分離可能な2次計画問題に対する非同期型 ブロック並列アルゴリズム*

山川 栄樹**・牧 英一***・福島 雅夫****

An Asynchronous Block-Parallel Algorithm for Separable Quadratic Programming Problems*

Eiki YAMAKAWA**, Eiichi MAKI*** and Masao FUKUSHIMA****

An asynchronous modification of the block-parallel algorithm is presented for quadratic programming problems with separable objective functions. In particular, an implementation strategy is proposed to effectively decentralize the communication overhead associated with a specific processor of a parallel computer. Through extensive numerical experiments on the Connection Machine CM-5, the efficiency of the proposed method is examined for large scale problems with some block structure.

1. はじめに

現実の大規模な最適化問題を効率よく解く方法として、複数のプロセッサに処理を分担させる並列アルゴリズムが注目を集めている。すでに、数千から数万個のプロセッサを装備した超並列計算機も開発され、その構造的特徴を活用したデータ並列型のアルゴリズムが多数提案されている^{4)-6),10),15)}。これらの方法においては、アルゴリズムを構成する主要な処理が、並列的に実行し得る単純な計算に細分される。そのため、プロセッサ間のデータ通信に多大な時間が消費され、十分な並列化効率が得られない場合も考えられる。そこで、アルゴリズムの骨格部分を並列処理可能ないくつかのブロックに分割し、複数のプロセッサを用いてこれらを独立に実行させるコントロール並列型の方法が提案されている^{2),8),9),14)}。コネ

クションマシン CM-5 のように、十数個から数十個のベクトル計算機を構成要素とする並列計算機を利用すると、大規模な問題に対してもこれらのアルゴリズムを効率よく実行できる。

ところで、粒度の粗いコントロール並列型のアルゴリズムにおいては、各プロセッサの処理時間にばらつきが生じる可能性があるため、一部のプロセッサが長時間にわたって遊休状態となり、処理効率が著しく低下する場合も少なくない。このような欠点を解消するためには、非同期並列の考え方(参考文献1), Chapters 6-7)が有効であると言われている。例えば De Leone ら³⁾は、線形相補性問題に対する並列型の逐次過緩和(SOR)法⁹⁾において、各プロセッサが探索点の更新を独自に進めることにより、アルゴリズムの実際的な収束性が改善されることを示している。

本論文では、分離可能な2次計画問題に対して最近提案されたブロック並列アルゴリズム¹⁴⁾を非同期型に拡張することを試みる。このアルゴリズムは、原問題の双対問題をブロック対角行列を用いて分解し、得られた複数の小さな問題を、変数の非負制約を取り扱えるように修正した共役勾配法⁷⁾によっていくつかのプロセッサ上で並列的に解く方法である。分解された個々の問題に対する共役勾配法は、もとの問題の制約条件の係数ベクトルに対する簡単な操作のみで構成されるため、疎で大規模な現実の問題を効率的に解くことができる。しかしなが

* 原稿受付 1996年7月1日

** 高松大学 経営学部 College of Business Administration, Takamatsu University; 960 Kasugacho, Takamatsu city, Kagawa 761-01, JAPAN

*** 松下電工(株) インフォメーションシステムセンター Information System Center, Matsushita Electric Works Ltd.; 1048 Kodama, Kadoma city, Osaka 571, JAPAN

**** 京都大学 大学院 工学研究科 Graduate School of Engineering, Kyoto University; Yoshidahonmachi, Sakyo ward Kyoto city, Kyoto 606-01, JAPAN

Key Words: asynchronous algorithm, parallel algorithm, conjugate gradient method, quadratic programming problem.

ら、各プロセッサにおける共役勾配法の実行時間は、解くべき問題の係数行列の条件数や非零要素数によって大きく異なる可能性がある。そこで、いくつかのプロセッサが問題を解き終わった段階で探索点を更新し、遊休状態となったプロセッサに新たな問題を分配して探索を進める非同期型のアルゴリズムを提案する。

本論文の構成は次のとおりである。まず2.では、分離可能な2次計画問題に対するブロック並列アルゴリズムを記述する。これをもとに、3.において非同期並列アルゴリズムを構築する。4.では、提案したアルゴリズムを並列計算機で実行するための計算モデルと、特定のプロセッサに通信負荷が集中することを回避する方策を述べる。5.においては、コネクションマシン CM-5 を用いた数値実験結果を示す。最後に6.で、本論文のまとめを行う。

2. ブロック並列アルゴリズム

本論文では、分離可能な2次計画問題

$$\begin{aligned} \text{目的関数: } & \frac{1}{2}x^\top Dx + c^\top x \rightarrow \text{最小} \\ \text{制約条件: } & a_i^\top x = b_i, \quad i \in E, \\ & a_i^\top x \leq b_i, \quad i \in I, \end{aligned} \quad (1)$$

について考える。ただし、 $D \in R^{n \times n}$ は正定値対角行列、 c と a_i は R^n のベクトル、 b_i は実数、 E と I は有限な添字集合、 $\top$ は転置記号である。いま、 $a_i^\top$ を第 i 行にもつような行列を A 、 b_i を第 i 要素とするベクトルを b と書く。このとき、問題 (1) の双対問題は

$$\begin{aligned} \text{目的関数: } & \frac{1}{2}z^\top Mz + q^\top z \rightarrow \text{最小} \\ \text{制約条件: } & z_i \geq 0, \quad i \in I, \end{aligned} \quad (2)$$

と書ける。ただし、 $M = AD^{-1}A^\top$ 、 $q = AD^{-1}c + b$ である。ここで、双対問題を通常の最大化問題ではなく、等価な最小化問題として定式化していることに注意しておく。問題 (2) の目的関数の最急降下方向を $r = (-Mz - q)$ と書くと、その1次の最適性条件は次のように表せる。

$$\begin{aligned} r_i &= 0, & i \in E, \\ r_i &\leq 0, \quad z_i \geq 0, \quad r_i z_i = 0, & i \in I. \end{aligned} \quad (3)$$

行列 M に対して、 $G + H$ が M に等しく、 $G - H$ が正定値であるような対称行列の組 (G, H) を考える。行列 M の半正定値性より、 G は正定値行列である。このとき、部分問題

$$\begin{aligned} \text{DSP}^{(k)}: \text{目的関数: } & \frac{1}{2}z^\top Gz + v^{(k)\top} z \rightarrow \text{最小} \\ \text{制約条件: } & z_i \geq 0, \quad i \in I, \end{aligned}$$

の解はただ一つ存在するのでそれを $z^{(k+1)}$ とし、点列 $\{z^{(k)}\}$ を生成するようなアルゴリズムを考えることができる。ただし、 $v^{(k)} = Hz^{(k)} + q$ である。このとき、原問題 (1) が実行可能ならば、点列 $\{z^{(k)}\}$ は双対問題 (2) の最適解の一つに収束する (参考文献14), Theorem 1)。なお、

$$x^{(k)} = -D^{-1}(A^\top z^{(k)} + c) \quad (4)$$

とおくと、点列 $\{x^{(k)}\}$ は原問題 (1) の最適解に収束し、

$$r^{(k)} = Ax^{(k)} - b \quad (5)$$

と書ける。

いま、 $J_\ell, \ell = 1, \dots, L$, を添字集合 $E \cup I$ の分割とする。必要ならば行と列を入れ換えて、部分問題 $\text{DSP}^{(k)}$ の係数行列 G をブロック対角行列 $G = \text{diag}\{G_1, \dots, G_L\}$ に選ぶ。ただし、 G_ℓ は $|J_\ell| \times |J_\ell|$ 対称行列で、その行と列はそれぞれ添字集合 J_ℓ の要素に対応しているものとする。このとき、部分問題 $\text{DSP}^{(k)}$ は次のような L 個の独立な部分問題に分割される。

$$\begin{aligned} \text{DSP}_\ell^{(k)}: \\ \text{目的関数: } & \frac{1}{2}z_{J_\ell}^\top G_\ell z_{J_\ell} + v_{J_\ell}^{(k)\top} z_{J_\ell} \rightarrow \text{最小} \\ \text{制約条件: } & z_i \geq 0, \quad i \in I \cap J_\ell. \end{aligned}$$

ここで、 $v_{J_\ell}^{(k)}, z_{J_\ell}$ はそれぞれ $v^{(k)}, z$ の添字集合 J_ℓ に対応する部分ベクトルである。

結局、問題 (1) に対するブロック並列アルゴリズム¹⁴⁾ は、次のように記述される。

【アルゴリズム BP】

ステップ 1 $z_i^{(1)} \geq 0, i \in I$, を満たすように初期探索点 $z^{(1)} \in R^{|E \cup I|}$ を選び、 $k := 1$ とおく。

ステップ 2 (4), (5) 式により、 $r^{(k)}$ を計算する。

ステップ 3 $z^{(k)}$ と $r^{(k)}$ が条件 (3) を満たすならば終了する。

ステップ 4 部分問題 $\text{DSP}_\ell^{(k)}, \ell = 1, \dots, L$, を並列的に解き、それらの解 $\bar{z}_{J_\ell}$ をもとに探索点を $z^{(k+1)} := (\bar{z}_{J_1}^\top, \dots, \bar{z}_{J_L}^\top)^\top$ と更新する。

ステップ 5 $k := k + 1$ としてステップ 2 へ戻る。□

なお参考文献14)では、変数の非負制約を取り扱えるように修正した前処理つき共役勾配法を用いて、各部分問題 $\text{DSP}_\ell^{(k)}$ を解いている。

3. 非同期型のアルゴリズム

アルゴリズム BP は、ステップ 4 において L 個の独立な部分問題 $\text{DSP}_\ell^{(k)}$ を並列的に解くことができる反面、すべての部分問題の解が得られない限りステップ 5 へ進めないという意味で、同期型のアルゴリズムである。こ

のため、係数行列 G_ℓ の条件数や非零要素数に大きなばらつきが存在すると、部分問題を解き終わった一部のプロセッサが長時間にわたって遊休状態となり、アルゴリズム全体としての処理効率が著しく低下する可能性がある。そこで、部分問題のいくつかにおいて解が得られた段階で、利用可能な最新の情報をもとに新たな部分問題を構成しなおし、遊休状態となっているプロセッサに分配して探索を再開させる非同期型のアルゴリズムを考える。

いま、十分な数のプロセッサが存在し、各部分問題はそれぞれ特定のプロセッサに独占的に割当てられていると仮定する。このとき、次のような規則に基づいて探索点を更新すれば、非同期アルゴリズムのプロトタイプが得られる。すなわち、 ℓ 番目のプロセッサにおいて部分問題の解 $\bar{z}_{J_\ell}$ が得られたならば、 $z^{(k+1)} := (z_{J_1}^{(k)\top}, \dots, z_{J_{\ell-1}}^{(k)\top}, \bar{z}_{J_\ell}^\top, z_{J_{\ell+1}}^{(k)\top}, \dots, z_{J_L}^{(k)\top})^\top$ により探索点を更新し、 $k := k+1$ とする。収束判定の後、更新した探索点に対して新たな部分問題を構成し、プロセッサ ℓ においてこれを解くための手続きを直ちに起動する。

この方法では、プロセッサが遊休状態となる時間は非常に短くなるものの、比較的速く部分問題を解き終わったプロセッサにおいては、探索点の大部分の成分が未更新の状態での新たな部分問題を構成するため、反復回数がかさむだけで十分な改善が得られない可能性がある。また、部分問題を解くために長い時間を要したプロセッサが与える結果は古い時点の情報に基づいているため、探索点の更新が効率よく行われないおそれもある。

そこで、探索点を更新して新たな部分問題を構成するタイミングを定めるパラメータを導入することにより、非同同期度が調整可能な実効あるアルゴリズムの構築を試みる。同期型のアルゴリズムに非同同期性を導入するには、さまざまな方法が考えられる。たとえば、一定の時間間隔で探索点を更新し、その時点の最新情報をもとに新たな部分問題を構成して遊休状態にあるプロセッサに割当てれば、非同同期度を連続的に調整できるアルゴリズムが得られる。しかし、一般に部分問題を解くために要する時間は問題によって異なるため、望ましい時間間隔を事前に決定することは困難である。ここでは、ある整数 $N \in \{1, \dots, L\}$ を定め、新たに N 個のプロセッサが部分問題を解き終わるたびに探索点を更新し、遊休状態にあるプロセッサにおいて新たな部分問題を構成して探索を再開させることにする。このとき、 $N=1$ とおけば上で述べた非同同期アルゴリズムのプロトタイプが得られ、 $N=L$ とおけば同期型のアルゴリズムと同じ動作をする。したがって、個々の問題の難易度には関係なく、 N の値によってアルゴリズムの非同同期度を相対的に定めることができる。またこの方法では、いずれかのプロセッサが部分問題の解を得

たことをトリガーとして探索点の更新が行われる。部分問題を解き終わったプロセッサが得られた解を他のプロセッサに伝えることにしておけば、このような処理は並列計算機のメッセージ受け渡し機能を用いて容易に実行できる。問題 (1) に対する非同同期型ブロック並列アルゴリズムは、次のように記述される。

【アルゴリズム ABP】

ステップ 1 $z_i^{(1)} \geq 0, i \in I$, を満たすように初期探索点 $z^{(1)} \in R^{E \cup I}$ を選ぶ。整数 $N \in \{1, \dots, L\}$ を定め、遊休状態にあるプロセッサ番号の集合 S の初期値を $S := \{1, \dots, L\}$ とする。 $k := 1$ とおく。

ステップ 2 (4), (5) 式により、 $r^{(k)}$ を計算する。

ステップ 3 $z^{(k)}$ と $r^{(k)}$ が条件 (3) を満たすならば終了する。

ステップ 4 各プロセッサ $\ell \in S$ において部分問題 $\text{DSP}_\ell^{(k)}$ を解くための手続きを独立に起動した後、 $S := \emptyset, y := z^{(k)}$ とおく。

ステップ 5 あるプロセッサ ℓ が部分問題 $\text{DSP}_\ell^{(k)}$ の解 $\bar{z}_{J_\ell}$ を得たならば、 $S := S \cup \{\ell\}$ として

$$y := (y_{J_1}^\top, \dots, y_{J_{\ell-1}}^\top, \bar{z}_{J_\ell}^\top, y_{J_{\ell+1}}^\top, \dots, y_{J_L}^\top)^\top$$

とおく。

ステップ 6 もし $|S| < N$ ならば、ステップ 5 へ戻る。さもなければ、 $z^{(k+1)} := y, k := k+1$ としてステップ 2 へ戻る。□

非同同期アルゴリズムの一般的な収束定理は、Bertsekas ら¹⁾に詳しい。いま、 X を集合 $X_1, \dots, X_n$ の直積空間とし、 X の要素を $x = (x_1, \dots, x_n)$ と書く。さらに、関数 $f_i: X \rightarrow X_i, i = 1, \dots, n$, を考え、 $f(x) = (f_1(x), \dots, f_n(x))$ とおく。もし、同期型の並列アルゴリズム

$$x_i^{(k+1)} := f_i(x^{(k)}), \quad i = 1, \dots, n,$$

によって生成される点列 $\{x^{(k)}\}$ の任意の集積点が関数 f の不動点となるならば、アルゴリズムを非同同期型に拡張しても、適当な仮定のもとで収束性が保たれる。すなわち、任意の時刻 $t \in \{0, 1, 2, \dots\}$ において、 x の少なくとも一つの成分が式

$$x_i(t+1) := f_i(x_1(\tau_1^i(t)), \dots, x_n(\tau_n^i(t)))$$

に従って更新されるものとする。ただし、

$$0 \leq \tau_j^i(t) \leq t, \quad \forall j,$$

である。このとき、次の命題が成り立つ (参考文献 1), §6.2 の Proposition 2.1)。

【命題 1】 x のすべての成分は無限回更新されるものとし、 $\{t_k\}$ を x_i が更新される時刻から成る列とするとき、

$$\lim_{k \rightarrow \infty} \tau_j^i(t_k) = \infty, \quad \forall j,$$

が成り立つものと仮定する。このとき、 $\{x(t)\}$ のすべての集積点は f の不動点である。□

命題 1 を利用すると、アルゴリズム ABP の収束性は次のように記述される。

【定理 1】 十分大きな正の定数 T が存在し、任意の k に対して $z^{(k)}$ の部分ベクトル $z_{J_\ell}^{(k)}$ は期間 $[k, T]$ の間に少なくとも 1 回更新されると仮定する。このとき、原問題 (1) が実行可能ならば、アルゴリズム ABP で生成された点列 $\{z^{(k)}\}$ の任意の集積点は双対問題 (2) の最適解である。

(証明) $G+H$ が M に等しく $G-H$ が正定値であるような対称行列の組 (G, H) に対して、アルゴリズム ABP に対応する同期型のアルゴリズム BP によって生成された点列 $\{z^{(k)}\}$ は、問題 (2) の最適解の一つに収束する¹⁴⁾。よって、命題 1 より非同期型のアルゴリズム ABP の収束性が従う。□

すべての $\ell=1, \dots, L$ に対して行列 G_ℓ は正定値であるから、部分問題 $\text{DSP}_\ell^{(k)}$ は常に唯一の解をもつ。したがって、前処理つき共役勾配法によりこれらの部分問題を解けば、理論上は一定の時間内にその解を得ることができるため、定理 1 の仮定は成り立つものと考えられる。

アルゴリズム BP の場合¹⁴⁾と同じように、 G_ℓ を M のブロック対角部分のスカラ倍に選べば、部分問題 $\text{DSP}_\ell^{(k)}$ を解くための前処理つき共役勾配法の手続きは、原問題の制約条件の係数ベクトルに対する簡単な演算のみで効率的に実行できる。行列 M のブロック対角部分が正定値でない場合は、単位行列のスカラ倍を加えて $G-H$ が正定値となるように修正すればよい。

4. 並列計算機での実行方法

アルゴリズム ABP は、そのステップ 4 において部分問題 $\text{DSP}_\ell^{(k)}$, $\ell \in S$, を独立に解かなければならないため、典型的なコントロール並列型のアルゴリズムである。しかし、各部分問題を前処理つき共役勾配法を用いて解くならば、個々のプロセッサはほとんど同一のプログラムを実行することになる。よって、同期型のアルゴリズム BP の場合¹⁴⁾と同様に、メッセージ受け渡しの機構を組み込んだ SPMD (Single-Program Multiple-Data) 型の計算モデルによって実現できる。メッセージには、発信元、宛先、メッセージの種類を記したタグがつけられていると仮定する。また、遊休状態にあるプロセッサは、

自分宛に発信されたメッセージの有無を常に監視し、メッセージを受信すればその発信元と種類を識別したうえでメッセージの内容を読み、必要な処理を起動できるものとする。

行列 G の各ブロック対角部分 G_ℓ に対して、プロセッサを一つずつ割当てる。これら L 個のプロセッサをワーカノードと呼び、割当てられた部分問題の処理に専念させる。一方、探索点の更新や収束判定のようにアルゴリズム全体の動きを調整する役割は、マスタノードと呼ばれる付加的なプロセッサに担当させる。

アルゴリズム ABP のステップ 2 においては、マスタノードが (4) 式により $x^{(k)}$ の値を計算し、これを遊休状態にあるワーカノード群 $\ell \in S$ に伝送する。アルゴリズム BP を実行する際には、 $r^{(k)}$ の値もマスタノードが計算して、その結果をもとに収束判定を行っていた¹⁴⁾。しかし、非同期型のアルゴリズム ABP では、とくに N の値が小さい場合、マスタノードはワーカノードから探索点の更新を頻繁に要求される。そのため、処理時間の長い $r^{(k)}$ の計算をマスタノードで行うことは、ワーカノードが遊休状態となる時間を削減するという非同期アルゴリズムのねらいを妨げることになる。そこで、メッセージとして $x^{(k)}$ を受け取った各ワーカノードが独立に

$$r_{J_\ell}^{(k)} = A_\ell x^{(k)} - b_{J_\ell}$$

を計算する。そして、 $z_{J_\ell}^{(k)}$ の値とともに、条件 (3) の添字集合 J_ℓ に対応する部分の成否を判定し、その結果を収束判定資料としてマスタノードに伝えることにする。

アルゴリズム ABP のステップ 3 は、マスタノードにおいて実行される。マスタノードは添字集合 J_ℓ ごとに条件 (3) の成立状況を蓄積し、各ワーカノードから新たな収束判定資料を受信するたびにこれを更新する。ただし、各ワーカノードからは一般に異なる k における収束判定資料が伝送されるため、受信した最新の収束判定資料がいずれも条件 (3) の該当部分の成立を指示した段階ですべてのワーカノードの同期をとり、改めてアルゴリズム全体としての終了判定を行うことにする。すなわち、現在部分問題を解いているワーカノードの処理が終了するのを待って、最新の $x^{(k)}$ の値をすべてのワーカノードに送り直し、返送される収束判定資料の内容を総合して条件 (3) の成立を判定する。なお、条件 (3) の対応する部分が成立したワーカノードにおいては、現在処理している部分問題も直ちに解けるため、同期をとることによる時間的損失はほとんど生じない。

マスタノードにおける収束判定と並行して、各ワーカノードはアルゴリズム ABP のステップ 4 を実行する。収束判定資料を伝送し終えたワーカノードは

$$v_{J_\ell}^{(k)} = -G_\ell z_{J_\ell}^{(k)} - r_{J_\ell}^{(k)}$$

を計算し、新たな部分問題を解き始める。部分問題を解き終わったワーカノードは、その解 $\bar{z}_{J_\ell}$ をもとに $D^{-1}A_\ell^T \bar{z}_{J_\ell}$ を計算してマスタノードに伝送する。ただし、 A_ℓ は添字集合 J_ℓ に対応する A の行から成る行列である。マスタノードは、各 ℓ に対する $D^{-1}A_\ell^T \bar{z}_{J_\ell}$ の値を蓄積し、新たな値を受信するたびにこれを更新する。そして、新たに N 個のワーカノードからメッセージを受け取った段階で、 $x^{(k)}$ の値を更新することになる。

しかし、上で述べた計算モデルにおいても、新たに $x^{(k)}$ を受け取ったワーカノードからマスタノードへ送られる収束判定資料の受信処理によって、その後部分問題を解きおわった他のワーカノードから送られてくる探索点の更新要求の実行が遅れることも考えられる。そこで、利用できるプロセッサの数に余裕があるならば、マスタノードの機能を二つのプロセッサに分担させることにする。ここでは、探索点の更新を担当するプロセッサをマスタノード A、アルゴリズム全体の収束判定を担当するプロセッサをマスタノード B と呼ぶ。また、このような計算モデルを二重マスタモデルと呼ぶ。これに対して、先に述べたマスタノードが一つの計算モデルを単独マスタモデルと呼んで区別する。

二重マスタモデルを用いたアルゴリズム ABP の実行状況の一例を、Fig.1 に示す。ワーカノードから受信した最新の収束判定資料がいずれも条件 (3) の成立を示した段階で、マスタノード B はマスタノード A にワーカノードの同期をとることを指示する。マスタノード A は、部分問題を解いている各ワーカノードからメッセージが到着した後、最新の $x^{(k)}$ をすべてのワーカノードに伝送する。一方、マスタノード B は、各ワーカノードから送られる収束判定資料をもとに終了判定を行い、収束が確認されればアルゴリズムの停止を指示する。

なお、各プロセッサにおいて原問題 (1) の係数行列の非零要素などを保持する方法は、アルゴリズム BP の場合¹⁴⁾と同じである。

5. 数値実験結果

この節では、コネクションマシン CM-5 上で行った数値実験結果を述べる。数値実験に用いた CM-5 は、浮動小数点演算用のベクトルユニットを4個ずつ搭載した32個のプロセッサから構成され、各プロセッサのピーク性能は128 MFLOPS と評価されている¹²⁾。また、データネットワークを介して個々のプロセッサ間でメッセージ受け渡しが可能であり、SPMD 型の計算モデルを用いてコントロール並列型のアルゴリズムを効率よく実行できる。

各プロセッサにおいて実行する手続きのコーディングには、大規模な配列要素同士の演算を高速に実行できるように開発されたプログラミング言語 CM Fortran¹¹⁾を用いた。また、プロセッサ間のメッセージ受け渡しには、ソフトウェア・ライブラリー CMMD¹³⁾を利用している。なお、すべての計算は倍精度で行った。

数値実験は、ランダムに生成された疎な2次計画問題 (1) に対して行った。ただし、制約条件はすべて不等式制約条件とし、 A の非零要素、 b, c の各要素および D の対角要素は、それぞれ区間 $[-5, 5]$, $[1, 10]$, $[-100, 100]$ および $[1, 10]$ から一様に選んだ。このとき、テスト問題は常に実行可能である。参考文献14)と同様に、行列 A に対して二つの典型的なブロック構造を考え、それぞれについて問題のサイズを変えながら実験を行った。なお、各サイズについて乱数の種類を変えて五つの例題を生成し、その平均を実験結果としている。

アルゴリズム ABP の初期探索点として $z^{(1)} = 0$ を選ぶ。また、ステップ3においては、すべての $i \in E \cup I$ に対して条件

$$|r_i^{(k)}| \leq 10^{-7} \|b\|_\infty$$

または

$$z_i^{(k)} = 0, \quad r_i^{(k)} \leq 10^{-7} \|b\|_\infty$$

が成り立ったときにアルゴリズム ABP を終了させる。一方、 $s_{J_\ell} = -G_\ell z_{J_\ell} - v_{J_\ell}^{(k)}$ とおくと、部分問題 $\text{DSP}_\ell^{(k)}$ の1次の最適性条件は

$$\begin{aligned} s_i &= 0, & i \in E \cap J_\ell, \\ s_i &\leq 0, \quad z_i \geq 0, \quad s_i z_i = 0, & i \in I \cap J_\ell, \end{aligned}$$

と書ける。そこで、部分問題 $\text{DSP}_\ell^{(k)}$ を解く前処理つき共役勾配法の手続きは、すべての $i \in J_\ell$ に対して条件

$$|s_i| \leq 10^{-7} \|b_{J_\ell}\|_\infty$$

または

$$z_i = 0, \quad s_i \leq 10^{-7} \|b_{J_\ell}\|_\infty$$

が成り立った時点で打ち切ることにする。

5.1 階段型制約条件

まず、問題 (1) の制約条件の係数行列 A が、多期間計画問題などにおいて現れる階段型の構造

 Fig. 1 Execution sequence for Algorithm ABP with double master model when $L=4$ and $N=2$

Table 2 Efficiency of Algorithm ABP

Model	Problem	$N=1$	$N=2$	$N=3$	$N=4$	$N=5$	$N=6$	$N=7$	$N=8$
Single	QP22	0.6366	0.7093	0.7123	0.7017	0.6856	0.6715	0.7088	0.7965
Master	QP23	0.4771	0.5298	0.5470	0.5358	0.5561	0.5480	0.5355	0.5047
Double	QP22	0.7014	0.6697	0.6497	0.6337	0.6153	0.6049	0.6418	0.7137
Master	QP23	0.5236	0.5140	0.5142	0.4890	0.5024	0.5075	0.4821	0.4515

$$A = \begin{pmatrix} A_{11} & A_{12} & & & & & & & \\ & A_{21} & A_{22} & A_{23} & & & & & \\ & & & A_{32} & A_{33} & A_{34} & & & \\ & & & & & & \ddots & & \\ & & & & & & & A_{Q-Q-1} & A_{Q-Q} \end{pmatrix}$$

をもつ場合を考える。ただし、 A_{qq-1} , $q=2, \dots, Q$, と A_{qq+1} , $q=1, \dots, Q-1$, は $\bar{m} \times \bar{n}_1$ 行列, A_{qq} , $q=2, \dots, Q-1$, は $\bar{m} \times \bar{n}_2$ 行列, また A_{11} , A_{QQ} はいずれも $\bar{m} \times (\bar{n}_1 + \bar{n}_2)$ 行列とする。行列 A の構造に対応して, 対角行列 D を

$$D = \text{diag}\{D_{11}, D_{12}, D_{22}, D_{23}, D_{33}, D_{34}, \dots, D_{Q-1Q}, D_{QQ}\}$$

と書く。このとき, 双対問題 (2) の係数行列 M はブロック三重対角行列となり, $A_{10} = A_{Q-Q+1} = 0$ とおくと, そのブロック対角部分は

$$M_{qq} = A_{qq-1} D_{q-1q}^{-1} A_{qq-1}^T + A_{qq} D_{qq}^{-1} A_{qq}^T + A_{qq+1} D_{qq+1}^{-1} A_{qq+1}^T, \quad q=1, \dots, Q$$

と書ける。そこで, $L=Q$ とおき, $G_\ell = M_{\ell\ell}$ として部分問題 $\text{DSP}_\ell^{(k)}$ を構成する。

行列 A の行方向のブロック ($A_{qq-1} A_{qq} A_{qq+1}$) を $A_q \in R^{\bar{m} \times (2\bar{n}_1 + \bar{n}_2)}$ と書く。数値実験では, ブロックの数を表す Q の値を 8 に固定し, 各ブロックのサイズを規定する $\bar{m}$, $\bar{n}_1$, $\bar{n}_2$, および A_q , $q=1, \dots, 8$, の非零要素数を Table 1 のように選んでテスト問題を生成した。パラメータ N の値を 1 から 8 まで変化させてアルゴリズム

Table 1 Characteristics of the test problems with staircase structure

Prob.	Each Block			Non-zeros		Matrix A	
	$\bar{m}$	$\bar{n}_1$	$\bar{n}_2$	A_1	$A_q, q>1$	Rows	Cols.
QP11	128	56	442	8192	8192	1024	4040
QP12	128	56	442	16384	16384	1024	4040
QP13	128	56	442	16384	8192	1024	4040
QP14	128	56	442	32768	8192	1024	4040
QP21	512	56	442	8192	8192	4096	4040
QP22	512	56	442	16384	16384	4096	4040
QP23	512	56	442	16384	8192	4096	4040
QP24	512	56	442	32768	8192	4096	4040

ム **ABP** を実行した結果を Fig. 2 に示す。なお, 破線は単独マスタモデルによる実行時間, 実線は二重マスタモデルによる実行時間である。横軸の右端 $N=8$ は, 同期型のアルゴリズム **BP** を実行した場合に相当する。

並列アルゴリズムの性能を評価する指標としては, 並列化効率がよく知られている¹⁾。これは, 基準となる逐次アルゴリズムの実行時間を, 並列アルゴリズムの実行時間と使用したプロセッサ数の積で割ったものである。そこで, 問題 QP22, QP23 について, 対応する双対問題 (2) をブロック分割せずに CM-5 の一つのプロセッサ上で前処理つき共役勾配法を用いて解かせた場合の実行時間を測定し, これを基準として並列化効率を計算した。その結果を Table 2 に示す。

一般に, 並列アルゴリズムを実際の問題に適用した場合の性能は, アルゴリズムそのものの能力のほかに, 並列計算機における実行方法にも左右される。数値実験結果は, いずれのテスト問題に対しても, また, どのような N の値に対しても, 二重マスタモデルによるアルゴリズムの実行時間が単独マスタモデルの場合と同等またはそれ以下になることを示している。この傾向は N の値が小さくなるほど顕著であり, とくに $N=1$ の場合には, 使用するプロセッサ数が一つ多いにもかかわらず, 二重マスタモデルの方が並列化効率も高くなっている。このことから, 二重マスタモデルは非同期アルゴリズムの有効な実行方法であることがわかる。

なお, QP11, QP12, QP21, QP22 のように, すべての q に対して A_q の非零要素数が同じ問題では, 二重マスタモデルを用いてもなお, 非同期性の導入に伴って実行時間が増加傾向を示す場合がある。これらはいずれも構造的に均一なテスト問題であり, もともと各プロセッサの遊休時間は少ない。このため, 反復回数の増加によるデメリットのみが表面に現れたものと考えられる。一方, QP13, QP14, QP23, QP24 は, A_1 の非零要素数のみが多い問題である。このような問題においては, N の減少に伴って各プロセッサの遊休時間が減少し, 実行時間が大幅に短縮されたものと考えられる。

5.2 結合制約つきのブロック対角型制約条件

つぎに, 原問題の制約条件の係数行列 A が, 多品種流問題においてあらわれるようなブロック構造

(a) Problem QP11

(e) Problem QP21

(b) Problem QP12

(f) Problem QP22

(c) Problem QP13

(g) Problem QP23

(d) Problem QP14

(h) Problem QP24

Fig. 2 Performance of the proposed algorithm for problems with staircase structure

$$A = \begin{pmatrix} A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_Q \\ \bar{A}_1 & \bar{A}_2 & \cdots & \bar{A}_Q \end{pmatrix}$$

をもつ場合を考える。ただし、 $A_q, q=1, \dots, Q$, はいずれも $\bar{m} \times \bar{n}$ 行列、また $\bar{A}_q, q=1, \dots, Q$, はいずれも $\bar{n} \times \bar{n}$ 行列である。行列 A の構造に対応して、対角行列 D を

$$D = \text{diag}\{D_1, D_2, \dots, D_Q\}$$

と書く。このとき、双対問題 (2) の係数行列 M も特徴的なブロック構造を示し、そのブロック対角部分は

$$M_q = A_q D_q^{-1} A_q^T, \quad q=1, \dots, Q,$$

$$M_{Q+1} = \sum_{q=1}^Q \bar{A}_q D_q^{-1} \bar{A}_q^T$$

と書ける。そこで、 $L=Q+1$ とおき、 $G_\ell = M_\ell$ として部分問題 $\text{DSP}_\ell^{(k)}$ を構成する。

問題 (1) の結合制約の係数行列 $(\bar{A}_1 \cdots \bar{A}_Q)$ を $\bar{A} \in R^{\bar{n} \times Q\bar{n}}$ と書く。数値実験では、 A_q の非零要素数を 8192、列数 $\bar{n}$ を 512 に固定し、行数 $\bar{m}$ 、ブロック数 Q 、および $\bar{A}$ の非零要素数を Table 3 のように選んでテスト問題を生成した。パラメータ N の値を変化させてアルゴリズム **ABP** の実行時間を測定した結果を Fig. 3 に示す。

Table 3 Characteristics of the test problems with block angular structure

Prob.	Q	Each Block		Non-zeros		Matrix A	
		$\bar{m}$	$\bar{n}$	A_q	$\bar{A}$	Rows	Cols.
QP31	3	128	512	8192	4096	896	1536
QP32	3	128	512	8192	8192	896	1536
QP33	7	128	512	8192	8192	1408	3584
QP34	7	128	512	8192	16384	1408	3584
QP41	7	256	512	8192	8192	2304	3584
QP42	7	256	512	8192	16384	2304	3584
QP43	15	256	512	8192	16384	4352	7680
QP44	15	256	512	8192	32768	4352	7680

まず、単独マスタモデルに比べて二重マスタモデルの計算時間が短いことは、階段型の場合と同様である。二重マスタモデルを用いた場合、パラメータ N の減少に伴って計算時間が増加傾向にある例は問題 QP33 や QP44 だけであり、問題 QP41 においては、 N を小さくしたときに非常に良好な結果が得られている。問題 QP41 について、ワーカーノード 1 と $L=8$ の実行状況を詳細に測定した結果を Table 4 に示す。表の # Iterations 欄は、各ワーカーノードにおいて部分問題が解かれた回数である。Table 4

より、結合制約に対応する双対変数を受け持つワーカーノード L が遊休状態にある時間の割合は、 $N=8, 4, 2, 1$ となるにつれて、順に 45.9%, 33.2%, 18.9%, 2.9% のように急激に減少していることがわかる。ワーカーノード 1 が遊休状態にある時間は、処理時間全体の 10% 前後の低い水準で推移していることに注意すると、アルゴリズムを非同期化することによって問題全体に対して影響力を持つワーカーノード L の遊休状態が解消し、非常に大きな効果が得られたものと考えられる。

Table 4 Details of the results for problem QP41

N		1	2	4	8
Elapsed Time [s]		160.5	167.7	171.8	183.0
Worker	Idle Time [s]	4.5	12.8	18.5	30.9
Node 1	# Iterations	110.8	106.4	103.4	83.8
Worker	Idle Time [s]	4.6	31.1	57.1	83.9
Node 8	# Iterations	186.6	132.2	111.0	83.8

一方、実行時間の変化が V 字型を示した問題 QP32 について、アルゴリズムの実行状況を詳細に測定した結果を Table 5 に示す。表からわかるように、非同期性の導入によって、ワーカーノード 1 において部分問題が解かれる回数が急激に増加するのに対して、ワーカーノード $L=4$ においては部分問題を解く回数が増えしていない。このような場合には、 N の値を必要以上に小さくすると、ワーカーノード L が時々与える情報によって他のワーカーノードにおいて頻繁に更新されていた情報が過去の時点にまで引き戻されてしまう可能性がある。そのため、非同期的に実行した多くの反復が十分に活かされず、収束性の改善がみられなかったものと考えられる。

Table 5 Details of the results for problem QP32

N		1	2	4
Elapsed Time [s]		123.2	122.0	124.7
Worker	Idle Time [s]	10.7	22.1	38.9
Node 1	# Iterations	367.6	244.6	70.4
Worker	Idle Time [s]	1.5	2.5	4.3
Node 4	# Iterations	84.2	76.0	70.4

6. おわりに

本論文では、分離可能な 2 次計画問題に対するブロック並列アルゴリズムを非同期型に拡張した。提案したアルゴリズムは、SPMD 型の計算モデルを用いて並列計算機で容易に実行できる。コネクションマシン CM-5 における数値実験から、個々のワーカーノードの負荷が一樣でないような問題に対しては、探索点の更新と収束判定を別々のプロセッサに担当させる二重マスタモデルを用

(a) Problem QP31

(e) Problem QP41

(b) Problem QP32

(f) Problem QP42

(c) Problem QP33

(g) Problem QP43

(d) Problem QP34

(h) Problem QP44

Fig. 3 Performance of the proposed algorithm for problems with block angular structure

いることによって、同期型のアルゴリズムに比べて実行時間が短縮されることが確かめられた。

提案したアルゴリズムの性能は、Tables 4, 5 からわかるように、非同期性の導入による遊休状態の解消がもたらすメリットと、探索点の更新が非効率的となって反復回数が増加することによるデメリットの相互関係により決定される。そのため、問題の構造によって最も適切な N の値があらかじめ求められれば、アルゴリズムの有効性を十分に生かすことができるものと考えられる。また、同期型のブロック並列アルゴリズムに対して、その実際的な収束性を高めるために、部分問題を解くための前処理つき共役勾配法の収束条件を緩和する方策が試みられている¹⁴⁾が、本論文で提案した非同期アルゴリズムにおいてもそのような方法が適用できるかどうかの検討は今後の課題である。

参考文献

- 1) D. P. Bertsekas and J. N. Tsitsiklis: *Parallel and Distributed Computation, Numerical Methods*, Prentice-Hall (1989)
- 2) R. J. Chen and R. R. Meyer: Parallel optimization for traffic assignment; *Mathematical Programming*, Vol. 42, pp. 327–345 (1988)
- 3) R. De Leone, O. L. Mangasarian and T.-H. Shiau: Multi-sweep asynchronous parallel successive overrelaxation for the nonsymmetric linear complementarity problem; *Annals of Operations Research*, Vol. 22, pp. 43–54 (1990)
- 4) J. Eckstein: The alternating step method for monotropic programming on the Connection Machine CM-2; *ORSA Journal on Computing*, Vol. 5, pp. 84–96 (1993)
- 5) J. Eckstein and M. Fukushima: Some reformulations and applications of the alternating direction method of multipliers; *Large Scale Optimization: State of the Art* (W. W. Hager, D. W. Hearn and P. M. Pardalos, Eds.), Kluwer Academic, pp. 115–134 (1994)
- 6) M. Fukushima, M. Haddou, V. H. Nguyen, J.-J. Strodiot, T. Sugimoto and E. Yamakawa: A parallel descent algorithm for convex programming; *Computational Optimization and Applications*, Vol. 5, pp. 5–37 (1996)
- 7) M. R. Hestenes: *Conjugate Direction Methods in Optimization*, Springer-Verlag (1980)
- 8) I. J. Lustig and G. Li: An implementation of a parallel primal-dual interior point method for block-structured linear programs; *Computational Optimization and Applications*, Vol. 1, pp. 141–161 (1992)
- 9) O. L. Mangasarian and R. De Leone: Parallel gradient projection successive overrelaxation for symmetric linear complementarity problems; *Annals of Operations Research*, Vol. 14, pp. 41–59 (1988)
- 10) S. S. Nielsen and S. A. Zenios: Massively parallel algorithms for singly constrained convex programs; *ORSA Journal on Computing*, Vol. 4, pp. 166–181 (1992)
- 11) Thinking Machines Corporation: *CM Fortran Programming Guide, Version 2.1* (1994)
- 12) Thinking Machines Corporation: *CM-5 CM Fortran Performance Guide, Version 2.1* (1994)
- 13) Thinking Machines Corporation: *CMMD Reference Manual, Version 3.0* (1993)
- 14) E. Yamakawa and M. Fukushima: A block-parallel conjugate gradient method for separable quadratic programming problems; *Journal of the Operations Research Society of Japan*, Vol. 39, pp. 407–427 (1996)
- 15) S. A. Zenios and Y. Censor: Massively parallel row-action algorithms for some nonlinear transportation problems; *SIAM Journal on Optimization*, Vol. 1, pp. 373–400 (1991)