TR - H - 135

ブロック構造を持つ 2次計画問題に対する 非同期並列型共役勾配法

山川　栄樹　　牧　英一

(奈良先端科学技術大学院大学)

1995. 3. 13

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　☎ 0774-95-1011

ATR Human Information Processing Research Laboratories

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

Telephone: +81-774-95-1011

Facsimile: +81-774-95-1008

ブロック構造を持つ2次計画問題に対する非同期並列型共役勾配法

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

牧 英一
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

1995 年 3 月 13 日

1 はじめに

並列処理が身近なものとなりつつある中で、数理計画問題に対しても既に多くの並列アルゴリズムが提案されている [2, 3] 。その多くは、数理計画における双対理論と関数 (行列) 分割の考え方を基礎としており、もとの問題の近似モデルを複数の独立な部分問題に分割し、各反復でこれらを並列的に解くことによって原問題の解に至るような点列を生成しようとするものである。特に、主問題あるいは双対問題の変数を単位とするような粒度の細かい分割が行われ、生成された部分問題の解が解析的に求められるような場合には、最近盛んに開発が進められている超並列計算機上での実行に適したデータ並列型のアルゴリズムが得られる。一方、現実の大規模問題が持つ構造的な特徴を利用して、効率的な並列アルゴリズムを得ることも可能である。文献 [8] では、分離可能な目的関数を持つ2次計画問題に対して、双対問題の目的関数の係数行列をブロック対角行列を用いて分割し、その結果得られる複数の小さな部分問題を共役勾配法を用いて解くようなブロック並列型のアルゴリズムが提案されている。多期間計画問題や多品種流問題などのように、もとの問題の制約条件の係数行列がある種のブロック構造を持つ場合には、その双対問題の目的関数の係数行列も何らかのブロック構造を示す。従って、この構造に沿う形で双対問題を分割すれば、比較的少ない反復数で収束するような並列アルゴリズムを得ることができる。このアルゴリズムは、もとの問題のブロックを単位とする比較的粗い分割に基づいており、各部分問題をそれぞれ反復解法を用いて解く必要があるため、コントロール並列型のアルゴリズムとなる。既に、コネクション・マシン CM-5 をはじめとして、 MIMD (Multiple-Instruction Multiple-Data) 型の並列計算機もいくつか商品化されており、コントロール並列型のアルゴリズムの実行も現実のものとなっている。特に CM-5 は、比較的少数のベクトル計算機を構成要素とするような並列計算機であり、粗い粒度で分割された各部分問題にはそれぞれデータ並列型の計算モデルを、アルゴリズム全体に対してはコントロール並列型の計算モデルを適用することができるため、非常に良好な結果を得ることができる。

コントロール並列型のアルゴリズムを並列計算機上で実行する際には、各部分問題を解くことに専念するいくつかの処理装置の他に、アルゴリズム全体の進捗管理および各処理装置への仕事の分配を行うもう一つの処理装置を置くという計算モデルを用いる場合が多い。何人かの部下を使って管理職が仕事を進める形態に似ていることから、前者の各処理装置はワーカー・ノード，後者の処理装置はマスター・ノードと呼ばれる。文献 [8] で提案されたブロック並列型のアルゴリズムでは、ワーカー・ノード群が計算する各部分問題の解がすべて揃うのを待って、マス

ター・ノードは収束判定と次の探索点における部分問題の構成を行う。従って、もとの問題のブロック構造が一様ではないなどの理由により各ワーカー・ノードが部分問題を解くために要する時間に大きな差が生じる場合には、遊休状態となる処理装置が多数発生して、並列化効率が著しく低下する可能性がある。

そこで、本論文では、すべてのワーカー・ノードの処理について同期をとることを強制しないような非同期並列型のアルゴリズムを提案する。すなわち、マスター・ノードは、いくつかのワーカー・ノードが部分問題を解き終わった段階で探索点を更新して部分問題を構成し直し、遊休状態となったワーカー・ノードにこれらを分配して探索を再開させる。また、アルゴリズムの収束判定は、マスター・ノードが探索点を更新するたびに行うこととし、収束したと判定された場合には、現在稼働中のワーカー・ノードが遊休状態となった時点でアルゴリズム全体を終了させることにする。なお、マスター・ノードが収束判定を行っている間に各ワーカー・ノードが遊休状態に陥る可能性も考慮して、マスター・ノードの機能を二つの処理装置に分割し、部分問題の再構成を行う第1のマスター・ノードと収束判定のみを行う第2のマスター・ノードを用いるモデルについても検討を行う。

以下では、まず2章において、基礎となるブロック並列型の共役勾配法のアルゴリズムを紹介する。次に3章では、このアルゴリズムを非同期並列型に拡張する。4章においては、非同期並列型の共役勾配法を実行するための計算モデルについて述べる。並列計算機 CM-5 上で行った数値実験の結果は、5章に掲げる。最後に6章において、本論文のまとめを行う。

なお、本論文では次のような表記法を用いる。J をある添字集合とする時、あるベクトル u に対してその要素 $u_i, i \in J$, により構成される部分ベクトルを u_J で表す。また、ある行列 A に対してその行 $a_i^\top, i \in J$, から成る部分行列を A_J と記す。

2 ブロック並列型の共役勾配法

次の2次計画問題について考える。

$$\begin{array}{lll} \text{目的関数：} & \dfrac{1}{2} x^\top D x + c^\top x & \rightarrow \quad \text{最小} \\ \text{制約条件：} & a_i^\top x = b_i, & i \in E, \\ & a_i^\top x \leq b_i, & i \in I. \end{array} \tag{1}$$

ただし、E, I は有限な添字集合であり、D は $n \times n$ 対角行列, c と a_i は n 次元実ベクトル, b_i は実数である。今、$a_i^\top$ を第 i 行に持つような行列を A, b_i を第 i 要素に持つようなベクトルを b と書く。このとき、問題 (1) の双対問題は

$$\begin{array}{lll} \text{目的関数：} & \dfrac{1}{2} z^\top M z + q^\top z & \rightarrow \quad \text{最小} \\ \text{制約条件：} & z_i \geq 0, & i \in I, \end{array} \tag{2}$$

と定義される。ただし、

$$\begin{aligned} M &= AD^{-1}A^\top, \\ q &= AD^{-1}c + b \end{aligned}$$

である。問題 (1) が実行可能ならばそれは唯一の最適解を持ち、双対定理により問題 (2) も解を持つ。また、問題 (1) の最適解を x^* とすると、問題 (2) の解 z^* との間には次のような関係が成り立つ。

$$x^* = -D^{-1}\left(A^\top z^* + c\right). \tag{3}$$

点 z における双対問題 (2) の目的関数の最急降下方向を r とすると、

$$r = -M z - q$$

である。このとき、問題 (2) の 1 次の最適性条件は

$$\begin{aligned} &r_i = 0, && i \in E, \\ &r_i \le 0, \quad z_i \ge 0, \quad r_i z_i = 0, && i \in I, \end{aligned} \tag{4}$$

と表すことが出来る。行列 M は半正定値対称なので、条件 (4) は問題 (2) に対する必要かつ十分な最適性条件である。

ブロック並列型の共役勾配法 [8] は、問題 (2) に対して行列分割法を適用することによって導かれる。行列 M に対して

$$M = G + H$$

を満たすような行列の対 (G, H) は M の分割と呼ばれる。特に、$G - H$ が正定値であるとき、分割 (G, H) は正則 (regular) であると言われ、M が半正定値ならば G は正定値となることがわかる。そこで、対称行列 G を用いて定義される問題 (2) の係数行列 M の正則な分割 (G, H) を用いて、現在の点 $z^{(k)}$ において次のような部分問題を解くことを考える。

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

ただし、

$$v^{(k)} = H z^{(k)} + q \tag{5}$$

である。このとき、もとの問題 (1) が実行可能ならば、部分問題 $\mathrm{DSP}^{(k)}$ の解を $z^{(k+1)}$ として生成した点列 $\{z^{(k)}\}$ は問題 (2) の最適解の一つに収束することが示される [8, Theorem 1]。

行列 G としてブロック対角行列を選ぶと、部分問題 $\mathrm{DSP}^{(k)}$ を複数の小さな問題に分割することができる。実際、$J_\ell, \ell = 1, \cdots, L$, を

$$\bigcup_{\ell=1}^{L} J_\ell = E \cup I \qquad \text{and} \qquad J_\ell \cap J_{\ell'} = \emptyset, \quad \ell \ne \ell',$$

をみたす添字集合とする。そして、必要に応じて行と列を並べ換えることにより、行列 M の正則な分割 (G, H) を構成する G として、次のようなブロック対角行列を考える。

$$G = \begin{pmatrix} G_1 & 0 & \cdots & 0 \\ 0 & G_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & G_L \end{pmatrix}.$$

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 $\mathrm{DSP}^{(k)}$ は次のような L 個の独立な部分問題に分割することができる。

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

ここで、$v_{J_\ell}^{(k)}$ は (5) で定義されるベクトル $v^{(k)}$ の部分ベクトルである。行列 G の正定値性より G_ℓ も正定値となり、各部分問題 $\mathrm{DSP}_\ell^{(k)}$ はすべての k に対して唯一の解を持つことがわかる。

式 (3) に対応して、現在の探索点 $z^{(k)}$ における主問題 (1) の解の推定値を

$$x^{(k)} = -D^{-1}\left(A^\top z^{(k)} + c\right) \tag{6}$$

とおく。さらに、点 $x^{(k)}$ における問題 (1) の制約条件の残差ベクトルを $r^{(k)}$ とすると、

$$r^{(k)} = A\,x^{(k)} - b \tag{7}$$

と書けるが、簡単な計算から

$$r^{(k)} = -M\,z^{(k)} - q$$

となることがわかる。この時、(5) で定義されるベクトル $v^{(k)}$ は

$$v^{(k)} = -G\,z^{(k)} - r^{(k)}$$

と書き直せ、G がブロック対角行列であることから、

$$v_{J_\ell}^{(k)} = -G_\ell\,z_{J_\ell}^{(k)} - r_{J_\ell}^{(k)}, \quad \ell = 1, \cdots, L, \tag{8}$$

を得る。すなわち、各部分問題 $\mathrm{DSP}_\ell^{(k)}$ は、ブロック ℓ に関係する情報のみを用いて定義することができる。

結局、問題 (1) に対するブロック並列型の共役勾配法のアルゴリズムは、次のように書くことができる。

アルゴリズム BP :

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

ステップ 2. 次式により残差ベクトル $r^{(k)}$ を計算する。

$$\begin{aligned} x^{(k)} &:= -D^{-1}\left(A^\top z^{(k)} + c\right), \\ r^{(k)} &:= A\,x^{(k)} - b. \end{aligned}$$

ステップ 3. $z^{(k)}$ と $r^{(k)}$ が条件 (4) を満たすならば終了する。

ステップ 4. $\ell = 1, \cdots, L$ に対して、部分問題 $\mathrm{DSP}_\ell^{(k)}$ を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並列的に解き、解 $z_{J_\ell}^{(k+1)}$ を得る。

ステップ 5. $k := k + 1$ としてステップ 2 へ戻る。 □

アルゴリズム BP がステップ 3 で終了したとき、$z^{(k)}$ と $x^{(k)}$ は、それぞれ問題 (2) と (1) の最適解である。ステップ 4 において適用する前処理つき共役勾配法の詳細な手続きについては、文献 [8, §3] を参照されたい。

3 非同期並列型への拡張

以下では、アルゴリズムを実行する並列計算機は十分な数の処理装置を保持しており、各部分問題を個々の処理装置に独占的に割り当てることができるような理想的な状況を想定する。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 $\mathrm{DSP}_\ell^{(k)}$ を並列的に解くことが可能である反面、すべての部分問題の解が得られない限りステップ 5 に進むことができないという意味で、同期型の並列アルゴリズムである。従って、特定の ℓ に対する部分問題を解くために非常に多くの処理時間を要するような場合には、部分問題を解き終わった他の処理装置が遊休状態となる期間が長期化し、アルゴリズム全体の効率が低下してしまう可能性がある。そこで、一部の部分問題の解が得られたならば、現時点で利用できる最新の情報を用いて部分問題を構成し直し、遊休状態に至った処理装置に対して直ちに探索の実行を指示するという非同期並列型のアルゴリズムを考えることにする。

各反復で、ブロック ℓ に対応する部分問題 $\mathrm{DSP}_\ell^{(\cdot)}$ の解を求めるための前処理つき共役勾配法の手続きを実行する処理装置に注目しよう。第 k_ℓ 反復が次のように記述されるアルゴリズムは、典型的な非同期型のアルゴリズムとなる。$z_i^{(k_\ell)} \in R, i \in E$, および $z_i^{(k_\ell)} \geq 0, i \in I$, を満たすような点 $z^{(k_\ell)}$ が与えられたとき、この処理装置は式 (6), (7) に基づいて残差ベクトル $r^{(k_\ell)}$ を計算する。$z^{(k_\ell)}$ と $r^{(k_\ell)}$ が条件 (4) を満たさないならば、式 (8) に従って $v_{J_\ell}^{(k_\ell)}$ を計算し、部分問題 $\mathrm{DSP}_\ell^{(k_\ell)}$ を前処理つき共役勾配法の手続きを用いて解く。得られた解を $\overline{z}_{J_\ell}$ と書くとき、

$$z_i^{(k_\ell+1)} := \begin{cases} \overline{z}_i & i \in J_\ell, \\ \tilde{z}_i & i \notin J_\ell, \end{cases}$$

とし、$k_\ell := k_\ell + 1$ とおいて直ちに次の反復に入る。ただし、$\tilde{z}_{J_{\ell'}}, \ell' \neq \ell$, はその時点で他のブロックにおいて得られている最新の双対変数 z の値である。

このアルゴリズムでは、部分問題を解き終わった処理装置には、直ちに次の部分問題が与えられるため、遊休状態となる時間は非常に短くなる。しかしながら、最も早く部分問題を解き終わった処理装置は、自分自身が担当するブロックの情報のみを更新した部分問題を解き直すことになるため、反復回数がかさむだけでさほど大きな効果が得られない可能性がある。また、最も遅く部分問題を解き終わった処理装置は、既に繰り返し更新を行った他のブロックの双対変数の値を用いて次の部分問題を構成することになり、古い情報をもとに求めた当該ブロックの解が他のブロックの情報との間で不整合をおこすおそれもある。さらに、各処理装置が残差ベクトル $r^{(k_\ell)}$ を個別に計算して探索の継続を判断することは、明らかに非効率であると考えられる。

そこで、ある整数 $N \in [1, L]$ に対して、新たに N 個の部分問題が解き終わった段階で、これらの部分問題を担当していた処理装置に解かせるべき部分問題を構成し直し、対応する処理装置にそれぞれ探索の再開を指示することにする。問題 (1) に対する非同期並列型の共役勾配法のアルゴリズムは次のように記述される。

アルゴリズム AP :

ステップ 1. 整数 $N \in [1, L]$ を定め、初期探索点 $z^{(1)}$ を $z_i^{(1)} \in R, i \in E$, および $z_i^{(1)} \geq 0, i \in I$, を満たすように選ぶ。集合 S を

$$S := \{1, \cdots, L\}$$

と設定し、$k_\ell := 1, \ell = 1, \cdots, L$, および $k := 1$ とおく。

ステップ 2. 次式により残差ベクトル $r_{J_\ell}^{(k_\ell)}$, $\ell \in S$, を計算する。

$$\begin{aligned} x^{(k)} &:= -D^{-1}\left(\sum_{\ell=1}^{L} A_{J_\ell}^{\top} z_{J_\ell}^{(k_\ell)} + c\right), \\ r_{J_\ell}^{(k_\ell)} &:= A_{J_\ell} x^{(k)} - b_{J_\ell}, \qquad\qquad \ell \in S. \end{aligned}$$

ステップ 3. $z_{J_\ell}^{(k_\ell)}$, $\ell = 1, \cdots, L$, と $r_{J_\ell}^{(k_\ell)}$, $\ell = 1, \cdots, L$, が条件 (4) を満たすならば終了する。

ステップ 4. $\ell \in S$ に対して、部分問題 $\mathrm{DSP}_\ell^{(k_\ell)}$ の解 $z_{J_\ell}^{(k_\ell+1)}$ を求めるための前処理つき共役勾配法の手続きを起動する。

ステップ 5. 集合 S を次のように設定し直す。

$$S := \left\{ \ell \;\middle|\; \text{部分問題 } \mathrm{DSP}_\ell^{(k_\ell)} \text{ を解くための前処理つき共役勾配法の手続きが終了} \right\}.$$

ステップ 6. $|S| < N$ ならばステップ 5 へ戻る。さもなければ、$k_\ell := k_\ell + 1$, $\ell \in S$, および $k := k + 1$ としてステップ 2 へ戻る。 □

この章の初めに述べた非同期並列アルゴリズムのプロトタイプは、アルゴリズム AP において $N = 1$ とした場合に相当する。一方、$N = L$ とおくと、アルゴリズム AP は同期型の並列アルゴリズム BP と全く同じ動作をする。また、ステップ 5 における前処理つき共役勾配法の手続きの終了は、部分問題の解 $z_{J_\ell}^{(k_\ell+1)}$ がブロック ℓ を担当する処理装置から他の処理装置へ伝送された時点で検出される。一般に処理装置間での情報伝達は直列に行われるため、ステップ 6 からステップ 2 へ戻る際の集合 S の要素数は、常に丁度 N 個となる。

4 並列計算機上での実行方法

この章では、非同期並列型の共役勾配法のアルゴリズム AP を並列計算機上で実行するための計算モデルについて考える。ステップ 4 において、部分問題 $\mathrm{DSP}_\ell^{(k_\ell)}$ を解くための前処理つき共役勾配法の手続きが各 $\ell \in S$ について独立に起動されることから、アルゴリズム AP はコントロール並列型のアルゴリズムと見なすことができる。従って、各ブロック ℓ に対して処理装置を一つづつ割当て、それぞれにおいて前処理つき共役勾配法の手続きを独立に実行させるという方式が、最も自然な並列計算モデルである。ただし、ステップ 2 における $x^{(k)}$ の計算やステップ 3 の収束判定は、L 個のブロック全体の情報を用いて行う必要がある。そこで、もう一つの特別な処理装置を用意して、ブロックごとに分割することができないこれらの処理を行わせることにする。以下では、アルゴリズム全体の動きを調整する役割を果たす後者の処理装置をマスター・ノード，各ブロックに対応する部分問題を解くことに専念する前者の処理装置をワーカー・ノードと呼ぶことにする。各ワーカー・ノードは、マスター・ノードから解くべき部分問題を定義する情報を受け取った時点で稼働し始め、得られた解をマスター・ノードへ送り返した後に遊休状態となる。すなわち、この計算モデルにおいては、ワーカー・ノードが行う通信の相手先はマスター・ノードのみに限定される。また、各ワーカー・ノードが実行する前処理つき共役勾配法の手続きは全く同一のものである。従って、アルゴリズム AP は、多くの MIMD 型の並列計算機において採用されている SPMD (Single-Program Multiple-Data) 型の計算モデルを用いて比較的容易に実現することができる。

上で述べた計算モデルは、アルゴリズム BP の構造に沿った非常に自然な実行方法であるが、N の値が小さい非同期並列アルゴリズムの場合は、マスター・ノードに対するメッセージの受渡しが頻繁に発生するため、マスター・ノードの負荷が大きくなって並列化効率が低下しないように十分な注意を払う必要がある。実際、マスター・ノードがステップ 2, 3 を実行している間は、ワーカー・ノード $\ell \in S$ は遊休状態となる。特に、ステップ 2 で現れる行列とベクトルの二通りの積 $A_{J_\ell}^{\top} z_{J_\ell}^{(k_\ell)}$ および $A_{J_\ell} x^{(k)}$ の計算には多くの処理時間が必要である。そこで、同期型のアルゴリズム AP の場合 [8, §4] と同様に、各ワーカー・ノードは、部分問題の解 $z_{J_\ell}^{(k_\ell+1)}$ の代わりに $D^{-1}A_{J_\ell}^{\top} z_{J_\ell}^{(k_\ell+1)}$ の値を計算した上でマスター・ノードに送ることにする。このとき、マスター・ノードは、次の反復のステップ 2 において、これらと $D^{-1}c$ の和をとることによって $x^{(k)}$ の値を簡単に求めることができる。さらに、マスター・ノードは $x^{(k)}$ の値そのものをワーカー・ノード $\ell \in S$ に伝送し、残差ベクトル $r_{J_\ell}^{(k_\ell)}$, $\ell \in S$, の値は各ワーカー・ノードにおいて計算することにする。一方、収束判定のステップ 3 についても、まず各ワーカー・ノードにおいて $z_{J_\ell}^{(k_\ell)}$ と $r_{J_\ell}^{(k_\ell)}$ が条件 (4) のブロック ℓ に対応する部分

$$\begin{aligned} &r_i = 0, && i \in (E \cap J_\ell), \\ &r_i \le 0, \quad z_i \ge 0, \quad r_i z_i = 0, && i \in (I \cap J_\ell), \end{aligned} \tag{9}$$

を満たすかどうかを判定し、その結果をマスター・ノードに送る。マスター・ノードは、すべてのワーカー・ノードにおいて条件 (9) が成り立ったことを確認した時点で、アルゴリズムを停止させればよい。

アルゴリズム AP は、ブロック ℓ を担当するワーカー・ノード上において次のような手続きとして実行される。

手続き WORKER$_\ell$:

ステップ 1. 初期探索点 $z_{J_\ell}^{(1)}$ を $z_i^{(1)} \in R, i \in (E \cap J_\ell)$, および $z_i^{(1)} \ge 0, i \in (I \cap J_\ell)$, を満たすように選び、$k_\ell := 1$ とおく。

ステップ 2. マスター・ノードから送られてきた $x^{(\cdot)}$ の値を読み込み、次式により残差ベクトル $r_{J_\ell}^{(k_\ell)}$ を計算する。

$$r_{J_\ell}^{(k_\ell)} := A_{J_\ell} x^{(\cdot)} - b_{J_\ell}.$$

ステップ 3. 次式により収束判定結果 $\delta_\ell^{(k_\ell)}$ を定め、マスター・ノードにその値を送る。

$$\delta_\ell^{(k_\ell)} := \begin{cases} 1 & z_{J_\ell}^{(k_\ell)} \text{ と } r_{J_\ell}^{(k_\ell)} \text{ は条件 (9) を満たす}, \\ 0 & \text{上記以外}. \end{cases}$$

ステップ 4. 部分問題 $\mathrm{DSP}_\ell^{(k_\ell)}$ を前処理つき共役勾配法を用いて解き、解 $z_{J_\ell}^{(k_\ell+1)}$ を得る。

ステップ 5. $D^{-1}A_{J_\ell}^{\top} z_{J_\ell}^{(k_\ell+1)}$ の値を計算し、マスター・ノードに送る。

ステップ 6. $k_\ell := k_\ell + 1$ としてステップ 2 へ戻る。 □

容易に確かめられるように、点 $z_{J_\ell}^{(k_\ell)}$ における部分問題 $\mathrm{DSP}_\ell^{(k_\ell)}$ の目的関数の最急降下方向は、$r_{J_\ell}^{(k_\ell)}$ となる。従って、ステップ 3 において $\delta_\ell^{(k_\ell)} = 1$ となった場合には、$z_{J_\ell}^{(k_\ell)}$ は部分問題 $\mathrm{DSP}_\ell^{(k_\ell)}$ の最適性条件も満たし、$z_{J_\ell}^{(k_\ell+1)} = z_{J_\ell}^{(k_\ell)}$ となる。手続き WORKER_ℓ は、マスター・ノードからメッセージが送られなくなった時点で終了する。

一方、マスター・ノードにおけるアルゴリズム AP の動きは、次のように記述できる。

アルゴリズム MASTER :

ステップ 1. 整数 $N \in [1, L]$ を定め、初期探索点 $z^{(1)}$ を $z_i^{(1)} \in R$, $i \in E$, および $z_i^{(1)} \geq 0$, $i \in I$, を満たすように選ぶ。集合 S を

$$S := \{1, \cdots, L\}$$

と設定し、$k_\ell := 1, \ell = 1, \cdots, L$, および $k := 1$ とおく。

ステップ 2. 次式により問題 (1) の解の推定値 $x^{(k)}$ を計算する。

$$x^{(k)} := -\sum_{\ell=1}^{L} D^{-1} A_{J_\ell}^{\mathsf{T}} z_{J_\ell}^{(k_\ell)} - D^{-1} c.$$

すべての $\ell \in S$ に対して、ワーカー・ノード ℓ に $x^{(k)}$ の値を送る。

ステップ 3. すべての $\ell \in S$ に対して、ワーカー・ノード ℓ から送られてきた $\delta_\ell^{(k_\ell)}$ の値を読み込む。

ステップ 4. $\delta_\ell^{(k_\ell)} = 1, \ell = 1, \cdots, L$, ならば、すべてのワーカー・ノードから送られてくるメッセージを受け取って終了する。さもなければ、$S := \emptyset$ とおく。

ステップ 5. あるワーカー・ノード $\ell \notin S$ から $D^{-1} A_{J_\ell}^{\mathsf{T}} z_{J_\ell}^{(k_\ell+1)}$ が送られてきたならばその値を読み込み、次式により集合 S を更新する。

$$S := S \cup \{\ell\}.$$

ステップ 6. $|S| < N$ ならばステップ 5 へ戻る。さもなければ、$k_\ell := k_\ell + 1, \ell \in S$, および $k := k + 1$ としてステップ 2 へ戻る。 □

アルゴリズム MASTER のステップ 1 で設定する $z^{(1)}$ の値は、各ワーカー・ノードで実行される手続き $\mathrm{WORKER}_\ell, \ell = 1, \cdots, L$, のステップ 1 における初期探索点 $z_{J_\ell}^{(1)}$ の値と整合性がとれていなければならない。$L = 2$, $N = 1$ の場合について、各処理装置間での情報の流れの一例を図 1 に示す。図からわかるように、マスター・ノードがいずれか一方のワーカー・ノードに $x^{(\cdot)}$ の値を伝送した後、そのワーカー・ノードがマスター・ノードに収束判定結果 $\delta^{(k,)}$ を送り返すまでの間は、部分問題を解き終わった他のワーカー・ノードに対する処理は行われない。これは、アルゴリズム MASTER のステップ 3 において、集合 S の要素に対応するすべてのワーカー・ノード ℓ から収束判定結果 $\delta_\ell^{(k_\ell)}$ が送られて来ない限り、次のステップに進めないからである。

このような無駄時間を少なくするためには、ワーカー・ノード $\ell \in S$ から $\delta_\ell^{(k_\ell)}$ の値が送られてきた任意の時点でアルゴリズム MASTER のステップ 4 の終了判定が実行できればよい。と

ところが、アルゴリズム MASTER のステップ 5 には既に $D^{-1}A_{J_\ell}^{\top}z_{J_\ell}^{(k_\ell+1)}$ の伝送を監視する機能が組み込まれているため、アルゴリズムの構成を変えることなく任意の時点で $\delta_\ell^{(k_\ell)}$ の到着を検出することは不可能である。そこで、マスター・ノードが受け持っている機能を二つの処理装置 A および B に分担させることを考える。すなわち、マスター・ノード A には $x^{(\cdot)}$ の計算を、マスター・ノード B にはアルゴリズムの収束判定を担当させる。

マスター・ノード A におけるアルゴリズム AP の動きは、次のようになる。

アルゴリズム MASTER-A :

ステップ 1. 整数 $N \in [1, L]$ を定め、初期探索点 $z^{(1)}$ を $z_i^{(1)} \in R$, $i \in E$, および $z_i^{(1)} \geq 0$, $i \in I$, を満たすように選ぶ。集合 S を

$$S := \{1, \cdots, L\}$$

と設定し、 $k_\ell := 1, \ell = 1, \cdots, L$, および $k := 1$ とおく。

ステップ 2. 次式により問題 (1) の解の推定値 $x^{(k)}$ を計算する。

$$x^{(k)} := -\sum_{\ell=1}^{L} D^{-1}A_{J_\ell}^{\top}z_{J_\ell}^{(k_\ell)} - D^{-1}c.$$

すべての $\ell \in S$ に対して、ワーカー・ノード ℓ に $x^{(k)}$ の値を送る。

ステップ 3. マスター・ノード B から収束を知らせるメッセージが送られてきているかどうかを調べる。メッセージが届いていれば、ステップ 4 へ進む。さもなければ、 $S := \emptyset$ とおいてステップ 5 へ進む。

ステップ 4. すべてのワーカー・ノードから送られてくるメッセージを受け取って終了する。

ステップ 5. あるワーカー・ノード $\ell \notin S$ から $D^{-1}A_{J_\ell}^{\top}z_{J_\ell}^{(k_\ell+1)}$ が送られてきたならばその値を読み込み、次式により集合 S を更新する。

$$S := S \cup \{\ell\}.$$

ステップ 6. $|S| < N$ ならばステップ 5 へ戻る。さもなければ、 $k_\ell := k_\ell + 1$, $\ell \in S$, および $k := k+1$ としてステップ 2 へ戻る。 □

また、マスター・ノード B は、次のようなアルゴリズムを実行する。

アルゴリズム MASTER-B :

ステップ 1. すべての $\ell = 1, \cdots, L$ に対して、 $\delta_\ell^{(0)} := 0$, $k_\ell := 0$ とおく。

ステップ 2. $\delta_\ell^{(k_\ell)} = 1, \ell = 1, \cdots, L$, ならば、マスター・ノード A に収束を知らせるメッセージを送って終了する。

ステップ 3. あるワーカー・ノード $\ell \in \{1, \cdots, L\}$ から $\delta_\ell^{(k_\ell+1)}$ が送られてきたならば、その値を読み込み、 $k_\ell := k_\ell + 1$ とおいてステップ 2 へ戻る。 □

一方、ワーカー・ノード ℓ では、手続き WORKER$_\ell$ のステップ 2, 5 における「マスター・ノード」を「マスター・ノード A 」に、ステップ 3 における「マスター・ノード」を「マスター・ノード B 」にそれぞれ修正した手続きを実行すればよい。マスター・ノードを分割した場合の各処理装置間の情報の流れの一例を、図 2 に示す。

以下では、アルゴリズム MASTER と手続き WORKER$_\ell$ による実行方法を単独マスター・モデル、マスター・ノードの機能を分割してアルゴリズム MASTER-A および MASTER-B を用いる実行方法を二重マスター・モデルと呼ぶことにする。

5 数値実験結果

数値実験は、エイ・ティ・アール人間情報通信研究所が保有する並列計算機 CM-5 を用いて行った。 CM-5 は 2 のべき乗個の処理ノード (processingnodes) から構成される大規模並列計算機である。各処理ノードは、標準的な SPARC プロセッサと合計 4 個の浮動小数点演算用のベクトル・ユニットを搭載しており、そのピーク性能は 128 MFLOPS と評価されている [6] 。また、各処理ノードには 8 Mbyte のローカル・メモリが 4 個装備されており、大規模なデータの取扱いが可能であるとともに、データ・ネットワークを介して各処理ノード間で 1 対 1 のデータ通信を行うことができる。 CM-5 は SPMD 型の計算モデルを採用しており、大規模なデータに対して一斉に処理を行うデータ並列型のプログラムと、各処理ノードがそれぞれ独立に処理を行うコントロール並列型のプログラムの双方を、効率良く実行することができる。

実験には 32 個の処理ノードを持つ CM-5 を用い、一つまたは二つの処理ノードをマスター・ノードに、残りの処理ノード群を必要に応じてワーカー・ノードに割当てた。各処理ノードにおいて実行するプログラムのコーディングには、データ並列型のプログラミング言語である CM Fortran [5] を用いた。 CM Fortran は、配列要素同士の演算を並列的に実行したり、配列要素の移動や部分和の計算を高速に実行できるように開発された FORTRAN 系統の言語であり、各ワーカー・ノードで実行しなければならない前処理つき共役勾配法の手続きを効率的に処理できるものと期待される。なお、共役勾配法の手続きを実行する際に必要となる行列とベクトルの積の計算には、 CM Fortran のユーティリティ・ライブラリー [4] において提供されている部分配列内の積算 (segmented-add-scan) および複写 (segmented-copy-scan) を実行するサブルーチンを用いた。一方、各処理ノード間でのメッセージ受渡し (message passing) は、ソフトウェア・ライブラリー CMMD [7] を利用して実現した。また、すべての計算は倍精度で行った。

テスト問題としては、ランダムに生成された不等式制約条件のみを持つ分離可能で疎な 2 次計画問題

$$\begin{array}{ll} \text{目的関数：} & \dfrac{1}{2}x^{\mathsf{T}}D\,x + c^{\mathsf{T}}x \quad \rightarrow \quad \text{最小} \\ \text{制約条件：} & A\,x \leq b \end{array} \tag{10}$$

を用いた。ただし、 D は $n \times n$ 対角行列, c は n 次元実ベクトル, A は $m \times n$ 行列, b は m 次元実ベクトルである。 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 $[1,10]$ の一様乱数を用いて生成した。この時、 $x = 0$ は制約条件 $A\,x \leq b$ を満たすので、テスト問題 (10) は常に実行可能となる。

添字集合 E を空集合とおくことにより、問題 (10) に対してもアルゴリズム AP を適用することができる。まず、ステップ 1 において、初期探索点としては $z^{(1)} = 0$ を選んだ。また、ス

テップ 3 においては、すべての $i \in J_\ell$, $\ell = 1, \cdots, L$, に対して条件

$$|r_i^{(k_\ell)}| \leq 10^{-7} \| b \|_\infty \tag{11}$$

または

$$z_i^{(k_\ell)} = 0, \quad r_i^{(k_\ell)} \leq 10^{-7} \| b \|_\infty \tag{12}$$

が成り立った場合にアルゴリズム AP を終了させることにした。一方、部分問題 $\mathrm{DSP}_\ell^{(k_\ell)}$ を解くための前処理つき共役勾配法の手続きも、z_{J_ℓ} と r_{J_ℓ} の各成分が条件 (11) または (12) を満足した段階で打ち切ることにしている。

数値実験は、制約条件の係数行列 A が次のような特徴的な構造を持つ問題に対して行った。

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

ただし、A_q, $q = 1, \cdots, Q$, はいずれも $\overline{m} \times \overline{n}$ 行列, また $\overline{A}_q$, $q = 1, \cdots, Q$, はいずれも $\overline{n} \times \overline{n}$ 行列で、$m = Q\,\overline{m} + \overline{n}$, $n = Q\,\overline{n}$ である。このようなブロック構造 (dual block angular structure) は、多品種流問題など現実の大規模問題においてしばしば現れる。. A の列方向の分割に対応して、対角行列 D を次のように書き直しておく。

$$D = \mathrm{diag}\left\{ D_1, D_2, \cdots, D_Q \right\}.$$

ただし、D_q, $q = 1, \cdots, Q$, はいずれも $\overline{n} \times \overline{n}$ 対角行列である。このとき、問題 (10) の双対問題の目的関数の 2 次の係数行列 M は

$$M = \begin{pmatrix} M_1 & & & \overline{M}_1 \\ & M_2 & & \overline{M}_2 \\ & & \ddots & \vdots \\ & & M_Q & \overline{M}_Q \\ \overline{M}_1^\top & \overline{M}_2^\top & \cdots & \overline{M}_Q^\top \quad M_{Q+1} \end{pmatrix}$$

と書ける。ただし、

$$\begin{aligned} M_q &= A_q\, D_q^{-1}\, A_q^\top, \qquad q = 1, \cdots, Q, \\ \overline{M}_q &= A_q\, D_q^{-1}\, \overline{A}_q^\top, \qquad q = 1, \cdots, Q, \\ M_{Q+1} &= \sum_{q=1}^{Q} \overline{A}_q\, D_q^{-1}\, \overline{A}_q^\top \end{aligned}$$

である。数値実験では、$L = Q + 1$ とおき、行列 M を

$$G_\ell = M_\ell, \qquad \ell = 1, \cdots, L,$$

を用いて定められるブロック対角行列 G によって分割した。

問題 (10) の結合制約 (coupling constraint) の係数行列は、

$$\overline{A} = \left[\overline{A}_1 \ \overline{A}_2 \ \cdots \ \overline{A}_Q \right]$$

で定められる $\overline{m} \times n$ 行列である。数値実験では、係数行列 A のブロック部分 A_q の非零要素数を 8192, 列数 $\overline{m}$ を 512 にそれぞれ固定し、行数 $\overline{m}$ を 128 と 256 の 2 通りに選んだ。また、そのそれぞれの場合に対して、結合制約の係数行列 $\overline{A}$ の非零要素数をさらに 2 通りに変化させている。Q の値は、3, 7, 15 の 3 通りについて実験を行った。テスト問題の特徴を表 1 に示す。

表 1: テスト問題

		各ブロック		非零要素数		全体	
問題名	Q	$\overline{m}$	$\overline{n}$	A_q	$\overline{A}$	m	n
A11	3	128	512	8192	4096	896	1536
A12	3	128	512	8192	8192	896	1536
A21	7	128	512	8192	8192	1408	3584
A22	7	128	512	8192	16384	1408	3584
A31	15	128	512	8192	16384	2432	7680
A32	15	128	512	8192	32768	2432	7680
B11	3	256	512	8192	4096	1280	1536
B12	3	256	512	8192	8192	1280	1536
B21	7	256	512	8192	8192	2304	3584
B22	7	256	512	8192	16384	2304	3584
B31	15	256	512	8192	16384	4352	7680
B32	15	256	512	8192	32768	4352	7680

パラメータ N の値を様々に変化させて各テスト問題を解き、アルゴリズム AP の実行時間を測定した。実験にあたっては、各サイズのテスト問題に対して乱数の種類を変えて 5 つの例題を生成し、それらの平均を求めた。図 3 から図 14 に、実験の結果を示す。各図において、破線は単独マスター・モデルによる結果, 実線は二重マスター・モデルによる結果である。図の右端は $N = L$, すなわち、同期型の並列アルゴリズム BP を実行した場合に相当する。一方、図の左端は $N = 1$ であり、各ワーカー・ノードを完全に非同期的に動作させた場合の結果を示している。図の中央部はこれらの中間的な動作をした場合であり、左にいくほど非同期性が増して、遊休状態で待機しているワーカー・ノードの個数が少なくなるように部分問題の更新をこまめに行ったことになる。

まず、実験を行ったすべての場合において、二重マスター・モデルによる実行の方が単独マスター・モデルによる実行に比べて計算時間が少なくなっていることがわかる。この傾向は、N の値が小さくなるほど顕著に現れており、マスター・ノードの機能を二つに分割することによって、マスター・ノードが収束判定結果を待っている間に遊休状態に至るワーカー・ノードの数を減らすという方策が有効に働いたものと考えられる。ただし、必要となる処理装置の数が一つ増えるので、並列化効率の観点から考えた場合には、必ずしも有効な方法であるとは限らない。実

際、計算時間と使用した処理装置の数の積で比較してみると、単独マスター・モデルの方が値は小さくなっていることがわかる。

次に、アルゴリズム AP の非同期度合をあらわすパラメータ N と計算時間の関係は、N の減少にともなって計算時間が増えるもの、減るもの、あるいはまた、V 字型の変化を示すものなど問題によって様々である。しかし、二重マスター・モデルを用いるならば、N の減少に伴って計算時間が極端に増加している例は問題 B12 だけである。従って、非同期型のアルゴリズム AP は、同期型のアルゴリズム BP とほぼ同等の性能を持つと考えてよい。特に、問題 B11 や B21 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。

二重マスター・モデルを用いた場合に N の減少に伴って計算時間が単調に減少しているいくつかのテスト問題について、各ワーカー・ノードの稼働状況をもう少し詳細に見てみよう。あるワーカー・ノードが実際に稼働している時間の実行時間全体に対する割合を稼働率と呼ぶことにすると、次のような二通りの傾向が認められる。まず、非同期性の導入により非常に大きな効果が得られた問題 B21 の場合、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 の稼働率は、$N = 8, 4, 2, 1$ となるにつれて、順に 54.1%, 66.8%, 81.1%, 97.1% のように急激な伸びを示している。一方、他のブロックに対応する双対変数を受け持つワーカー・ノード、例えばワーカー・ノード 1 の稼働率は、83.1%, 89.2%, 92.4%, 97.2% のように高い水準で推移している。すなわち、同期型のアルゴリズムにおいては、ワーカー・ノード L だけが部分問題を早く解き終わり、他のブロックの処理を待っている状態であったことがわかる。非同期性を導入した場合、問題全体に対して影響力を持つと思われる結合制約に対する処理を受け持つワーカー・ノード L の稼働率が上昇することによって、非常に大きな効果が得られたものと考えられる。これに対して、問題 A22 の場合は、全く対照的な傾向が観測される。実際、ワーカー・ノード L の稼働率は $N = 8, 4, 2, 1$ と変化させても 97.6%, 97.9%, 98.2%, 98.5% のように高い値のままである。一方、ワーカー・ノード 1 の稼働率は、46.5%, 66.0%, 76.9%, 82.5% のように N の減少に伴って急激に増加する。すなわち、同期型のアルゴリズムにおいては、ワーカー・ノード L が他のブロックに処理の終了を待たせていたが、、非同期性を導入することによって、ワーカー・ノード 1 から $L-1$ が独自に探索を進めることができるようになり、処理時間の短縮が得られたものと考えられる。ただし、問題題全体に対して影響力を持つと思われる結合制約に対応する双対変数の探索の進み具合には大きな改善が見られないため、問題 B21 の場合ほどの大きな効果は得られなかったものと考えられる。

計算時間の変化が V 字型になっているテスト問題も、同期型のアルゴリズムにおいては後者の傾向を持つ問題である。実際、各ワーカー・ノードにおける手続き WORKER_ℓ の実行回数を調べてみると、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 の反復回数が、他のワーカー・ノードの反復回数に比べて少なくなっている傾向が見られる。しかしながら、このようなテスト問題においては、N の値を必要以上に小さくすると、ワーカー・ノード L が時たま与える情報によって、それまで他のワーカー・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まで引き戻されてしまう可能性がある。従って、非同期的に実行した多くの反復が十分に活かされず、マスター・ノードにおいて $x^{(\cdot)}$ の更新に費やした計算時間に相当する分だけ、収束が遅くなっているのではないかと考えられる。

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型の共役勾配法を非同期並列型に拡張した。提案したアルゴリズムは、コントロール並列型の計算モデルを用いることによって並列計算機上に容易に実現することができる。実際に並列計算機 CM-5 を用いて行った数値実験によって、行列分割の考え方を用いて構成した複数の部分問題のうち、いくつの問題の解が得られた段階で新たな部分問題を構成するかを適切に選択すれば、問題の構造によっては同期型の並列アルゴリズムに優るとも劣らない性能を持つことが確かめられた。

今回の実験においては、非同期型のアルゴリズムが有効となる場合について、各ワーカー・ノードの稼働状況からその特徴を考察した。一般的には、いくつかの非常に密な行ブロックが存在するような問題に対して非同期型のアルゴリズムが有利になると考えられるが、制約条件の構造との間に明確な因果関係を見い出すためには、さらに系統的な実験が必要である。実際、具体的な相関関係が確認できれば、その結果は非同期並列アルゴリズムのパラメータ設定にも利用できるものと期待される。

また、同期型のブロック並列アルゴリズムの実行にあたっては、外側の反復の探索点がもとの問題の最適解から遠く離れていると予想される反復の初期の段階において前処理つき共役勾配法の収束判定条件を緩和することにより、実際の計算における収束性を加速するという方策を用いた [8, §5]。非同期並列アルゴリズムの場合には、各ブロックごとに共役勾配法の適用回数が異なるため、収束判定条件の緩和度合を決定する統一的な規則を導入することは必ずしも容易ではない。しかし、アルゴリズム AP のカウンタ k の値や、各ブロックごとのカウンタ k_ℓ の最小値を用いて、同期型の場合と同じような規則を構成することが考えられる。

ここで提案した非同期並列アルゴリズムにおいては、探索点のどの成分がいつ更新されるかを明確にとらえることができないため、その収束性を理論的に示すことは非常に難しいと考えられる。よく知られているように、何らかの評価関数が単調に減少するような反復が周期的に含まれているならば、アルゴリズムの大域的収束性を示すことができる。実際文献 [1] では、非対称線形相補性問題に対する逐次過緩和法 (successive overrelaxation method) において、数反復ごとに同期をとるような非同期並列アルゴリズムが示されている。ここで提案した非同期並列アルゴリズムにこのような考え方を導入した場合の性能とその収束性の解析は、今後の課題である。

参考文献

[1] De Leone, R., Mangasarian, O.L. and Shiau, T.-H.: Multi-Sweep Asynchronous Parallel Successive Overrelaxation for the Nonsymmetric Linear Complementarity Problem. *Annals of Operations Research*, Vol. 22 (1990), 43-54.

[2] 福島雅夫：非線形最適化における並列アルゴリズム. システム／制御／情報, Vol. 34 (1990), 223-231.

[3] 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, 15-28.

[4] Thinking Machines Corporation: *CM Fortran Libraries Reference Manual, Version 2.1.* Cambridge, Massachusetts, 1994.

[5] Thinking Machines Corporation: *CM Fortran Programming Guide, Version 2.1.* Cambridge, Massachusetts, 1994.

[6] Thinking Machines Corporation: *CM-5 CM Fortran Performance Guide, Version 2.1.* Cambridge, Massachusetts, 1994.

[7] Thinking Machines Corporation: *CMMD Reference Manual, Version 3.0.* Cambridge, Massachusetts, 1993.

[8] Yamakawa, E. and Fukushima M.: A Block-Parallel Conjugate Gradient Method for Separable Quadratic Programming Problems. *Journal of the Operations Research Society of Japan*, submitted for publication.

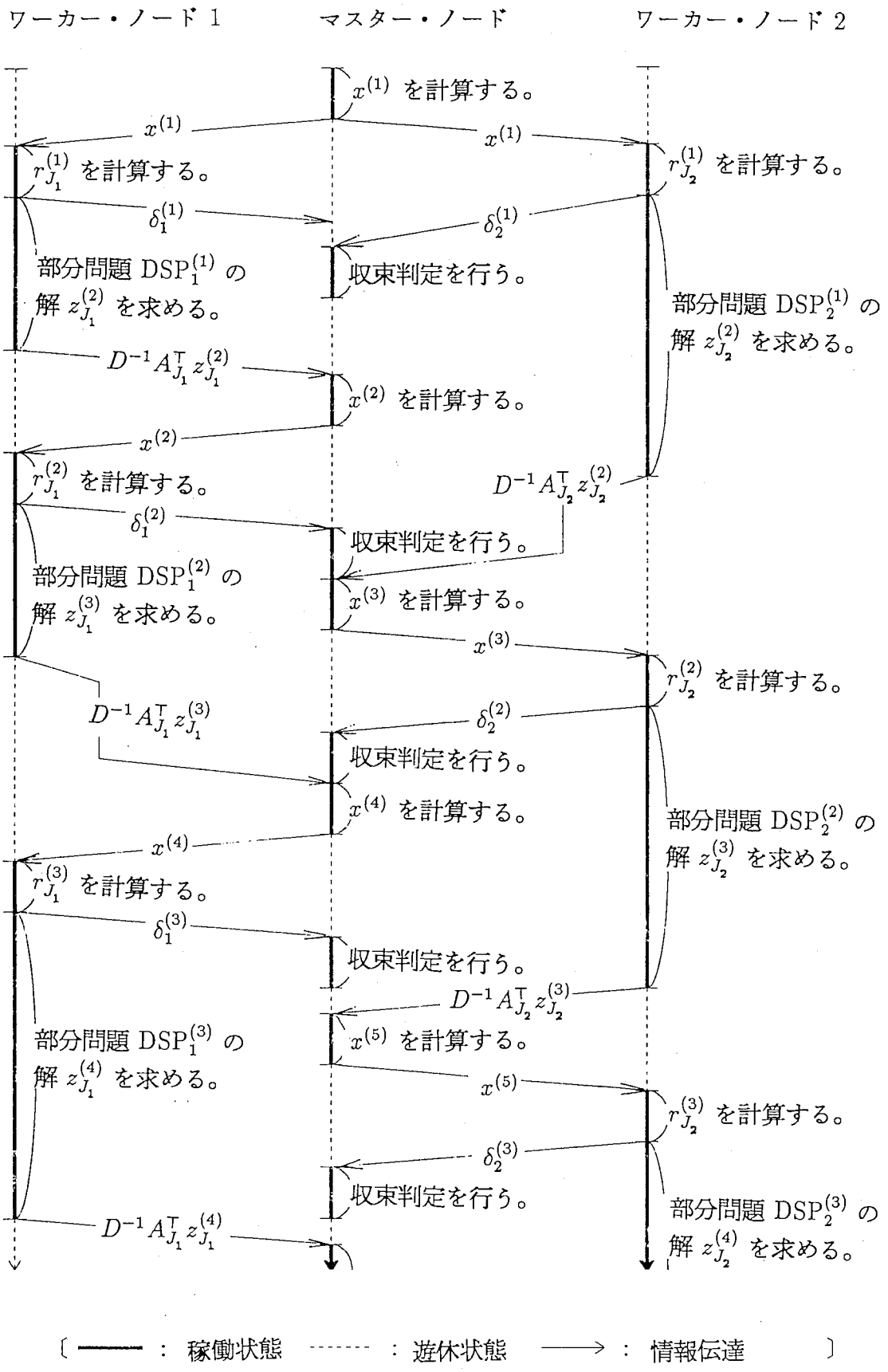

図 1: アルゴリズム AP における処理装置間の情報の流れ：$L = 2$, $N = 1$ の場合

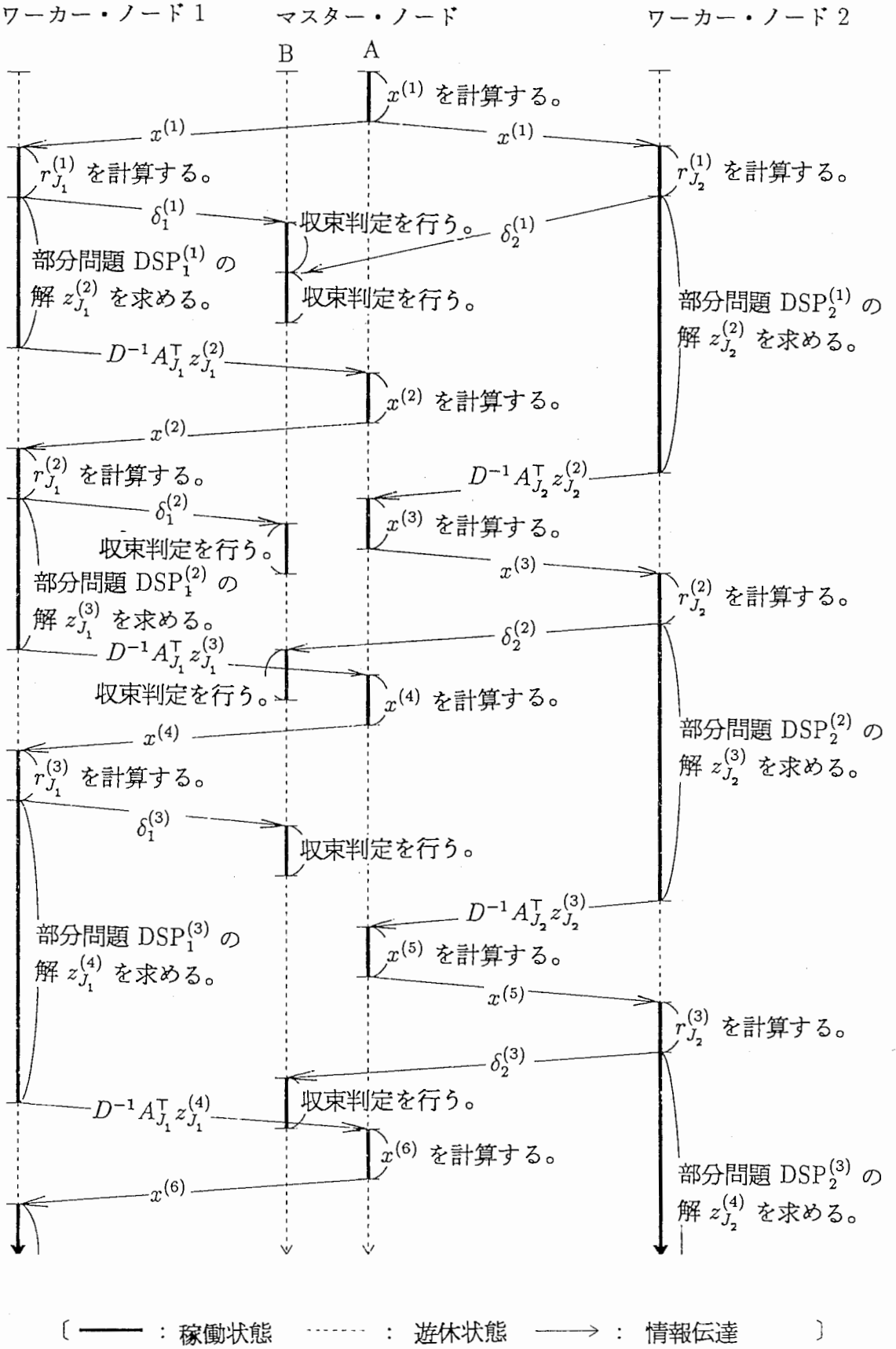

図 2: マスター・ノードを分割した計算モデルにおける情報の流れ：$L = 2, N = 1$ の場合

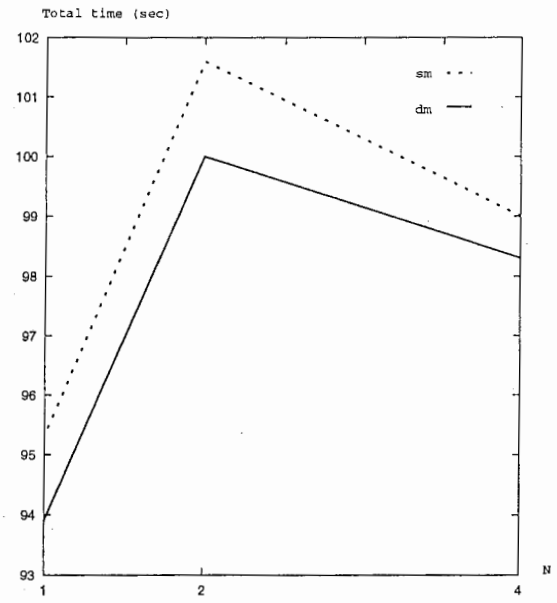

図 3: 問題 A11 に対する実験結果

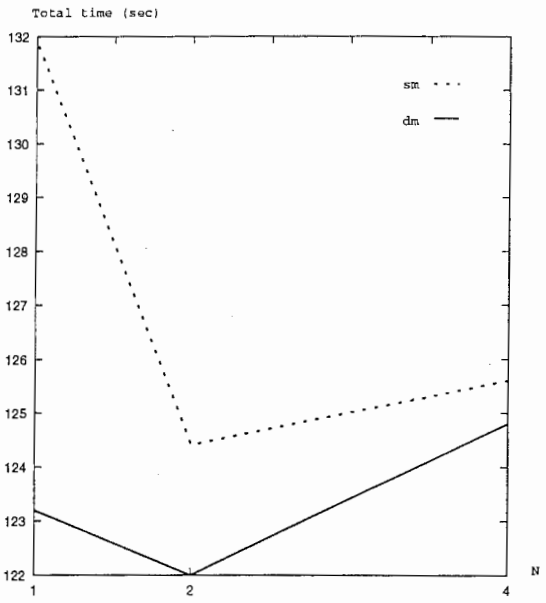

図 4: 問題 A12 に対する実験結果

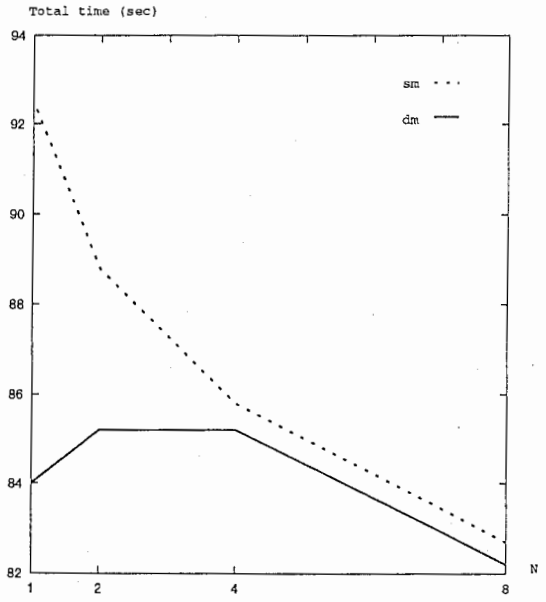

図 5: 問題 A21 に対する実験結果

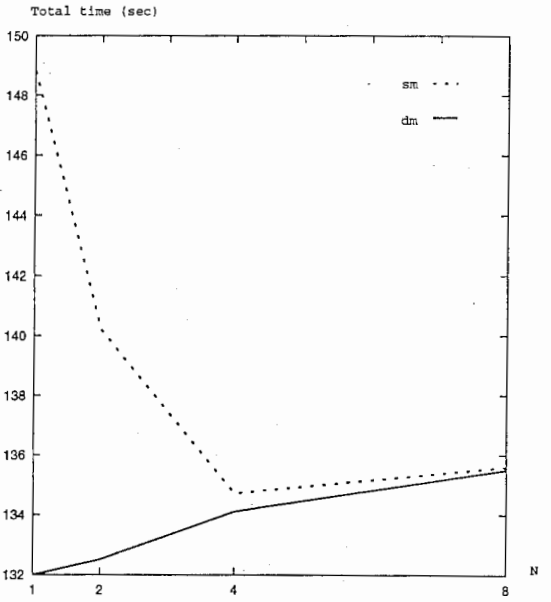

図 6: 問題 A22 に対する実験結果

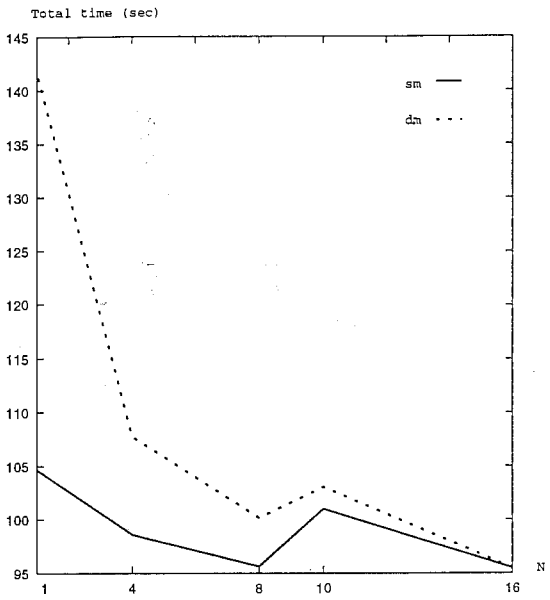

図 7: 問題 A31 に対する実験結果

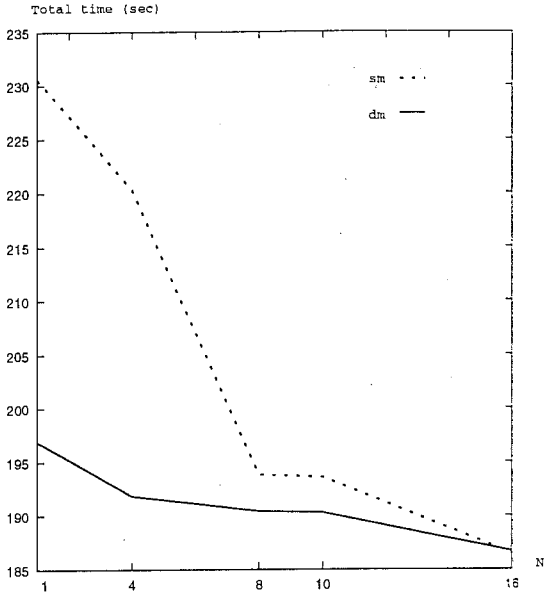

図 8: 問題 A32 に対する実験結果

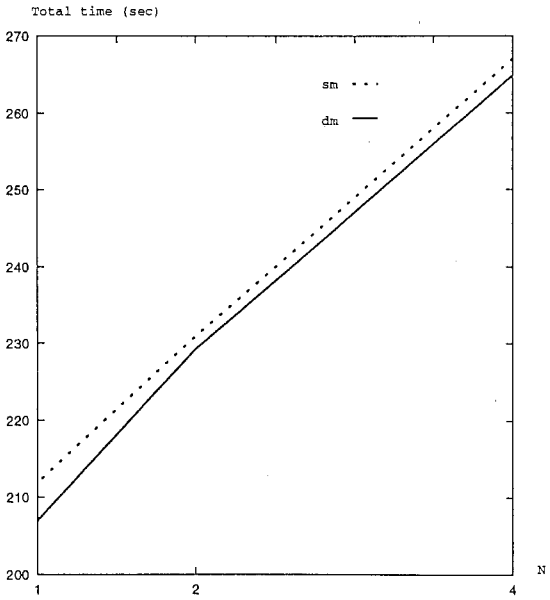

図 9: 問題 B11 に対する実験結果

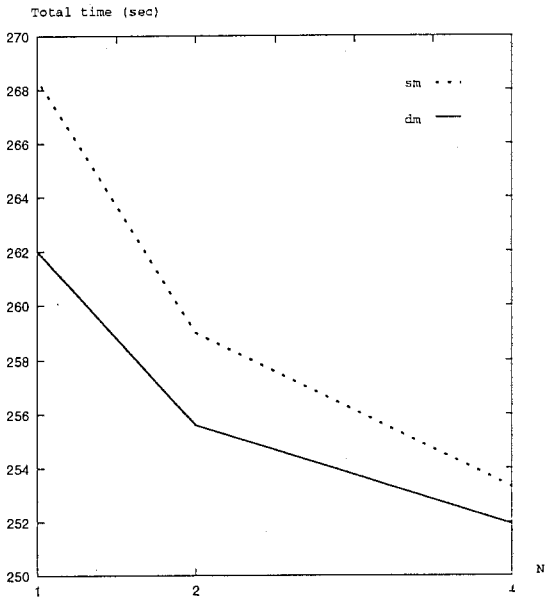

図 10: 問題 B12 に対する実験結果

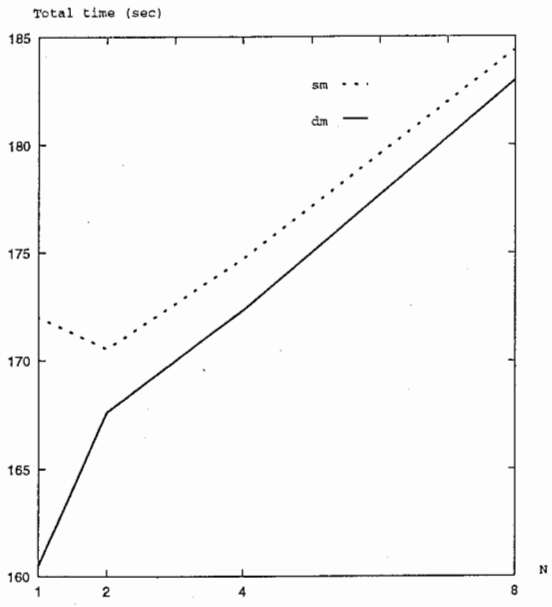

図 11: 問題 B21 に対する実験結果

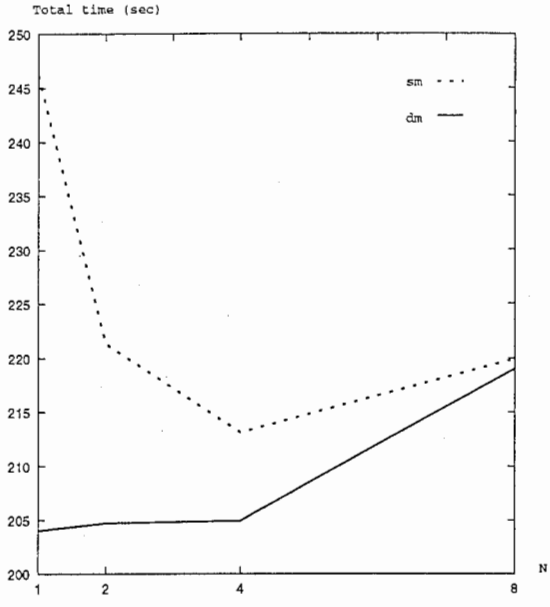

図 12: 問題 B22 に対する実験結果

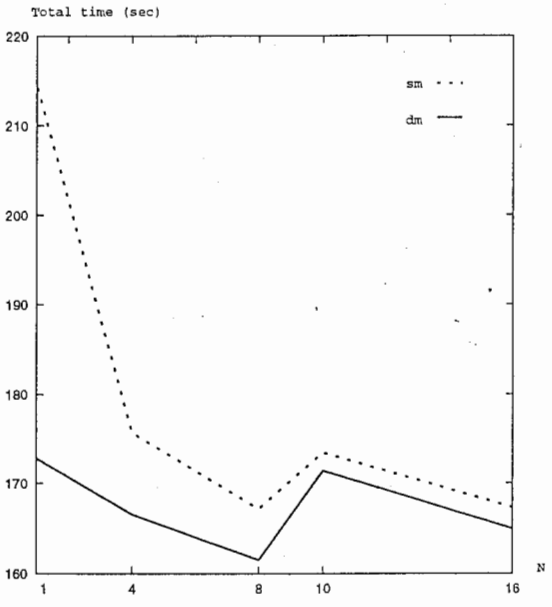

図 13: 問題 B31 に対する実験結果

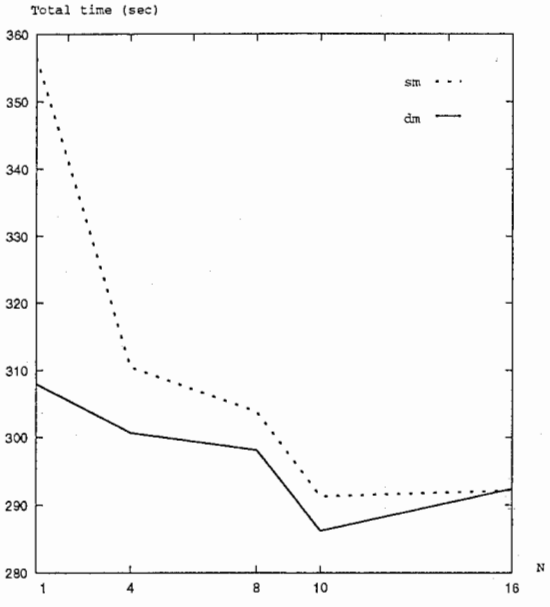

図 14: 問題 B32 に対する実験結果