Vol.2016-HPC-157 No.19
2016/12/22

Parareal法と領域分割法による拡散問題での時空間並列性能評価

今村 成吾[1,a)]　飯塚 幹夫[2]　小野 謙二[2,3]　横川 三津夫[1]

概要：CPU のメニーコア化などによってスーパーコンピュータの並列度が増加し，空間方向の領域分割だけでは並列化要素が不十分になりつつある．そのため，時間並列計算法が近年注目されている．本研究では，その有用な解法の一つである Parareal 法を放物型偏微分方程式の一例として拡散問題に適用し，並列加速率の挙動の調査と考察を行った．京コンピュータでの評価では，時間方向の並列数が 100 の場合，Speedup モデルから推定される最大 Speedup 性能 14 倍に対して 8.5 倍の実行性能を得られた．また，領域分割法による空間並列と Parareal 法の時間並列の組み合わせによる大規模並列性能評価を行った．64 空間並列を 64 ノードで並列化した場合，逐次計算に対して 13.5 倍の速度向上が得られ，さらに Parareal 法による 100 並列を組み合わせることにより，64×100 ノードでの並列で 101 倍の速度向上が得られた．

キーワード：時間並列，放物型偏微分方程式，ハイパフォーマンスコンピューティング

1. はじめに

GPGPU やメニーコアプロセッサによる並列度数の増加によって近年のスーパーコンピュータ（スパコン）の演算性能はさらに向上している．また次世代の exa-scale 級の計算機でも更なる演算性能の向上のために並列度数は増えるであろう．

数値流体力学で扱われる並列化手法として領域分割法が主流である．しかし，最新のスパコンでは大規模並列時に，通信コストの増加により並列性能は飽和傾向にある．そのため，次世代のスパコンでの高速化のための並列化要素として空間並列だけでは不十分である．そこで，空間並列に加えて，時間並列が注目されている．時間並列は 2001 年に Lions らによって提案された Parareal 法 [1] をベースとして，PFASST や MGRIT[2], [3] などの Parareal よりも収束性の良いアルゴリズムも提案されている．しかし，Parareal 法は時間並列計算法の中でも非常にシンプルであるため，既存のシミュレーションコードの時間並列化が比較的容易であり実装のコストを考慮しても十分期待ができる．

Parareal 法の性能評価は様々な問題を用いて行われている [4], [5]．Parareal 法は双曲型偏微分方程式は対しては収束性に課題があるが [6]，放物型の偏微分方程式に対しては良い収束性をもつ．しかし，大規模並列や時空間並列での性能評価は不十分である．

本研究では，拡散方程式を対象に Parareal 法の概要と並列加速率の性能モデルについて説明し，数値実験を行い Parareal 法の時間並列の性能評価と，空間領域分割との組み合わせによる時空並列の性能評価を述べる．

2. Parareal法

Parareal 法は時間発展方程式の並列計算法である．Parareal 法による時間並列の計算概要を Fig. 1 に示す．計算は (a) 全計算時間を部分時間領域に分割，(b) 領域毎に仮の初期値を設定，(c) 各領域で並列に領域内の逐次積分，(d) 反復毎に生じる領域 $n-1$ の終端時刻の値と領域 n の始端時刻の値の食い違い量を計算，(e) 食い違い量の伝搬計算を粗視化積分演算により行い修正計算，(f) 修正量を用いて各領域の開始/終端時刻の値を修正の順で進める．以上の計算を反復計算により近似解を得る．

Parareal 法の導出と計算方法について説明をする [7]．次の方程式を用いて説明を行う：

$$u_t = f(u),\ t \in [0, T]. \tag{1}$$

時間領域 $[0,T]$ を N_t 分割した時間小領域 $[T_{n-1}, T_n]$ $(1 \le n \le N_t, T_0 = 0 < T_1 < \cdots < T_{N_t} = T)$ を Time slice と呼ぶ．Time slice 上の端点の変位 $u_n = u(T_n)$ とし，従来の時間積分方法で u_n を求める場合，

[1] 神戸大学大学院 システム情報学研究科
[2] 理研 計算科学研究機構
[3] 九州大学 情報基盤研究開発センター
[a)] s-imamura@stu.kobe-u.ac.jp

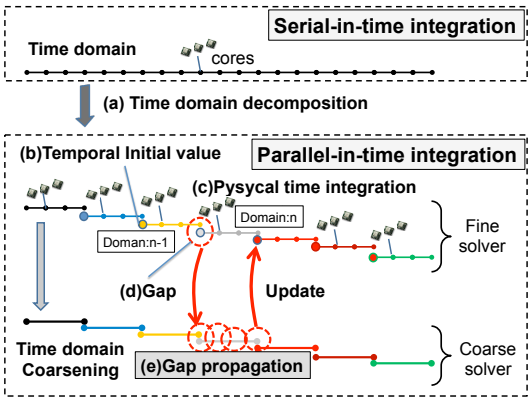


図 1 Parareal 法の概要
Fig. 1 Overview of Parareal method.

$$u_{n+1} = \mathcal{F}(T_{n+1}, T_n, u_n) \tag{2}$$

と表現される．ここで Fine solver $\mathcal{F}(\cdot)$ はユーザーが望ましい精度をもつ時間積分をする作用素とする．$\boldsymbol{U} \equiv (u_0, \ldots, u_{N_t})^\top$ として，(2) を変形する：

$$f(\boldsymbol{U}) = \begin{pmatrix} u_0 - u_0 \\ u_1 - \mathcal{F}(T_1, T_0, u_0) \\ \vdots \\ u_{N_t} - \mathcal{F}(T_{N_t}, T_{N_t-1}, u_{N_t-1}) \end{pmatrix} = 0. \tag{3}$$

$f(\boldsymbol{U}) = 0$ となる $\boldsymbol{U}$ を求める Newton 法を用いると，k 反復目に得られる近似解 $\boldsymbol{U}^k \equiv (\boldsymbol{u}_0^k, \ldots, \boldsymbol{u}_{N_t}^k)^\top$ とする．ただし，初期値より $u_0^k = u_0$ とする．

$$\boldsymbol{U}^{k+1} = \boldsymbol{U}^k - [f'(\boldsymbol{U}^k)]^{-1} f(\boldsymbol{U}^k) \tag{4}$$

(4) の両辺に左から $f'(\boldsymbol{U}^k)$ をかけて整理すると，

$$\begin{cases} u_0^k & = u_0 \\ U_{n+1}^{k+1} & = \mathcal{F}(T_n, T_{n-1}, u_n^k) \\ & \quad -\mathcal{F}'(T_n, T_{n-1}, u_n^k)[\mathcal{F}(T_n, T_{n-1}, u_n^{k+1}) \\ & \quad -\mathcal{F}(T_n, T_{n-1}, u_n^k)] \end{cases} \tag{5}$$

(5) の 2 式の右辺第 2 項を Fine solver よりも計算コストの低い Coarse solver $\mathcal{G}(\cdot)$ を用いて近似する：

$$\begin{aligned} &\mathcal{F}'(T_n, T_{n-1}, u_n^k)[\mathcal{F}(T_n, T_{n-1}, u_n^{k+1}) \\ &-\mathcal{F}(T_n, T_{n-1}, u_n^k)] \\ &\approx \mathcal{G}(T_n, T_{n-1}, u_n^{k+1}) - \mathcal{G}(T_n, T_{n-1}, u_n^k). \end{aligned} \tag{6}$$

これにより，よく知られる Parareal アルゴリズムが得られる [6]：

$$\begin{aligned} u_{n+1}^{k+1} &= \mathcal{G}(T_{n+1}, T_n, u_n^{k+1}) \\ &\quad +\mathcal{F}(T_{n+1}, T_n, u_n^k) - \mathcal{G}(T_{n+1}, T_n, u_n^k). \end{aligned} \tag{7}$$

$\mathcal{G}(T_n, T_{n-1}, u_n^{k+1})$ の計算は逐次計算になるが Fine solver よりも低コストに抑えることができ，また，$F(T_n, T_{n-1}, u_n^k)$ の計算は時間方向の依存性がなく並列計算が可能となっている．

実装概要を Algorithm 1 に計算の流れを Fig. 2 に示す．wrkCn, wrkCb, wrkF はワーク領域である．Parareal にはデータの通信時間 t_{com} と通信の待ち時間 t_{wait} がある．この 2 つの時間を合わせた通信時間を t'_{com} として数値実験で計測を行う：

$$t'_{com} = t_{com} + t_{wait}. \tag{8}$$

Algorithm 1 Parareal method

Receive u_{n-1}^0 from previous time slice $[T_{n-2}, T_{n-1}]$
$u_n^0 = \mathcal{G}(T_n, T_{n-1}, u_{n-1}^0)$
Send u_n^0 to next time slice $[T_n, T_{n+1}]$
Copy u_n^0 to wrkCn
while $k < N$ **do**
  wrkF $= \mathcal{F}(T_n, T_{n-1}, u_{n-1}^k)$
  Copy wrkCn to wrkCb
  Recieve u_{n-1}^{k+1} from previous time slice $[T_{n-2}, T_{n-1}]$
  wrkCn $= \mathcal{G}(T_n, T_{n-1}, u_{n-1}^{k+1})$
  Summation vectors u_n^{k+1} = wrkCn + wrkF − wrkCb
  Send u_n^{k+1} to next time slice $[T_n, T_{n+1}]$
end while

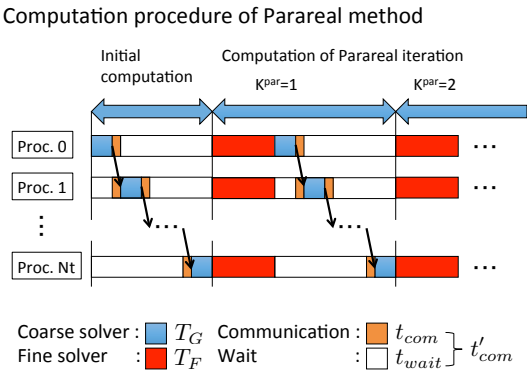


図 2 Parareal 法の計算の流れ
Fig. 2 Computation procedure of Parareal method.

2.1 残差について

Parareal の収束判定では以下の残差式を用いる：

$$res_n^k = u_n^k - u_n^{k-1}. \tag{9}$$

残差式 (9) はこれまでに示されていないが文献 [6] の誤差の式を参照して，以下のように示される．

$$res_n^k = \mathcal{G} \cdot res_{n-1}^k + (\mathcal{F} - \mathcal{G}) res_{n-1}^{k-1} \tag{10}$$

これから，残差は次のように上限を抑えることができる：

$$\max_{1 \le n \le N_t} |res_n^k| \le \frac{(CT)^{(k-1)}}{(k-1)!} \Delta T^{p(k-1)} \max_{1 \le n \le N_t} |res_n^1|. \tag{11}$$

ここで，C は定数である．この残差式を使い，

$$\epsilon > res(K^{par}), \tag{12}$$

から，許容誤差 ϵ を下回る残差となる反復数 K^{par} の推定が可能となる．

2.2 並列加速率について

Parareal 法のアルゴリズムから理論的な並列加速率について説明する [8], [9]．N_t を Time slice の数（並列数），T_G と T_F を Time slice$[T_{n-1}, T_n]$ での Coarse と Fine solver の実行時間とする．並列加速率の基準となる逐次実行時間を Fine solver で逐次的に $[0, T]$ を時間積分するときの実行時間とし，$N_t T_F$ と表される．Parareal 法では，まず初期値の計算に Coarse solver での $[0, T]$ の時間積分を行う．Coarse solver の実行時間は通信時間を含めて $N_t(T_G + t_{com})$ となる．また，1 反復に Coarse Solver と Fine solver の実行時間が $N_t(T_G + t_{com}) + T_F$ となる．K^{par} を Parareal 法の反復数とすると Speedup は $\alpha(N_t, K^{par}, T_G, T_F, t_{com})$ は，

$$\alpha(N_t, K^{par}, T_G, T_F, t_{com}) = \frac{N_t T_F}{N_t(T_G + t_{com}) + K^{par}(N_t(T_G + t_{com}) + T_F)}, \quad (13)$$

となる．これを変形して

$$\alpha(N_t, K^{par}, T_G, T_F, t_{com}) = \frac{N_t}{N_t(T_G/T_F + t_{com}/T_F) + K^{par}(N_t T_G/T_F + 1)}. \quad (14)$$

となる．t_{com} は T_F より十分小さいとき無視することができる：

$$\alpha(N_t, K^{par}, T_G, T_F) = \frac{N_t}{N_t T_G/T_F + K^{par}(N_t T_G/T_F + 1)}. \quad (15)$$

Speedup モデル (15) から，この上限は N_t/K^{par}[8] とわかるが未知数が多くこのモデルの挙動の検討は難しい．そのため T_G, T_F を近似して未知数を減らして検討を行う．Coarse solver と Fine solver で同じ時間積分法で扱った場合，それぞれの実行時間の比を時間ステップ幅 $\delta T, \delta t (< \delta T)$ を使った比 $R_{fc} = T_F/T_G \approx \delta T/\delta t$ で近似できるすることで，Speedup モデルを近似する：

$$\alpha(N_t, K^{par}, R_{fc}) = \frac{N_t}{N_t/R_{fc} + K^{par}(1 + N_t/R_{fc})}. \quad (16)$$

時間粗視化率 R_{fc} はユーザーが指定できるため既知であるが，K^{par} は問題によって収束性が異なるため予測が難しく未知数である．そのため数値実験で得られる反復数 K^{par} を用いた式 (16) により並列加速率を検討する．

3. 並列加速率の挙動の推定

残差による Parareal 法の計算の打ち切り式 (12) から反復数 K^{par} を求め，式 (16) を使うことにより，Parareal 法の並列加速率の挙動を推定できる．正確な式 (12) が得られていないため，理論的な推定は困難である．しかし，以下のように定性的に並列加速率の挙動は推定できる．R_{fc} を大きくすると，逐次計算部分が減少し並列加速率が増加するが，その一方で修正計算である Coarse solver の計算精度が低下して，並列加速率は低下する．この 2 つの効果から，並列加速率は，R_{fc} に対してピークを持つような挙動を持つと推定できる．その概要を Fig. 3 に示す．

また，Parareal 法の並列加速率の式には，空間並列の影響は入っていないため，空間並列に関係なく，時間並列の効果を得ることが出来ると推測される．

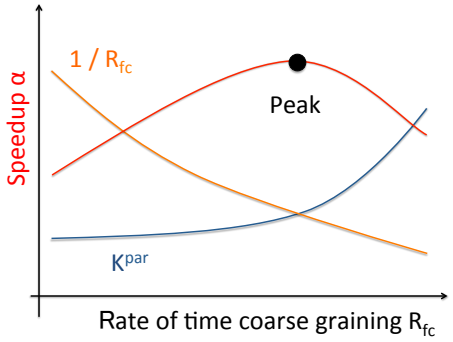


図 3 Parareal 法の Peak 性能
Fig. 3 Peak performance of Parareal method.

4. 数値実験

数値実験を行い，前節で説明した並列加速率のモデル (16) と数値実験結果に基づき，拡散問題での Parareal 法の性能評価を行う．

4.1 拡散問題

今回扱う拡散問題，初期値と境界条件は，

$$u_t(\boldsymbol{x}, t) = \frac{1}{3}\Delta u(\boldsymbol{x}, t), \ \boldsymbol{x} \in \Omega = [0, 1]^3, \ t \in [0, T], \quad (17)$$

$$u(\boldsymbol{x}, 0) = \sin(\pi x)\sin(\pi y)\sin(\pi x), \quad u(\boldsymbol{x}, t) = 0, \quad \boldsymbol{x} \in \partial\Omega, \quad (18)$$

としている [10]．離散化は差分法，時間積分法として Coarse, Fine solver ともに Crank-Nicolson 法を用いており，また Crank-Nicolson 法から得られる連立一次方程式の線形 solver として Red-Black SOR 前処理付 Bi-CGstab 法を利用している．

4.2 実行環境

京コンピュータを使用し，その詳細は Table 1 に示す．また，計測には理研の Performance Monitor library (PMlib)[11] を使用している．コンパイルオプションは-Kfast,parallel,ocl,preex,array_private,auto -Kopenmp,simd=2,uxsimd としている．使用言語は C++ と Fortran で，プログラムの制御部分と計測を C++で，線形 solver などの数値計算の部分は Fortran で実装している．

空間並列を MPI と OpenMP のハイブリッド並列で実装し，Parareal 法は MPI で実装している．空間並列は k,j,i の空間三重ループについて，最外側の k ループを MPI で領域分割し，次に k,j の二重ループを OpenMP の Collapse 指示句を用いて一重化しスレッド並列化している．

表 1 京コンピュータの詳細
Table 1 Specification of K computer.

Archtecture	SPARC64^{TM}VIIIfx
CPU performance	128GFLOPS (16GFLOPS × 8 core)
L2 cache	6MB
Memory band width	64GB/s
Frequency	2GHz
ノード間帯域	5GB/s

4.3 パラメータ

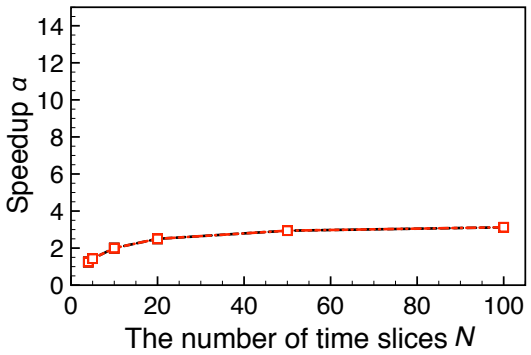
数値実験で使用したパラメータリストを Table 2 に示す．なお，Parareal の収束判定は各 Time slice n の端点の残差の L2 ノルム $||\boldsymbol{u}_n^{k+1} - \boldsymbol{u}_n^k||_2$ のうちの最大値を用いている．

表 2 パラメータリスト
Table 2 Parameter list.

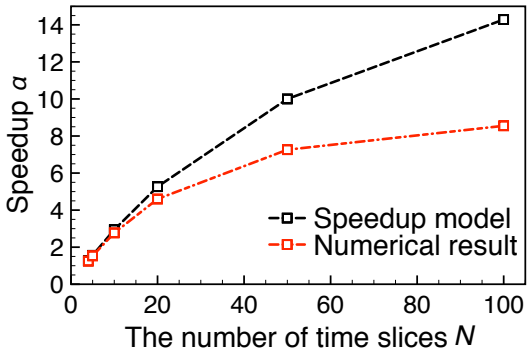
Coarse, Fine solver の時間積分法	Crank-Nicolson 法
格子数	128^3
時間領域	$[0:T] = [0:1], [0:0.2]$
Time Slice の数 N_t	4,5,10,20,50,100
領域分割の数 N_s	1, 2, 4, 8, 16, 32, 64
$R_{fc}(\delta T)$	500(0.5), 100(0.01), 50(0.05), 10(0.001)
δt	0.0001
Parareal 法の収束判定	$\max_n ||\boldsymbol{u}_n^{k+1} - \boldsymbol{u}_n^k||_2 < 10^{-6}$
線形 solver	Red-Black SOR 前処理付 Bi-CGstab 法
線形 solver の収束判定	$||\boldsymbol{r}||_2/||\boldsymbol{b}||_2 < 10^{-12}$ （$\boldsymbol{r}$ は残差ベクトル）

4.4 Speedup モデルの性能挙動

ここでは領域分割なし ($N_s = 1$) の計算条件で，Speedup モデル (15) の検証を行う．$R_{fc} = 10, 100$ のときの Speedup の実験結果を Fig. 4 に，また Parareal 法の反復数を Table 3 に示す．Speedup モデルの K^{par} は未知数であるので，実験結果の値を用いて算出している．また実行結果のグラフは時間方向に逐次計算した実行時間をもとに示している．まず，$R_{fc} = 10$ のときはモデル通りの性能が得られることがわかる．次に $R_{fc} = 100$ の場合，時間方向に 100 並列することによって，最大 8.5 倍の Speedup が得られることがわかる．しかし，Speedup モデルから得られる理想的な値からは離れている．各 Time step での Coarse solver の反復数を見るを Tab. 4 に示す．$K^{par} = 0$ は Parareal の初期値の計算で扱う Coarse solver の計算を示す．$R_{fc} = 10$ では，線形 solver の反復数が一定になっている．しかし，$R_{fc} = 100$ では Time domain の後半部で反復数が増えており，また Parareal の反復数の方向にも線形 solver の反復数が増えている．これによりロードバランスが均一でなくなったため，モデルとの乖離が見られた．



(a) $R_{fc} = 10(\delta T = 0.001)$.



(b) $R_{fc} = 100(\delta T = 0.01)$.

図 4 $T = 1$ での Time slice の数の変化による Speedup
Fig. 4 Speedup according to the number of time slices at $T = 1$.

表 3 Parareal の反復数 K^{par}
Table 3 The number of iterations of Parareal K^{par}.

T	R_{fc}	The number of time slice N					
		4	5	10	20	50	100
1	500	4	4	5	5	-	-
	100	3	3	3	3	3	3
	50	2	3	3	3	3	3
	10	2	2	2	2	2	2
0.2	100	3	3	3	3	3	3

Time domain を $[0, 0.2]$ に変えた時の Speedup を Fig. 5 に示す．Time domain$[0, 1]$ に比べて，$[0, 0.2]$ はモデルに非常に近くなっていることがわかる．このことから，理想的には Speedup モデル通りの加速が期待できる．

表 4 Coase solver での線形 solver の反復数
Table 4 The number of iterations of coarse solver.

R_{fc}	K^{par}	T				
		0.2	0.4	0.6	0.8	1.0
100	0	2	2	3	4	4
	1	2	3	4	5	6
	2	3	5	5	7	8
	3	4	5	7	9	10
10	0	2	2	2	2	2
	1	2	2	2	2	2
	2	2	2	2	2	2

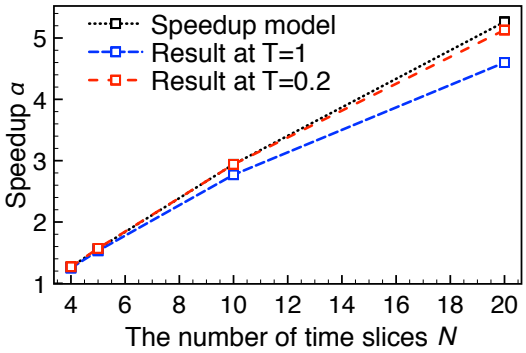


図 5 Time domain による Speedup の変化
Fig. 5 Speedup according to time domain.

Time Slice の数を 10 で固定し，R_{fc} を変化させたときの Speedup を Fig. 6 に示す．これから Parareal 法は R_{fc} によって Speedup の Peak が存在することがわかる．そのため，最適なパラメータを設定する方法が必要となり，K^{par} を予測する式 (12) が必要となる．

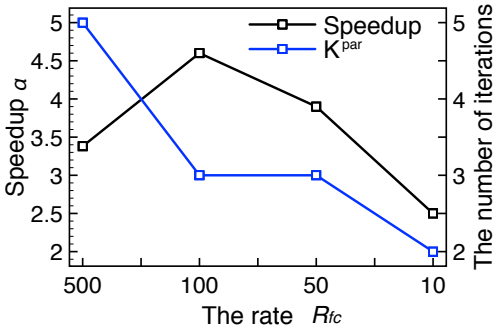
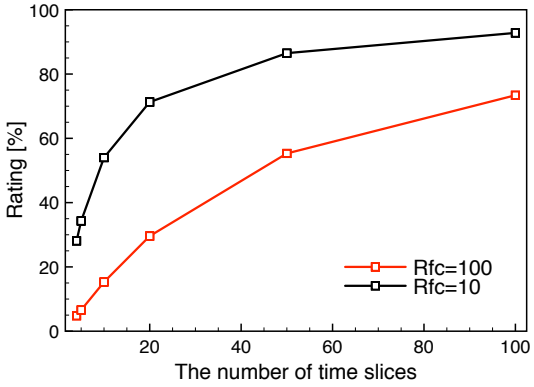
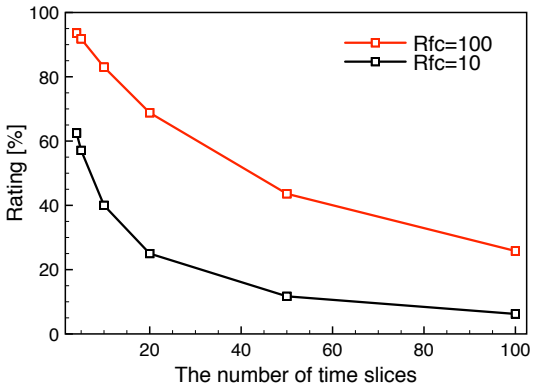


図 6 R_{fc} による Speedup の変化
Fig. 6 Speedup according to R_{fc}.

次に，実行時間を占める通信時間 t'_{com} と Fine solver の実行時間 T_F の比率を Fig. 7 に示し，また，Tab. 5,6 に実行時間と内訳を示している．Time slice の数が増えるほど，Coarse solver の計算待ちによる通信時間 t'_{com} が増えている．また，R_{fc} が大きいほど，通信時間 t'_{com} の比率が減少し，かつ Fine solver の時間の比率が大きくなっており並列性能が上がっていることがわかる．そのため Coarse solver には計算負荷が小さくかつ R_{fc} が増加しても反復数 K^{par} を少なくできるような高精度計算方法が必要とされる．



(a) 通信時間 t'_{com}



(b) Fine solver の実行時間 T_F

図 7 実行時間のうち通信時間と Fine solver の時間が占める比率
Fig. 7 Rating of communication time and the time of Fine solver in execution time.

表 5 $R_{fc} = 100$ のときの実行時間と詳細 [sec]
Table 5 Execution time and its details [sec] with $R_{fc} = 100$.

N_t	実行時間	Fine solver	Coarse solver	t'_{com}
1	2.31×10^3	–	–	–
4	1.85×10^3	1.73×10^3	2.92×10^1	8.87×10^1
5	1.51×10^3	1.38×10^3	2.48×10^1	9.93×10^1
10	8.34×10^2	6.92×10^2	1.41×10^1	1.27×10^2
20	5.03×10^2	3.46×10^2	7.80×10^0	1.49×10^2
50	3.19×10^2	1.39×10^2	3.57×10^0	1.76×10^2
100	2.70×10^2	6.98×10^1	1.98×10^0	1.99×10^2

4.5 領域分割法との組み合わせ

$R_{fc} = 100$，Time slice の数 $N_t = 10, 100$ のときの領域分割との組み合わせによる Speedup を Fig. 8 に示す．Serial-in-time が示すグラフは領域分割のみによる並列によって得られた Speedup であり，64 ノードでの並列で 13.5 倍の Speedup と既に飽和傾向あることがわかる．これは格

表 6　$R_{fc} = 10$ のときの実行時間と詳細 [sec]
Table 6　Execution time and its details [sec] with $R_{fc} = 10$.

N_t	実行時間	Fine solver	Coarse solver	t'_{com}
4	1.84×10^3	1.15×10^3	1.73×10^2	5.19×10^2
5	1.61×10^3	9.22×10^2	1.38×10^2	5.54×10^2
10	1.15×10^3	4.61×10^2	6.92×10^1	6.23×10^2
20	9.23×10^2	2.31×10^2	3.46×10^1	6.58×10^2
50	7.86×10^2	9.23×10^1	1.39×10^1	6.80×10^2
100	7.41×10^2	4.61×10^1	6.93×10^0	6.88×10^2

子数が 128^3 と少ないため，64 分割時には計算量に対して通信の割合が増加し並列性能が低下するためである．しかし，そこから Parareal 法による時間方向に 100 並列することによって，つまり，6400($= 64 \times 100$) ノードでの並列で 101 倍の Speedup が得られることがわかる．Parareal 法は領域分割法ほど計算資源に応じた並列性能がでないが，領域分割が飽和したあとでも並列性能を得られることがわかる．そのため，exa-scale 級のコア数が多いスパコンを見据えると有用な並列化手法である．

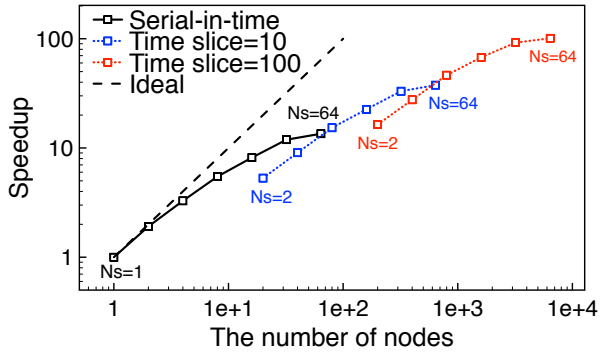
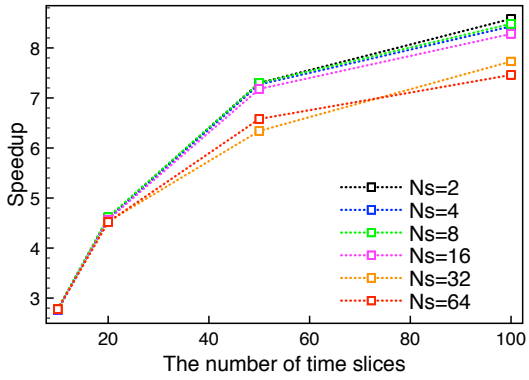


図 8　$R_{fc} = 100$ のときの領域分割との組み合わせによる Speedup.
Fig. 8　Speedup by combination to domain decomposition with $R_{fc} = 100$.

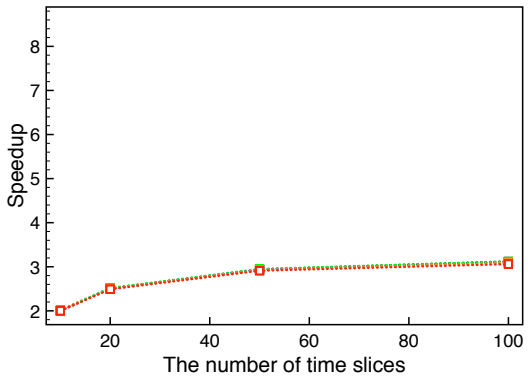
領域分割による並列後, 時間並列による効果について調べる．領域分割数を固定した場合の時間並列による Speedup を Fig. 9 に示す．$R_{fc} = 10$ のときは時間並列の効果は領域分割数に依存していないが，$R_{fc} = 100$ のときは領域分割数が増えるとともに少し下降傾向にあることがわかる．領域分割による影響が Coarse solver の反復数に影響したためと思われる．

5. まとめ

拡散方程式を対象に大規模並列と時空間並列での Parareal 法の性能評価を行った．Parareal 法で時間方向に 100 並列した場合，性能モデルでは 14 倍の Speedup が予想されたが，数値実験では最大 8.5 倍であった．この性能低下の原因として，Coarse solver での反復数が Fine solver での反



(a) $R_{fc} = 100(\delta T = 0.01)$.



(b) $R_{fc} = 10(\delta T = 0.001)$.

図 9　領域分割数 N_s を固定したときの Parareal による Speedup
Fig. 9　Speedup by Parareal with fixed the number of domain decomposition N_s.

復数に比べて増加したためである．そのため時間粗視化率 R_{fc} に反比例した Coarse solver の実行時間の短縮を得ることができなかったためである．粗視化率が小さい $R_{fc} = 10$ の場合は性能モデル通りの挙動を確認できた．時空間並列では，性能モデルから予想されるように，時間並列は空間並列に独立に効果があることを示した．その結果，空間並列性能が飽和後に時間並列によるさらに並列性能が向上することが確認できた．

今後の課題として，時間粗視化率 R_{fc} が大きい場合でも Coarse sovler で扱う線形 solver の反復数の増加がないような対応をする必要がある．その一例として，より高精度な 4 次精度後退差分法を Coarse solver へ適用し実験する予定である．今回は最も単純で使い易いオリジナルの Parareal 法を対象に性能評価を実施した．一方で，Parareal 法の逐次計算部分を短縮するために Pipelined Parareal 法や，SDC 法が提案されており [8]，より一層の性能向上が期待できる状況である．ただし，Pipeleind Parareal 法では並列の制御が複雑であったり，SDC 法ではユーザーアプリの時間積分法を更新することが必要となるためユーザーの時間並列計算コードの開発負担は増加する．今後，これらの手法を用いた性能評価も実施する予定である．

謝辞 本研究の成果の一部は，文部科学省科学技術試験研究委託事業「近未来型ものづくりを先導する革新的設計・製造プロセスの開発」と JSPS26390130 科研費の助成を受けたものです.

参考文献

[1] Lions, J.-L., Maday, Y., and Turinici, G.: A “ parareal ” in time discretization of PDE ’s, *C. R. Acad. Sci. Paros Ser. I Math.*, 332 (2001): 661–668.
[2] Speck, R., et al.: Integrating an N-Body Problem with SDC and PFASST, *Domain Decomposition Methods in Science and Engineering XXI*, 98 (2014): 637–645.
[3] Falgout, R.D., et al.: Parallel time integration with multigrid, *SIAM Journal on Scientific Computing*, 36.6 (2014): C635-C661.
[4] Kreienbuehl, Andreas, et al.: Numerical simulation of skin transport using Parareal, *Computing and visualization in science* 17.2 (2015): 99–108.
[5] Ruprecht, D., Speck, R. and Krause, R.: Parareal for diffusion problems with space-and time-dependent coefficients, *Domain Decomposition Methods in Science and Engineering XXII*, (2016): 371–378.
[6] Gander, M.J., Vandewalle, S.: Analysis of the parareal time-parallel time-integration method, *SIAM J. Sci. Comput.*, 29 (2007): 556–578.
[7] 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 HPC-131-6 (2011): 1–8.
[8] Minion, M.: A hybrid parareal spectral deferred corrections method, *Communications in Applied Mathematics and Computational Science*, 5.2 (2010): 265–301.
[9] 飯塚 幹夫, 小野謙二, 加藤千幸: Parareal 法による拡散方程式の時間並列計算, 第 29 回数値流体力学シンポジウム (2015).
[10] Minion, M. L., Speck, R., Bolten, M., Emmett, M., and Rupercht, D.: Interweaving PFASST and Paralel Multigrid, *SIAM J. Sci. Comput.* 37.5 (2015): S244–S263.
[11] `http://avr-aics-riken.github.io/PMlib/`