Vol.2013-HPC-138 No.12
2013/2/21

時間領域並列化法の通信パターンと効率実装

高見 利也[1,2,a)]　福留 大貴[3]

概要：偏微分方程式の初期値問題を時間方向に並列化する Parareal-in-Time 法は，空間方向の領域分割による高速化が飽和する場合に効果的に導入することができる．本研究報告では，大規模な並列計算機での現実的な利用を想定し，MPI による並列化を実施する場合の通信パターンについて考察する．例として，量子力学の時間発展計算に現れるユニタリ変換をモデル化し，空間と時間の両方向の通信パターンを提示する．可能な限り詳細な実測値に基づき，通信と計算のオーバーラップを含む，効率的な実装について検討した結果を報告する．

1. はじめに

現在，国内でも京をはじめとして多くの超並列計算機が利用可能になっており，物理学や化学などの科学研究分野においても，大規模な並列化が実施されている．例えば，熱，流体，電場，音場などの時間変化を扱う偏微分方程式の初期値問題は，空間自由度を有限要素法などによって離散化することで，時間に関する常微分方程式の形に変形することが出来るため，空間方向の大規模な並列性を利用して効果的な並列計算が構成され，科学シミュレーションなどとして広く実施されている．この場合の並列化は，多数の空間要素を領域毎に分割することで実施され，領域間の微小な共通部分 (袖領域などと呼ばれる) の情報を交換し，全体としての整合性を維持している．この場合に効率的な並列計算が実施できるのは，現実的な問題の場合は空間方向の要素数が膨大で，超並列計算機を使って領域分割しても，各領域に含まれる要素数が一定数以上確保できることから，並列化によって増大する情報交換のための通信コストが，その領域内の計算コストに比べて，比較的小さく押さえられているためである．このような，空間方向の独立性を最大限に利用した大規模並列計算では，通信に関連するコストの低減，あるいは，隠蔽に関して，さまざまな手法が研究されている．

ところが，現時点の大規模計算機ですでに並列コアの数が 100 万前後になっており，今後もエクサフロップスの計算機に向けてさらに増えていく可能性が高い．この状態では，特別に大規模な空間要素数を持つ計算ではない限り，これらのコアを効率的に利用することが難しくなってきている．複雑な空間構造を持つ問題に対しては，さらに詳細な空間分割が要求されていることが多いことから，計算機リソースに応じた大規模分割を実施することで，並列実行効率の維持は可能である．この場合の問題は，空間表現の詳細さに応じて時間方向も詳細な計算が必要になることであり，現実的な現象を再現するための数値計算を完了するためには，より多くの回数の時間ステップが必要になる．一方，計算の手法や対象とする問題によっては，指定された問題サイズを変えられないものも存在し，このような問題に対しては，一定数以上の並列コア数を利用しても，計算の効率を上げることは不可能である．結局，全体として最も効率的に問題を解くためには，空間分割を一定数以下に押さえることが必要になり，これ以上の並列計算機は不要であるという結論を招く．

本報告は，現実問題への応用を考えるとこれ以上の超並列計算機は不要である，という議論を展開することが目的ではない．上記は，空間分割による並列化という一つの方向にのみ、問題解決の方法を求めたことにより、生じた結果である．ここでは，空間方向の離散化によってしわ寄せのいった時間積分に対して，高速化を図るための時間領域分割手法について，時空間並列化による通信パターンとその効率化という観点から議論する．

時間領域分割を実現するための手法の一つが，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いる [2]．また，最近の大規模並列計算機で大規模に時空間並列化計算を実装し，性能向上のための研究を行っている例もある [3], [4], [5]．この方法の収束性などについての理論的な

[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
[a)] takami@cc.kyushu-u.ac.jp

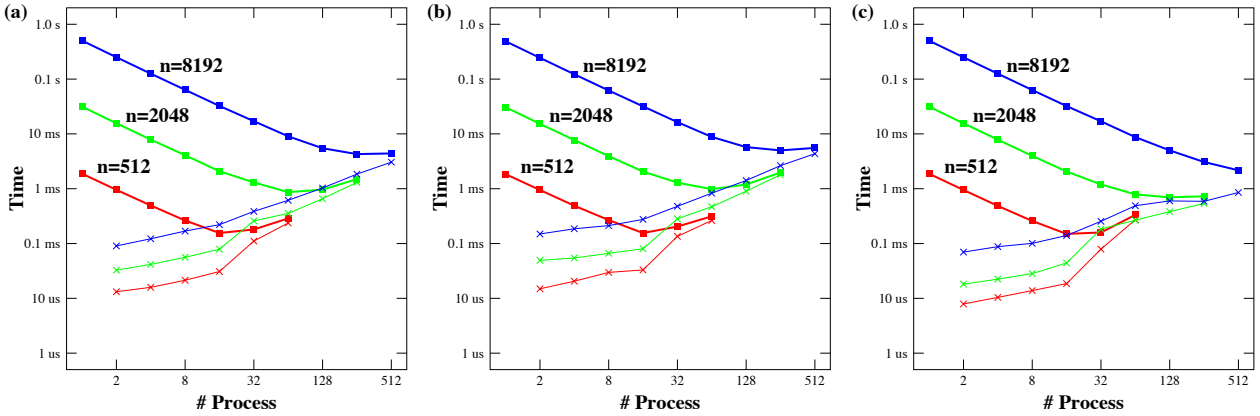

図 1　行列ベクトル積のフラット MPI による並列計算における時間短縮効果: (a) Bcast/Gather, (b) Scatter/Reduce, (c) Allgather. ■は計算と通信を合わせた時間, ×は通信部分の時間を表す.

解析は，既に多数の文献が出版されているため，そちらを参照いただくこととする．また，線形計算に適用した場合の Parareal-in-Time 法の収束性や，これによる高速化率などについては，筆者らの検討結果 [6], [7] もあるが，本稿では，Parareal-in-Time 法の並列計算機での実装に関して，特に通信部分にしぼって検討する．

この報告の構成は以下の通りである．まず第 2 章では，ここで対象とする計算問題について解説し，次の第 3 章で，この並列化法の実装方法とスピードアップについて理論的な予想を行う．第 4 章では，実測結果をもとに，更なる効率化のための方向を検討する．

2. 少数自由度系の時間発展計算

時間領域分割の手法と実装に入る前に，本研究で対象としている計算について簡単に解説し，通常の並列化だけを利用する場合の性能劣化の様子を示すこととする．物理学の研究に限定しても，計算科学にはさまざまな対象と手法があるが，ここでは，比較的少数自由度の系を，詳細に計算するような場合を対象とする．例えば，量子状態制御のための最適制御外場を求める計算 [8], [9], [10] など，非常に長時間の時間発展計算を必要とするようなものである．

実際の研究で使用する計算内容にできるだけ近い問題を使って評価するため，ここでは量子系の時間発展計算を評価対象とする．波動関数 $|\psi(t)\rangle$ で表される量子系を制御するために，外部から弱い古典外場を付加するとき，解くべき Schödinger 方程式は，

$$i\hbar\frac{\partial}{\partial t}|\psi(t)\rangle = \left[\hat{H}_0 + \varepsilon(t)\hat{V}\right]|\psi(t)\rangle \qquad (1)$$

である．外場が与えられているためにエネルギーは保存しないが，時間的に変動する外場 $\varepsilon(t)$ が弱いときは，孤立系の量子力学とほとんど同様のシンプレクティック積分法を構成することが出来る．以下，計算量の評価を行うために，波動関数 $|\psi(t)\rangle$ は n 成分の複素数ベクトルであるとする．時間発展演算子が座標と運動量のそれぞれの表現に分離できる場合は，高速フーリエ変換 (FFT) による表示の変換で高速な時間発展計算が可能で，この場合の時間発展 1 ステップにかかる計算量は，$O(n\log n)$ の次数となる．一方，エルミート演算子 $\hat{H}_0$ や $\hat{V}$ の固有基底による表現を取っている場合は，ユニタリ演算子 $\exp\left[\hat{H}_0\delta t/i\hbar\right]$ と $\exp\left[\frac{\varepsilon(t)\delta t}{i\hbar}\hat{V}\right]$ に分割する形のシンプレクティック法を適用することになるが，この場合の基底変換に要する計算量は，通常は $O(n^2)$ となる．このように，n 成分の量子状態ベクトルに対する時間発展計算は，$O(n\log n)$ または $O(n^2)$ の計算が必要であるが、この後の評価では $O(n^2)$ の場合を検討することとし，簡単のため，$n\times n$ のユニタリ行列 $U\equiv\exp[-i\delta tH]$ による時間発展計算

$$|\psi(t+\delta t)\rangle = U|\psi(t)\rangle \qquad (2)$$

を対象とする．この場合，時間ステップ 1 回に含まれる計算量は，複素数の積が n^2 回，和／差が n^2 である．これは，浮動小数点の計算数では合計 $8n^2$ 回の計算に相当する．

では，この計算を並列に実施する場合の性能は，現在の計算機では，どの程度のものになっているだろうか．C/C++ で書いたプログラムを MPI で並列化し，クラスタ計算機で実測した結果を示す．図 1 は，複素数の行列ベクトル積の計算時間を，様々な並列数で測定した結果である．問題サイズとしては，$n=512, 2048, 8192$ の三通りについて，全体の時間と，そのうちの通信に要した時間を表示してある．集団通信を使った行列ベクトル積の並列化には様々な方法があるが，ここでは，(a) Bcast-Gather によるもの，(b) Scatter-Reduce によるものに加えて，ベクトルがあらかじめ分散配置されている場合を想定した，(c) Allgather を利用するものの三種類について示す．この結果は，フラット MPI の並列プログラムの性能を，32 ノードの SandyBridge (ノード内 16 コア) クラスタ (InfiniBand FDR) 上で計測した．一度の集団通信だけですむ (c) の場合が若干高速であ

るが，傾向としてはほぼ同じで，$n = 512$ (赤色) では 16 並列，$n = 2048$ (緑色) では 64 並列から 128 並列，$n = 8192$ (青色) で 256 から 512 並列あたりまで並列性能が伸びているが，いつまでも性能が伸び続けるわけではないことは明らかである．

問題サイズを固定した計算のスケーラビリティに限界が生じるのは，並列化によってコアあたりの計算量は確実に小さくなるものの，グラフにも示されているとおり通信時間は逆に増大してしまうためである．特に $n < 1000$ 程度の小規模な問題では，比較的早い段階で並列化による効果が期待できなくなるため，超並列計算機を利用しても，並列数を上げるだけではこれ以上の高速化を行うことは不可能である．

3. 時間領域分割手法の実装

前章のように比較的小規模な問題を並列計算機上でさらに高速化するためには，厳密な依存関係が存在する時間方向の並列計算を実施することになる．アルゴリズムや計算方法を変えないままで，時間方向に効率的な並列化手法を導入することは難しいため，Parareal-in-Time 法のようになんらかの近似的な手法を導入することとなる．この章では，簡単に Parareal-in-Time 法の計算手順を解説した後，それらのスピードアップ比について検討する．

Parareal-in-Time 法を，式 (2) のように直前の状態に依存する時間発展計算など，漸化式

$$x_{k+1} = \mathcal{F}_k(x_k) \tag{3}$$

で表される計算に適用すると，この関係式を直接計算する代わりに，

$$x_{k+1}^{(r+1)} = \mathcal{G}_k(x_k^{(r+1)}) + \mathcal{F}_k(x_k^{(r)}) - \mathcal{G}_k(x_k^{(r)}), \tag{4}$$

という計算を行うこととなる．ここで $\mathcal{G}_k$ は，もともとの解法 $\mathcal{F}_k$ の近似解法で，$x_k^{(r)}$ は，時刻 k の状態に対する r 次の近似解を表す．$\mathcal{F}_k(x_k^{(r)})$ 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ため，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，並列計算による高速化が可能になる．この手法の収束性や適用範囲に関する解析は，既に多くの文献が出版されているため，それらを参照されたい．

3.1 Parareal-in-Time 法の実装とスピードアップ比

さまざまな文献で検討されているように，時間方向の分割数を多くとると，収束性に問題が生じることとなり，r 方向の繰り返し回数を多くとる必要が生ずる．この場合は計算の効率という点で不利になるため，ここでは，実用的な観点から，比較的少ない分割数での実装と効率について検討する．この場合，空間方向に関しては，既に示したように最も高速化が可能な領域で並列計算が行われているものとし，これに加えて最小限の時間方向並列化計算を適用するという形にする．

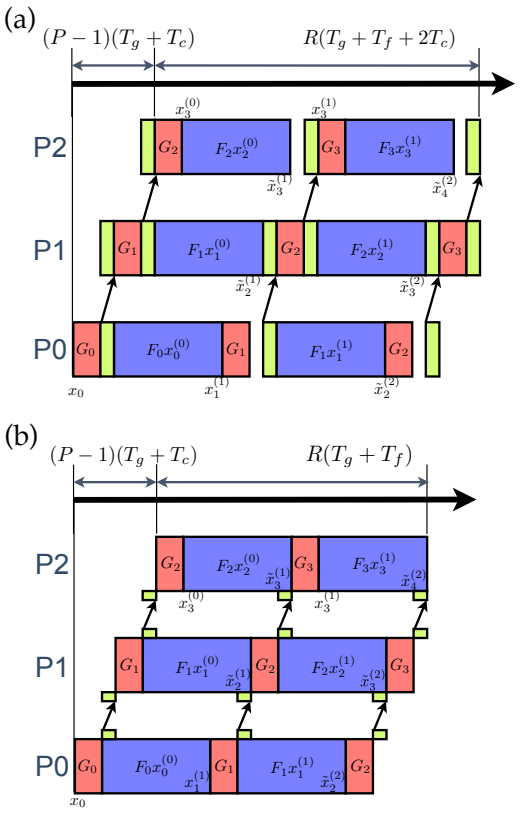

図 2 Parareal-in-Time 法の実装: (a) 同期通信を利用した場合，(b) 非同期通信による効率化を図った場合．

MPI の同期通信の `MPI_Send` と `MPI_Recv` を利用した通常の実装では，図 2(a) のような手順での実行となる．この図では横方向で実時間を表し，時間とともに実行される計算の内容を示している．縦方向の P0 や P1 などは計算機リソースグループを表し，それぞれが内部で並列計算を行っている．ここでは，時間方向に 3 並列で実行する場合を表している．リソース間の矢印は，`MPI_Send/MPI_Recv` を利用した同期通信を表し，$x_k^{(r)}$ を隣のリソースにパイプライン的に転送している．一方，非同期通信を使って実装する例を図 2(b) に示す．計算の順序などは全く同じであるが，非同期通信を導入することにより，各リソースでの実行内容に隙間がなくなり，効率的に実装できることがわかる．

時間方向の並列リソース数を P とし，収束までの繰り返し計算が R 回である場合，$K = P + R - 1$ ステップの時間発展計算が実施されることに注意すると，同期通信で実装した場合のスピードアップ比は，

$$S_{\mathrm{b}}(P, R) = \frac{(P + R - 1)T_f}{(P - 1)(T_g + T_c) + R(T_g + T_f + 2T_c)} \tag{5}$$

と表すことが出来る．ここで，T_g は近似 $\mathcal{G}_k$ の計算時間，T_f は $\mathcal{F}_k$ の計算時間，T_c は 1 回の通信にかかる時間を表す．同様に，非同期通信を利用した場合のスピードアップ

比は，

$$S_{\rm nb}(P,R) = \frac{(P+R-1)T_f}{(P-1)(T_g+T_c)+R(T_g+T_f)} \qquad (6)$$

となる．

Parareal-in-Time 法による並列計算のスピードアップ比に関して，いくつかコメントしておく．式 (5) と (6) からは，並列数 P が十分に大きい場合のスピードアップ比は $S(P,R) \approx T_f/(T_g+T_c)$ であり，通信コストを無視すると，高速な近似計算を導入したことによるメリットが，そのまま反映するように見えるが，P が大きいとき，収束性の問題から繰り返し回数 R を大きく取る必要があるため，現実にはこの理想値にはならない．また，実際には，通信のコストが比較的大きく，これによる性能の悪化も考慮する必要がある．同期通信と非同期通信の利用による実装を比較するとき，スピードアップ比に関しては，通信時間 T_c に関する部分にわずかな違いが出るだけである．非同期通信を実現するための wait などによる付加的なコストは，ここでは無視した形の式になっているため，現実にどのような形で効率化が図れるかについては，実測によってこれら二つの実装を比較する必要がある．

並列化による効率 (リソースを P 使ってスピードアップが $S=P$ のとき，効率 100%とする) という面からみると，通常の空間並列化と異なり，Parareal-in-Time 計算では，決して 100%にはならない．これは，R 回の繰り返し計算が必要となるため，元々の計算量に比べて R 倍計算量が大きくなっていることによる．十分なスピードアップ比を得るためには，P をある程度以上大きく取る必要があり，この場合には，非常に効率の悪い計算となることに注意しておく．

3.2 Template Class ‘Parareal <T>’

並列計算機による時間領域計算のベンチマークコードとしてだけでなく，計算科学の問題に簡単に Parareal-in-Time 法を導入することができるように，筆者らは，C++ の Template Class として，`Parareal <T>`を開発中である．これは，`PararealSequence <T>` と合わせて利用することで，MPI による時空間並列計算の実行を支援するものである．ただし，時間発展させる状態ベクトル x_k を表す `class T` は，抽象クラス `class PararealElement` を実装する形で，利用者が定義する必要がある．詳細は，後日公開する予定である．

3.3 時空間並列計算での通信パターン

ここで実装した時空間並列計算について，その通信部分の詳細を明記しておく．図 2 の各計算リソース (P0∼P2) 内では，ユニタリ変換による複素ベクトルの時間発展計算をリソース内の集団通信を利用した並列計算で実行している．本原稿執筆時点で時空間並列実装が完了しているのは，Bcast-Gather タイプだけであるので，ここでは，この方法で実装した場合の通信パターンについて検討する．

まず，Parareal-in-Time 法を導入するための近似計算 $\mathcal{G}_k$ は，ここでは $\mathcal{G}_k \equiv I$ (恒等変換) と定義した．$\mathcal{F}_k$ は元々のユニタリ変換である．

$$U = \exp[-i\delta t H] = I + (\exp[-i\delta t H] - I) \qquad (7)$$

と書けるため，δt が小さい場合には，この近似が有効に働く．この方法による Parareal-in-Time 法の収束性については，既に文献 [7] で確認してある．この場合，$\mathcal{F}_k$ は $O(n^2)$ 回の浮動小数点計算，$\mathcal{G}_k$ は，$O(n)$ 回の配列のコピーなどである．

図 3 に，時空間並列実装での通信パターンについて示す．隣のグループから n 要素の複素数ベクトル $\mathcal{G}_k x_k^{(r+1)}$ を受け取る所から計算が始まる．ここで実装した $\mathcal{F}$ の計算は，Bcast-Gather タイプの行列ベクトル積であるため，グループ間のベクトルの受け渡しはグループ内の代表プロセスのみが実施する．同じグループで既に実施した計算結果 j$\mathcal{F}_k x_k^{(r)} - \mathcal{G}_k x_k^{(r)}$ に加えることで，

$$x_{k+1}^{(r+1)} = G_k x_k^{(r+1)} + \left[\mathcal{F}_k x_k^{(r)} - \mathcal{G}_k x_k^{(r)}\right] \qquad (8)$$

を求めることが，最初の仕事である．次に，ここで得られた $x_{k+1}^{(r)}$ に対して $\mathcal{G}_{k+1}$ 演算を行い (この実装では恒等変換なので何も演算しない)，次のグループに転送する．さらに，$x_{k+1}^{(r)}$ に対して $\mathcal{F}_{k+1}$ の演算を実施する．このために，`MPI_Bcast` でグループ内の全プロセスに送付し，各プロセスでは，$O(n \times m)$ 程度の計算を実施した後，結果として得られる m 個の複素数を，再び代表プロセスに `MPI_Gather` して完了する．ここで，m は，n をグループ内のプロセス数で割った値である．最後に，$\mathcal{F}_{k+1} x_{k+1}^{(r+1)}$ から $\mathcal{G}_{k+1} x_{k+1}^{(r+1)}$ の引き算を行い一連の仕事が終了する．

以上のように，ある計算グループの周辺での通信パターンは，他のグループからの `MPI_Recv`，グループ内での `MPI_Bcast`，および，`MPI_Gather`，他のグループへの `MPI_Send` の実施となる．ここでは，グループ内の代表プロセスが，グループ間通信を担うという実装にしたが，ベクトル $x_k^{(r)}$ をグループ内のプロセスが分散して保持する形の実装も可能で，この時は，グループ内の計算は，Allgather タイプで実装する形が効率が良いため，他のグループからの `MPI_Recv`，グループ内での `MPI_Allgather`，他のグループへの `MPI_Send` という通信パターンとなる．後者の実装では，$O(n)$ の回数のベクトルの和や差もグループ内で並列化が可能であることに加えて，グループ間通信も分散化されるため，前者に比べて効率が良くなることが予想される．ただ，後者の実装がまだ完了していないため，次章での実測では，前者の実装によるものだけを扱うこととする．

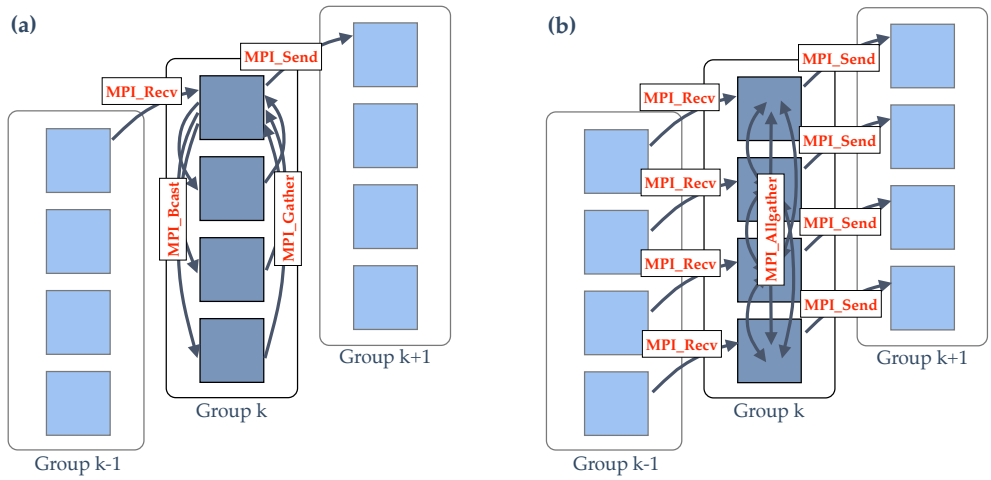

図 3　グループ内／グループ間通信のパターン．色の付いた四角はプロセスを表している．

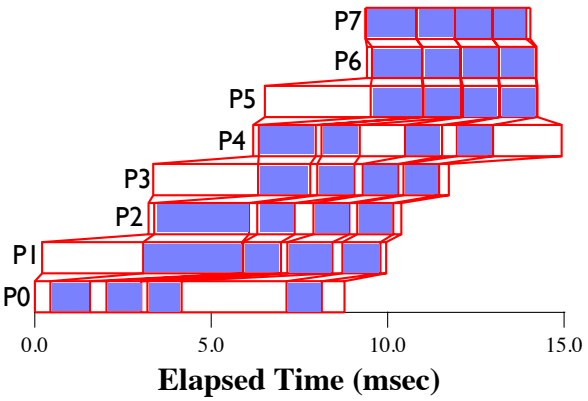

図 4　実測値に基づく各計算グループ (P0∼P7) の実行状況．横軸は経過時間 (msed) を表す．塗りつぶしてある場所が，演算子 $\mathcal{F}$ による計算を実施している部分である．

4. 計測結果

実際に並列計算機で Parareal-in-Time 法の計算を実行してタイミングデータなどを取得し，効率的な実装に向けた解析を行う．計算機はこれまでに利用したものと同じ SandyBridge (ノード内 16 コア) のクラスタ (InfiniBand FDR) であるが，ここでは時間方向まで含めて最大 512 プロセスまでの，フラット MPI による並列計算を行った．

図 4 に，複数グループでの実行の一例を示す．ここでは，P0 から P7 までの 8 グループに 2 ノードずつを割り当て，それぞれのグループ内では，32 プロセスでユニタリ変換の計算を実施した．Parareal-in-Time 法の繰り返し計算は 4 回としており，この場合には，$8+4-1=11$ ステップの計算を，8 グループによる時間領域並列化計算で実施したことになる．

横軸は開始からの経過時間を表し，最初に `MPI_Barrier` を実行した時点を $t=0$ として，ミリ秒で表示してある．P0 から順にパイプライン的な実行になっていることがわかるが，想定したような効率の良い実行にはなっておらず，所々に原因不明の待ちによる空白が生じている．時間計測の精度はマイクロ秒程度であるため，これらの空白は時間計測誤差によるものではないと考えられる．しかし，同じネットワークスイッチの下に他の計算ジョブが流れている状態での計測であるため，外的要因による通信遅れの可能性は否定できない．今後，もう少し静かな環境で実測を行い，詳しい解析を行う必要がある．

5. まとめ

Parareal-in-Time 法という時間領域分割計算を時空間並列計算として実用的な実装を行うために，空間方向，時間方向の通信パターンを検討し，スピードアップ比などについて考察した．いくつかの方法でテスト実装した検証用のプログラムを使って並列計算機で実測した結果を提示した．

本稿で検討した Parareal-in-Time 法による時間領域分割計算は，全体での同期を行わない計算であることに特徴がある．実測結果を模式的に表示した図 4 でわかるように，この計算は各リソースでの計算結果をパイプライン的に隣のリソースに転送していく形で実行される．我々がこの計算手法を検討の対象の一つにしているのは，パイプライン的な実行パターンを持っているためである．一般に，パイプライン的な実行パターンは，開始時期の異なる複数の関連する計算を，並列計算機で実行しようとする時に利用される．計算の効率や性能を追求するばかり，完全に独立な大規模計算だけが超並列計算機で実行される問題であるということにならないように，少しでも可能性がある場合には，様々な計算のパターンについて検討しておきたいと考えている．パイプライン的な計算パターンについても，独立な計算を多数含む問題に比べると効率の悪化はさけられないが，通信効率化などの実装技術を整備して，最大限の性能を引き出したい．

理論的には，非同期通信を利用することで，隙間のない

効率的な実行が可能になると考えられるが，実測結果を見てもわかるように，一部のリソースで予定外の遅れが生ずると，全体的な計算の遅れに直結する可能性が高い．これは，パイプライン的な実行パターンでは，ある程度さけて通れない部分である．今回の計算では，ノード内のコア数と同じだけのプロセスを立ち上げているため，CPU リソースに全く余裕がない状態で動いていることも要因の一つであると考えられる．非同期通信の効果的な利用によって，少なくとも通信部分で発生する擾乱には左右されないような構成も考えられる．個人的には，外部からの擾乱を完全に排除して効率的な実行を目指すよりも，予定外のタイミングの遅れを後ろに伝搬させないような実装技術について，より魅力を感じる．

謝辞　数値計算は，九州大学情報基盤研究開発センターの CX400 を利用して実施した．

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. **332**, 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] E. Aubanel, “Scheduling of tasks in the parareal algorithm,” Parallel Computing **37**, 172–182 (2011).
[4] W. R. Elwasif, S. S. Foley, D. E. Bernholdt, L. A. Berry, D. Samaddar, D. E. Newman, and R. Sanchez: “A Dependency-Driven Formulation of Parareal: Parallel-in-Time Solution of PDEs as a Many-Task Application,” in Proc. MTAGS2011, p02 (2011).
[5] R. Speck, D. Ruprecht, R. Krause, M. Emmett, M. Minion, M. Winkel, P. Gibbon: “A massively space-time parallel N-body solver,” in Proc. SC12, No. 91, 1–11 (2012).
[6] 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).
[7] T. Takami and A. Nishida, “Parareal Acceleration of Matrix Multiplication,” Adv. Par. Comp. **22**, 437–444 (2012).
[8] Y. Maday, G. Turinichi, “Parallel in Time Algorithms for Quantum Control: Parareal Time Discretization Scheme,” IJQC **93**, 223–228 (2003).
[9] T. Takami and H. Fujisaki, “Analytic approach for controlling quantum states in complex systems,” Phys. Rev. E**75**, 036219 (2007).
[10] 高見利也，藤崎弘士: 複雑量子系の最適制御，日本医科大学紀要 Vol. 41, pp. 27–56 (2012).