TR－H－090

並列計算機 CM-5 を用いた 逐次線形化法に対する数値実験

山川 栄樹
阿部 敦（奈良先端科技院大）

1994. 8. 15

ATR人間情報通信研究所

〒619-02 京都府相楽郡精華町光台2-2　☎07749-5-1011

ATR Human Information Processing Research Laboratories

2-2, Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02 Japan

Telephone: +81-7749-5-1011
Facsimile: +81-7749-5-1008

並列計算機 CM-5 を用いた逐次線形化法に対する数値実験[1]

山川 栄樹
株式会社 エイ・ティ・アール人間情報通信研究所 第六研究室

阿部 敦
奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 システム基礎講座

1994 年 8 月 15 日

1 はじめに

実際に現われる大規模非線形計画問題では、制約条件の大半が線形関数であり、その係数行列が疎であるという性質を持つ場合が多い。逐次線形化法 (Successive Linearization Method) は、反復計算の過程において原問題の疎な性質を保持することによって、現実の大規模問題を効率的に取扱おうとするアルゴリズムである [3] 。逐次線形化法のアルゴリズムは、各反復で目的関数, 制約条件の双方を線形近似することによって得られる部分問題を解く。ただし、大域的収束性を保証するために、部分問題の目的関数に凸 2 次項を付加することによって、ステップサイズのコントロールを行う。その際、2 次項を分離可能となるように構成すれば、得られた 2 次計画部分問題は、係数行列の各行ベクトルに対する簡単な演算のみで構成される所謂Row-Action Algorithm を少し修正することにより、効率的に解くことができる。

本論文の目的は、逐次線形化法の部分問題を解くために最も基本的な並列アルゴリズムの一つである射影 Jacobi 法を用い、これを SIMD (Single Instruction-Multiple Data) 型の並列計算機上に実現して、逐次線形化法に対する並列アルゴリズム適用の有効性を数値実験を通して検証することである。射影 Jacobi 法自体は、射影 SOR (Successive Over Relaxation) 法とともに線形相補性問題等に対する Row-Action Algorithm の一つとして、古くから研究されている [2] 。逐次計算機上での実現を考える場合には、本質的に逐次的な手続きによって構成される射影 SOR 法に比べて、射影 Jacobi 法は必ずしも優れた方法ではない。しかし、細粒度型の大規模並列計算機を用いる場合には、並列化度合が増すにつれて並列アルゴリズム本来の有効性を発揮できるものと期待される。勿論、実際に十分な並列化効率を引出す為には、係数行列の大半の要素が零であるという現実の大規模問題特有の性質を十分に考慮して、プロセッサの割当てやデータの配置を行うことが必須条件となる。

以下では、先ず第 2 章において逐次線形化法のアルゴリズムとその性質について簡単に紹介する。次に第 3 章において、2 次計画部分問題を解くための射影 Jacobi 法について述べる。第 4 章では、射影 Jacobi 法を並列計算機上に実現するに際して、原問題の疎な性質をどのように利用するかを中心に考える。テスト問題の生成方法と、並列計算機 Connection Machine Model CM-5 による数値実験結果については、第 5 章にまとめる。最後に第 6 章において、数値実験を通じて明らかになった問題点についての考察を行う。

[1]本論文は、筆者の一人 (阿部) が学外実習生としてエイ・ティ・アール人間情報通信研究所に滞在中 (1993.12.14-1994.3.31) に得た結果を基に、筆者のもう一人 (山川) が更に詳細な数値実験を加えた上でまとめたものである。

2 逐次線形化法のアルゴリズムとその性質

本論文では次のような非線形計画問題について考える。

$$\begin{aligned} \text{NLP:}\quad & \text{minimize} \quad f(x), \\ & \text{subject to} \quad g_i(x) = 0, \quad i = 1, \cdots, \overline{m}, \\ & \qquad\qquad\ \ g_i(x) \leq 0, \quad i = \overline{m}+1, \cdots, m. \end{aligned}$$

ただし、x は n 次元の実数ベクトルであり、f および $g_i, i = 1, \cdots, m$ は R^n で定義された連続微分可能な実数値関数とする。以下では、各制約条件に対する Lagrange 乗数を $u_i, i = 1, \cdots, m$ で表し、問題 NLP に対する Lagrange 関数を

$$L(x, u) = f(x) + \sum_{i=1}^{m} u_i\, g_i(x) \tag{1}$$

と書くことにする。

問題 NLP に対する正確な L_1 ペナルティ関数は

$$p(x) \equiv f(x) + \sum_{i=1}^{\overline{m}} r_i\, |\, g_i(x)\, | + \sum_{i=\overline{m}+1}^{m} r_i \max\{\, 0,\, g_i(x)\,\} \tag{2}$$

で定義される [4]。ここに、$r_i > 0, i = 1, \cdots, m$ はペナルティ・パラメータである。式 (2) の右辺第 1 項は問題 NLP の目的関数そのものである。一方、その第 2 項，第 3 項は、それぞれ問題 NLP の等式制約，不等式制約に対応しており、x がこれらすべての条件を満足していればその値は 0 であるが、実行可能領域から離れるにつれて次第に大きな値をとるようになる。従って、ペナルティ関数 $p(x)$ は、問題 NLP の目的関数値の減少と実行可能性の度合とを総合的に判断するための指標と考えることができ、その性質は以前からよく研究されている。

定理 1 [4] (x^*, u^*) を、最適性の 2 次の十分条件を満たす問題 NLP の局所最適解と、対応する Lagrange 乗数ベクトルの組とする。この時、$r_i \geq |u_i^*|, i = 1, \cdots, m$ ならば、x^* は $p(x)$ の局所最小点となる。

定理 1 は、問題 NLP の解が $p(x)$ の制約なし最小点となるための必要条件を与えているに過ぎない。しかし、問題 NLP が凸計画問題の場合には、$p(x)$ の局所最小点が問題 NLP の最適解となるための十分条件も知られている。

定理 2 [6] 問題 NLP は凸計画問題で有限な解を持ち、緩和された Slater の制約想定

$$\{\, x \mid g_i(x) < 0, i \in I_1;\ g_i(x) \leq 0, i \in I_2 \,\} \neq \phi$$

を満足すると仮定する。ここに、

$$I_1 \equiv \{1, \cdots, m\} \setminus I_2, \quad I_2 \equiv \{\, i \mid g_i(x) : \text{線形}, i = 1, \cdots, m\}$$

である。問題 NLP のある最適 Lagrange 乗数を u^* とする時、ある $r_i > |u_i^*|, i = 1, \cdots, m$ に対して x^* が $p(x)$ の局所最小点ならば、x^* は問題 NLP の最適解である。

これらの定理から、問題 NLP に有限な解が存在し、対応する Lagrange 乗数ベクトルも存在するならば、十分大きな有限の r に対して $p(x)$ の制約なし最小化問題を解くことにより、原問題 NLP の局所最適解が得られるものと期待できる。

ペナルティ関数 $p(x)$ には微分不可能な点が存在し、しかも非線形であるために、これを直接最小化することは容易ではない。そこで、現在の探索点 x^k において、$p(x^k+d) < p(x^k)$ なるステップ d を逐次求める反復解法を考える。まず、$p(x)$ を構成する f および g_i を線形近似することによって得られる関数

$$\bar{p}(x,d) \equiv f(x) + \nabla f(x)^T d \ + \sum_{i=1}^{\overline{m}} r_i \, | \, g_i(x) + \nabla g_i(x)^T d \, | \\ + \sum_{i=\overline{m}+1}^{m} r_i \max\left\{ 0,\, g_i(x) + \nabla g_i(x)^T d \right\}$$

を、ペナルティ関数 $p(x)$ の線形近似関数と考えよう。逐次線形化法 [3] は、現在の探索点 x^k において、関数 $\bar{p}(x^k,d)$ をステップサイズ d を制限するための凸2次項と共に最小化する問題

$$\mathrm{SP}(x^k, C^k, \lambda_k): \quad \text{minimize} \quad \frac{\lambda_k}{2} d^T C^k d + \bar{p}(x^k, d)$$

を解き、点列 $\{x^k\}$ を得ようとする方法である。ここに、$\lambda_k > 0$ は、ステップサイズをコントロールするためのパラメータであり、C^k は正定値対称行列にとる。問題 SP の目的関数は狭義凸関数であり、唯一の解 d^k を持つが、第2項に微分不可能な点が存在し、そのままでは最小化し難いように見える。しかし、人為変数

$$\begin{aligned} t_i &\equiv \max\left\{ 0,\, g_i(x^k) + \nabla g_i(x^k)^T d \right\}, \quad i = 1, \cdots, m, \\ w_i &\equiv \max\left\{ 0,\, -g_i(x^k) - \nabla g_i(x^k)^T d \right\}, \quad i = 1, \cdots, m \end{aligned}$$

を導入することにより、問題 SP は次のような等価な2次計画問題に書き直すことができる。

$$\begin{aligned} \text{minimize} \quad & \frac{\lambda_k}{2} d^T C^k d + \nabla f(x^k)^T d + \sum_{i=1}^{\overline{m}} r_i \, (t_i + w_i) + \sum_{i=\overline{m}+1}^{m} r_i \, t_i, & \\ \text{subject to} \quad & g_i(x^k) + \nabla g_i(x^k)^T d - t_i + w_i = 0, & i = 1, \cdots, m, \\ & t_i \geq 0, \quad w_i \geq 0, & i = 1, \cdots, m. \end{aligned} \tag{3}$$

原問題 NLP が構造的に疎な場合、C^k として正定値対角行列を採用すれば、問題(3)においても疎な構造が反復を通して維持されるため、効率的な解法が構成できる。

人為変数 t_i, w_i の導入によって、各反復で解くべき部分問題(3)の規模は $2m+n$ 変数, $3m$ 制約となり、原問題に比べてかなり大きくなってしまう。しかし、凸2次計画問題(3)に対する Lagrange の双対問題を考えると、それは m 個の双対変数に対する上下限制約のみを持つ次のような凸2次計画問題となる。

$$\begin{aligned} \mathrm{DSP}(x^k, C^k, \lambda_k): \quad \text{minimize} \quad & \phi^k(u) \\ \text{subject to} \quad & -r_i \leq u_i \leq r_i, \quad i = 1, \cdots, \overline{m}, \\ & 0 \leq u_i \leq r_i, \quad i = \overline{m}+1, \cdots, m. \end{aligned}$$

ここに、

$$\phi^k(u) = \frac{1}{2\lambda_k} \nabla L(x^k, u)^T \left(C^k \right)^{-1} \nabla L(x^k, u) - \sum_{i=1}^{m} u_i \, g_i(x^k) \tag{4}$$

であり、主問題(3)すなわち問題 SP の解 d^k は、双対問題 DSP の解を u^k とすると、

$$d^k = -\frac{1}{\lambda_k} \left(C^k \right)^{-1} \nabla L(x^k, u^k) \tag{5}$$

により求められる。

逐次線形化法のアルゴリズムは、次のように書き下すことができる。

アルゴリズム SLM

ステップ 1 パラメータ：$0 < \mu_0 < \mu_1 < \mu_2 < 1, \gamma > 1$ をセットする。初期探索点：$x^1 \in R^n$, 正定値対称行列：C^1 および $\lambda_1 > 0$ を選び、$k := 1$ とおく。

ステップ 2 問題 DSP(x^k, C^k, λ_k) の解 u^k を求め、式 (5) により探索幅 d^k を得る。

ステップ 3 $p(x)$ の線形近似関数の減少量

$$\Delta\bar{p}_k := p(x^k) - \bar{p}(x^k, d^k)$$

を求める。もし、$\Delta\bar{p}_k = 0$ ならば終了。さもなければ、ペナルティ関数の実際の減少量

$$\Delta p_k := p(x^k) - p(x^k + d^k)$$

を求める。

ステップ 4 ペナルティ関数の実際の減少量と線形近似関数の減少量との比

$$\rho_k := \Delta p_k \,/\, \Delta\bar{p}_k$$

を求める。もし、$\rho_k < \mu_0$ ならば、

$$\begin{aligned} x^{k+1} &:= x^k, \\ \lambda_{k+1} &:= \lambda_k \end{aligned}$$

さもなければ、

$$\begin{aligned} x^{k+1} &:= x^k + d^k, \\ \lambda_{k+1} &:= \begin{cases} \gamma \cdot \lambda_k, & \rho_k \le \mu_1 \quad \text{のとき}, \\ \lambda_k, & \mu_1 < \rho_k \le \mu_2 \quad \text{のとき}, \\ \lambda_k/\gamma, & \rho_k > \mu_2 \quad \text{のとき} \end{cases} \end{aligned}$$

とする。また、C^{k+1} を正定値対称となるように決定する。

ステップ 5 $k := k + 1$ とし、ステップ 2 に戻る。

ところで、定理 1, 定理 2の議論から、ペナルティ・パラメータ r_i は、条件

$$r_i > |u_i^*|, \qquad i = 1, \cdots, m$$

を満たしていなければならない。しかも、数値計算の観点からすると、その値はできる限り小さい値に止めることが望ましい。しかしながら、原問題 NLP の最適 Lagrange 乗数 u^* の値を事前に予測することは不可能と言わざるを得ない。ところが幸いなことに、アルゴリズム SLM は点列 $\{x^k\}$ のほかに、Lagrange 乗数の列 $\{u^k\}$ を生成する。そこで各反復において、ペナルティ・パラメータ r_i が、条件

$$r_i > |u_i^k|, \qquad i = 1, \cdots, m$$

を満たすよう、必要に応じて r_i の値を大きくしていくという戦略が考えられる。ただし、アルゴリズム SLM のステップ 2 において解かれる双対問題 DSP の制約条件から、

$$|u_i^k| \le r_i, \qquad i = 1, \cdots, m$$

が常に成立っていることに注意すると、実際にはアルゴリズム SLM のステップ 2 を次のように変更すればよいことがわかる。

ステップ 2-1 問題 $\mathrm{DSP}(x^k, C^k, \lambda_k)$ の解 u^k を求める。

ステップ 2-2 もし、

$$|u_i^k| < r_i, \qquad i = 1, \cdots, m$$

ならば、式 (5) により探索幅 d^k を計算し、ステップ 3 へ進む。さもなければ、

$$\begin{array}{lll} r_i & := & \nu\, r_i, \quad \text{for all} \quad i \in \left\{ i \mid r_i = |u_i^k|, i = 1, \cdots, m \right\}, \\ x^{k+1} & := & x^k, \\ \lambda_{k+1} & := & \lambda_k, \\ C^{k+1} & := & C^k \end{array}$$

としてステップ 5 へ行く。

ここに、$\nu > 1$ はペナルティ・パラメータを更新するための定数である。

アルゴリズム SLM は、次のような性質を持つことが知られている。

補題 1 [3] アルゴリズム SLM で生成された点列 $\{x^k\}$ は有界、係数行列の列 $\{C^k\}$ は有界かつ一様正定値であると仮定する。この時、次の各性質が成立つ。

1. $\Delta\bar{p}_k = 0$ ならば、またその場合に限り $d^k = 0$ である。この時 x^k は $p(x)$ の停留点となる。
2. ある $s_i^k \in [-1, 1], i = 1, \cdots, m$ が存在し、

$$\lambda_k C^k d^k + \nabla f(x^k) + \sum_{i=1}^{m} s_i^k\, r_i \nabla g_i(x^k) = 0$$

を満たす。従って、ある正の定数 τ が存在し、

$$\| d^k \| = \tau / \lambda_k, \quad \forall\, k$$

が成立つ。

3. ある $\overline{\lambda} > 0$ が存在し、$\lambda_k \geq \overline{\lambda}$ ならば

$$\rho_k = \Delta p_k \,/\, \Delta\bar{p}_k \geq \mu_1$$

を満たす。これにより、2 次項の係数の列 $\{\lambda_k\}$ は有界となる。

補題 1の 1 は、λ_k および C^k の各要素の値が正の有限な区間内に止まっているならば、$\Delta\bar{p}_k$ の相対的な大きさを停止基準として用いることができることを示している。また補題 1の 2 は、パラメータ λ_k がステップ d^k の大きさをコントロールする働きをしていることを示している。更に補題 1の 3 は、パラメータ λ_k の値を十分大きくとることによって、部分問題 SP の解 d^k がペナルティ関数 $p(x)$ の値を十分に減少させることを保証している。これらの性質により、アルゴリズム SLM は大域的収束性を持つことが示される。

定理 3 [3] アルゴリズム SLM で生成された点列 $\{x^k\}$ は有界、係数行列の列 $\{C^k\}$ は有界かつ一様正定値であると仮定する。このとき、$\{x^k\}$ の任意の集積点は $p(x)$ の停留点である。

3　2次計画部分問題を解くための射影 Jacobi 法

ここでは、部分問題 SP の 2 次の係数行列 C^k を正定値対角行列にとることにする。文献 [3] では、C^k の構造的特徴に注目し、射影 SOR 法または共役勾配法を少し修正したアルゴリズムを用いることによって、原問題 NLP の疎な性質を保持しながらアルゴリズム SLM の各反復で問題 DSP を効率的に解く方法を提案している。これに対して我々は、射影 Jacobi 法を用いて部分問題 DSP を並列的に解くことにより、より大規模な現実の非線形計画問題を高速に解くアルゴリズムを構築することを目指す。なお、以下ではアルゴリズム SLM の反復を表す添字 k と区別するために、射影 Jacobi 法の反復は括弧付の添字 (j) で表すことにする。

解くべき問題 DSP の目的関数 $\phi^k(u)$ は、式 (1), (4) より次のような u に関する 2 次関数であることがわかる。

$$\begin{aligned}\phi^k(u) &= \frac{1}{2\lambda_k}\nabla L(x^k,u)^T\left(C^k\right)^{-1}\nabla L(x^k,u)-\sum_{i=1}^{m}u_i\,g_i(x^k)\\ &= \frac{1}{2}u^TM^ku+\left(q^k\right)^Tu+z^k.\end{aligned}$$

ここに、M^k は $m\times m$ 行列, $q^k\in R^m$, $z^k\in R$ であり、それぞれ

$$\begin{aligned}M^k &= \nabla g(x^k)^T\left(\lambda_k\,C^k\right)^{-1}\nabla g(x^k),\\ q^k &= \nabla g(x^k)^T\left(\lambda_k\,C^k\right)^{-1}\nabla f(x^k)-g(x^k),\\ z^k &= \frac{1}{2}\nabla f(x^k)^T\left(\lambda_k\,C^k\right)^{-1}\nabla f(x^k)\end{aligned}\qquad(6)$$

と書き下すことができる。なお、$\nabla g(x^k)$ は $n\times m$ 行列

$$\nabla g(x^k)=\left(\nabla g_1(x^k),\nabla g_2(x^k),\cdots,\nabla g_m(x^k)\right)$$

である。今、行列 M^k に対して分割

$$M^k=B^k+(M^k-B^k)$$

を考える。問題 DSP の制約条件を満たす探索点 $u^{(j)}$ に対して、分割の第 2 項 M^k-B^k に対応して q^k を修正して得られるベクトル

$$v^{(j)}=(M^k-B^k)\,u^{(j)}+q^k$$

を用いて構成される部分問題

$$\begin{aligned}&\text{minimize} && \frac{1}{2}u^TB^ku+\left(v^{(j)}\right)^Tu\\ &\text{subject to} && -r_i\le u_i\le r_i, \qquad i=1,\cdots,\overline{m},\\ &&& 0\le u_i\le r_i, \qquad i=\overline{m}+1,\cdots,m\end{aligned}\qquad(7)$$

の解を次の探索点 $u^{(j+1)}$ とおくことによって点列 $\{u^{(j)}\}$ を生成する反復法は、問題 DSP に対する行列分割法 [2] と呼ばれる。特に行列 B^k として、M^k の対角成分を緩和パラメータ $\omega>0$ により修正した対角行列

$$B^k=\omega^{-1}\mathrm{diag}\left\{\nabla g_i(x^k)^T\left(\lambda_k\,C^k\right)^{-1}\nabla g_i(x^k)\right\}_{i\in\{1,\cdots,m\}}\qquad(8)$$

を用いる方法は射影 Jacobi 法 [2] となり、問題 (7) の解 $u^{(j+1)}$ は、

$$u_i^{(j+1)}=\begin{cases}\mathrm{mid}\{-r_i,u_i^+,r_i\}, & 1\le i\le\overline{m} \quad \text{のとき},\\ \mathrm{mid}\{\ 0,\ u_i^+,r_i\}, & \overline{m}+1\le i\le m \quad \text{のとき}\end{cases}\qquad(9)$$

により求められる。ここに、midは3数の中央値を表し、各iに対して(6)の各式と式(8)より

$$\begin{aligned} u_i^+ &= \left\{u^{(j)} - \left(B^k\right)^{-1}\left(M^k u^{(j)} + q^k\right)\right\}_i \\ &= u_i^{(j)} - \omega \cdot \frac{\nabla g_i(x^k)^T (C^k)^{-1} \nabla L(x^k, u^{(j)}) - \lambda\, g_i(x^k)}{\nabla g_i(x^k)^T (C^k)^{-1} \nabla g_i(x^k)} \end{aligned} \tag{10}$$

である。式(10), (9)は各iに対して独立に計算することができるので、射影Jacobi法は細粒度型の大規模並列計算機上での実現に適していると考えられる。なお、式(10)の右辺第2項の分母$\nabla g_i(x^k)^T (C^k)^{-1} \nabla g_i(x^k)$は射影Jacobi法の全反復を通じて変化しないので、各i毎に予めその値を計算しておくものとする。一方、式(10)の右辺第2項の分子に表れる$\nabla L(x^k, u^{(j)})$は、射影Jacobi法の前反復において独立に求めた$u_i^{(j)}, i = 1, \cdots, m$を集約して得られる情報であり、その計算をいかに効率的に行うかがアルゴリズム全体の並列化効率を左右することになる。

一方、射影Jacobi法の収束性を保証する十分条件の一つとして、行列$2B^k - M^k$の正定値性があげられる[2]。式(8)より行列B^kは正定値対角行列であり、その対角成分の値は緩和パラメータ$\omega > 0$の値に反比例する。従って正の数ωの値を十分小さくとれば、対称行列$2B^k - M^k$の対角要素の値を、対応する行の非対角要素の絶対値の和よりも大きくすることができる。この時、いわゆるDiagonal Dominance Theorem [7]により、行列$2B^k - M^k$は正定値となる。以下では、このようなωの値の最大値を、$\overline{\omega}$と記述することにする。なお、式(10)からもわかるように、緩和パラメータωは変数uに関する探索においてステップサイズの役割を果たしている。従って、収束性を阻害しない範囲でできるだけ大きな値を用いるのが望ましい。

ところで、式(5)に対応して、

$$d^{(j)} = -\frac{1}{\lambda_k}\left(C^k\right)^{-1} \nabla L(x^k, u^{(j)}) \tag{11}$$

とおくと、問題DSPの目的関数$\phi^k(u)$の$u^{(j)}$における最急降下方向は、(6)の各式より

$$\begin{aligned} -\nabla \phi^k(u^{(j)}) &= -M^k u^{(j)} - q^k \\ &= -\nabla g(x^k)^T (\lambda_k\, C^k)^{-1} \nabla L(x^k, u^{(j)}) + g(x^k) \\ &= \nabla g(x^k)^T d^{(j)} + g(x^k) \end{aligned}$$

となる。そこで、

$$z^{(j)} = \nabla g(x^k)^T d^{(j)} + g(x^k) \tag{12}$$

とおき、添字集合

$$\begin{aligned} I^{(j)} \equiv\ & \{\, i \,|\, (u_i^{(j)} = -r_i \text{ and } z_i^{(j)} \le 0) \text{ or } (u_i^{(j)} = r_i \text{ and } z_i^{(j)} \ge 0),\ i = 1, \cdots, \overline{m}\} \\ \cup\ & \{\, i \,|\, (u_i^{(j)} = 0 \text{ and } z_i^{(j)} \le 0) \text{ or } (u_i^{(j)} = r_i \text{ and } z_i^{(j)} \ge 0),\ i = \overline{m}+1, \cdots, m\} \end{aligned} \tag{13}$$

を定義すると

$$z_i^{(j)} = 0, \qquad \forall i \notin I^{(j)}$$

ならば、$u^{(j)}$は問題DSPの解であることがわかる。なお、式(11), (12)より、式(10)は

$$u_i^+ = u_i^{(j)} + \omega \cdot \frac{z_i^{(j)}}{\nabla g_i(x^k)^T (\lambda_k C^k)^{-1} \nabla g_i(x^k)} \tag{14}$$

と書き直すことができる。

以上をまとめると、問題DSPに対する射影Jacobi法のアルゴリズムは次のようになる。

アルゴリズム 射影Jacobi法

ステップ 1 緩和パラメータ $\omega \in (0, \overline{\omega})$ をセットする。問題 DSP の制約条件を満たす初期探索点 $u^{(1)} \in R^m$ を選び、$j := 1$ とおく。

ステップ 2 式 (10) または (14) により、$i = 1, \cdots, m$ について独立に u_i^+ を計算する。さらに式 (9) により、u^+ の各成分を問題 DSP の対応する制約条件の区間内に射影して次の探索点 $u^{(j+1)}$ を求める。

ステップ 3 式 (11), (12) に基づいて $d^{(j+1)}$, $z^{(j+1)}$ を順次計算する。また、定義 (13) に従って添字集合 $I^{(j+1)}$ を求める。もし、$z_i^{(j+1)} = 0, \forall i \notin I^{(j+1)}$ ならば終了。さもなければ、$j := j + 1$ としてステップ 2 に戻る。

アルゴリズム SLM では、その計算量の大部分が問題 DSP を解くために費やされる。従って現実の大規模問題に対して適用する場合、現在の探索点 x^k が最適解から遠く離れていると考えられる初期の反復においては、問題 DSP の解を求める射影 Jacobi 法のアルゴリズムを、その良い近似解が得られたと判断された段階で打切るのが得策である [3]。そこで、許容誤差の初期値 $\varepsilon > 0$ とその更新乗数 $\sigma \in (0, 1)$ を用いて、アルゴリズムのステップ 3 を次のように修正する。

ステップ 3-1 式 (11), (12) に基づいて $d^{(j+1)}$, $z^{(j+1)}$ を順次計算する。また、定義 (13) に従って添字集合 $I^{(j+1)}$ を求める。

ステップ 3-2 もし、$|z_i^{(j+1)}| < \varepsilon, \forall i \notin I^{(j+1)}$ ならばステップ 3-3 へ。さもなければ、$j := j+1$ としてステップ 2 に戻る。

ステップ 3-3 もし

$$p(x^k) - \bar{p}(x^k, d^{(j+1)}) \geq \delta \lambda_k \left(d^{(j+1)}\right)^k C^k d^{(j+1)} \tag{15}$$

を満たせば終了。さもなければ、$\varepsilon := \sigma\varepsilon$, $j := j + 1$ としてステップ 2 に戻る。

ここに、$\delta \in (0, 1/2)$ は、問題 DSP を解くことによって本来得られる筈のペナルティ関数 $p(x)$ の線形近似関数の減少量 $\Delta\bar{p}_k$ を緩和する意味を持つ定数であり、部分問題 SP の厳密な解 d^k に対しては、

$$\Delta\bar{p}_k \geq \frac{\lambda_k}{2} \left(d^k\right)^T C^k d^k$$

となることが保証されている [3]。

一般に射影 Jacobi 法では、射影 SOR 法や共役勾配法と比べて、解を得るまでにより多くの反復数を必要とする傾向がある。しかも、射影 Jacobi 法のステップ 3 またはステップ 3-1 から 3-3 にかけての終了判定は、ステップ 2 において各 i について独立に求めた情報を集約して行う処理であり、並列計算機ではプロセッサ間通信など多大な負荷を要する場合が多い。そこで、射影 Jacobi 法のアルゴリズムでは、ステップ 3 を毎回行うのではなく、ある一定周期で停止基準が満たされるか否かを判定するという戦略を用いるのが有効である。ただし、式 (14) により u^+ を求める場合には、各反復で $d^{(j+1)}$ とこれに基づき $z^{(j+1)}$ を計算する必要がある。一方、式 (10) による場合も、$\nabla L(x^k, u^{(j)})$ の計算と各 i に対して $\nabla g_i(x^k)^T (C^k)^{-1}$ との内積を求める計算が反復毎に必要となることに変わりはない。

4 並列計算機上への実現

大規模な問題に対するアルゴリズムを実現する際には、行列をどのような形態でメモリ上に保持するかがしばしば問題となる。逐次型の計算機上でのプログラミングにおいては、いわゆる List Structure を用いて疎行列を取扱う方法が一般的であり、特に大規模線形計画問題を対象とする内点法の分野で様々な技法が研究されている [1] 。List Structure は、疎行列の非零要素のみを 1 次元配列に格納するとともに、格納されたデータの行番号または列番号を指示する補助的な配列と、ある行または列の非零要素の格納位置を示すポインタを保持する配列を用意することによって、疎行列とベクトルとの積を効率良く行うためのデータ構造である。今回我々が並列計算機上に実現しようとしている射影 Jacobi 法のアルゴリズムにおいても、式(10), (11), (12)から推察されるように、疎な $n \times m$ 行列 $\nabla g(x^k)$ の取扱いと、$\nabla g(x^k)$ またはその転置行列にベクトルを掛ける演算を効率的に実行できるような仕組みを構築することが要求される。また、Connection Machine System は共有メモリを持たず、配列の各要素に対応する形でローカルメモリを持つプロセッサが割当てられ、プロセッサ間のデータ転送をネットワーク経由で行なう並列計算機であるため、通信負荷をできる限り抑えるような工夫も必要になる。従って、ポインタを次々にたどりつつ必要なデータを検索する List Structure の考え方では、通信負荷の増大により十分な性能を引出すことができない。むしろ、疎な行列との積をとるベクトルの保持形態に冗長性を持たせて、その成分を実際に掛け算を行なう行列の非零要素と同じ位置に格納することにより、同じサイズの配列要素同士の演算を並列的に実行できるという Connection Machine System の特徴が生かせるようなデータ構造を採用することが望ましい。

以下では、我々が射影 Jacobi 法のアルゴリズムをプログラミングする際に採用した手法を具体的に説明する。$n = 4, m = 3$ で、$\nabla g(x)$ が恒等的に 0 でない要素を 6 個持つとしよう。実際、ある反復 k において $\nabla g(x^k)$ が次のような行列である場合を考える。

$$\nabla g(x^k) = \begin{pmatrix} 3 & 0 & 5 \\ 0 & 2 & 0 \\ 0 & 1 & 6 \\ 0 & 0 & 4 \end{pmatrix}.$$

行列 $\nabla g(x^k)$ の非零要素を、列番号，行番号の順にソートして 1 次元配列 GGRD_I に格納しよう。実際の計算には用いないが、GGRD_I が保持している行列 $\nabla g(x^k)$ の要素の列番号，行番号を格納する 1 次元配列をそれぞれ GCLM_I，GROW_I とすると、それらは次のようになる。

GGRD_I :	3	2	1	5	6	4

GCLM_I :	1	2	2	3	3	3

GROW_I :	1	2	3	1	3	4

ここで、GCLM_I において左隣とは異なる値を持つ要素には論理値 T (True)，同じ値持つ要素には F (False) を格納する 1 次元配列 SEGM_I を構成する。ただし、左端の要素の左隣は、右端の要素であると考える。この時、SEGM_I は

SEGM_I :	T	T	F	T	F	F

のような論理配列となり、GGRD_I が保持しているデータの行列 $\nabla g(x^k)$ における区切りを示すことから、Segment と呼ばれれる。

さて、1 次元配列 GGRD_I，GCLM_I，GROW_I を GROW_I の要素を第 1 キー，GCLM_I の要素を第 2 キーとしてソートした 1 次元配列を、それぞれ GGRD_J，GCLM_J，GROW_J とすると、

GGRD_J :

3	5	2	1	6	4

GCLM_J :

1	3	2	2	3	3

GROW_J :

1	1	2	3	3	4

を得る。当然、GGRD_J は、行列 $\nabla g(x^k)$ の非零要素を行番号，列番号の順にソートして格納した1次元配列である。先程と同じように、GROW_J において左隣とは異なる値を持つ要素には論理値 T (True)，同じ値持つ要素には F (False) を格納する1次元配列 SEGM_J を構成すると、

SEGM_J :

T	F	T	T	F	T

を得る。これは、GGRD_J に対する Segment である。

さて、1 から 6 までの整数を順に格納した長さ 6 の1次元配列

1	2	3	4	5	6

を用意し、これを GCLM_J の要素を第1キー，GROW_J の要素を第2キーとしてソートした1次元配列を SEND_J とすると、

SEND_J :

1	3	4	2	5	6

を得る。これは一種のポインタで、1次元配列 SEND_J は、GGRD_I の各要素が GGRD_J の何番目に格納されているかを示している。実際、SEND_J の第3要素は 4 であるから、GGRD_I の第3要素の 1 は、GGRD_J においては第4要素に格納されていることがわかる。これを式で書くと、

$$\mathtt{GGRD\_I} = \mathtt{GGRD\_J}\,(\mathtt{SEND\_J}) \quad \text{または} \quad \mathtt{GGRD\_J}\,(\mathtt{SEND\_J}) = \mathtt{GGRD\_I}$$

となる。我々がアルゴリズムの実現に用いた言語 CM Fortran では、実際にこのようなプログラム記述を認めており、GGRD_J の添字 SEND_J を Vector-Valued Subscript、この操作によるデータ転送を Send Operation と呼んでいる。Send Operation は非常に便利で一般的な機能である反面、Router を経由した1対1通信となるため、処理速度の面である程度のマイナスになることを覚悟しておく必要がある [10] 。

ある行列 A とその転置行列 A^T を取扱う必要となるアルゴリズムを並列計算機上に実現する際には、ここで考えたような2種類の並びを持つ1次元配列を併用し、さらに Send Operation を用いてそれぞれの並び順で保持された情報の整合性をとるという考え方でプログラムを構成する場合が多い [13] 。しかし我々は、Send Operation の使い方を工夫することによって GGRD_I，SEGM_I，SEGM_J，SEND_J のみを用いてアルゴリズムを実現し、メモリ使用量の削減を図ることにした。射影 Jacobi 法のアルゴリズムにおいて $\nabla g(x^k)$ が関与する処理としては、先ず式(10)の右辺第2項の分子または式(11)の右辺に表れる

$$\nabla L(x^k, u^{(j)}) = \nabla f(x^k) + \nabla g(x^k) u^{(j)}$$

の計算がある。$\nabla f(x^k)$, $u^{(j)}$ はそれぞれ R^n, R^m のベクトルであるが、いずれも長さ 6 の1次元配列 FGRD_J，UVAL_I に、それぞれ GROW_J，GCLM_I が示す添字に対応して、

FGRD_J :

$\frac{\partial f(x^k)}{\partial x_1}$	$\frac{\partial f(x^k)}{\partial x_1}$	$\frac{\partial f(x^k)}{\partial x_2}$	$\frac{\partial f(x^k)}{\partial x_3}$	$\frac{\partial f(x^k)}{\partial x_3}$	$\frac{\partial f(x^k)}{\partial x_4}$

UVAL_I :

$u_1^{(j)}$	$u_2^{(j)}$	$u_2^{(j)}$	$u_3^{(j)}$	$u_3^{(j)}$	$u_3^{(j)}$

という並び順でその値が保持されているものとする。この時、共に GCLM_I が示す添字順に従って値を保持する UVAL_I と GGRD_I との間で対応する要素同士の掛け算を並列的に行なった後、SEND_J を Vector-Valued Subscript とする Send Operation を行なってその結果を GCLM_J が示す添字順に並べ変える。そして、SEGM_J が示す Segment 毎に配列要素の部分的な和を計算すれば、GROW_J が示す添字順で $\nabla g(x^k)\, u^{(j)}$ の値を得ることができる。さらに、その結果と FGRD_J との間で対応する要素同士の足し算を並列的に行なえば、$\nabla L(x^k, u^{(j)})$ の値が求められる。ただし、得られた結果は GROW_J が示す添字順となっているので、式(10) または式(11), (12) において $g(x^k)^T$ との内積を求めるために、再び SEND_J を Vector-Valued Subscript とする Send Operation を行なって、GCLM_I の示す添字順に戻しておく必要がある。

一方、$\nabla g(x^k)^T$ が関係する処理の例として、式(12) の計算を考える。$d^{(j)}$, $g(x^k)$ はそれぞれ R^n, R^m のベクトルであるが、いずれも長さ 6 の 1 次元配列 DVAL_I, GVAL_I に、GCLM_I が示す添字に対応して、

DVAL_I :	$d_1^{(j)}$	$d_2^{(j)}$	$d_2^{(j)}$	$d_3^{(j)}$	$d_3^{(j)}$	$d_3^{(j)}$
GVAL_I :	$g_1(x^k)$	$g_2(x^k)$	$g_2(x^k)$	$g_3(x^k)$	$g_3(x^k)$	$g_3(x^k)$

という並び順でその値が保持されているものとする。この場合は、共に GCLM_I が示す添字順に従って値を保持する GGRD_I と DVAL_I との間で対応する要素同士の掛け算を並列的に行なった後、SEGM_I が示す Segment 毎に配列要素の部分的な和を計算すれば $\nabla g(x^k)^T d^{(j)}$ の値を直ちに得ることができる。あとは、その結果と GVAL_I との間で対応する要素同士の足し算を並列的に行なうことにより、$z^{(j)}$ の値が求められる。なお、C^k を対角行列に選んだことに注意すると、式(10) または(14) の右辺分母に表れる $\nabla g_i(x^k)^T (C^k)^{-1} g_i(x^k)$, $i = 1, \cdots, m$ についても、1 次元配列 GGRD_I の各要素の自乗を並列的に求めた上で SEGM_I が示す Segment 毎に配列要素の部分的な和を計算すれば、各 i に対して同時にその値を求めることができる。また、アルゴリズムの他の部分において、$p(\cdot)$ あるいは $\bar{p}(x^k, \cdot)$ 等の値が必要となる場合が存在するが、これらについてもほぼ同様な手順によりその値を計算することができる。

このように、非零要素のみを 1 次元配列に格納した行列またはその転置行列とベクトルとの積を求めるためには、それぞれ SEGM_J または SEGM_I が示す Segment 毎に配列要素の部分的な和を求める必要がある。この演算は Segmented Scan Operation [13] と呼ばれる処理の一つであり、次のような方法で実現できる。先ず、あるベクトル $w = (w_1, w_2, \cdots, w_l)^T$ に対して、$s_1 = w_1$, $s_h = s_{h-1} + w_h$, $h = 2, \cdots, l$ によりベクトル $s = (s_1, s_2, \cdots, s_l)^T$ を求める。次に、$s = (s_1, s_2, \cdots, s_l)^T$ に対して、$t_h = s_l$, $h = l, l-1, \cdots, 1$ によりベクトル $t = (t_1, t_2, \cdots, t_l)^T$ を求める。前者は昇順の Scan Add Operation、後者は降順の Scan Copy Operation と呼ばれる処理である。容易に確かめられるように、これらの演算によってベクトル t の各要素には s の要素の値の総和が格納される。Segmented Scan Operation は、Scan Add Operation および Scan Copy Operation を与えられた Segment 単位で行なう処理であり、CM Fortran のユーティリティ・ライブラリ [9] には、それぞれ CMF_SCAN_ADD および CMF_SCAN_COPY の名前でサブルーチンが提供されている。次章に示す数値実験では、これらのサブルーチンを用いてアルゴリズムの構築を行なった。

5 数値計算結果

この章では、逐次線形化法の各反復で、2 次計画部分問題を射影 Jacobi 法を用いて並列的に解く場合の数値実験結果を示す。使用した計算機は、エイ・ティ・アール人間情報通信研究所の Connection Machine Model CM-5 である。この並列計算機 CM-5 には 32 個のプロセッシング・

ノードが搭載されており、各プロセッシング・ノード毎に4個のベクトル・ユニットが装備されている。各ベクトル・ユニットのピーク性能は32MFLOPSであるが、アルゴリズムSLMのようにスカラー値に対する判定処理や繰返し制御を含む実際のプログラムにおいては、我々の予備的な実験によると5MFLOPS, 最高でも10MFLOPS程度が限界であると言われている [11] 。アルゴリズムのプログラミングは、Connection Machine SystemにおいてSIMD型の並列計算機構を実現するために用意されているCM Fortranを用いて行った。CM Fortranは、科学技術計算向きの言語であるFORTRANを基本とし、Connection Machine Systemのノード間ネットワークを利用して非常に大きなサイズの配列要素同士の演算を並列的に実行すると共に、各配列要素の値を効率的に集約する機能を追加したプログラミング言語である [10] 。

文献 [3] と同様に、アルゴリズムSLMのパラメータはそれぞれ

$$\mu_0 = 0.1, \quad \mu_1 = 0.25, \quad \mu_2 = 0.75, \quad \gamma = 2$$

にセットし、正定値対称行列 C^k は全反復を通して単位行列 I に固定して実験を行った。また、ペナルティ・パラメータの値は $r_i = 100, i = 1, \cdots, m$ とし、ステップ2-2で必要となるペナルティ・パラメータの更新乗数の値としては $\nu = 2$ を用いた。なお、初期探索点としては $x^1 = (1, 1, \cdots, 1)^T$ を選び、部分問題SPの2次項の係数の初期値は $\lambda_1 = 100$ とした。一方、アルゴリズムSLMの停止基準は、補題1の1に基づいて

$$\Delta\bar{p}_k \leq 10^{-8} \left(|\, p(x^k) \,| + 1 \right) \tag{16}$$

とし、連続する2回の反復でこの条件が満たされた場合にアルゴリズムを終了させることにした。この条件は、文献 [3] における数値実験と比較して、やや厳しい条件となっている。

一方、部分問題DSPを解く射影Jacobi法のアルゴリズムにおいては、ステップ3-2および3-3に示す収束判定を10反復に1回の割合で行い、許容誤差の初期値 ε , その更新乗数 σ , ペナルティ関数の線形近似関数 $\bar{p}(x^k, \cdot)$ の減少量に対する緩和係数 δ としては、それぞれ

$$\varepsilon = 10^{-4}, \quad \sigma = 0.1, \quad \delta = 0.1$$

を用いた。なお、初期探索点 $u^{(1)}$ は、部分問題 $\mathrm{DSP}(x^1, C^1, \lambda_1)$ に対しては $(0, 0, \cdots, 0)^T$、以後 $k > 1$ の場合は u^{k-1} を用いることにした。実際、アルゴリズムSLMのステップ2-2において、ある r_i の更新が行われた場合でも、$\nu > 1$ より u^{k-1} は $\mathrm{DSP}(x^k, C^k, \lambda_k)$ の実行可能解となることがわかる。

我々は、目的関数が分離可能な2次関数で、制約関数として分離可能な2次関数, 一般的な線形関数そして変数の非負制約を併せ持つような構造的に疎なテスト問題

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2} x^T D x + c^T x \\ \text{subject to} \quad & \frac{1}{2} x^T G_h x + a_h^T x \leq b_h, \quad h \in H, \\ & a_i^T x \leq b_i, \qquad\qquad i \in I, \\ & x_j \geq 0, \qquad\qquad j \in J \end{aligned} \tag{17}$$

をランダムに生成して実験を行った。ここに、H, I, J はそれぞれ2次制約, 線形制約, 非負制約の添字集合であり、$|J| \leq n$ とする。また、D は $n \times n$ 正定値対角行列, $G_h, h \in H$ は各要素が非負の $n \times n$ 対角行列, c, $a_h, h \in H$, $a_i, i \in I$ は n 次元の実ベクトル, $b_h, h \in H$, $b_i, i \in I$ は実数である。このようなテスト問題を生成する方法としては、いわゆるRosen-Suzukiの方法 [8] がよく知られており、文献 [3] における数値実験においても採用されているが、我々は非常に大規模な問題を効率的に生成することを狙って、テスト問題(17)そのものも直接CM-5上で並

列的に生成することにした。なお、$a_h, h \in H$, $a_i, i \in I$ の非零要素はいずれも区間 $[-5,5]$ から、$b_h, h \in H$, $b_i, i \in I$ はいずれも区間 $[1,10]$ から、c の各要素は区間 $[-100,100]$ から、D の対角要素は区間 $[1,10]$ から、また $G_h, h \in H$ の対角要素は区間 $[0,10]$ からそれぞれ一様に選択している。この時、$b_h > 0, h \in H$, $b_i > 0, i \in I$ より $x = 0$ はすべての制約条件を満足するため、生成されたテスト問題は必ず実行可能となる。

我々は、変数の総数と制約条件の総数との比が 8 : 1, 4 : 1, 2 : 1 の 3 ケースを想定し、変数の総数を 2048 から順次 2 倍して 16384 までの 4 通り、合計 12 種類の問題を生成した。なお、2 次制約の数，線形制約の数そして非負制約の数の比は 1 : 2 : 1 に固定している。また、$a_h, h \in H$, $a_i, i \in I$ の非零要素の数は、制約関数のすべての係数でゼロ以外の値をとる成分の総数が 262144 個になるように決定している。各々のサイズについて乱数の種類を変えて 5 通りの問題例を生成し、そのそれぞれに対して、緩和パラメータ ω の値を 0.05 から順次 2 倍して 0.8 まで変化させながら実験を行った。表 1 から表 5 に、各 ω の値に対する実験の結果を示す。表の各欄に示す数字は、ランダムに生成された 5 通りの問題例に対する結果を平均した値であり、「外反復」は停止基準 (16) が 2 回連続して成立つまでに要したアルゴリズム SLM の反復回数、「内反復」は打切り条件 (15) が成立つまでに要したアルゴリズム射影 Jacobi 法の反復回数を、アルゴリズム SLM の全反復について平均した値をそれぞれ表している。また、「CPU Time」はアルゴリズムが停止するまでに要した実行時間、「Send」は Router 経由のデータ転送である Send Operation が実行時間に占める割合をそれぞれ表している。さらに「精度」欄に示す値は、アルゴリズムが停止した時点における制約条件の逸脱度合

$$\max\left\{ \max_{1\le i\le \bar{m}} \left\{ |g_i(x^k)| \right\}, \max_{\bar{m}+1\le i\le m} \left\{ g_i(x^k) \right\} \right\}$$

を、10 の冪乗単位で評価した値である。なお、「内反復」欄の表示が「> 10000」となっているケースは、アルゴリズム SLM の実行中に部分問題 DSP を解くための射影 Jacobi 法が収束せず、解が得られなかった問題例が少なくとも一つ存在したことを示している。

表に示す結果からわかるように、緩和パラメータ ω の値を大きくするにつれて、より短い時間で解を得ることができるようになる反面、ω の値をある値以上にすると、射影 Jacobi 法が収束せず解を得ることができなくなってしまう。この場合、射影 Jacobi 法における探索点の挙動は、振動または発散状態となっていることが観測される。問題の規模が全体として大きくなるにつれて、射影 Jacobi 法が破綻をきたす ω の閾値は大きくなっているが、これは制約関数の係数でゼロ以外の値をとるものの総数を固定したため、問題の規模が大きくなるにつれて行列 M^k が構造的に疎となり、$2B^k - M^k$ の非対角要素の数が相対的に減少したためと考えられる。また、Send Operation の占める比率が問題サイズに関係なく 56% 前後の値となった理由も、非零要素数を固定したことにより、Router 経由で転送すべきデータ量が各反復でほぼ一定であったためと推察される。

6 おわりに

本論文では、逐次線形化法の各反復で解くべき 2 次計画部分問題に対して、最も基礎的な並列アルゴリズムの一つである射影 Jacobi 法を適用し、これを SIMD 型の並列計算機上で実際に構築することによってその有効性を検証した。並列計算機 CM-5 を用いた数値実験の結果から、変数の総数が数千から 1 万余りの大規模問題に対しても、緩和パラメータ ω の値を適切に選択することにより、数十分以内で解が得られることを確認した。逐次線形化法は、部分問題 SP の 2 次の係数行列が対角行列となるように選ぶことによって原問題の疎な構造が全反復を通して維持されるため、現実の大規模問題を効率的に処理できることを一つの謳い文句にしている。しかしながら、逐次線形化法では原問題の 2 次の情報を全く使用していないため、逐次 2 次計画法 [5]

のように理論的な速い収束性を期待することはできない。実際、数値実験の結果からも、問題の規模が大きくなるにつれて「外反復」の回数が非常に多くなっていくことがわかる。また、実験結果の表には挙げなかったが、最適性の1次の条件の方程式系を構成する $\nabla L(x^k, u^k)$ の L_∞ ノルムの値は、2048 変数の場合で 10^{-2} 前後、16384 変数の場合には 0.5 前後に止まっている。これは、$\nabla L(x^k, u^k)$ の値が十分に小さくなっていない段階でも、λ_k や C^k の対角要素の値が大きい場合には、式(5)よりステップ d^k が非常に小さくなり、探索の進み具合が悪くなるためと推察される。また、C^k を単位行列に固定した場合でも、部分問題 SP の目的関数の第1項の大きさは $||\nabla L(x^k, u^k)||^2 / 2\lambda_k$、第2項はペナルティ・パラメータに制約条件の逸脱度合を掛けた程度の大きさとなるため、λ_k, r_i 共に 100 前後という数値実験のパラメータ設定においては、制約条件の逸脱度合が 10^{-6} 程度であっても、$\nabla L(x^k, u^k)$ の大きさは 10^{-1} 前後で釣合ってしまうことによるものと考えられる。

一方、CM-5 上でのプログラミングの観点から今回の実験を振返ってみると、Send Operation に要する処理時間の割合が全体の半分以上を占めていることからもわかるように、疎な行列 $\nabla g(x^k)$ およびその転置行列 $\nabla g(x^k)^T$ とベクトルとの積をいかに効率的に行うかが課題として残る。射影 Jacobi 法のアルゴリズムにおいては、$\nabla g(x^k)$ および $\nabla g(x^k)^T$ にあるベクトルを掛ける処理が交互に発生する。今回の実験では、$\nabla g(x^k)$ の非零要素を、その列をソートキーとした並び順により1次元配列に格納し、Segmented Scan Operation と Send Operation とを組合せることによりアルゴリズムを構築した。しかし、疎な行列およびその転置行列とベクトルとの積については、CMSSL (Connection Machine Scientific Software Library) [12] にサブルーチンが用意されており、これらを用いて必要な機能が実現できるかについて更に検討を加える必要があると考えられる。

謝辞 Connection Machine Model CM-5 上でのプログラミングにあたり、有益な助言を戴いたアドバンストシステムズ株式会社の 木目沢 司 氏に感謝致します。

参考文献

[1] I. Adler, N. Karmarkar, M.G.C. Resende and G. Veiga, "Data Structures and Programming Techniques for the Implementation of Karmarkar's Algorithm," *ORSA Journal on Computing,* Vol. 1 (1989) pp. 84-106.

[2] 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.

[3] M. Fukushima, K. Takazawa, S. Ohsaki and T. Ibaraki, "Successive Linearization Methods for Large-Scale Nonlinear Programming Problems," *Japan Journal of Industrial and Applied Mathematics,* Vol. 9 (1992), pp. 117-132.

[4] S.-P. Han and O.L. Mangasarian, "Exact Penalty Functions in Nonlinear Programming," *Mathematical Programming,* Vol. 17 (1979) pp. 251-269.

[5] 茨木俊秀, 福島 雅夫, *FORTRAN 77* 最適化プログラミング, 岩波書店, 1991.

[6] O.L. Mangasarian, "Sufficiency of Exact Penalty Minimization," *SIAM Journal on Control and Optimization,* Vol. 23 (1985), pp. 30-37.

[7] J.M. Ortega and W.C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables,* Academic Press, New York, 1970.

[8] J.B. Rosen and S. Suauki, "Construction of Nonlinear Programming Test Problems," *Communications of the ACM,* Vol. 8 (1965) p. 113.

[9] Thinking Machines Corporation, *CM Fortran Libraries Reference Manual,* Cambridge, Massachusetts, 1994.

[10] Thinking Machines Corporation, *CM Fortran Programming Guide,* Cambridge, Massachusetts, 1994.

[11] Thinking Machines Corporation, *CM-5 CM Fortran Performance Guide,* Cambridge, Massachusetts, 1994.

[12] Thinking Machines Corporation, *CMSSL for CM Fortran: CM-5 Edition, Volume I,* Cambridge, Massachusetts, 1993.

[13] 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.

表 1: $\omega = 0.05$ の場合の数値実験結果

問題サイズ				反復回数		CPU Time	Send	精
変数	2 次制約	線形制約	非負制約	外反復	内反復	(sec)	(%)	度
2048	64	128	64	41.0	310.6	2154.58	56.0	10^{-7}
	128	256	128	44.4	367.2	2748.28	55.9	10^{-5}
	256	512	256	48.8	519.3	4270.11	55.9	10^{-7}
4096	128	256	128	45.8	289.7	2244.85	56.0	10^{-6}
	256	512	256	47.0	380.6	5445.88	55.8	10^{-4}
	512	1024	512	47.2	493.0	3912.91	55.8	10^{-4}
8192	256	512	256	83.6	299.3	4223.55	55.9	10^{-4}
	512	1024	512	82.8	367.5	5129.09	55.8	10^{-4}
	1024	2048	1024	72.2	512.1	6221.48	55.8	10^{-4}
16384	512	1024	512	612	277.2	28606.92	55.8	10^{-5}
	1024	2048	1024	393	375.0	24839.65	55.8	10^{-6}
	2048	4096	2048	400	499.0	33595.40	55.8	10^{-6}

表 2: $\omega = 0.1$ の場合の数値実験結果

問題サイズ				反復回数		CPU Time	Send	精
変数	2 次制約	線形制約	非負制約	外反復	内反復	(sec)	(%)	度
2048	64	128	64	—	> 10000	—	—	—
	128	256	128	—	> 10000	—	—	—
	256	512	256	—	> 10000	—	—	—
4096	128	256	128	41.6	158.8	1123.90	55.9	10^{-4}
	256	512	256	44.8	198.2	1502.37	55.8	10^{-4}
	512	1024	512	44.8	251.4	1901.41	55.8	10^{-4}
8192	256	512	256	83.6	161.0	2286.04	55.9	10^{-4}
	512	1024	512	82.0	206.9	2866.64	55.8	10^{-4}
	1024	2048	1024	71.2	272.8	3277.24	55.8	10^{-4}
16384	512	1024	512	616	152.8	15973.37	55.7	10^{-6}
	1024	2048	1024	401	201.8	13752.76	55.7	10^{-6}
	2048	4096	2048	401	366.2	24779.49	55.8	10^{-6}

表 3: $\omega = 0.2$ の場合の数値実験結果

問題サイズ				反復回数		CPU Time	Send	精
変数	2 次制約	線形制約	非負制約	外反復	内反復	(sec)	(%)	度
2048	64	128	64	—	> 10000	—	—	—
	128	256	128	—	> 10000	—	—	—
	256	512	256	—	> 10000	—	—	—
4096	128	256	128	—	> 10000	—	—	—
	256	512	256	—	> 10000	—	—	—
	512	1024	512	—	> 10000	—	—	—
8192	256	512	256	83.8	84.4	1213.43	55.8	10^{-4}
	512	1024	512	86.8	106.0	1573.25	55.7	10^{-6}
	1024	2048	1024	68.2	138.0	1598.65	55.8	10^{-4}
16384	512	1024	512	619	80.4	8544.53	55.7	10^{-6}
	1024	2048	1024	400	106.2	7258.38	55.8	10^{-6}
	2048	4096	2048	398	141.1	9546.01	55.8	10^{-6}

表 4: $\omega = 0.4$ の場合の数値実験結果

問題サイズ				反復回数		CPU Time	Send	精
変数	2 次制約	線形制約	非負制約	外反復	内反復	(sec)	(%)	度
2048	64	128	64	—	> 10000	—	—	—
	128	256	128	—	> 10000	—	—	—
	256	512	256	—	> 10000	—	—	—
4096	128	256	128	—	> 10000	—	—	—
	256	512	256	—	> 10000	—	—	—
	512	1024	512	—	> 10000	—	—	—
8192	256	512	256	—	> 10000	—	—	—
	512	1024	512	—	> 10000	—	—	—
	1024	2048	1024	—	> 10000	—	—	—
16384	512	1024	512	565	41.0	4067.14	55.7	10^{-6}
	1024	2048	1024	397	55.0	3794.17	55.8	10^{-6}
	2048	4096	2048	419	78.0	5620.36	55.8	10^{-6}

表 5: $\omega = 0.8$ の場合の数値実験結果

問題サイズ				反復回数		CPU Time	Send	精
変数	2 次制約	線形制約	非負制約	外反復	内反復	(sec)	(%)	度
2048	64	128	64	—	> 10000	—	—	—
	128	256	128	—	> 10000	—	—	—
	256	512	256	—	> 10000	—	—	—
4096	128	256	128	—	> 10000	—	—	—
	256	512	256	—	> 10000	—	—	—
	512	1024	512	—	> 10000	—	—	—
8192	256	512	256	—	> 10000	—	—	—
	512	1024	512	—	> 10000	—	—	—
	1024	2048	1024	—	> 10000	—	—	—
16384	512	1024	512	—	> 10000	—	—	—
	1024	2048	1024	—	> 10000	—	—	—
	2048	4096	2048	—	> 10000	—	—	—