$		$p=2$	$p=4$	$p=8$
2048	2048	16384	38.9	54.6	66.7	74.1
4096	4096	32768	50.0	73.2	90.0	101.3
8192	8192	65536	69.3	98.3	127.9	144.7
16384	16384	131072	78.3	110.5	141.3	165.6
32768	32768	262144	99.9	142.8	193.3	218.0
65536	65536	524288	100.6	144.1	187.7	221.2
1024	1024	16384	35.3	51.5	65.7	75.9
2048	2048	32768	42.9	67.5	88.8	104.9
4096	4096	65536	54.0	87.2	115.1	136.5
8192	8192	131072	70.9	110.3	159.4	192.3
16384	16384	262144	77.7	119.8	169.6	217.5
32768	32768	524288	101.9	197.0	220.5	276.6

によりすべてのプロセッサに転送する通信のオーバヘッドによるものと考えられる。なお，並列処理の対象とならなかった部分の処理時間は，使用したプロセッサ数に関係なく一定であり，表１の“ベクトル処理”の欄に括弧つきで示している。この処理時間を取り除いたとき，すなわち，ステップ３のみにおける並列化の効率を計算すると，$p=2$ で0.85前後，$p=4$ で 0.7 前後，$p=8$ で 0.5 前後であることがわかる。

文献［Eckstein and Fukushima（1994)］には，交互方向乗数法のアルゴリズムをコネクションマシン CM-5 上で同じようなテスト問題に対して実行した数値実験結果が示されている．それによると，ピークで 32 MFLOPS の性能をもつベクトル演算装置256個から成る CM-5 上で，$|\mathcal{V}_1|=|\mathcal{V}_2|=32768$，$|\mathcal{E}|=262144$ の問題を解くために25.31秒を要している。単純に計算すると，この CM-5 の性能は５プロセッサから成る VPP500 の性能と同等である。ところが，本稿の数値実験では，VPP500 の４プロセッサを用いたベクトル並列処理により，同じサイズの問題がわずか7.49秒で解けるなど，極めて高い性能を示している。これは，つぎのような理由によるものと考えられる。文献［Eckstein and Fukushima（1994)］では，すべての情報を枝数と同じサイズをもつ１次元配列に保持し，その要素を各プロセッサに均等に分配している。そして，$\pi_j^{(k+1)}$ のように一方の節点に関する添字だけをもつ情報を生成する場合には，“segmented scan”［Zenios and Censor(1991)］と呼

ばれる処理を用いて配列要素の部分和を計算し，添字が同じ値の要素へ結果を転送している。また，各反復で部分問題（2.15）を解く際は，係数 $\bar{c}_{ih}^{(k+1)}$ を $i \in \mathcal{V}_1$ について完全にソートした後，すべての情報をこのソート結果に対応して並べ換えるという処理を行っている。それゆえ，同じ $i \in \mathcal{V}_1$ に対する情報を固めて保持しても，これらの処理を行うためにはある程度のプロセッサ間通信が必要になる。一方，本稿の実行方法では，$\pi_j^{(k+1)}$ はそれぞれのプロセッサにおいてベクトル処理を用いて計算される。また，部分問題（2.15）を解く際も，同じプロセッサに保持された $\bar{c}_{ih}^{(k+1)}$ と $\bar{d}_{ih}$ を部分的に整列するだけであり，プロセッサ間通信を行う必要がない。さらに，ベクトルアンローリングなどのベクトル化チューニングを施したことによって，良好な性能が得られたものと考えられる。

5．おわりに

本稿では，分離可能な2次のコストをもつ輸送問題に対する交互方向乗数法をベクトル並列計算機 VPP500 上で実行する方法を示した。とくに，交互方向乗数法が粒度の細かい並列アルゴリズムであることに注意して，ベクトル化が効果的に行われるように処理手順やデータの格納方法を検討した。一方，複数のプロセッサを用いた処理の並列化については，プロセッサ間通信の発生による性能の低下を最小限にとどめるために，比較的まとまった処理に対して補完的に導入するにとどめた。そして，数値実験により，提案した実行方法が大規模な問題を効率的に処理できることを確認した。

ここで提案した実行方法では，アルゴリズムの実行に必要なすべての情報を，使用するそれぞれのプロセッサに重複して持たせている。しかしながら，より規模の大きな問題を解くためには，各プロセッサに情報を分割して並列処理の対象範囲を拡大する必要がある。このとき，通信によるオーバヘッドを抑えながら効率的に並列化を行わなければならないが，その具体的な方法の開発は今後の研究課題である。

謝辞

日頃よりご指導いただいている京都大学大学院情報学研究科の福島雅夫教授に感謝致します。なお，本研究の一部は，京都大学大型計算機センター開発計画によるものである。

参考文献

[1] D. P. Bertsekas：Constrained Optimization and Lagrange Multiplier Methods, Academic Press, New York, 1982.

[2] D. P. Bertsekas and J. N. Tsitsiklis：Parallel and Distributed Computation; Numerical Methods, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[3] Y. Censor and S. A. Zenios：Parallel Optimization; Theory, Algorithm and Applications, Oxford University Press, New York, 1997.

[4] J. Eckstein：“The Alternating Step Method for Monotropic Programming on the Connection Machine CM-2”, ORSA Journal on Computing, Vol. 5 (1993), pp. 84－96.

[5] J. Eckstein and M. Fukushima：“Some Reformulations and Applications of the Alternating Direction Method of Multipliers”, in Large Scale Optimization: State of the Art (eds. W. W. Hager, D. W. Hearn and P. M. Pardalos), Kluwer Academic Publishers, 1994, pp. 115－134.

[6] 富士通株式会社：UXP/V VP プログラミングハンドブック, V10用, 1997.

[7] 富士通株式会社：UXP/V VPP プログラミングハンドブック, V10用, 1997.

[8] M. Fukushima：“Application of the Alternating Direction Method of Multipliers to Separable Convex Programming Problems”, Computational Optimization and Applications, Vol. 1 (1992), pp. 93－112.

[9] 福島雅夫：“数理計画問題に対する並列アルゴリズム,” 第5回 RAMP シンポジウム論文集, 1993, pp. 15－28.

[10] 平野彰雄：“MSP ユーザのための VPP 入門,” 京都大学大型計算機センター広報, Vol. 28 (1995), pp. 63－75.

[11] 茨木俊秀：アルゴリズムとデータ構造, 昭晃堂, 1989.

[12] 一森哲男：数理計画法－最適化の手法－, 共立出版, 1994.

[13] 今野浩, 山下浩：非線形計画法, 日科技連, 1978.

[14] R. T. Rockafellar：Convex Analysis, Princeton University Press, Princeton, New Jersey, 1970.

[15] 山川栄樹：“並列最適化に関する最近の研究から,” 第7回 RAMP シンポジウム論文集, 1995, pp. 181－196.

[16] S. A. Zenios and Y. Censor : "Massively Parallel Row-Action Algorithms for Some Nonlinear Transportation Problems," SIAM Journal on Optimization, Vol. 1 (1991), pp. 373－400.

Implementing and Running the Alternating Direction Method of Multipliers for Quadratic Transportation Problems on a Vector Parallel Computer

Eiki Yamakawa

Abstract

The alternating direction method of multipliers is one of the sophisticated parallel algorithms for mathematical programming problems with some network structure. This paper presents implementation strategies of a special reformulation of the method for quadratic transportation problems on a vector parallel computer VPP500. Performance of the method is examined through extensive computational experiments.

高 松 大 学 紀 要

第 30 号

平成10年10月28日　印刷
平成10年10月30日　発行

編集発行

高　　松　　大　　学
高 松 短 期 大 学
〒761-0194　高松市春日町960番地
TEL（087）841－3255
FAX（087）841－3064