2004－HPC－100　(3)
2004／12／17

ロトカ・ボルテラ系による特異値計算アルゴリズムの並列化

誉田 太朗 †,††　高田 雅美 †,††　岩崎 雅史 †,††　辻本 諭 ††　中村 佳正 †,††
konda@amp.i.kyoto-u.ac.jp

† 科学技術振興機構 さきがけ
†† 京都大学大学院 情報学研究科 数理工学専攻

概要

本稿ではロトカ・ボルテラ系による特異値計算アルゴリズムの並列化手法を提案する．行列の特異値計算はデータ検索や画像処理などに広く利用されている．必要となるデータ量の増大に対応し，より高度な情報処理を行うためには，高速かつ高精度な特異値計算手法が必要となる．このアルゴリズムには演算順序に規則性があるため，行列データを等分割する並列化手法を適用した場合，通信オーバヘッドが生じる．そこで，非同期式通信を用いて通信待機中に計算を続行させることによって，並列化効率の向上を図る．本手法の有効性を確認するために，並列化ライブラリ Message Passing Interface を用いて並列プログラムを開発し，32 プロセッサで構成される分散メモリ型並列計算機において数値実験を行う．

A Parallelization of Singular Value Computation Algorithm by the Lotka–Volterra System

Taro Konda,†,††　Masami Takata,†,††　Masashi Iwasaki,†,††　Satoshi Tsujimoto ††
and Yoshimasa Nakamura †,††

† PRESTO，JST
†† Department of Applied Mathematics and physics,
Graduate School of Informatics，Kyoto University

Abstract

A parallelization of singular value computation algorithm by the Lotka–Volterra system is presented. The singular value computation plays an important role in, for example, data search systems and image data processings. A large–scale processing for the advanced applications needs a better algorithm with respect to both convergence speed and numerical accuracy. A straightforward data–splitting method invokes interprocessor communications every step. As a result, it parallelizes the algorithm inefficiently. We here propose a parallel version of the algorithm with non–blocking communications, which enables us to introduce processing in the waiting. Numerical results of the parallel algorithm implemented with the Message Passing Interface on a distributed–memory computer with 32 processors are also presented.

1　はじめに

数値シミュレーションの基礎である線形数値計算アルゴリズムのいくつかは，可積分系と深い関わりがある．ここで可積分系とは，線形化可能あるいは線形系と関連付けられる非線形力学系の総称で，十分な数の保存量を持ち，初等関数や特殊関数によって解を具体的に書き下すことができる．時間変数の適切な離散化がなされた可積分系においてもこれらの性質が失われることはない．注目すべきは，既存の数値計算アルゴリズムの漸化式の中に離散可積分系と一致するものがみられることである．一歩進んで，可積分系の離散化によって新たなアルゴリズムの定式化にも成功している．

線形計算において特異値計算・特異値分解は極めて重要な行列演算の一つである．特に，画像処理 [1] やデータ検索 [2] をはじめ，最小二乗問題を扱う分野に広く応用されている．扱うデータ量の増大に伴い，より高速かつ高精度な数値計算手法が求められている．そ

こで本稿では，可積分系であるロトカ・ボルテラ系の離散化によって新たに定式化された特異値計算アルゴリズム (dLV アルゴリズム) の並列化によりこの要求に応じることを目的とする．

特異値を高速に計算する既存の手法として qd アルゴリズムがある [3]．これは国際的な数値計算ライブラリ LAPACK[4] に採用されているが，数値的な危険性があり相対誤差にはばらつきが見られる．一方，dLV アルゴリズムはより高い精度を保つ特異値計算が可能であるため，高速 · 高精度な反復計算が可能となる．

大規模な行列の特異値を計算する場合，効率的な並列化が必要とされる．qd アルゴリズムの並列化として Parlett らは，n 個の行列積を $O(\log_2 n)$ で計算する Parallel Prefix 法 [5] を適用した [3, 6]．しかし Parallel Prefix 法のもつ数値不安定性 [7] に加え，大量のプロセッサ間通信に起因する並列化効率の低下が避けられないためか，Parallel Prefix 法の有用性を主張するような実験結果は報告されていない．

以下，第 2 章では dLV アルゴリズムについて概説する．第 3 章において，このアルゴリズムの並列化のための領域分割法を定式化するとともに，非同期式通信を適用することによって，並列化効率の向上を図る．第 4 章では，分散メモリ型並列計算機を用いた数値実験により，並列 dLV アルゴリズムの評価を行う．

2　ロトカ・ボルテラ系による特異値計算アルゴリズム

時間連続なロトカ・ボルテラ系

$$\frac{d}{dT}u_k(T) + u_k(T)(u_{k-1}(T) - u_{k+1}(T)) = 0, \qquad (1)$$
$$T > 0,\ k = 0, \pm 1, \pm 2, \cdots$$

は生態系を記述する代表的な数理モデルである．ここで，T は時間を，$u_k(T)$ は時間 T における k 番目の生物の個体数を表す．

可積分系にはルンゲ・クッタ法のような近似手法とは全く異なる特有の離散化手法がある．系 (1) について可積分な離散化を行えば，時間離散なロトカ・ボルテラ系

$$u_k^{(t+\delta)} - u_k^{(t)} = \delta\{u_k^{(t)}u_{k+1}^{(t)} - u_k^{(t+\delta)}u_{k-1}^{(t+\delta)}\}, \qquad (2)$$
$$t = 0, 1, 2, \cdots$$

が得られる [8]．ここで t，δ はそれぞれ離散時間，差分間隔を表し，連続時間 T との間には $T = t\delta$ が成り立つ．また，$u_k^{(t)}$ は離散時間 t における k 番目の生物の個体数を表す．なお式 (2) は $T = t\delta$ を保ったまま $\delta \to 0$ とすれば式 (1) に帰着する．

N 次元上 2 重対角行列 B の非零成分を b_k $(k = 1, \cdots, 2N-1)$ とし，$w_k^{(0)} = b_k^2$ とおく．任意の $\delta > 0$ での有限離散ロトカ・ボルテラ系 (有限 dLV 系) を

$$\begin{cases} u_1^{(t)} = w_1^{(t)} \\ u_k^{(t)} = w_k^{(t-1)}/(1 + \delta * u_{k-1}^{(t)}) \\ w_{k-1}^{(t)} = u_{k-1}^{(t)} * (1 + \delta * u_k^{(t)}) \\ w_{2N-1}^{(t)} = u_{2N-1}^{(t)} \end{cases} \quad k = 2, \cdots, 2N-1 \quad (3)$$

と定義する．そのとき $w_{2k-1}^{(t)}$ は $t \to \infty$ で B の特異値 σ_k の 2 乗に収束する [9]．ここで，$*$ は乗算とする．なお，B の成分に零がある場合は，別処理によって零を成分に持たないいくつかの上 2 重対角行列に分割する．これは dLV アルゴリズムと呼ばれ，以上のように適切な境界条件と初期値を与え時間発展をさせることにより，行列の特異値計算を高精度で求めることができる [10]．

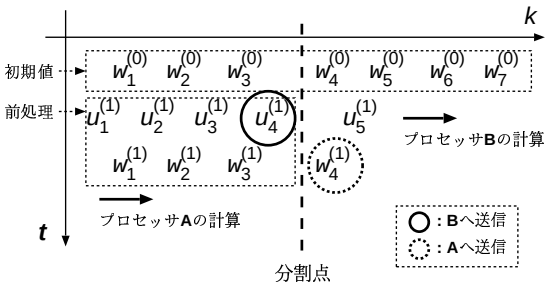

図 1: 2 台のプロセッサによる並列計算 (N=4)

3　並列特異値計算アルゴリズム

大規模な行列の特異値を高速に計算する場合，効率的な並列化が必要である．本章では dLV アルゴリズムを並列化するための手法について述べる．以下，3.1 節では領域分割法を用いて dLV アルゴリズムを並列化する．3.2 節では非ブロッキング通信を用いて高い並列化効率を目指す．3.3 節では並列化指標を用いて並列 dLV アルゴリズムの性能を理論的に評価する．

3.1　領域分割法による並列化

第 2 章で示した式 $u_k^{(t)} = w_k^{(t-1)}/(1 + \delta * u_{k-1}^{(t)})$ は dLV アルゴリズムの核となる演算の 1 つである．これは $u_k^{(t)}$ を求めるには必ず $w_k^{(t-1)}$ および $u_{k-1}^{(t)}$ が既知でなければならないことを意味する．$w_k^{(t-1)}$ と $u_{k-1}^{(t)}$ より $u_k^{(t)}$ を先に求めることはできない．

本節では上述した演算特性を把握した上で，dLV アルゴリズムの並列化を行う．並列化手法として，前処理により各プロセッサの計算開始のタイミングを調整し，適切なプロセッサ間の通信を選定する．

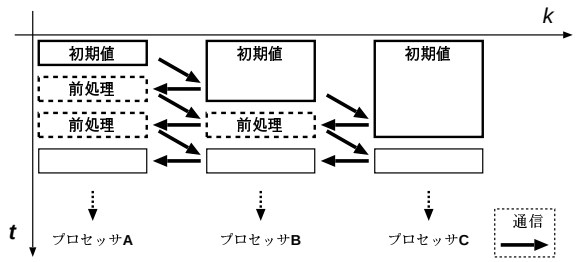

図 2: 3 台のプロセッサによる並列計算

図 1 は 4 次元行列の処理の流れを模式的に示したものである．与えられた初期値 $w_k^{(0)}$ の空間領域を 2 つに分割し，左側をプロセッサ A の計算領域，右側をプロセッサ B の計算領域とする．前処理として，プロセッサ A は計算ステップを一段階時間発展させ，$u_k^{(1)}$，$w_k^{(1)}$ を計算する．次に，プロセッサ A はプロセッサ B に $u_4^{(1)}$ を送信する．プロセッサ B はこれを受信し，計算が可能となった $w_4^{(1)}$ を求め，その結果をプロセッサ A に送信する．領域境界において以後，同様の送受信を繰り返すことにより，各プロセッサの計算を並列に進めることができる．プロセッサ A で $w_k^{(t)}$ を計算中，プロセッサ B では $w_k^{(k-1)}$ を求めるという仕組になっている．

3 台のプロセッサを用いる場合，図 2 のように各プロセッサにおいて前処理をもう一段進める．このときプロセッサ B は 2 つの分割点をもつため，プロセッサ A，C とそれぞれ通信を行う必要がある．プロセッサ B では，通信待機時のデッドロックに注意する必要がある．各時間発展においてデッドロックは，プロセッサ A から u を受信後に w を送信し、次にプロセッサ C に u を送信後に w を受信するという順序で通信を行うことによって回避することができる．

任意台数のプロセッサにおいて dLV アルゴリズムを並列実行する場合，空間領域を等分割し，図 3 に示されるように 3 種類のプロセッサのタイプに応じた処理を行えばよい．左端の分割領域の計算を行う L 型 (Left type) は右隣のプロセッサのみとの通信を必要とする．同様に右端の分割領域の計算を行う R 型 (Right type) は左隣のプロセッサとのみ通信が生じる．両隣のプロセッサと通信を行う C 型 (Center type) におけるアルゴリズムの違いは分割点のみである．そこで，プロセッサ番号 p (MPI での rank) を識別に用いることでアルゴリズムの共有化を図る．

各プロセッサが扱う空間領域が十分に大きければ，領域境界での通信コストの影響は小さくなる．また空間領域が分割され，各プロセッサが用いる記憶空間が小さくなるにつれ，キャッシュメモリをより有効に活用できる．これらより，上記提案手法を適用すれば，dLV アルゴリズムを効率的に並列実行できると考えられる．

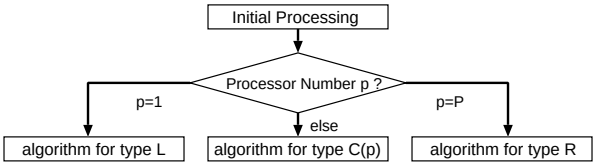

図 3: プロセッサのタイプに応じた処理の分岐

3.2 非ブロッキング通信による高速化

分散メモリ型並列計算機のための通信ライブラリである MPI(Message Passing Interface) には，ブロッキング通信と非ブロッキング通信がある．ブロッキング通信ではひとたび通信を開始するとその手続きが全て完了するまで他の処理を行うことができない．一方，非ブロッキング通信はバッファを介して通信を行うため，手続きの終了を待たずに次の処理を進めることができる [11]．

計算機上で動作している他プロセスや OS のオーバヘッドの影響により，同性能のプロセッサであっても実際の演算性能にはばらつきがある．このため，送信側と受信側の通信開始時刻にずれが生じ，その分プロセッサに待ちが生じる．例えば，L 型プロセッサにおける w の受信時が挙げられる．このずれはプロセッサ台数が多くなるほど蓄積され，それに伴い各プロセッサの待ち時間も長くなってしまう．待ち時間は並列 dLV アルゴリズムの並列化効率を悪化させる主な原因である．

通信待機中に計算を行うことができれば，より効率的に並列アルゴリズムとなる．そこで，並列 dLV アルゴリズムに非同期式通信を用いる．これにより例えば，L 型は w の受信を待たずに，C 型との境界を除いた領域の計算を続行できる．R 型であれば，一段階前の時間発展において u の受信命令をあらかじめ呼び出しておき，計算を進めることができる．

非ブロッキング通信を用いた並列 dLV アルゴリズムを次ページに示す．

3.3 並列化指標の理論による評価

本節では，以下の並列化指標を用いて並列 dLV アルゴリズムの理論的評価を行う [12, 13, 14]．

- **並列化率** r $(0 \le r \le 1)$: プログラムのうち, 完全に並列実行が可能なステートメントの割合を表す.
- **並列加速率** $S(N,P)$ $(0 < S(N,P))$: 問題の逐次処理による実行時間と並列処理実行時間の比を表す. すなわち, $T_\sigma(N)$ を逐次処理によるオーダ N の実行時間, $T_\pi(N,P)$ を P 個のプロセッサからなる並列処理による実行時間とするとき,

$$S(N,P) = T_\sigma(N)/T_\pi(N,P) \qquad (4)$$

非ブロッキング等分割 dLV 並列アルゴリズム

通信は全て非ブロッキング通信とする．

$\mathrm{dLV}(k=\cdots)$ は引数 k で漸化式計算

$$\begin{cases} u_k^{(t)} = w_k^{(t-1)}/(1+\delta * u_{k-1}^{(t)}) \\ w_{k-1}^{(t)} = u_{k-1}^{(t)} * (1+\delta * u_k^{(t)}) \end{cases}$$

を実行する．

1. **初期設定 (typeA，B，C に共通)**
   分割点の設定 $s_i = \lceil n/P \times i \rceil \quad (i = 1, 2, \cdots, P-1)$
   (ただし P は並列処理を行うプロセッサの台数)

2. **以下、値が収束するまで時間発展** $(t = 1, 2, \cdots)$

   - **type L** (プロセッサ 1):
     $u_1^{(t)} = w_1^{(t)}$
     $\mathrm{dLV}(k = 2, \cdots, 2s_1)$
     受信 (1) の終了を待つ $(t \neq 1$ のとき)
     $\mathrm{dLV}(k = 2s_1 + 1)$
     送信 (1) の終了を待つ
     プロセッサ 2 に $u_{2s_1+1}^{(t)}$ を送信 (1)
     プロセッサ 2 から $w_{2s_1+1}^{(t)}$ を受信 (1)
   - **type C(p)** (プロセッサ $2, 3, \cdots, P-1$):
     プロセッサ $p-1$ から $u_{2s_{p-1}+1}^{(t)}$ を受信 (2)
     $(t = 1$ のとき)
     受信 (2) の終了を待つ
     $\mathrm{dLV}(k = 2s_{p-1} + 2)$
     送信 (2) の終了を待つ $(t \neq 1$ のとき)
     プロセッサ $p-1$ に $w_{2s_{p-1}+1}^{(t)}$ を送信 (2)
     プロセッサ $p-1$ から $u_{2s_{p-1}+1}^{(t)}$ を受信 (2)
     $\mathrm{dLV}(k = 2s_{p-1} + 3, 2s_{p-1} + 4, \cdots, 2s_p)$
     受信 (3) の終了を待つ $(t \neq 1$ のとき)
     $\mathrm{dLV}(k = 2s_p + 1)$
     送信 (3) の終了を待つ
     プロセッサ $p+1$ に $u_{2s_p+1}^{(t)}$ を送信 (3)
     プロセッサ $p+1$ から $w_{2s_p+1}^{(t)}$ を受信 (3)
   - **type R** (プロセッサ P):
     プロセッサ $P-1$ から $u_{2s_{P-1}+1}^{(t)}$ を受信 (4)
     $(t = 1$ のとき)
     受信 (4) の終了を待つ
     $\mathrm{dLV}(k = 2s_{P-1} + 2)$
     送信 (4) の終了を待つ $(t \neq 1$ のとき)
     プロセッサ $P-1$ に $w_{2s_{P-1}+1}^{(t)}$ を送信 (4)
     プロセッサ $P-1$ から $u_{2s_{P-1}+1}^{(t)}$ を受信 (4)
     $\mathrm{dLV}(k = 2s_{P-1} + 3, 2s_{P-1} + 4, \cdots, 2n-1)$
     $w_{2n-1}^{(t)} = u_{2n-1}^{(t)}$

である．もし $S(N,P) = P$ ならば, プログラムは線形スピードアップをもつといい, そうでない場合はスローダウンと呼ぶ．

- **並列化効率** $E(N,P)$ $(0 < E(N,P))$: 逐次プログラムと比較したときの並列プログラムのプロセッサ利用の計量である．これは並列加速率を, 使用しているプロセッサ台数で割った値である．

$$E(N,P) = S(N,P)/P \tag{5}$$

まず，通信に要する時間を無視した場合の dLV アルゴリズムの計算量を考える．漸化式一回の計算時間を α とすると，反復計算一回に要する時間は $\alpha(2N-1)$ である．この反復が I 回行われるとすると，単一プロセッサによる計算時間は $T_\sigma(N) = \alpha I(2N-1)$ となる．また P 台のプロセッサで計算する場合，これが等しく分割されるので計算時間は $T_\pi(N,P) = \alpha I(2N-1)/P$ となる．よって並列加速率は $S(N,P) = T_\sigma(n)/T_\pi(N,P) = P$ であり，並列化効率は $E(N,P) = P/P = 1$ となる．

次に，通信時間を考慮する．反復一回の間に発生するプロセッサ一台あたりの送受信時間を β とすると，プロセッサ P 台で反復一回に要する計算時間は $\alpha(2N-1)/P + \beta$ である．よって並列加速率は $S(N,P) = P/(1 + \alpha^{-1}\beta P/(2N-1))$ となる．ここで $N \gg 1$, $\beta/(2\alpha) = c$ とすると並列化効率は

$$E(N,P) \approx \frac{1}{1+\beta P/(2N\alpha)} = \frac{1}{1+cP/N} \tag{6}$$

となる．一方 α, β を用いると, 並列化率は $r = \alpha N(\alpha N + \beta)^{-1}$ なので, $\beta/(\alpha N) = (1-r)/r$ となり, 並列化効率は $E(N,P) = (1 + 2P(1-r)/r)^{-1}$ で与えられる.

式 (6) より，計算時間に対する通信時間の比である c が小さいほど，並列 dLV アルゴリズムの並列化効率は理想値に近づくことがわかる．また，行列サイズが大きくなるにつれ並列化効率は高くなり，逆にプロセッサが増えると低くなってしまう．並列化効率は反復回数 I には依存しないので，特異値への収束速度に無関係であることも分かる．

4 PC クラスタを用いた数値実験

本章では，分散メモリ型並列計算機を用いた数値実験により並列 dLV アルゴリズムの評価を行う．並列化のための通信ライブラリである MPI を用いてアルゴリズムを実装し，行列サイズとプロセッサ台数に対する計算時間と並列化指標の変化を計測する．

実験には分散メモリ型スカラ並列計算機 (PC クラスタ) である Appro 社の HyperBlade を使用した．これは 2 プロセッサ/node の SMP16 台で構成され，合計 32 台のプロセッサを持つ．1 台の SMP はプロセッサとして AMD Opteron 2.0GHz(L1D:64KB, L2:1024KB)，主

表 1: 計算時間と並列化効率 ($N = 25000$)

プロセッサ台数 (CPU/node)	1 (1)	2 (1)	4 (1)	8 (1)	16 (1)	32 (2)
計算時間 [s]	109.51	49.00	26.41	14.52	8.13	5.88
並列加速率	1.00	2.23	4.15	7.54	13.47	18.62
並列化効率	1.00	1.12	1.04	0.94	0.84	0.58

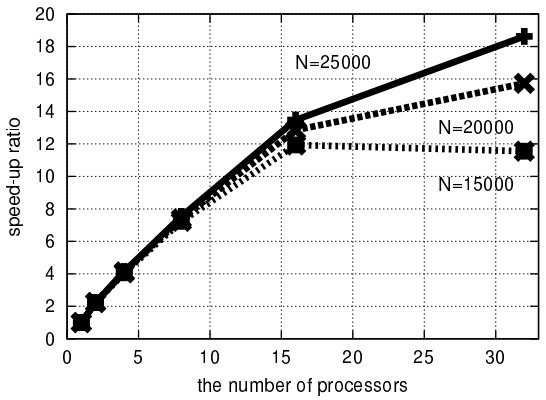

図 4: 列加速率:非ブロッキング通信

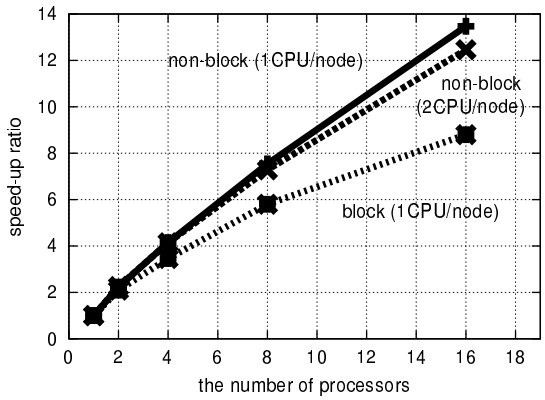

図 5: 並列加速率:両通信方式の比較 ($N = 25000$)

記憶装置として2GB を搭載している．OS には SuSE 8 を，コンパイラには pgcc を，実行環境として score5.6.0 を使用した．

並列 dLV アルゴリズムの並列化効率は 3.3 節に示したように，行列の種類には依存しない．そこで，実験対象の行列は対角成分が 2.0，非対角成分が 2.001 の上 2 重対角行列とした．この行列は近接特異値および零特異値を持たない．同じ試行を 10 回繰り返し，最速の結果を採用した．並列 dLV アルゴリズムの差分間隔は $\delta = 1.0$ とした．

まず，プロセッサ台数に対する計算時間と並列化指標の変化について調べた，得られたデータを表 1 に示す．並列加速度については，図 4 に図示する．実行モードは 1 プロセッサ/node であるが，32 プロセッサでの実行時のみ 2 プロセッサ/node とした．3.3 節で示したように，プロセッサ台数が増加するほど並列化効率が低下していることが分かる．一方，行列サイズが大きくなるほど並列化効率の向上が見られた．非ブロッキング通信を用いた 1 プロセッサ/node では，線形スピードアップに近い高速化が実現されている．特にプロセッサ 4 台までについては，スーパーリニアな結果が得られた．これは，キャッシュメモリによる演算速度向上が通信のオーバヘッドよりも勝ったためと考えられる．ところが，32 プロセッサによる実行時には並列化効率の比較的大きな低下が見られる．要因として，2 プロセッサ/node では 2 台のプロセッサによってイーサネットと主記憶が共有されていることが挙げられる．この結果，排他制御のオーバーヘッドが発生し並列化効率の低下を招く．

図 5 はブロッキング通信と非ブロッキング通信の違いによるプロセッサ台数に対する並列加速率を示す．1 プロセッサ/node とし，非ブロッキング通信に関しては 2 プロセッサ/node とした場合の測定も行った．ただし行列サイズを $N = 25000$ とする．プロセッサ 16 台の実行においては，非ブロッキング通信の導入により，50% 以上の差が生じた．

次に，ブロッキング通信と非ブロッキング通信の優劣を明確にするために，両通信方式の実行時間全体に占める通信時間の割合を比較した．その結果を図 6 に示す．非ブロッキング通信の通信時間は，ブロッキング通信に比べ 60%以上削減されていることがわかる．両者の差は，主に非ブロッキング通信を用いることにより削減された通信待機時間であると考えられる．ここで，待ち時間をプロセッサ台数 P に依存する関数として $f(P)$ と表すと，並列加速率は $S(n, P) = P/(1 + \alpha^{-1}P(\beta - f(P))/(2n - 1))$ となる．これより並列化効率は

$$E(n, P) \approx \frac{1}{1 + P(2c + f(P))/2n} \qquad (7)$$

で与えられる．ブロッキング通信ではプロセッサ台数 P が大きくなるほど $f(P)$ は増加を示すため，並列化効率 $E(n, p)$ の低下が避けられない．並列 dLV アルゴリズムでは，非ブロック通信を採用することで通信待機時間が削減され，理想的な並列化に成功したと考えられる．

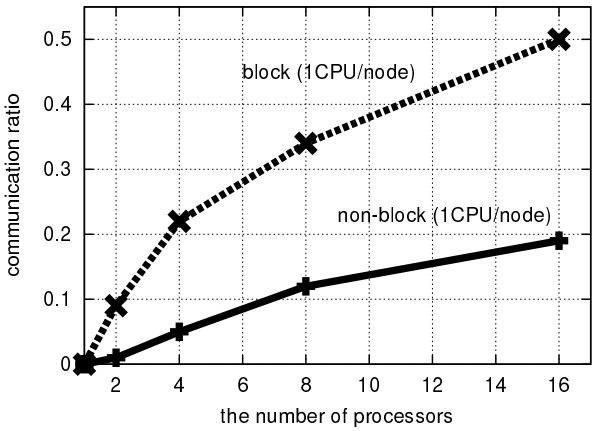

図 6: 実行時間に占める通信の割合 ($N = 25000$)

5 まとめと今後の課題

本稿では，高精度な特異値を高速に計算するため，dLV アルゴリズムの並列化手法を提案した．ポイントは dLV アルゴリズム特有の演算規則に着目することにある．適切な前処理とプロセッサ間通信を導入し，領域分割法による並列化を行った．さらに，各プロセッサの通信待機時間が引き起こす並列化効率の低下を解消するため，MPI の非ブロッキング通信を採用した．その結果，線形スピードアップに近い並列化効率を達成できた．並列 dLV アルゴリズムは，実行するプロセッサ台数が多くなるほど並列化効率が低下し，取り扱う問題サイズが大きくなるほど高い並列化効率を示した．場合によってはキャッシュメモリが有効に活用され，並列化効率が理論値を超える場合も確認できた．

今後の課題として，分割点を調整するなどのチューニングを重ね，より高い並列化効率を実現することが挙げられる．dLV アルゴリズムを，特異ベクトルを計算する適切なアルゴリズムと組み合わせることで，新しい並列化手法の展開も検討する．以上より ScaLAPACK[15] の特異値分解ルーチンとの比較実験を行いたい．

謝辞

本研究にあたり実験環境の整備に御尽力下さった峯崎征隆助手，予備的な研究を行われた京都高度技術研究所の森貴士さんに紙面をお借りして感謝の意を表します．

参考文献

[1] 斎藤恒雄：画像処理アルゴリズム，近代科学社 (1993).

[2] 北研二, 津田和彦, 獅々堀正幹：情報検索アルゴリズム, 共立出版 (2002).

[3] Fernando, K. V. and Parlett, B. N.: Accurate singular values and differential qd algorithms, *Numerische Mathematik*, Vol. 67, pp. 191–229 (1994).

[4] Anderson, E., Bischof, Z., Demmeml, J., Dongarra, J., Croz, J. D., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., D. Sorensen 著小国 力訳：行列計算パッケージ LAPACK 利用の手引, 丸善株式会社 (1995).

[5] Codenotti, B. and Leoncini, M.: *Parallel Complexity of Linear System Solution*, World Scientific (1991).

[6] Parlett, B. N.: The New qd Algorithms, *Acta Numerica*, pp. 459–491 (1995).

[7] Mathias, R.: The Instability of Parallel Prefix Matrix Multiplication, *Preprint* (1997).

[8] 中村佳正編著:可積分系の応用数理, 裳華房 (2000).

[9] Iwasaki, M. and Nakamura, Y.: On the convergence of a solution of the discrete Lotka-Volterra system, *Inverse Problems*, Vol. 18, pp. 1569–78 (2002).

[10] Iwasaki, M.: *Studies of Singular Value Decomposition in Terms of Integrable Systems, Doctor Thesis*, Kyoto Univ. (2004).

[11] FORUM, M. P. I.: MPI: A message passing interface standard, *Internat. J. Supercomput. Appl.*, Vol. 8, pp. 159–416 (1994).

[12] Fox, G., Johnson, M., Lyzenga, G., Otto, S., Salmon, J. and Walker, D.: *Solving Problems on Concurrent Processors*, Prentice-Hall (1988).

[13] Kumar, V., Grama, A., Gupta, A. and Karypis, G.: *Introduction to Parallel Computing -Design and Analysis of Algorithms*, The Benjamin/Cummings Publishing Company, Inc. (1994).

[14] P. パチェコ 著 秋葉 博訳：MPI 並列プログラミング, 培風館 (2001).

[15] Blackford, L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D. and Whaley, R.: *ScaLAPACK Users' Guide*, SIAM (1997).