Vol.2014-HPC-143 No.12
2014/3/3

恒等変換による時間並列化法Identity Parareal の性能とバケツリレー通信

高見 利也[1,2,a)]　福留 大貴[3]

概要：時間並列化法として広く知られる Parareal-in-Time 法の問題点を解析し、その改良のために、近似関数として恒等変換 (identity) を利用した Identity Parareal 法を提案する。この新しい構成方法による時間発展計算の収束性とスピードアップを調べるとともに、この手法を並列に実装するために利用されるバケツリレー通信について、その性質と高速化手法の実測結果について報告する。

キーワード：時間並列化, Parareal-in-Time 法, Identity Parareal 法, バケツリレー通信

1. はじめに

シミュレーション等における時間発展計算、すなわち、常微分方程式、偏微分方程式の初期値問題の数値計算では、通常、時間について逐次的に計算が実行される。離散化した時刻を一つ進める計算は、領域分割などによって大規模な並列計算が可能であるため、これまで時間発展計算の並列化と言えば、空間的な並列化を意味してきた。しかし、特定の問題を高速に解くために、より多くの計算機リソースを利用しても、強スケーリングにはどこかで限界が来ることは明らかであるため、今後どこかの段階で、時間発展計算そのものを並列に計算することを考えなくてはならなくなる。近年、大学等の計算機センターでも比較的大きな台数の並列計算が容易に利用可能になっているため、研究者が日常的に研究している問題規模では、空間並列化によるスピードアップの飽和が見られる場合が多い。一方、京コンピュータなど超大規模な計算機でも、問題規模を大きくすれば空間並列化の効果は高くなるが、大きな問題になればなるほど現象再現のための時間スケールはのびる傾向にあるため、より長い時間発展計算を実施する必要が生ずる。単なるベンチマークではなく実問題を対象として、現実的な時間内に意味のある計算結果を得るという条件を付けると、やはりどこかの段階で、空間並列化だけに頼ることが出来なくなり、時間方向にも並列化を検討することが必要となる。

[1] 九州大学 情報基盤研究開発センター
[2] JST CREST
[3] 九州大学 大学院 システム情報科学府
[a)] takami@cc.kyushu-u.ac.jp

時間方向並列計算のアルゴリズムとして広く知られているのは、2001 年に Lions たちによって提案された Parareal-in-Time 法 [1] である。この手法による並列計算は、分子動力学や量子状態制御などの常微分方程式系 [2], [3] から流体力学などの偏微分方程式系 [4], [5], [6], [7], [8], [9], [10] まで、多くの種類の時間発展計算において収束性と有効性が調べられており、大規模な計算機上での応用 [11], [12], [13], [14] もされている。しかし、従来からこの手法の問題点が指摘されており、常に時間方向並列化の効果が得られるとは言えない。本稿では、この手法の問題点に対処するために新しい構成法を提案し、この構成による場合の収束性や適用範囲を明らかにするとともに、いくつかの問題に対して適用する場合の効果について検討する。また、このアルゴリズムを実装するときの通信部分を、バケツリレー通信として独立させ、この通信パターンの性質と高速化の方法に関して実験した結果を示す。

2. 時間並列化法 Parareal-in-Time の問題点

Parareal-in-Time 法は、離散化した時間 t_k での系の運動状態を表す時系列 $\{x_k\}$ $(x_{k+1} \equiv F_k(x_k))$ を計算する問題に対して、時間の方向 (k の方向) に並列計算を実施するためのアルゴリズムである。この手法は、k 方向の依存関係を近似関数 $G_k(x_k)$ を使って解消することにより、時間のかかる計算 $F_k(x_k)$ を並列に実施するもので、もともとの時系列の代わりに、

$$x_{k+1}^{(r+1)} = G_k(x_k^{(r+1)}) + F_k(x_k^{(r)}) - G_k(x_k^{(r)}) \qquad (1)$$

で定義される r 次近似の時系列 $\{x_k^{(r)}\}$ を求める、という形で記述される。ただし、$r \geq k$ に対しては、$x_k^{(r)} = x_k$ と

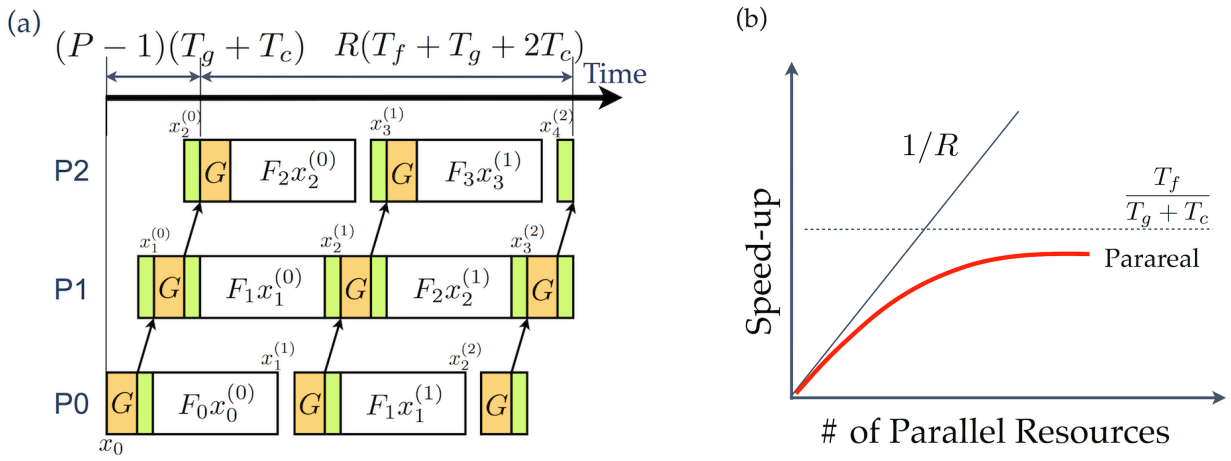

図 1　(a) 一般的な Parareal-in-Time 法の実行方法と (b) スピードアップの限界 (本文中では $T_g = 1/M$, $T_c = t$ と書かれている)。

定義しておく。

Parareal-in-Time 法による時間方向並列化計算では、依存関係の解消のための近似計算のコストと、収束した時系列を得るための r 方向の繰り返し計算のコストが新たに必要となるため、通常の並列計算のように 100%に近い効率を目指すことは困難である。この節では、時間並列計算を少しでも利用しやすいものとするために、式 (1) の問題点を洗い出して検討することとする。

2.1　一般化への障害

漸化式 (1) を利用するためには、近似計算 $G_k(x_k)$ を定義する必要がある。時間発展計算 $F_k(x_k)$ は、通常、空間離散化やモデル化によって定義されたものであるため、離散化のためのパラメータなどを調整することによって、より高速な計算方法 $G_k(x_k)$ を定義することが可能である。しかし、これらのパラメータは問題に応じて異なり、一般的な形で近似関数を定義することは難しい。

例えば、漸化式 (1) の並列計算を支援するためにアプリケーションプログラムインターフェイス (API) を作成する時、与えられた時間発展計算 $F_k(x_k)$ に加えて近似関数 $G_k(x_k)$ も、関数ポインタなどの形でライブラリに渡す必要がある。近似関数の定義は漸化式の収束性を大きく左右する重要な点であり、この点を利用者の定義にゆだねることは問題である。

2.2　スピードアップの限界

近似関数の善し悪しは、漸化式の安定性や収束性だけでなく、並列計算によるスピードアップにも影響する。一般的な近似関数の定義方法が与えられない状態では、時間並列化による効果さえも予測が困難になる。唯一、時間粗視化による方法 (時間刻み dt で M 回繰り返す計算を $F_k(x_k)$ とし、大きな時間刻み Mdt での計算を $G_k(x_k)$ とする) は、$F_k(x_k)$ を与えることで機械的に近似計算 $G_k(x_k)$ を決めることが可能である。陽的な時間発展計算の場合、この方法による近似計算は、通常、M 倍高速な計算となる。ここでは、この時間粗視化による方法を採用する場合に、スピードアップの限界を見ておく。

Parareal-in-Time 法の漸化式 (1) の一般的な並列実装は、図 1 のようになる。P 倍の並列リソースを利用し、R 回の繰り返し計算を M 倍速い近似計算で実現する場合に予想されるスピードアップ比は、

$$S(P,R,M,t) = \frac{KM}{K(1+Mt)+MR(1+t)} \qquad (2)$$

である。t は、データの転送など雑多なコストを表す。$P+R-1 \equiv K$ は、一度の計算で進められる時間ステップ数で、明らかにこの値が大きいほど高速化が可能である。通信などのコストを無視すると、K が十分に大きい場合の Parareal-in-Time 法のスピードアップの限界は、近似計算の高速化率 M に等しいことがわかる。

3.　Identity Parareal 法の提案と収束性

前節で明らかになった Parareal-in-Time 法の問題点を解決するために、近似関数として恒等変換を利用した Identity Parareal 法を提案し、その収束性を調べる。これまでも、恒等変換による近似は、個別の問題に対して検討したことがあったが [15], [16]、改めて名前をつけてその性質全般を調べることはしていなかった。この節では主に、収束性とスピードアップについて調べ、並列計算機への実装と高速化に関しては第 4 節で検討する。

3.1　近似関数としての恒等変換

Parareal-in-Time 法では、逐次計算部分に近似関数 $G(x)$ を使って高速化することが出来たが、問題に応じた $G(x)$ の調整の煩雑さやスピードアップ比の限界の存在など問題点の多くは、この $G(x)$ に関連するものが多いことがわかった。そこで、我々は、有効な近似関数 $G(x)$ を定義する

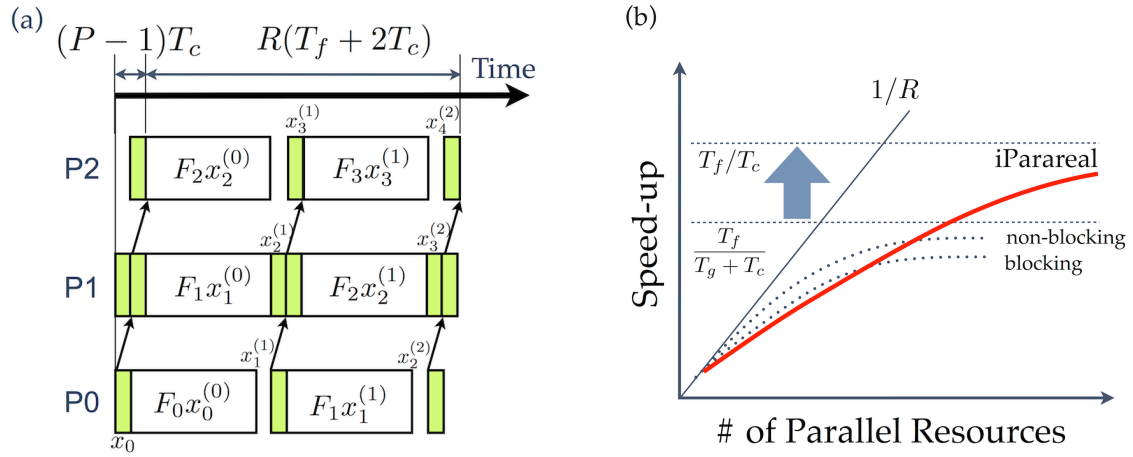

図 2　(a) Identity Parareal 法の実行方法と (b) スピードアップの性質

代わりに、恒等変換 $G(x)=x$ を使うことを提案する。これは、時間発展計算においては、前の時刻の状態をそのまま使うこと、すなわち、何も近似計算を実行しないことを意味する。如何に近似の精度を高めて Parareal-in-Time 法の収束を速くするかを研究してきた従来の方向とは、全く逆の試みであり、近似という意味では、明らかに改悪していることになる。また、この方法は、すべての依存関係のある計算 $x_{k+1}=F(x_k)$ に適用できる訳ではないことに注意する必要がある。しかし、時間発展計算の内、時間に対して連続性のあるほとんどすべての計算にこのような大胆な提案が可能であることは、次のようにして示される。常微分方程式 (空間部分を離散化した偏微分方程式も含む)、

$$\frac{d}{dt}x(t)=f(t,x(t)) \tag{3}$$

を、時間を離散化して形式的に表現すると、

$$x(t+dt)=x(t)+\frac{\partial f}{\partial x}dt+O(dt^2) \tag{4}$$

となるが、右辺第一項は前の時刻の状態と同じ $x(t)$、すなわち、恒等変換による結果である。時間の積分に関して高次の解法を利用する場合でも、右辺の第二項までは全く同じである。一見乱暴な近似に見えるが、時間に関して連続性が保証されている場合、つまり、有限の時間刻みを使って計算できる問題に対しては、恒等変換が正当化されるのである。

近似関数として恒等変換を利用して Parareal-in-Time 法を構成する時、繰り返し計算の漸化式は、

$$x_{k+1}^{(r+1)}=x_k^{(r+1)}+F(x_k^{(r)})-x_k^{(r)} \tag{5}$$

となる。この漸化式では恒等変換 (identity) を利用していることから、この方法による時間並列計算を Identity Parareal (iPararеal) 法と呼ぶこととする。

3.2　線形近似の範囲内での誤差評価

時間発展演算子が線形の場合 [15], [16], [17] には、この方法による収束性が証明出来るため、ここでは簡単に示しておくこととする。

形式的に時間発展を陽的に表現し、時間推進演算子 F_k が線形であるとすると、

$$x_{k+1}=F_kx_k=[I+(F_k-I)]\,x_k \tag{6}$$

と書ける。時間発展計算は右辺の線形演算子 $I+(F_k-I)$ を繰り返し状態ベクトルに適用することであるから、全部で k 回の適用では、

$$x_k=\prod_{k'=0}^{k-1}[I+(F_{k'}-I)]x_0 \tag{7}$$

である。ただし、右辺の演算子の積は時間の順序を保存する形で計算されるものとする。今、時間推進演算子 F_k は恒等変換に近い演算子であったことから、F_k-I による作用は微小であるため、右辺の積を展開し F_k-I の低次から並べると、

$$I+\sum_k(F_k-I)+\sum_{k>j}(F_k-I)(F_j-I)+... \tag{8}$$

という摂動的な展開となる。ここで、r 次までの近似演算子によって得られる時系列を $x_k^{(r)}$ と定義すると、

$$x_{k+1}^{(r+1)}=x_k^{(r+1)}+F_kx_k^{(r)}-x_k^{(r)} \tag{9}$$

という漸化式が満たされることと、有限次数での打ち切り誤差 $|x_k^{(r)}-x_k|$ が、

$$\frac{|x_k^{(r)}-x_k|}{|x_0|}\le\sum_{j=r+1}^{k}\left(\begin{array}{c}k\\j\end{array}\right)[\rho(F-I)]^j \tag{10}$$

となることがわかる。ただし、$\rho(A)$ は演算子 A のスペクトル半径とし、$\rho(F_k-I)$ の最大値を $\rho(F-I)$ とした。つまり、F_k が恒等変換に近い演算子であれば、有限の次数までの計算で、真の時系列 $\{x_k\}$ との差を十分に小さくすることが出来るのである。

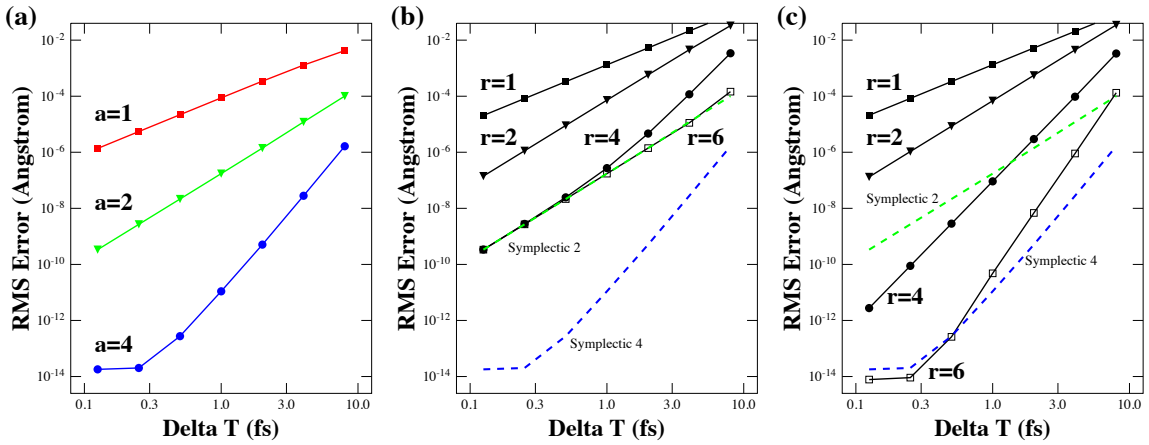

図 3　16 時間ステップの分子動力学計算の誤差: (a) 1 次、2 次、4 次のシンプレクティック (SI) 法による誤差、(b) $F(x)$ として 2 次の SI 法を使った場合の iPararеal 法の誤差、(c) $F(x)$ として 4 次の SI 法を使った場合の iPararеal 法の誤差。

3.3　時間発展計算への適用例

しかし、一般の時間発展計算は非線形であるため、この証明をそのまま適用することは出来ない。ただ、離散的な時間刻みを使った陽的時間発展計算が実施されている問題では、オイラー法 (4) の右辺第二項は、一般にそれほど大きくないことが期待される。以下で、代表的な問題に対して iPararеal 法を適用し、収束性を調べることとする。

3.3.1　分子動力学

統計力学的な環境のもとでのタンパク質などミクロスケールの運動や化学反応を記述する分子動力学計算 (Molecular Dynamics (MD)) は、統計力学、分子科学、分子生物学、薬学などの広い分野で実用化され、様々なシミュレーションが実施されている。一般的に、空間領域分割による並列化で効率の良い計算が実施されているが、高精度の計算ではフェムト秒 (fs, 10^{-15} sec) 程度の時間刻みが必要とされるにも関わらず、生体分子の運動はマイクロ秒 (μs, 10^{-6} sec) からミリ秒 (ms, 10^{-3} sec) のオーダーで観測されることが知られており、時間方向に非常に長い繰り返し計算を必要とする。

MD 計算で広く使われる二次のシンプレクティック積分法 (Velocity Verlet) では、$t_{k+1} = t_k + dt$ の運動状態を表す $6N$ 個の力学変数 (原子の位置座標 x_j と速度あるいは運動量 v_j) は、

$$x_j(t_{k+1}) = x_j(t_k) + \left[v_j(t_k) + \frac{f_j(t_k)}{2m_j} dt \right] dt \qquad (11)$$

$$v_j(t_{k+1}) = v_j(t_k) + \frac{f_j(t_k) + f_j(t_{k+1})}{2m_j} dt \qquad (12)$$

と表される。右辺の第一項は、ここでも前の時刻の状態そのものである。溶液状態や気体の状態の MD 計算は、初期値鋭敏性を強く持つ計算であることが知られているため、この問題での収束性を示しておくことは重要である。

図 3 に、iPararеal 法による誤差の収束の様子を示した。図 3(a) のシンプレクティック (SI) 法による離散化誤差の様子と比較すると明らかなように、時間発展計算として 2 次の SI 法を使った場合 (b) は (a) の 2 次の誤差曲線に、4 次の SI 法を使った場合 (c) は (a) の 4 次の誤差曲線に漸近する結果が得られている。この結果から、通常の MD 計算では $r = 4$ 程度の繰り返し、高精度の計算では $r = 6$ 程度の繰り返しが必要とされることがわかる。

3.3.2　Burgers 方程式

偏微分方程式を対象とした例として、Burgers 方程式

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \frac{1}{Re}\frac{\partial^2 u}{\partial x^2} \qquad (13)$$

での収束性を評価する。これは流体運動を表す Navier-Stokes 方程式から圧力項を除いて一次元にしたもので、流体的な運動を表す最も簡単な系の一つである。Re はレイノルズ数で、これが大きい程、粘性の小さい流体を表す。簡単のため、空間領域 $0 \le x < 1$ での流速 $u(x)$ を N 点等間隔メッシュ上の値 $u_j \equiv u(x_j)$ $(x_j = j/N, j = 0, 1, ..., N-1)$ で表現し、周期境界 $u_N = u_0$ とする。Euler 法の場合も同様であるが、中点法で時間積分する場合、

$$u_j^* = u_j(t) + \left[\frac{1}{Re}\frac{\delta^2 u_j(t)}{\delta x^2} - u_j \frac{\delta u_j(t)}{\delta x} \right] \frac{dt}{2} \qquad (14)$$

$$u_j(t+dt) = u_j(t) + \left[\frac{1}{Re}\frac{\delta^2 u_j^*}{\delta x^2} - u_j^* \frac{\delta u_j^*}{\delta x} \right] dt \qquad (15)$$

より、近似として恒等変換が利用できることがわかる。

レイノルズ数 $Re = 100$ の場合に、Euler 法・中点法 (2nd Runge-Kutta) による数値解と厳密解との差をプロットすると、図 4(a) のようになる。ここでは、初期条件として $u(x) = \sin(2\pi x)$ を与え、$t = 0.25$ での誤差を評価した。点線で示した Euler 法の誤差は時間刻みによって変動するが、中点法では十分に高精度な時間積分になっているため時間刻みによらず、安定に計算出来る範囲内では空間差分 (5 点中間差分) の誤差で決まる一定値となる。

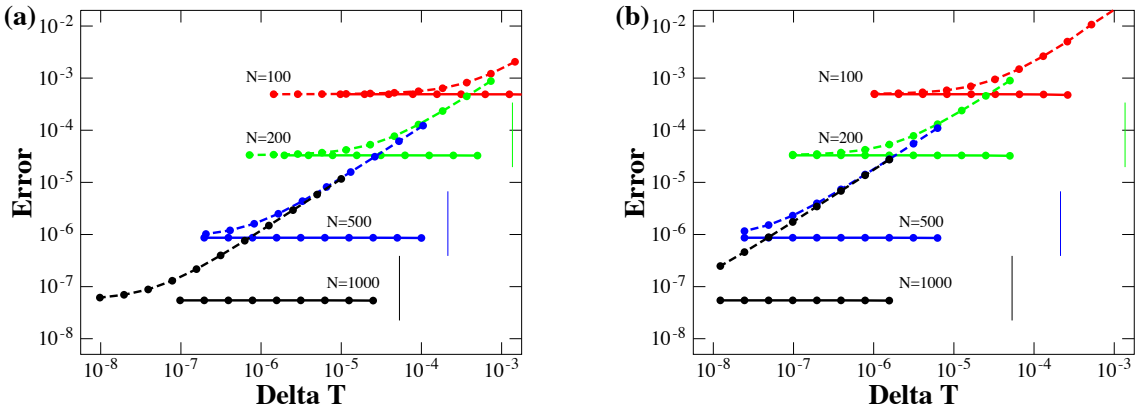

図 4　(a) バーガース方程式の離散化誤差 (点線は Euler 法、実線は中点法によるもの)、
(b) iPararеal 法 ($P = 16$) による誤差 (点線は $R = 1$、実線は $R = 2$ の結果)

図 4(b) は、$F_k(x_k)$ として中点法による時間積分を利用した iPararеal 法の誤差を厳密解と比較したものである。$R = 1$ では収束が不十分だが、$R = 2$ では、通常の中点法による結果と同様に、時間刻みによらず収束していることがわかる。なお、ここには示していないが、時間粗視化による通常の Paraeral-in-Time 法は十分に速く収束し、iPararеal 法の $R = 2$ の結果とほぼ同じ誤差となる。つまり、時間粗視化による近似関数を使った計算では $R = 1$ で収束し、我々が導入した iPararеal 法では $R = 2$ で収束するという結果である。

最後に安定性についてコメントをしておく。この問題設定の場合に時間刻み幅が満たすべき安定性条件は、

$$dt \leq \frac{Re}{2N^2} \tag{16}$$

となり、図 4(a) と (b) に、この限界値を縦の線で示した。陽的な時間発展計算では、誤差の増幅を招くためこれより大きい dt を使うことは不可能である。図に示した数値計算では、出来るだけ大きな dt で計算し、安定に解が得られたところまでを示したが、明らかに iPararеal では、安定性の限界が低くなっていることが読み取れる。図中には示していないが、この安定性限界の低下は、時間粗視化による Parareal-in-Time 法の計算でも同様に見られた。移流方程式系の陽的時間発展では、高次風上差分法を利用することで安定性が確保できることが知られているため、時間並列化を効果的に適用するためには、安定性限界の高い方法との組み合わせで実装することが必要となる。

3.4　スピードアップ比とデータ量、計算量

我々が導入した iPararеal 法を計算科学の問題に適用した場合のスピードアップ比がどのように予想されるかを、転送データ量と計算量の関係から検討する。図 2(a) の実行方法をとる場合のスピードアップ比は、

$$S(P, R, t) = \frac{K}{Kt + R(1 + t)} \tag{17}$$

である。ここでも $K = P + R - 1$ は有限の値に制限されるが、通信などのコスト t が十分小さければ、スピードアップの限界値はかなり高いところにあることがわかる。

1 時間ステップの計算量を 1 としたときの転送データ量に依存するコスト t が、どの程度かを見積もっておくことが重要である。例えば MD 計算の例では、転送量は $6N$ 個の力学変数 ($48N$ Bytes) であるが、計算量は $O(N^2)$ のオーダーであるため、ある程度 N が大きい場合は、$t \ll 1$ となる。長距離相互作用を近似して $O(N \log N)$ 程度の計算量に押さえている場合でも、比較的小規模の FFT 計算の場合は計算の実行性能があまり高くないために、やはり通信は無視できる程度である。流体計算などの場合は、データ転送量が $8N$ Bytes であるのに対して、計算量が $O(N)$ であることから、$t \approx 1$ となることもあり得る。

すでに見たように、Burgers 方程式を時間並列化する場合は安定性が問題になったが、流体などの偏微分方程式系で、データ転送量と計算量が同程度となる問題は、性能の面からも注意深く適用すべきである。流体系の数値計算でも、乱流の時間発展計算など初期値鋭敏性の高い系では、状況が異なってくることが考えられるため、それぞれの問題の特性を詳しく解析する必要がある。

4.　バケツリレー通信

前節までで検証した iPararеal 法を並列計算機上に実装するための通信パターンに関して、MPI による実装と性能測定結果を示し、さらなる性能向上に向けての検討結果を示す。iPararеal 法を分散並列計算する時は、漸化式 (5) を

$$y_k^{(r+1)} = F_k(x_k^{(r)}) - x_k(r) \tag{18}$$

$$x_{k+1}^{(r+1)} = x_k^{(r+1)} + y_k^{(r+1)} \tag{19}$$

と分割し、前半をプロセス毎の計算、後半をバケツリレー通信関数として実装する [19]。例えば、MPI の同期通信で書くと、次のような形となる。

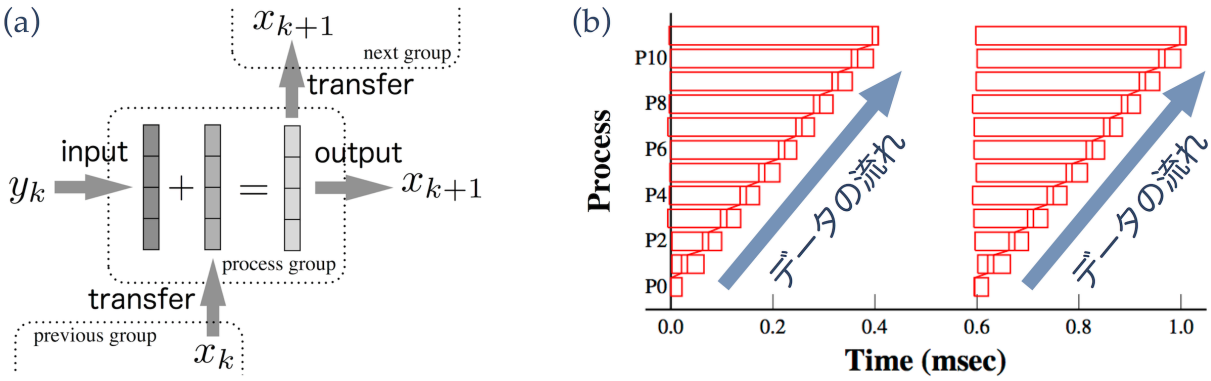

図 5　(a) バケツリレーの概念図と、(b) 実行のイメージ。

Procedure: バケツリレー通信

```
1. void bb(double *x, double *y, int n,
           int prev, int next) {
2.     MPI_Recv(x, n, MPI_DOUBLE, prev, ..);
3.     for (int i = 0;  i < n;  i++)
4.         x[i] += y[i];
5.     MPI_Send(x, n, MPI_DOUBLE, next, ..);
6. }
```

ここで定義したバケツリレーは、図 5(a) に示すように、複数プロセス間で一方向にデータを転送しながら演算を行う、最も単純なパイプラインパターンである。これは、Reduce 的な隣接集団通信の一種で、特定のプロセス間にのみ一方向の転送が実行されることが特徴的である。

バケツリレーの様子を時系列として可視化したものを図 5(b) に示す。12 プロセス間のバケツリレーを 2 回繰り返して測定したものであるが、各プロセスで現れる三個の長方形は、受信待ち、計算、送信待ちの状態に対応したものである。この図から読み取れるように、P0 でバケツリレーが開始されてから P11 で完了するまで、約 0.4msec の時間がかかっている。この通信と計算のパターンは、あるプロセスでの遅延が、後のプロセスへと蓄積されてしまう形であるため、各部分での効率的な実行が重要である。

4.1　非同期通信の利用

バケツリレー通信のレイテンシを隠蔽するために、非同期通信の利用を検討する。最も単純には、先に示した同期通信関数によるバケツリレー通信を非同期通信で書き換えれば良い。このようにすることで、バケツリレー通信の前後にある別の演算とオーバーラップさせ、特に受信待ちのレイテンシを削減することが出来る (図 6(a) の上半分)。

この場合、隣接集団通信の非同期化 [20] と同様、集団通信の外部の演算とのオーバーラップを実現するためには、バケツリレー通信関数の非同期化と同期ポイントの設定、集団通信アルゴリズム進行の管理など煩雑な処理が増えてしまうこととなる。ここでは、この方法による高速化は採用せず、次の節で見るようにバケツリレー通信内部の演算と通信のオーバーラップによる高速化を試みる。

4.2　データの細分化

バケツリレー通信はデータ転送と単純な和の計算を組み合わせた通信パターンである。しかし、すべてのデータを受信するまで待って演算を開始し、演算がすべて終わった後に転送するという方法では、無駄な待ち時間が大きくなってしまう。このような場合は、データを細分化してパイプライン的に実行することで、通信、あるいは、演算の部分を隠蔽できる (図 6(a) の下半分)。バケツリレーは片方向の通信であるため、受信と送信をオーバーラップさせることも可能である。データの細分化と非同期通信を利用して実装することで、大きいデータの場合に高速化されることが実測出来る。

図 6(b) は、12 プロセス間で N 個の倍精度実数データを、$m = 2,\ 4,\ ...,\ 64$ などに細分化してバケツリレー転送し、実測したものである。データサイズが小さい場合は平均的に 0.1 msec 前後のレイテンシが必要で、12 プロセス全体のスループットは 10 MBytes/sec 程度であるが、データサイズが大きい $N = 10^6$ と $N = 10^7$ の場合は、分割数を大きくすれば、約 4 msec、40 msec で転送を完了することが出来、2 GBytes/sec 程度のスループットが得られている。全体 (P0 は演算をしないため 11 プロセス) で 3 GFlops 弱の演算を行っていることになるが、異なるプロセスの演算は異なる CPU で同時に実行されるため、決して過大な性能ではない。通信と計算のチューニングにより、さらに高性能を目指すことは可能であると考えられる。

ここで採用した高速化の方法は、バケツリレー関数内部でのオーバーラップを実施したものであるため、プログラムの構造は煩雑にならずにすむ。時間並列計算など、パイプライン転送のレイテンシが性能向上に重要となる問題に適用するとき、特に流体計算など、計算量に比べて時間方向の転送データ量が大きい場合には効果的である。

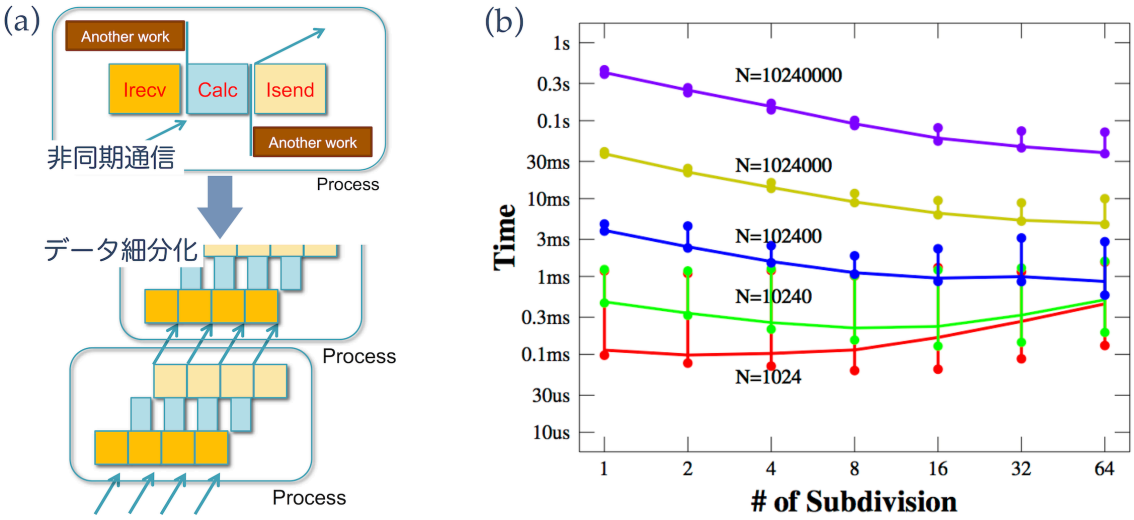

図 6　(a) 非同期通信とデータ細分化による高速化、(b) 転送時間の実測結果

5. 関連研究

Parareal-in-Time 法に関して、効果的な近似関数を導入して性能向上を図る試みは、ドイツの Jülich Supercomputer Center のグループが精力的に実施している [13], [14]。彼らは、時間の粗視化と同時に空間方向にもマルチグリッド的なアプローチを行うことによって、時空間マルチグリッド法として時間並列化計算を行っている。彼らの手法は、まさに正攻法として近似関数をするものである。近似計算の高速化と同時に空間粗視化によるデータ量の削減も達成できるため、収束性だけでなく性能面も考慮した手法となっている。拡散方程式や流体系などで解法の安定性が問題になる時に、さらに威力を発揮することが期待される。

我々の iPararеal 法は、彼らとは全く逆の方向を狙ったもので、近似関数を極限まで単純化する代わりに、パイプライン化による通信性能の向上を期待して実装するというもので、プロセス間のデータ転送を積極的に利用する形の構成となっている。通常の分散並列計算では、通信量を削減することが至上命令となっていることが多いが、この意味で我々の方法はあまり例を見ないものであると考える。

パイプライン的なアルゴリズムに関連しては、マルチコアプロセッサ向けの線形演算の高速化手法が存在する [21]。分散並列環境でパイプラインを利用した高速化手法についても広く研究されており、MPI の集団通信において、データの細分化と転送により高速化を達成するという研究がなされている [22]。ここで我々が導入したバケツリレー転送は、`MPI_Reduce` と比べても単純なパターンであるため、時間並列計算以外の場面での汎用性はわからないが、依存関係を解決する手段として通信が利用される以上、このような通信パターンに合致する数値計算も存在すると考える。他の応用例に思い当たる方は、情報をお寄せいただけるとありがたい。

6. まとめ

恒等変換 (identity) を近似関数として利用した Parareal-in-Time 法 (Identity Parareal 法) を提案し、分子動力学と Burgers 方程式での収束性などを調べた。また、時間並列化による性能向上に関して、運動状態を表す力学変数のサイズと計算量の関係も重要であることを指摘した。この計算の分散並列化で利用する通信パターンをバケツリレー通信として構成し、パイプライン化による通信と演算のオーバーラップの結果、大幅にスループットが向上することを実測により示した。我々が提案した iPararеal 法では、この性能向上の恩恵が受けられるため、安定性の問題を解決すれば、流体など偏微分方程式系でも効果的な時間並列化を実施することが可能になる。

謝辞　本研究は、科学研究費補助金基盤 (C)「身近な非線形現象に対するマルチスケール的解析手法の確立と応用」(課題番号 23540454) の支援を受けている。また、本研究の数値計算には、九州大学情報基盤研究開発センターの CX400、12 ノードを利用した。

参考文献

[1] Lions, J., Maday, Y., and Turinici, G.: A ‘parareal’ in time discretization of PDE’s. C. R. Acad. Sci., Ser. I: Math. **332**, 661–668 (2001).
[2] Baffico, L., Bernard, S., Maday, Y., Turinici, G., and Zérah, G.: Parallel-in-time molecular-dynamics simulations. Phys. Rev. E **66**, 057701 (2002).
[3] Maday, Y., Turinichi, G.: Parallel in Time Algorithms for Quantum Control: Parareal Time Discretization Scheme. IJQC **93**, 223–228 (2003).
[4] Farhat, C. and Chandesris, M.: Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluidstructure applications. Int. J. Numer. Meth. Engng. **58**, 1397–1434 (2003).
[5] Fischer, P. F., Hecht, F., and Maday, Y.: A Parareal in

Time Semi-implicit Approximation of the Navier-Stokes Equations. LNCSE 40, 433–440 (2005).
[6] Gander, M. J. and Vandewalle, S.: On the Superlinear and Linear Convergence of the Parareal Algorithm. LNCSE **55**, 291–298 (Springer, 2007).
[7] Gander, M. J. and Vandewalle, S.: Analysis of the parareal time-parallel time-integration method. SIAM J. Sci. Comput. 29, 556–578 (2007).
[8] Samaddar, D. and Newman, D. E., and Sánchez, R.: Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm. J. Comp. Phys. 229, 6558–6573 (2010).
[9] Duarte, M., Massot, M., and Descombes, S.: Parareal operator splitting techniques for multi-scale reaction waves: Numerical analysis and strategies. ESAIM: Math. Mod. Num. Analysis 45, 825–852 (2011).
[10] Ruprecht, D. and Krause, R.: Explicit parallel-in-time integration of a linear acoustic-advection system. Computers & Fluids **59**, 72–83 (2012).
[11] Aubanel, E.: Scheduling of tasks in the parareal algorithm. Parallel Computing **37**, 172–182 (2011).
[12] Elwasif, W. R., Foley, S. S., Bernholdt, D. E., Berry, L. A., Samaddar, D., Newman, D. E., and Sanchez, R.: A dependency-driven formulation of parareal: parallel-in-time solution of PDEs as a many-task application. in Proc. MTAGS'11, 15–24 (2011).
[13] Speck, R., Ruprecht, D., Krause, R., Emmett, M., Minion, M., Winkel, M., and Gibbon, P.: A massively space-time parallel N-Body Solver. in Proc. SC12, Technical Paper No. 92, 1–11 (2012).
[14] Speck, R., Ruprecht, D., Emmett, M., Bolten, M., Krause, R.: A space-time parallel solver for the three-dimensional heat equation. International Conference on Parallel Computing, B5-1 (Munich, 10-13, Sep. 2013).
[15] 高見利也，西田 晃: 時間方向並列化の線形計算への適用可能性, 情報処理学会研究報告 Vol. 2011-HPC-131, No.6, 1–8 (2011).
[16] T. Takami and A. Nishida, "Parareal Acceleration of Matrix Multiplication," Adv. Par. Comp. **22**, 437–444 (2012).
[17] Bal, G.: On the Convergence and the Stability of the Parareal Algorithm to Solve Partial Differential Equations. LNCSE 40, 425–432 (2005).
[18] 高見利也，福留大貴: 時間並列化法の通信パターンと効率実装, 情報処理学会研究報告 Vol. 2013-HPC-138, No.12, 1–6 (2013).
[19] Fukudome, D. and Takami, T.: Parallel Bucket-Brigade Communication Interface for Scientific Applications. in proc. EuroMPI'13, 135–136 (2013).
[20] Hoefler, T., Lumsdaine, A., and Rehm, W.: Implementation and Performance Analysis of Non-Blocking Collective Operations for MPI. in Proc. SC07 (IEEE Computer Society/ACM, 2007).
[21] Kurzak, J. and Dongarra, J.: Implementing Linear Algebra Routines on Multi-core Processors with Pipelining and a Look Ahead. LNCS 4699, 147–156 (2007).
[22] Worringen, J.: Pipelining and overlapping for MPI collective operations. in Proc. IEEE Conference on Local Computer Network, 548–557 (2003).