Vol.2011-HPC-131 No.6
2011/10/6

時間方向並列化の線形計算への適用可能性

高見利也[†1]　西田晃[†1]

時間方向並列化手法として知られている Parareal-in-Time 法は，常微分方程式や偏微分方程式の時間発展問題に適用した場合の収束性を中心に研究されてきたが，本研究では，行列ベクトル積や反復法など，線形計算への適用を検討する．まず，時間方向並列化の直接の発展として，行列ベクトル積により定義されたベクトル列を並列に計算する問題を扱う．この手法を摂動展開として理解することで収束性に関する解析を行い，さらに，並列計算によるスピードアップ比の測定を通して，適用範囲，および，限界を明らかにする．また，一般の反復法への応用可能性についても検証し報告する．

Applicability of Time-domain Parallelism to Iterative Linear Calculus

Toshiya Takami[†1] and Akira Nishida[†1]

The time-domain parallelism, known as Parareal-in-Time algorithm, has been applied to scientific problems described by ordinary differential equations or partial differential equations. In this report, applicability of this algorithm to simple linear transformations such as matrix-vector multiplications, iterative calculus, etc., is studied through convergence and speed-up. At first, as a direct application of this algorithm, convergence to a series of vectors defined by matrix multiplications is analyzed from the viewpoint of perturbation. The speed-up ratio by this algorithm on a distributed parallel machine is measured and appropriate problem sizes for this scheme are analyzed. In addition to these analysis, applicability to general iterative calculations is reported.

†1 九州大学, kyushu University

1. Introduction

空間領域分割による並列化は，大規模計算科学における数値積分に広く応用されており，問題サイズが大きい場合には高い並列化効率が期待できる．一方，時間発展計算は直前の計算結果に強く依存するため，時間領域分割による並列計算はあまり使われていない．このような手法の一つが，Parareal-in-Time と呼ばれる方法で,[1] 最初の提案から現在までの 10 年ほどの間に，計算科学の問題への適用可能性と収束性に関しては様々な観点から調べられてきており,[2] 分子動力学計算や偏微分方程式への適用もされている.[3),4)] 一般に時間方向並列計算は，空間離散化法との組み合わせや特定の問題に特化して開発されている場合が多いが,[5),6)] Parareal-in-Time 法は，これらと独立に導入できることが特徴である．

この手法では，一般に依存関係のある時系列 $\{x_0, x_1, \ldots, x_k\}$ を並列に計算することが出来る．ただし，時間発展を記述する厳密な積分演算子 $\mathcal{F}_k$ に対して，近似演算子 $\mathcal{G}_k$ が存在することが必要である．時間発展問題を陽解法で解く場合には，時間刻み Δt を適切に設定することにより，$\mathcal{F}_k$ と $\mathcal{G}_k$ を容易に導入出来ることに注意しておく．通常，このアルゴリズムは，一種の予測子-修正子法，あるいは，Newton 法による繰り返し計算 $\{x_k^{(r)}\} \to \{x_k^{(r+1)}\}$ として理解され，十分大きな繰り返し数 r に対して，厳密な時系列 $\{x_k\}$ へ収束することが期待される.[7)] この収束が十分速い場合には，時間のかかる $\mathcal{F}_k$ の計算を並列に実行することにより，時間発展問題の並列高速化が実現する．

しかし，この手法は，大規模応用計算の現場で十分に活用されていないのが現状である．実は，これには理由が存在しており，一つには，この手法による高速化の限界が，$\mathcal{G}_k$ の計算コストに大きく依存すること,[8)] もう一つは，繰り返し列 $\{x_k^{(r)}\}$ の収束性が，$\mathcal{G}_k$ の近似の精度に大きく影響を受けることがあげられる．つまり，厳密な解法 $\mathcal{F}_k$ に対して適切な近似 $\mathcal{G}_k$ を導入できるか，という個別の問題毎に解決しなければならない部分の存在が，この手法の普及と応用を妨げていると考えられる．

本研究は，Parareal-in-Time 法の線形繰り返し問題への適用を通して，その適用範囲と実用性に関して定量的な評価を試みることを目的とする．まず，第 2 章では，この並列化アルゴリズムに関して，従来型の説明に加えて摂動の高次並列計算法としての解説を加える．第 3 章では，行列ベクトル積により定義されるベクトル列を求める問題に適用して，その収束性に関して解析し，第 4 章では，並列クラスタでのスピードアップ比の測定結果から，このアルゴリズムの適用範囲に関して考察する．また，その他の繰り返し問題についても検証を行い，並列化の一手法として線形計算ライブラリへの実装の可能性を検討する．

Vol.2011-HPC-131 No.6
2011/10/6

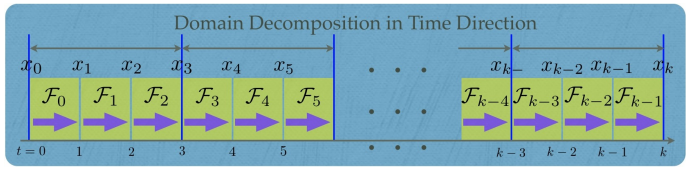

図 1 直前の結果に依存する時系列計算では，一般に時間方向の領域分割による並列化は困難である．

2. Parareal-in-Time 法のアルゴリズム

科学シミュレーションでは，偏微分方程式や常微分方程式の初期値問題の解法として，時間発展計算が主要な演算である．適切な離散化の後，時系列 $\{x_k\}$ を得るための漸化式

$$x_{k+1} = \mathcal{F}_k(x_k). \tag{1}$$

として表現される (図 1)．ある時刻の状態 x_k は，その直前の時間ステップの計算結果 x_{k-1} に直接依存しているため，時間軸方向に領域分割する方法では並列化は困難である．

Parareal-in-Time 法は，このような計算を一定の条件下で並列計算するための手法であり，厳密な時間積分演算 $\mathcal{F}_k$ が，粗視化された演算 $\mathcal{G}_k$ で近似できるとき，

$$x_{k+1}^{(r+1)} = \mathcal{G}_k(x_k^{(r+1)}) + \mathcal{F}_k(x_k^{(r)}) - \mathcal{G}_k(x_k^{(r)}), \tag{2}$$

という形で定義される．ここで $x_k^{(r)}$ は，時刻 k の状態に対する r 次の近似解であり，$\mathcal{F}_k(x_k^{(r)})$ の計算は，異なる k に属する計算とは独立に実行可能である．これにより，近似列 $\{x_k^{(r)}\}$ $(r = 1,\ 2,\ ...)$ が $r \to \infty$ で時系列 $\{x_k\}$ に収束するならば，並列計算が可能になる．

2.1 Newton 法としての Parareal-in-Time

上記のように定義される Parareal-in-Time の系列 $\{x_k^{(r)}\}$ が，$r \to \infty$ で $\{x_k\}$ に収束することは，次のようにして説明されている．[7),8)] r 回反復後の近似列を一つの列ベクトルとして $X^{(r)} \equiv (x_0^{(r)},\ x_1^{(r)}, ...\ x_k^{(r)})^T$ と書き，定義式 (1) との差を

$$f(X^{(r)}) \equiv \begin{pmatrix} x_0^{(r)} - x_0 \\ x_1^{(r)} - \mathcal{F}_0(x_0^{(r)}) \\ \vdots \\ x_k^{(r)} - \mathcal{F}_{k-1}(x_{k-1}^{(r)}) \end{pmatrix} \tag{3}$$

と表すとき，$\{x_k\}$ は，$f(X^{(r)}) = 0$ を満たすベクトル $X^{(r)}$ の要素として求められる．

$f(X)$ の Jacobian を $f'(X)$ と書く時，$f(X) = 0$ を解くための Newton 法は，

$$X^{(r+1)} = X^{(r)} - \left[f'(X^{(r)})\right]^{-1} f(X^{(r)}) \tag{4}$$

と定義されるが，

$$f'(X^{(r)}) = \begin{pmatrix} 1 & & & & 0 \\ \mathcal{F}_0'(x_0^{(r)}) & 1 & & & \\ & \mathcal{F}_1'(x_1^{(r)}) & \ddots & & \\ & & \ddots & 1 & \\ 0 & & & \mathcal{F}_{k-1}'(x_{k-1}^{(r)}) & 1 \end{pmatrix} \tag{5}$$

を，式 (4) の両辺に左から演算して整理すると，最終的に，

$$x_0^{(r)} = x_0 \tag{6}$$

$$x_{k+1}^{(r+1)} = \mathcal{F}_k(x_k^{(r)}) + \mathcal{F}_k'(x_k^{(r)}) \left[x_k^{(r+1)} - x_k^{(r)}\right] \tag{7}$$

が得られる．ここで，時間積分の Jacobian $\mathcal{F}_k'(\cdot)$ を含む項を，粗視化積分 $\mathcal{G}_k(\cdot)$ を使って，

$$\mathcal{F}_k'(x_k^{(r)}) \left[x_k^{(r+1)} - x_k^{(r)}\right] \approx \mathcal{G}_k(x_k^{(r+1)}) - \mathcal{G}_k(x_k^{(r)}) \tag{8}$$

と近似したものが，Parareal-in-Time 法である．つまり，式 (2) は近似的に Newton 法と同等と考えられ，十分に良い初期値から始めれば，厳密列 $\{x_k\}$ を得ることが出来る．

2.2 摂動展開としての Parareal-in-Time

これまで，Parareal-in-Time 法のアルゴリズムの有効性と収束については，非線形な場合も含めて広く調べられてきている．[9),10)] しかし，十分に短い時間刻みに対して，時間発展演算が有界でほとんど線形であると見なせる場合には，比較的容易に理解することが可能である．[9)] ここでは，時間発展問題が線形変換の繰り返しである場合に，つまり，$\mathcal{F}_k$ や $\mathcal{G}_k$ が線形演算子である場合に，よりわかりやすい解釈を試みる．科学計算の応用領域においても，量子力学的な時間発展問題[11),12)] に対しては，このような例が数多く存在している．

今，厳密な時間発展演算子 $\mathcal{F}_k$ が，近似演算子 $\mathcal{G}_k$ と，それに対する修正 $\mathcal{F}_k - \mathcal{G}_k$ に分解できる場合に，時系列 x_k は

$$x_k = \left[\mathcal{G}_{k-1} + (\mathcal{F}_{k-1} - \mathcal{G}_{k-1})\right] x_{k-1} = \prod_{j=0}^{k-1} \left[\mathcal{G}_j + (\mathcal{F}_j - \mathcal{G}_j)\right] x_0, \tag{9}$$

Vol.2011-HPC-131 No.6
2011/10/6

と表される．ここで，一般にすべての演算子 $\mathcal{F}_j$ と $\mathcal{G}_j$ は，時間順序 (j の順) を保存する形で演算される．$\mathcal{G}_j$ を非摂動運動の時間発展と見なし，$\mathcal{F}_j - \mathcal{G}_j$ をこの運動に加えられた摂動と考えると，もし，各 $\mathcal{G}_j$ と $\mathcal{F}_j - \mathcal{G}_j$ が有界な演算子ならば，時間発展を摂動表現することが出来る．これは，式 (9) の右辺を展開して，項別に計算することに相当する．

$\mathcal{F}_j - \mathcal{G}_j$ をちょうど r 回含む項を r 次摂動項と呼ぶ時，r 次までの摂動項をすべて含み，それ以上の高次項を落とした近似時系列 $x_k^{(r)}$ に対して，

$$x_{k+1}^{(r+1)} = \mathcal{G}_k x_k^{(r+1)} + (\mathcal{F}_k - \mathcal{G}_k) x_k^{(r)} \tag{10}$$

が成立することは，簡単に確かめられる．これはまさに Parareal-in-Time 法 (2) であり，逆に Parareal-in-Time 法とは，摂動展開で高次項を落とした近似に等しいことがわかる．

ところで，長さ k の時系列に対する摂動項の数は，r とともに急速に増大し

$$\left(\begin{array}{c} k \\ r \end{array} \right) = \frac{k!}{(k-r)!r!} \approx \sqrt{\frac{r}{2\pi(k-r)r}} \left(\frac{ek}{r} \right)^r, \tag{11}$$

と書ける．ただし，$1 \ll r \ll k$ と仮定して，Stirling の公式を使った．項数が増大しても，摂動の各項が十分小さい値であれば，低次までの摂動近似が成立し，数値的にも有効桁数内での計算が安定に実行できることになる．すなわち，Parareal-in-Time 法による計算の収束性は，各摂動項の大きさと，項数の急速な増大との兼ね合いで決まることがわかる．

2.3 Parareal-in-Time 法の計算手順

有限項での打ち切り誤差の解析は第 3 章で行うが，ここでは，式 (2) のような簡単な漸化式で，多数の摂動項を効率的に計算できる仕組みについて解説しておく．

右方向に時間の向きを，下向きに摂動の次数を表す形で，Parareal-in-Time 法の計算を模式的に書いたものが図 2 である．一つ一つの矢印が線形演算に対応しており，右向きの矢印が非摂動計算 $\mathcal{G}$，右斜め下への矢印が摂動 $\mathcal{F} - \mathcal{G}$ を表す．また，二つの矢印が集っている所は，式 (2) に従って和を取ることとする．この時，$r = k = 0$ の左上隅が初期状態 x_0 を表し，計算しようとしている $x_k^{(r)}$ は対角に位置する右下になる (図では $x_k^{(3)}$ が示されている)．式 (9) を展開する時，r 次には，図のような平行四辺形の街路を左上から右下まで移動するために取りうるすべての最短経路の数と同じだけの項がある (全部で k 回の遷移のうち，斜め方向にちょうど r 回移動することになるため，全体で式 (11) に示す経路数になる)．これを，式 (2) に従うと，$(r+1)(k-r)$ 回程度の $\mathcal{G}$ 演算と，$r(k-r)$ 回程度の $\mathcal{F} - G$ 演算で計算することが出来るのである．ただし，この計算量は，展開しない計算の

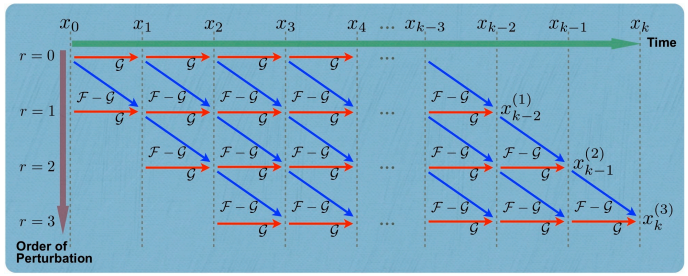

図 2　Parareal-in-Time 法の計算手順

場合に比べると，ほぼ r 倍程度に増えていることに注意する．

Parareal-in-Time 法に従って並列計算を行う場合には k 方向に並列化し，r 方向には逐次計算を行うことになる．つまり，図 2 の経路に関する和の計算を，上から r の順に計算して行く．この時，右斜め下向きの矢印に相当する演算は独立に実行できるため，並列計算が可能になる．しかし，全体の計算量が r 倍に増大しているため，十分効率的な並列計算が出来なければ，逐次計算に比べて必ずしも高速化されるわけではない．

3. 行列ベクトル積に対する Parareal-in-Time 計算の収束性

これまで Parareal-in-Time 法は，計算科学の問題を中心に様々な観点から研究されて来たが，本報告では線形計算を対象とする．そこで，この章では行列ベクトル積

$$x_{k+1} = [G + \varepsilon F]\, x_k = [G + \varepsilon F]^k\, x_0, \tag{12}$$

で定義されるベクトル列 $\{x_k\}$ の計算に適用し，その収束性を調べることとする．ここで，G と F は $N \times N$ 行列，ε は小さなパラメタである．前節までの表記方法と異なり，あらわに摂動パラメタ ε ($\ll 1$) が入っていることに注意する．これに対する Parareal 計算は，線形演算子 $[G + \varepsilon F]^k$ を展開し，高次摂動計算を行うことと同じであり，r 次の Parareal 計算における時刻 k のベクトル $x_k^{(r)}$ は，ε^r までの多項式

$$x_{k+1}^{(r)} = \left[G^k + \varepsilon \sum_{j=0}^{k-1} G^j F G^{k-1-j} + \cdots + \varepsilon^r \sum (F \text{ を } r \text{ 回含む項}) \right] x_0 \tag{13}$$

として与えられる．

G と F のスペクトル半径 $\rho(G)$ と $\rho(F)$ を使うと，r 次の Parareal 近似 $x_k^{(r)}$ の誤差は，

残りの項の和

$$\frac{\left|x_k - x_k^{(r)}\right|}{[\rho(G)]^k |x_0|} \leq \sum_{j=r+1}^{k} \left(\begin{array}{c} k \\ j \end{array} \right) \left[\frac{\varepsilon\rho(F)}{\rho(G)}\right]^j . \tag{14}$$

で押さえられることがわかる．$1 \ll r \ll k$ に対して ε^r の項の大きさは，近似的に

$$\left(\begin{array}{c} k \\ r \end{array} \right) \left[\frac{\varepsilon\rho(F)}{\rho(G)}\right]^r \approx \sqrt{\frac{k}{2\pi(k-r)r}} \left(\frac{ek}{r}\right)^r \left[\frac{\varepsilon\rho(F)}{\rho(G)}\right]^r \tag{15}$$

となるため，これから必要な次数 r を見積もることが可能である．$\varepsilon\rho(F) \ll \rho(G)$ の場合には，有限次の摂動で高精度に x_k を近似できることがわかるが，以下では，具体的な行列ベクトル積の計算で，収束の様子を定量的に調べることとする．

3.1 実対称行列

実対称行列による行列ベクトル積は，実固有値問題の計算などで広く使われているクリロフ部分空間法での主要な計算になっている．ここでは，実対称行列による行列ベクトル積 (12) により定義されるベクトル列 $\{x_k\}$ に対する Parareal アルゴリズムの収束性を，1024×1024 のランダムな行列を使って調べる．G は -1 から 1 までの値の一様乱数を対角要素に持つ行列，F は Gaussian Orthogonal Ensemble (GOE) に従う実対称のランダム行列で，要素のガウス乱数の幅は，固有値が -1 から 1 に分布するようにスケールする．

$$\langle |F_{ii}|^2 \rangle = \frac{1}{2N}, \qquad \langle |F_{ij}|^2 \rangle = \frac{1}{4N} \quad (i \neq j) \tag{16}$$

図 3(a) に，$\varepsilon = 0.01$ に対する収束の結果を示す．図中の実線は r 次近似と逐次計算の結果との差で，点線は式 (15) を $\rho(G) = \rho(F) = 1$ と仮定して計算した値である．両者の差は，行列 F と G の絶対値最大の固有値に対応する固有ベクトルが一般には異なることによって生ずると考えられる．いずれにしても，$r = 15$ 程度で $k = 200$ までの計算がほぼ可能であることがわかる．

3.2 ユニタリ行列

ユニタリ変換は，一般に量子力学の時間発展計算 $x_{k+1} = \exp\left[\frac{\Delta t}{i\hbar} H\right] x_k$ で多用される．[11),12)] ここで，H はエルミート行列で，x_k と x_{k+1} は規格化された複素ベクトルである．十分に小さい時間刻み $\Delta t/\hbar \equiv \varepsilon$ に対して，近似的に

$$\exp(-i\varepsilon H) = I - [\exp(i\varepsilon H) - 1] \approx I - i\varepsilon H \tag{17}$$

と書けることに注意する．つまり，時間発展演算子を単位行列 I で近似して $G = I$ と置き，残りの $i\varepsilon H$ を摂動部分とする ($\varepsilon F = -i\varepsilon H$)．これは厳密なユニタリ行列ではないが，ここでは簡単のためにこの形式を調べる．以下の数値計算では，H は Gaussian Unitary Ensemble (GUE) に従うランダム行列で，-1 から 1 の実固有値を持つように，

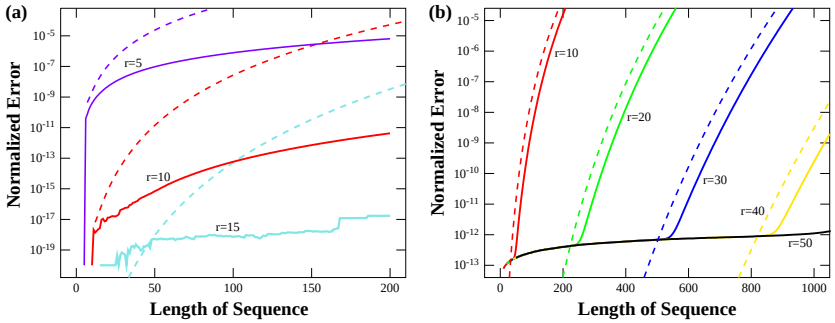

図 3 行列ベクトル積に対する Parareal-in-Time 法の収束性．(a) 実対称行列 $G + \varepsilon F$ によるベクトル列の誤差．(b) ユニタリ行列 $\exp(i\varepsilon H)$ によるベクトル列の誤差．点線は，$r + 1$ 次の項，式 (15) の大きさを表す．

$$\langle |H_{ii}|^2 \rangle = \langle |H_{ij}|^2 \rangle = \frac{1}{4N} \tag{18}$$

とスケールしておく．

近似ベクトル列 $x_k^{(r)}$ と，直接計算した $x_k \equiv (1 - i\varepsilon H)^k x_0$ の差を，図 3(b) に示す (ただし，$\varepsilon = 0.01$)．これによると $k = 1000$ までのベクトル列に対しても，$r = 50$ であれば正確な値を与えることがわかる．$r + 1$ 次項に対する式 (15) の値を同じ図に点線で表示してあるが，数値計算の結果求められた誤差とほぼ一致している．誤差が 10^{-13} より小さくならないのは，倍精度実数での数値表現では有効桁数に限界があるためである．逐次計算による行列積と比較すると，Parareal-in-Time 法では多数の計算結果の和を求めることとなるため，桁落ちに対して相対的に敏感になる可能性がある．

式 (15) から期待される誤差と数値計算の誤差が完全に一致しないのは，$r + 1$ 次の項のうちで，一部，$r + 2$ 次の項と打ち消す部分があるなど，上限の評価では限界があるためである．実対称行列の場合に比べて差が小さくなったのは，ユニタリ行列では $G = I$ と近似したため，$\rho(G) = \rho(F) = 1$ という想定が現実に近かったことによる．ただ，この他にも，異なる次数間の干渉や数値誤差の蓄積の仕方の違いなど，前節のスペクトル半径の解析だけでは捉えきれない部分があるため，打ち切り次数をより精密に推定するためには，行列の対称性などに応じて，さらなる解析と分類が必要である．

Vol.2011-HPC-131 No.6
2011/10/6

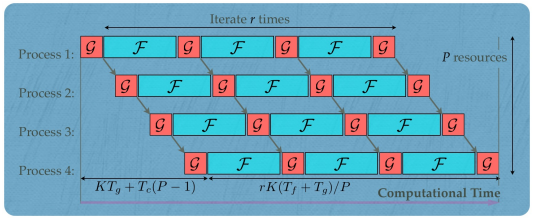

図 4　長さ K の時系列を，P 個の並列プロセスで計算する場合のタイミング．T_g は非摂動部分，T_f は摂動部分の時間，T_c は転送に要する時間を表す．

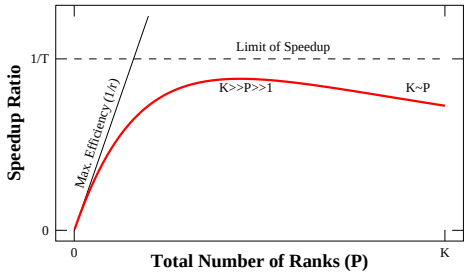

図 5　Parareal-in-Time 法の効率とスピードアップ比の概略．効率 (グラフの傾き) の最大値は $1/r$ である．$K \gg P \gg 1$ でスピードアップ比の限界 $1/T$ に近づくが，$K \sim P$ では通信時間 t が無視できなくなる．

4. 行列ベクトル積に対する Parareal-in-Time 計算のスピードアップ比

Parareal-in-Time アルゴリズムの並列化効率は，実装方法に依存する部分がある．[13] 様々な実装方法が考えられるが，通常は，バケツリレー的にデータを受け渡す形で実装される．本報告でも，MPI_Send/Recv による通信を使い，計算中にはプロセス全体での同期を行うことなくスピードアップ比を測定した．ここでの測定は，64 ノード (Westmere, 12 コア／ノード) の Xeon クラスタ (InfiniBand QDR) 上で実施した．

測定対象は，前節の行列ベクトル積で定義されたベクトル列の計算とする．この場合，$\mathcal{G}$ による非摂動計算とベクトルの和の計算は $O(N)$ 程度であるのに対して，摂動計算 $\mathcal{F}$ は $O(N^2)$ となる．これに加えて，ベクトルの受け渡しのための通信時間が影響する．

4.1 並列実装方法と効率

行列ベクトル積 (12) により定義されるベクトル列を，r 次の Parareal 近似

$$x_{k+1}^{(r+1)} = Gx_k^{(r+1)} + \varepsilon F x_k^{(r)}, \tag{19}$$

で計算するために，MPI で並列化する．ただし，境界条件 $x_0^{(r)} = x_0$ を満たすものとする．

既に図 2 で解説した通り，Parareal 法では，近似の次数 r の順に逐次計算することになるが，式 (19) は同次の量 $x_k^{(r+1)}$ に依存しており，このままでは右辺のすべてを並列に計算することができない．そこで，式 (19) の右辺を次のステップに分けることとする．

$$\text{並列ステップ:} \quad \tilde{x}_{k+1}^{(r+1)} \equiv \varepsilon F x_k^{(r)} \tag{20}$$

$$\text{逐次ステップ:} \quad x_{k+1}^{(r+1)} \equiv \tilde{x}_{k+1}^{(r+1)} + Gx_k^{(r+1)}, \tag{21}$$

前半の「並列ステップ」は計算コストの大きい部分で，図 2 の斜め矢印に相当する．これは，r 次の結果 $x_k^{(r)}$ にだけ依存するので，すべての k に関して並列計算が可能である．一方，「逐次ステップ」は，同次の $x_k^{(r+1)}$ に依存するため，k の順に逐次実行される．

スピードアップ比を見積もるために，F の計算と G の演算にかかる時間を，それぞれ T_f と T_g とし，一つのベクトルを隣に転送するのに必要な時間を T_c とする．長さ K のベクトル列を，もともとの漸化式 (12) に従って，逐次計算で求めるための時間は $K(T_f + T_g)$ である．一方，Parareal-in-Time 法 (19) に従って r 次まで求める時，並列ステップ部分 (20) の計算にかかる時間は rKT_f，逐次ステップ部分 (21) の時間は $(r+1)KT_g$ となる (図 4 に示すように，最初の $x_{k+1}^{(0)} = Gx_k^{(0)}$ の準備に KT_g の計算が必要なため)．このとき，P 個の並列リソースを利用することにすると，並列ステップ部分は $1/P$ 倍に短縮される．逐次ステップ部分は k の方向に並列化は出来ないが，異なる r 間の依存関係がないため，計算をオーバーラップさせることにより，計算時間は $[(r/P)+1]KT_g$ となる．これに，P 個の並列リソース間でデータの転送に要する時間 $T_c(P-1)$ を加えて，P 並列で実行した場合のスピードアップ比は，

$$S(r,K,P) = \frac{K(T_f+T_g)}{\dfrac{rKT_f}{P} + \left(\dfrac{r}{P}+1\right)KT_g + T_c(P-1)} = \frac{P}{r + TP + \dfrac{P(P-1)}{K}t}, \tag{22}$$

と表すことが出来る．ここで，$T \equiv T_g/(T_f+T_g)$，および，$t \equiv T_c/(T_f+T_g)$ である．

この式によると，通信時間 t が無視できるような高性能な並列リソースが十分にあっても，スピードアップは $1/T \approx T_f/T_g$ が限界であることがわかる．また，並列化による効率

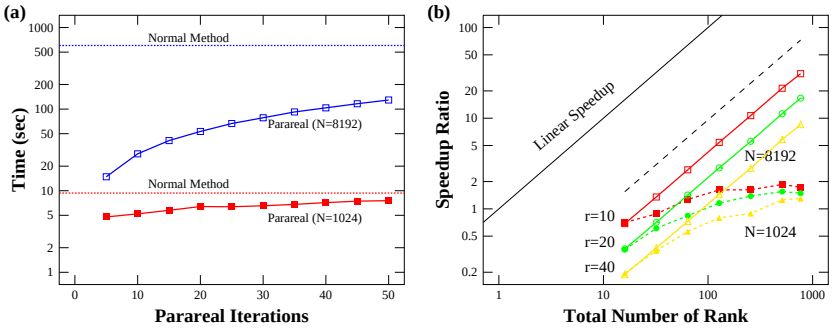

図 6 Parareal-in-Time 法と逐次計算の比較: (a) r 次計算を 512 並列で実行した場合の経過時間，点線は逐次計算の場合; (b) 並列計算でのスピードアップ比，黒の実線は逐次計算を基準とした場合の線形スピードアップ，黒の点線はスピードアップ比の推定値．塗りつぶしたマークは小規模行列の問題 ($N = 1024$)，中抜きのマークは比較的大規模な問題 ($N = 8192$) を表す．四角，丸，三角は，それぞれ，$r = 10$，20，40 に対応する．

は $S(r, K, P)/P$ と表されるから，たとえ $T \ll 1$ かつ $t \ll 1$ であるような理想的な状況でも，最大で $1/r$ でしかないことがわかる．これは，r 次の Parareal-in-Time 法では，もともとの計算より r 倍多い計算を実行していることによる．

計算時間の比 T は，G と F の計算量を比べることで見積もることは可能だが，実際の実行時間は他の様々な要因から予測が難しい場合もある．この手法が有効か，そうでないかを決定するには，データ量などに応じた現実の計算性能を正しく考慮する必要がある．

4.2 並列計算でのスピードアップ比

行列演算を Parareal-in-Time 法で並列化した場合の経過時間を，並列クラスタ上で MPI 並列プログラムにより測定した．結果を図 6 に示すが，ここでは，ユニタリ変換 (第 3.2 節) を対象とし，長さが 1536 の複素ベクトル列 ($K = 1536$) の計算に対して実施した．図 6(a) では，フラット MPI での 512 並列実行時の計算時間を，Parareal-in-Time 法の次数 r に対してプロットしてある．水平の線は，並列化していない従来法での時間を表す．$N = 8192$ の場合，次数 $r = 5$ から 50 に対して，14.7 秒から 130.1 秒の計算時間になっており，図ではわかりにくいが r にほぼ線形の関係にある．また，1 CPU での従来法での計算が 602.5 秒であるので，スピードアップは 41.0 倍から 4.6 倍である．一方，小さい行列 ($N = 1024$) に対しては，r に対して線形ではなく，スピードアップ比もせいぜい 2.0 程度である．

図 6(b) には，次数 $r = 10, 20, 40$ に対するスピードアップ比を示す．フラット MPI による分散並列化で，16 プロセスから 768 プロセスまでの結果である．$N = 8192$ の場合に，

表 1 並列計算によるスピードアップ比

	P	16	32	64	128	256	512	768
$N = 1024$	$r = 10$	0.71	0.88	1.27	1.61	1.62	1.86	1.73
	$r = 20$	0.36	0.61	0.84	1.16	1.38	1.55	1.49
	$r = 40$	0.19	0.35	0.56	0.80	0.89	1.25	1.31
$N = 8192$	$r = 10$	0.69	1.35	2.70	5.41	10.68	21.38	30.97
	$r = 20$	0.36	0.71	1.42	2.83	5.56	11.20	16.63
	$r = 40$	0.19	0.37	0.73	1.44	2.80	5.82	8.52

$t = 0$ と通信の時間を無視した時の，式 (22) の値を図中に点線で示すが，実測値とは約二倍の差である．表 1 には，これらのデータに対応するスピードアップ比の数値を示しておく．

スピードアップ比は並列ランク数に比べて低い (つまり効率が低い) が，$N = 8192$ の場合はリニアに伸びていることがわかる．つまり，特定の次数の Parareal-in-Time 法は，一定以上大きい行列の計算に対してはかなり高効率で並列実行できることがわかる．しかし，$N = 1024$ の場合にはリニアな伸びは見られない．これは，式 (22) 中の t に相当する通信時間の割合が，$N = 1024$ の時は無視できない事によると思われる．

4.3 Parareal-in-Time 法の適用範囲

上記の測定と解析の結果，Parareal-in-Time 法が効率的に適用できる行列のサイズと時系列の長さについて，大まかに図 7 に示すような分類が出来る．通常の線形ライブラリでは，行列の各要素に対する演算の独立性を利用して，複数のスレッド，あるいは，複数のプロセスで並列化されている．行列の対角化が $O(N^3)$ 程度の演算であることより，小さな行列演算で定義された非常に長い時系列を対象とする場合は，固有状態表現を通して解かれるべきである．大きな行列に対しては $O(N^3)$ の計算量は大きすぎるため，Parareal-in-Time 法による時間方向並列化は，通常の要素方向の並列化法と合わせて解かれるべきである．

では，これまで検討してきた行列ベクトル積以外の繰り返し計算に対して，Parareal-in-Time 法を適用することは可能だろうか．例えば，大規模な線形方程式 $Ax = b$ を解くための繰り返し法には，逐次加速緩和 (Successive Over-relaxation, SOR) 法，共役勾配 (Conjugate Gradient, CG) 法など様々な手法があるが，ここでは SOR 法を検討する．D を対角行列，L，U をそれぞれ下三角，上三角行列とするとき，線形方程式

$$[D + \varepsilon(L + U)]\, x = b \tag{23}$$

に対する SOR 法は，加速パラメータ w を適切に選択して

$$x_{k+1} = (1 - w)x_k + w(D + \varepsilon U)^{-1}\,[b - \varepsilon L x_k] \tag{24}$$

Vol.2011-HPC-131 No.6
2011/10/6

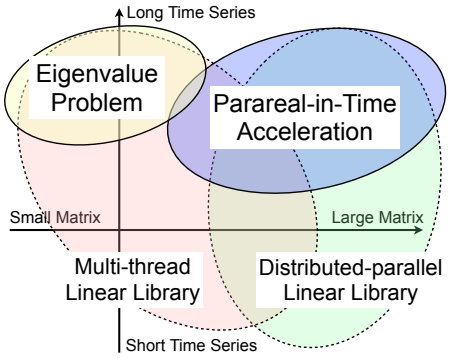

図 7　Parareal-in-Time 法の守備範囲，および，他の手法との関連．

という繰り返し計算で表される．ただし，ε は，対角優位になるように設定される小さな数値であるとする．右辺は行列積ではないが，陽的な時間発展問題と同様，直前の状態に依存する繰り返し計算であるため，この部分に Parareal-in-Time 法が適用できれば，依存関係のために高速化が難しい部分の並列計算が可能になる．そのために，右辺の変換を近似できる高速な変換 G を設定する必要があるため，以下の二つの場合を考える．

Model 1: $G = I$ (恒等演算) と定義し，残りの演算を摂動と設定する．

$$x_{k+1}^{(r+1)} = x_k^{(r+1)} + w\left[(D+\varepsilon U)^{-1}(b - \varepsilon L x_k^{(r)}) - x_k^{(r)}\right] \tag{25}$$

Model 2: $\varepsilon = 0$ の場合の変換を G と定義し，残りを摂動とする．

$$x_{k+1}^{(r+1)} = (1-w)x_k^{(r+1)} + wD^{-1}b + w\left[(D+\varepsilon U)^{-1}(b - \varepsilon L x_k^{(r)}) - D^{-1}b\right] \tag{26}$$

これらの Parareal-in-Time 法の収束性に関して，まだ十分な解析が終わっていないが，これまでに調べた範囲で報告する．D の要素 D_{jj} を -1 から 1 の一様乱数とし，L と U の各要素は，$\langle |a_{ij}|^2 \rangle = 1/4N$ を満たすガウス乱数とする．ここでは，ちょうど 100 回程度で収束する問題になるように ε の値を決めた ($N = 100$，$\varepsilon = 0.1$)．実ベクトル b も乱数から作成するが，$|b|^2 = 1$ と規格化しておく．最初に，SOR 法が収束することを確かめた上で，このベクトル列を Parareal-in-Time 法で再現できるかを調べる．

図 8 の (a) と (b) は，同じ行列 $D+\varepsilon(L+U)$ と右辺ベクトル b の問題を設定し，同じ初期ベクトル x_0 からスタートする SOR 法のベクトル列 $\{x_k\}$ と，(a) Model 1 と (b) Model 2 に対応する r 次の Parareal-in-Time 法で計算されるベクトル列 $\{x_k^{(r)}\}$ との差を表示したものである．一見して Model 1 の Parareal 系列は Model 2 のものよりも $\{x_k\}$ への収束が遅いことがわかるが，Model 1 の場合は，$K = 100$ の系列を計算するのに $r = 50$ でも収束していない状態であるから，十分な結果とは言えない．Model 2 は収束しているため，高速化が可能に見えるが，実はこれらの結果は $w = 0.2$ に対して得られたものである．これまでに調べた範囲では，SOR の加速パラメータ w に関して，Gauss-Seidel と等価な $w = 1$，および，一般に収束が加速される $1 < w < 2$ の場合には，Parareal 法が正しい系列 $\{x_k\}$ に収束する例を確認出来なかった．

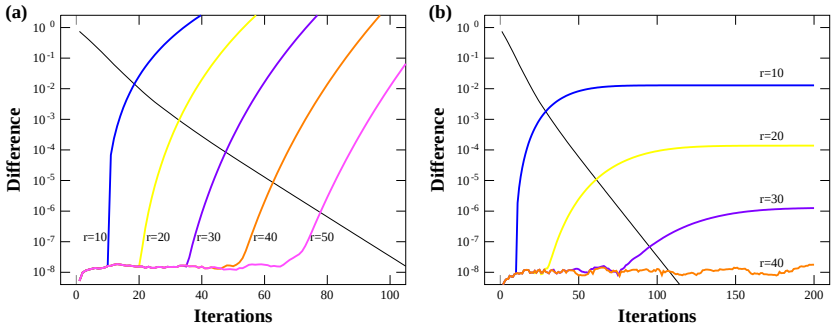

図 8　SOR の時系列に対する Parareal-in-Time 時系列の誤差: (a) Model 1，(b) Model 2．黒のカーブは，もともとの SOR 法でのベクトルの変化量 $|x_k - x_{k-1}|$．

通常，十分速く収束する問題に対しては，わざわざ繰り返し回数が長くなるような w の値を導入することはあり得ないため，少なくともこれまでに調べた範囲では，SOR 法に対して Parareal-in-Time 法は適した方法ではないということになる．

5. まとめと今後の展望

行列ベクトル積に Parareal-in-Time 法を適用し，その収束性とスピードアップを調べた．この問題は線形であることから，収束性の解析は，非線形の一般的な問題と比較して容易で，行列のスペクトル半径を利用して数値計算の結果得られる誤差と比較した．また，実際に並列計算機を使って，行列積の場合のスピードアップ比を測定し，この手法の有効性を検証した．ただし，1,000 コア近くを使って数十倍のスピードアップが限界であるため，計算効率の意味からはそれほど良い結果とは言えない．それでも，もともと並列化が難しい問題

の高速化であるため，低効率であっても必要とされる場合があると考えている．

行列計算の並列化は，通常，行，あるいは，列の独立性を利用して実行されることが多く，既に多くの線形ライブラリ[14] 内で実装されている．この場合，スピードアップの限界は，独立に計算できる行や列の数で決まるため，小規模行列に対しては期待したほどの性能向上が期待できない．本研究で検討した Parareal-in-Time 法は，空間方向とは独立に導入できるため，通常の並列化では性能向上に限界がある場合に効果を発揮する．データの並列性に加えて，時間方向のタスク並列性を利用して高速化を実施していることになる．もう一つの興味深い点は，通信の形態が，全体での同期通信を必要とせず，依存する方向への受け渡しだけで良いことである．この性質により，全体での集団通信やバリア同期のコストが莫大になる超並列機でも利用可能となる．

本研究では，これまでアプリケーションレベルで主に使われて来た Parareal-in-Time 法が，依存関係のある線形繰り返し計算の高速化に適用できることを示した．線形方程式の解法では様々な反復法が使われており，これらはほとんどすべて依存関係のある繰り返し計算であるため，本稿で検討した時間方向並列計算の対象となるが，すべての手法に Parareal-in-Time 法が適用可能であるわけではない．実際，まだ予備的な結果ながら，SOR 法に対して適用は可能なものの，高速化という目的にはそぐわないことがわかって来ている．一般に，繰り返し計算による変化量が比較的小さく，収束の遅いものほど Parareal-in-Time 法が適用しやすいと推定されるが，特定の線形計算に対する適用可能性は，ここで実行したようなスペクトル半径などによる解析を行うことで可能となる．今後，適用可能な反復法に対してスピードアップ比などの測定を行い，ある程度以上効果が期待できるものに関しては，線形ライブラリとして実装して行くことを考えている．

計算科学の分野への応用では，時間発展計算の並列化以外に，繰り返し計算を含む問題全般 (例えば，変分法を利用する大規模量子化学計算[15]) への適用可能性を探りたい．また，いわゆるマルチスケール現象[10] との関連からは，計算の効率化としてのブロック行列アルゴリズムやマルチグリッド離散化手法と，マルチスケール現象を捉えるための空間の粗視化記述との関連を研究しようとする動きと対応して，時間方向並列化手法の適用可能性と，動力学現象の記録・解析のためのデータ圧縮[16] との関連へと発展させたい．

謝　　辞

本研究は，2011 年度文部科学省科学研究費補助金，基盤研究 (C) (課題番号 23540454) による援助を受けている．数値計算は，九州大学情報基盤研究開発センターで実施した．

参 考 文 献

1) J. Lions, Y. Maday, and G. Turinici, “A ‘parareal’ in time discretization of PDE’s,” C. R. Acad. Sci., Ser. I: Math. **232**, 661–668 (2001).
2) M. J. Gander and S. Vandewalle, “On the Superlinear and Linear Convergence of the Parareal Algorithm,” LNCSE **55**, 291–298 (Springer, 2007).
3) L. Baffico, S. Bernard, Y. Maday, G. Turinici, and G. Zérah, “Parallel-in-time molecular-dynamics simulations,” Phys. Rev. E**66**, 057701 (2002).
4) G. Bal and Q. Wu, “Symplectic Parareal,” LNCSE **60**, 401–408 (Springer, 2008).
5) T. Hara, T. Naito, and J. Umoto, “Time-periodic finite element method for nonlinear diffusion equations,” IEEE Trans. Magn., MAG-21, 2261–2264 (1985).
6) K. Burrage, “Parallel and Sequential Methods for Ordinary Differential Equations,” (Oxford University Press Inc., NewYork, 1995).
7) M. J. Gander, “Analysis of the Parareal Algorithm Applied to Hyperbolic Problems Using Characteristics,” Bol. Soc. Esp. Mat. Apl. **42**, 21–35 (2008).
8) M. J. Gander and E. Hairer, “Nonlinear Convergence Analysis for the Parareal Algorithm,” LNCSE **60**, 45–56 (Springer, 2008).
9) G. A. Staff and E. M. Rønquist, “Stability of the Parareal Algorithm,” LNCSE **40**, 449–456 (Springer, 2005).
10) M. Duarte, M. Massot and S. Descombes, “Parareal Operator Splitting Techniques for Multi-scale Reaction Waves: Numerical Analysis and Strategies,” Math. Model. Num. Anal. **45** (5), 825–852 (2011).
11) Y. Maday, G. Turinichi, “Parallel in Time Algorithms for Quantum Control: Parareal Time Discretization Scheme,” IJQC **93**, 223–228 (2003).
12) T. Takami and H. Fujisaki, “Analytic approach for controlling quantum states in complex systems,” Phys. Rev. E**75**, 036219 (2007).
13) E. Aubanel, “Scheduling of tasks in the parareal algorithm,” Parallel Computing **37**, 172–182 (2011).
14) A. Nishida, “Experience in Developing an Open Source Scalable Software Infrastructure in Japan,” Proc. ICCSA 2010, Part II, LNCS **6017**, 448–462 (2010).
15) Y. Inadomi, T. Takami, J. Maki, T. Kobayashi, and M. Aoyagi, “RPC/MPI Hybrid Implementation of OpenFMO — All Electron Calculations of a Ribosome,” in Proc. ParCo2009, Adv. Par. Comp. **19**, 220–227 (2010).
16) 高見利也，戸田幹人，福水健次，“一次データを保存しない大規模科学計算の可能性,” 情報処理学会 研究報告 HPC-129-3, 1–8 (2011).