Vol.2018-HPC-163 No.27
2018/3/2

空間並列度および時間並列度の割り当て方に対する考察

依田 凌[†1]　藤井 昭宏[†1]　田中 輝雄[†1]　中島 研吾[†2]

概要：従来では時間積分において，空間方向のみの並列化が注目されてきた．多数の演算器により性能向上を図る近年の環境では，空間方向並列化のみを適用した場合，その莫大な並列数を効率的に使用することは難しい．そこで時間方向の並列化を検討することで，十分な自由度を確保しさらなる並列性の抽出が見込める．しかし時間方向並列化の効果や，空間並列度と時間並列度の割り当て方による性能への影響は分かっていない．本研究では，東京大学の FX10 スーパーコンピュータシステム上で時間発展する熱拡散問題を対象に，空間方向の並列化として Additive Correction Multigrid(ACM) を前処理にした共役勾配法 (ACM-CG)，時間方向の並列化として Multigrid Reductoin in Time(MGRIT) を適用し，時間方向並列化の有効性の確認し，時空間並列度の割り当てによる性能への影響を考察した．この実験により時間方向並列化は，空間方向並列化と比較して同等または同等以上の並列性の抽出ができていることがわかった．

キーワード：parallel-in-time integration, multigrid

1. はじめに

近年，使用できるコア数は急上昇し，超並列環境を以前に比べて身近に使えるようになった．そのために現在では，超並列環境で動くアプリケーションがより求められている．本研究においてはアプリケーションの一例として時間発展シミュレーションを対象に扱う．従来では時間発展シミュレーションに対して空間方向のみの並列化が用いられてきた．しかし超並列環境において空間方向のみの並列化を適用すると，その莫大な並列数を効率的に扱うことが難しい．空間方向並列化に加えて時間方向並列化を検討することによってさらなる並列性の抽出が見込める．

時空間並列を適用した例は少なく，空間方向並列化と時間方向並列化を合わせた際に時間方向並列化はどれほどの効果があるのか，また空間方向並列度と時間方向並列度の割り当てによる性能への影響は分かっていない．

本研究では時間方向並列化の有効性を確認するために，また時空間並列における空間並列度と時間並列度の割り当て方による性能への影響を調べるために，時間発展する線形熱拡散問題を扱い，東京大学の FX10 スーパーコンピュータシステム (Oakleaf-FX) 上で実験を行った．空間方向並列化として B. R. Hutchinson，G. D. Raithby により提案された Additive Correction Multigrid(ACM) [4][5][6][7] を前処理とした共役勾配法 (ACM-CG) 適用し，時間方向並列化として R. D. Falgout らにより提案された Multigrid Reduction in Time(MGRIT) [12][13][14] を適用した．1 コア 1 プロセスの Flat MPI で行ない，最大 1 万を超える並列度で計測した．

[†1] 工学院大学
Kogakuin University
[†2] 東京大学
The University of Tokyo

2. 時空間ソルバ

2.1 Additive Correction Multigrid

連立一次方程式 $A\boldsymbol{\phi} = \boldsymbol{b}$ の解法として，与えられた行列からマルチレベルを構成する AMG 法がある [1][3]．AMG 法では，粗いレベルの係数行列を作成するためのガラーキン近似 RAP(R:制限行列，P:補間行列) が計算時間の多くを占める．この改善案として，粗いレベルの行列を元の行列要素の線形和で作る Additive Correction Multigrid [4][6][7] がある．ガラーキン近似に比べて近似の精度が下がるために解法部の反復回数が増える可能性がある [5]．しかしながら，生成部で大きく時間削減ができるために全体の実行時間を減少できる場合がある．

$A\boldsymbol{\phi} = \boldsymbol{b}$ から 2 レベルでの ACM を導出する．はじめは幾何学的に考え，次に代数的に拡張する．幾何学的グリッド格子として，図 1 のような未知数の隣接関係を考える．細かいレベルの未知数は正方形の各セルの中心に置かれている．太線で囲まれた 4 つのセルをまとめて 1 つのブロックとし，1 ブロックが 1 つの粗いレベルの未知数に対応している．細かいレベルでは未知数を $\boldsymbol{\phi}$，index を (i, j) とし，粗いレベルでは未知数を $\boldsymbol{\Phi}$，index を (I, J)

とする．5 点差分を用いて係数行列を作成すれば，隣接関係は *West*,*East*,*South*,*North* で表現できる．$\phi_{i,j}$ に対応する行列 A の各行は，中心点を $a^p_{i,j}$ と置けば，その他は $a^w_{i,j}$,$a^e_{i,j}$,$a^s_{i,j}$,$a^n_{i,j}$ を用いて式 1 のように書ける．

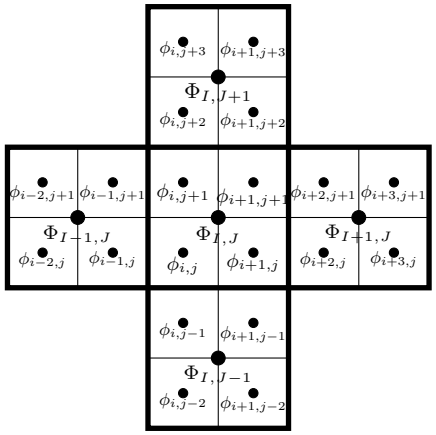

図 1 状態の隣接関係

$$A_{i,j} = \begin{pmatrix} -a^s_{i,j} & \cdots & -a^w_{i,j} & a^p_{i,j} & -a^e_{i,j} & \cdots & -a^n_{i,j} \end{pmatrix} \quad (1)$$

式 1 を用いて $A\boldsymbol{\phi} = \boldsymbol{b}$ は式 2，また近似解を $\hat{\boldsymbol{\phi}}$ とすると残差ベクトル $\boldsymbol{r} = \boldsymbol{b} - A\hat{\boldsymbol{\phi}}$ は式 3 のように書ける．

$$a^p_{i,j}\phi_{i,j} = a^w_{i,j}\phi_{i-1,j} + a^e_{i,j}\phi_{i+1,j} + a^s_{i,j}\phi_{i+1,j} + a^n_{i,j}\phi_{i,j+1} + b_{i,j} \quad (2)$$

$$r_{i,j} = b_{i,j} - a^p_{i,j}\hat{\phi}_{i,j} + a^w_{i,j}\hat{\phi}_{i-1,j} + a^e_{i,j}\hat{\phi}_{i+1,j} + a^s_{i,j}\hat{\phi}_{i,j-1} + a^n_{i,j}\hat{\phi}_{i,j+1} \quad (3)$$

次に粗いレベルの方程式 $\boldsymbol{A_c\Phi} = \boldsymbol{B}$ を，式 4 のように構成する．右下付きの文字が大文字の (I, J) の場合は粗いレベルの係数行列 $\boldsymbol{A_c}$ の要素を表している．

$$A^p_{I,J}\Phi_{I,J} = A^w_{I,J}\Phi_{I-1,J} + A^e_{I,J}\Phi_{I+1,J} + A^s_{I,J}\Phi_{I,J-1} + A^n_{I,J}\Phi_{I,J+1} + B_{I,J} \quad (4)$$

式 4 の各要素は図 2，3 の色と対応した場所を参照して構成される．図 2 と 4 セルと，図 3 の中心の 4 セルは図 1 の中心の太線で囲まれた 4 つのセルに対応しており，図の中心に粗いレベルの未知数 $\Phi_{I,J}$ がある．

- $A^p_{I,J} = a^p_{i,j} - a^e_{i,j} - a^n_{i,j} + a^p_{i+1,j} - a^w_{i+1,j} - a^n_{i+1,j}$
  $+ a^p_{i,j+1} - a^e_{i,j+1} - a^s_{i,j+1} + a^p_{i+1,j+1} - a^w_{i+1,j+1} - a^s_{i+1,j+1}$

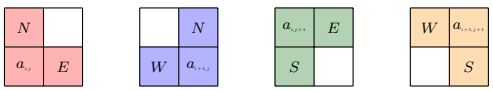

図 2 ACM で行列構築する際の参照箇所 1

- $A^w_{I,J} = a^w_{i,j} + a^w_{i,j+1}$, $A^e_{I,J} = a^e_{i+1,j} + a^e_{i+1,j+1}$
- $A^s_{I,J} = a^s_{i,j} + a^s_{i+1,j}$, $A^n_{I,J} = a^n_{i,j+1} + a^n_{i+1,j+1}$
- $B_{I,J} = r_{i,j} + r_{i+1,j} + r_{i,j+1} + r_{i+1,j+1}$

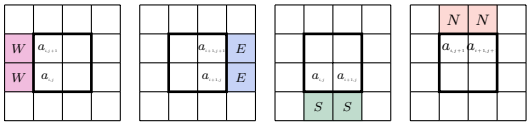

図 3 ACM で行列構築する際の参照箇所 2

残差方程式を構成した後に，その解である $\boldsymbol{\Phi}$ が求まったとする．細かいレベルへの補正は，そのブロックに属する細かいレベルの未知数全てにそのまま足しこむ．

$$\begin{aligned} \phi_{i,j} &\leftarrow \phi_{i,j} + \Phi_{I,J} & \phi_{i+1,j} &\leftarrow \phi_{i+1,j} + \Phi_{I,J} \\ \phi_{i,j+1} &\leftarrow \phi_{i,j+1} + \Phi_{I,J} & \phi_{i+1,j+1} &\leftarrow \phi_{i+1,j+1} + \Phi_{I,J} \end{aligned} \quad (5)$$

以上で格子グリッド上での 2 レベルの ACM を導出できた．代数的に拡張するには，まずは未知数の隣接関係を，与えられる係数行列から得られるグラフの隣接関係に置き換える．次にアグリゲート戦略によりアグリゲートを作成し，1 つのアグリゲートを 1 つの粗いレベルの未知数に対応させる．最後に粗いレベルの隣接関係をアグリゲートの隣接関係に置き換えれば良い．

2.2 ACM 前処理付き共役勾配法

共役勾配法の反復回数は対象とする係数行列の固有値分布に依存する．適切な前処理行列 C を導入して，固有値分布を改善する前処理付き共役勾配法がある [8][9]．前処理付き共役勾配法のアルゴリズムを図 4 に示す．

$\boldsymbol{x} = \boldsymbol{x_0}$; // 初期値
$\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{x}$; $\boldsymbol{p} = \left(CC^T\right)^{-1}\boldsymbol{r}$; $\rho = (\boldsymbol{r}, \boldsymbol{p})$;
while $||\boldsymbol{r}|| \geq \epsilon||\boldsymbol{b}||$
  $\boldsymbol{q} = A\boldsymbol{p}$; $\gamma = (\boldsymbol{p}, \boldsymbol{q})$; $\alpha = \frac{\rho}{\gamma}$;
  $\boldsymbol{x} = \boldsymbol{x} + \alpha\boldsymbol{p}$;
  $\boldsymbol{r} = \boldsymbol{r} - \alpha\boldsymbol{q}$;
  $\boldsymbol{q} = \left(CC^T\right)^{-1}\boldsymbol{r}$; $\mu = (\boldsymbol{q}, \boldsymbol{r})$; $\beta = \frac{\mu}{\rho}$; $\rho = \mu$;
  $\boldsymbol{p} = \boldsymbol{q} + \beta\boldsymbol{p}$;

図 4 PCG Algorithm

上記の赤色で囲まれた部分が前処理部分である．この部分の $(CC^T)^{-1}\boldsymbol{r}$ の代わりに，$A\boldsymbol{p} = \boldsymbol{r}$ に Multigrid 法を適用した解 $\boldsymbol{q}$ を使用する MGCG 法がある [10][11]．ACM も Multigrid 法の一種のため同様の前処理法が適用できる．以降，ACM を前処理に適用した共役勾配法を ACM-CG と記述する．

2.3 Multigrid Reduction in Time

時間方向に Multigrid 法を適用した Multigrid Reduction in Time(MGRIT)[12][13][14] がある．時間進展のシミュレーションにおいて陰解法を用いる多くの場合，タイムステップ 0 での初期状態と，タイムステップ i と $i+1$ の関係式で記述される．以下の MGRIT のアルゴリズムの説明のために定義される Φ, A は，2.1 節で定義した記号と違う意味を表していることに注意する．Φ はタイムステップ i と $i+1$ での未知数同士の関係を表す関数を表す．T を最

大タイムステップ数として，A は T タイムステップ分の未知数間の関係を表す行列と定義する．状態ベクトル $\boldsymbol{u}$ と，外部流入ベクトル $\boldsymbol{g}$ を用いて，各タイムステップ間の関係を式 6，時間進展を表す $A\boldsymbol{u}=\boldsymbol{g}$ を式 7 と書ける．

$$\begin{cases} g^0 &= u^0 \\ \Phi u^{i+1} &= u^i + g^{i+1} \quad (i=0,1,\cdots,T-1) \end{cases} \tag{6}$$

$$A\boldsymbol{u} = \begin{bmatrix} I & & & \\ -I & \Phi & & \\ & \ddots & \ddots & \\ & & -I & \Phi \end{bmatrix} \begin{bmatrix} u^0 \\ u^1 \\ \vdots \\ u^T \end{bmatrix} = \begin{bmatrix} g^0 \\ g^1 \\ \vdots \\ g^T \end{bmatrix} = \boldsymbol{g} \tag{7}$$

右上の数字はタイムステップ数を表す．式 7 は T タイムステップ分の関係式をまとめて記述した式と見なせる．式 6 より i ステップと $i+1$ ステップの間に依存性があることがわかる．ここで並列性を抽出するために，M ステップごとに依存関係を切り，それにより生まれる誤差を粗いレベルを用いて補正する方法を考える．粗いレベルでは，依存関係を切った M 個おきの未知数のみを持つために，M を粗格子率と呼ぶ．時間進展を表す式 7 と粗格子率 M の関係を図 5 に示す．

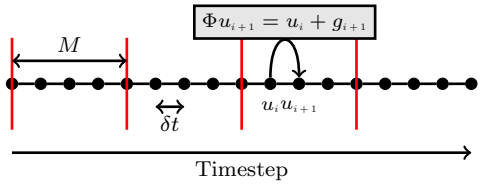

図 5　時間進展と粗格子率の関係

次に M 個おきのタイムステップの状態を C 点 (●)，それ以外の状態を F 点 (•) とおき，2 種類のスムーザを定義する．図 6，7 では $M=4$ としている．

- F-relaxation: C 点を基準に F 点を逐次的に更新

図 6　F-relaxation

- C-relaxation: 直前の F 点を基準に C 点を更新

図 7　C-relaxation

この 2 種類のスムーザは並列に実行可能である．MGRIT の V-cycle におけるプレスムージングとして F-relaxation，C-relaxation，F-relaxation の順に適用した FCF-relaxation を，ポストスムージングとして F-relaxation を用いる．

次に制限行列と補間行列を定義する．制限行列 R を M 個おきにタイムステップを抜き出す操作として図 8，補間行列 P を元の変数に代入する操作として図 9 に示す．

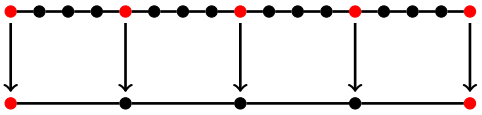

図 8　制限行列 R による操作

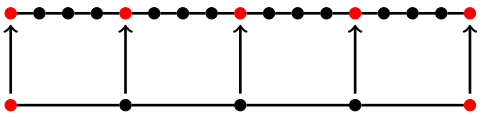

図 9　補間行列 P による操作

この制限行列と補間行列を用いて残差方程式を Full Approximation Scheme(FAS) で作る．右下付き Δ は粗いレベルの変数を表す．本研究の実験のみでは，この FAS を用いずに残差方程式を $A_\Delta \boldsymbol{e_\Delta} = \boldsymbol{r_\Delta}$ としても問題ないが，今後の研究において非線形問題に拡張できるようにこのスキーマを採用した．$\boldsymbol{u_\Delta} \leftarrow R\boldsymbol{u}$，$\boldsymbol{r_\Delta} \leftarrow R(\boldsymbol{g}-A\boldsymbol{u})$ として残差方程式を $A_\Delta \boldsymbol{v_\Delta} = A_\Delta \boldsymbol{u_\Delta} + \boldsymbol{r_\Delta}$ を式 8 のように構築する．

$$\overset{A_\Delta}{\begin{bmatrix} I & & & \\ -I & \Phi_\Delta & & \\ & \ddots & \ddots & \\ & & -I & \Phi_\Delta \end{bmatrix}} \overset{\boldsymbol{v_\Delta}}{\begin{bmatrix} v_\Delta^0 \\ v_\Delta^1 \\ \vdots \\ v_\Delta^{T_\Delta} \end{bmatrix}} = \overset{A_\Delta}{\begin{bmatrix} I & & & \\ -I & \Phi_\Delta & & \\ & \ddots & \ddots & \\ & & -I & \Phi_\Delta \end{bmatrix}} \overset{\boldsymbol{u_\Delta}}{\begin{bmatrix} u_\Delta^0 \\ u_\Delta^1 \\ \vdots \\ u_\Delta^{T_\Delta} \end{bmatrix}} + \overset{\boldsymbol{r_\Delta}}{\begin{bmatrix} r_\Delta^0 \\ r_\Delta^1 \\ \vdots \\ r_\Delta^{T_\Delta} \end{bmatrix}} \tag{8}$$

粗いレベルのタイムステップ数は $T_\Delta = \frac{T}{M}$ で表される．Φ_Δ は $\Delta t = M\delta t$ を用いて再度離散化して作られる．残差方程式の解 $\boldsymbol{v_\Delta}$ が求めた後は，補正項 $\boldsymbol{e_\Delta}$ を $\boldsymbol{v_\Delta} - \boldsymbol{u_\Delta}$ と計算し，$\boldsymbol{u} \leftarrow \boldsymbol{u} + P\boldsymbol{e_\Delta}$ として細かいレベルに補正する．以上で 2 レベルでの MGRIT を導出できた．これを再帰的に用いることでマルチレベルを構成できる．

3. 時空間並列化の実装

3.1 空間ソルバの並列化実装

問題行列はブロック行で分割され，分散されて保持する方法を取ると仮定した．与えられた問題行列からグラフの対称な隣接関係を作るために，未知数間の隣接関係を決める指標には式 9 を用いている．ここで $a_{i,j}$ は問題行列 A の i 行 j 列目の要素を表し，θ はしきい値を表す．

$$|a_{ij}| \geq \theta \sqrt{a_{ii} a_{jj}} \tag{9}$$

アグリゲート戦略として，1 つの頂点を選び，その頂点と隣接する頂点をまとめて 1 つのアグリゲートを作成し，次に頂点が選ばれる指標となる優先度をその頂点から距離が 3 の場所に +1 する方法を取った．

並列アグリゲート戦略での確保する領域については，分割した領域を独立に保持する Domain Decomposition ACM を使用した．この手法において逐次に解いた場合と並列に解いた場合では，作られるアグリゲート集合は異なる．例えば図 10，11 においては，逐次で解いた場合では同じアグリゲートに属していた未知数が，並列に解いた場合では独立領域を超えないために切り離される．

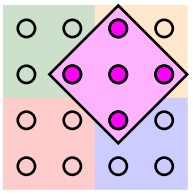

図 10　逐次にアグリゲートを作成した場合

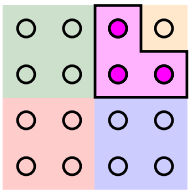

図 11　並列にアグリゲートを作成した場合

領域を独立に保持する方法では，並列度を上げるほど各レベルの未知数は増える．これを実際に確認するために，N^2=256^2=65536 の領域での熱拡散問題において，全ての要素がグラフの辺として貼られる θ を選択して，5 レベルを構築した場合の各レベルごとの未知数の割合を表 1，2 にまとめた．

表 1　並列数 $P = 1$

level	N	rate[%]
0	65536	100
1	21761	33.2
2	5850	8.93
3	1888	2.88
4	518	0.79

表 2　並列数 $P = 256$

level	N	rate[%]
0	65536	100
1	21921	33.4
2	7784	11.9
3	2732	4.18
4	871	1.33

最下位レベルに注目すれば，未知数の数が $P = 1$ では 518 に対して，$P = 256$ においては 871 まで増えていることがわかる．そして最下位部分では，分散して持っている行列データを 1 プロセスに集めて直接解くために，未知数が多くなるほど最下位部のコストが上昇することに気をつけなければならない．

また共役勾配法の前処理では，前処理行列も対称である必要がある．そして Multigrid 法を前処理として用いる場合，V-cycle が対称であればその制約を満たせることが知られている [11]．V-cycle のプレスムージングには 1 行目から N 行目に依存関係が生まれる gauss-seidel 法，ポストスムージングには N 行目から 1 行目に依存関係が生まれる gauss-seidel 法を採用し，V-cycle 全体として対称となるようにした．

3.2　時間ソルバの並列化実装

MGRIT では，各スムーザ，残差計算，制限と補正が全て並列にできる．しかし，最下位レベルの直接解法部分では時間方向に逐次で解かなければならない．たとえば 64 タイムステップの問題を 4 並列で分割し，粗格子率 $M = 4$ で 3 レベルを構築したとする．図 12 では各プロセスが担当する場所を色分けした．最下位レベルでは面しているプロセスが共有して保持していることを表す．

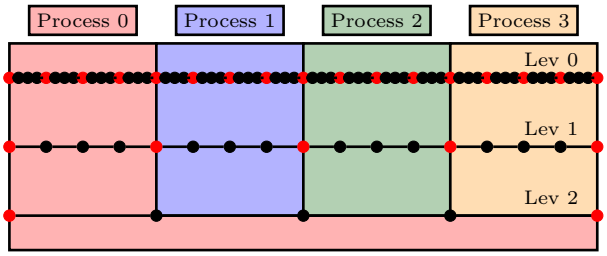

図 12　MGRIT の各プロセスが担当する箇所

本研究の並列化手法は，図 12 のように，最下位レベルは全てのタイムステップをプロセス 0 が担当させた．そして各プロセスが 1 タイムステップしか持たなくなるまで粗くしたレベルを最下位レベルとした．この方法では最下位レベルのタイムステップ数は並列度と等しくなるり，並列度が上がるにつれて逐次解法部分のコストが増えることに注意する．

3.3　Flat MPI 上のコミュニケータ分割

数万プロセスに及ぶ高並列度環境では，並列化によるオーバーヘッドが大きくなる．そのためにグローバルコミュニケータを分割し，ローカルコミュニケータで通信を行なう方法が考えられる．横軸に空間方向，縦軸に時間方向を取れば，状態空間を 2 次元で記述できる．これを用いれば従来用いられてきた空間方向のみの並列化は図 13，時空間並列化は図 14 のように表せる．

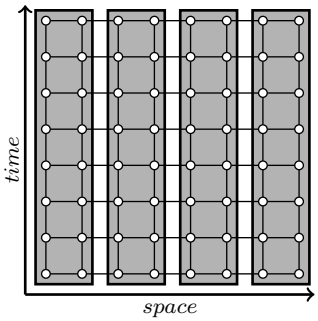

図 13　空間並列化

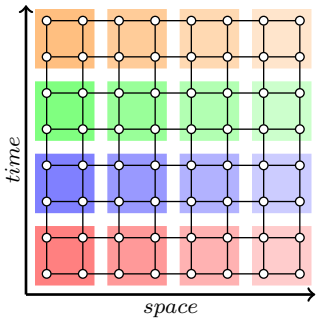

図 14　時空間並列化

空間方向並列度を P_s，時間方向並列度を P_t とすれば，全体並列度 P は $P = P_s \times P_t$ を満たす．図 14 は $P_s = 4$，$P_t = 4$ の場合であり，その色は，同じ空間コミュニケータに属することを示す．空間ソルバにおいての必要な通信 (例えば内積計算など) はグローバルなコミュニケータを介さずに，ローカルのコミュニケータ内で行われる．この実装により高並列度においても性能向上が期待される．

4. 数値実験

本研究では，時間発展の 2D 線形熱拡散方程式を東京大学の FX10 スーパーコンピュータシステム (Oakleaf-FX) 上で行った．最大 1024 ノードを Flat MPI で利用し，最大 16384 プロセスで評価を行った．格子点数 $N = 128^2 = 16384$，$\frac{k\Delta t}{(\Delta x)^2} = \frac{1.0 \times 0.1}{4.0^2} = 0.00625$，最大タイムステップ数 $T = 16384$ とした．初期温度は $\left(\frac{N}{2}, \frac{N}{2}\right)$ を中心とするガウス分布 $\mathcal{N}(0, 1)$ を 100 倍に拡大した分布に設定した．境界条件は温度を 0 で固定するディリクレ境界条件で行った．

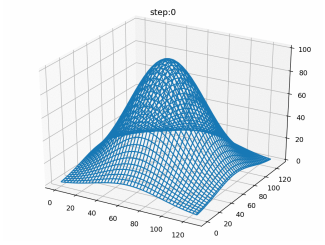

図 15　初期値，$T = 0$ の場合

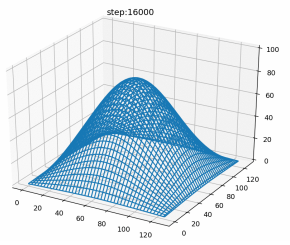

図 16　$T = 16000$ の場合

図 15 は初期値である $T = 0$ 時点の温度分布で，図 16 は $T = 16000$ 時点の温度分布である．熱のわき出しは設定していないため，最終的には全て境界条件と同じ 0 になるはずであるが，終盤のタイムステップでは定常状態には達していないことが確認できた．

この問題に対して以下の 3 つの実験を行った．

- 空間方向のみ並列化による性能評価
- 時間方向並列化の有効性の確認のために，空間並列度を固定し，時間方向に弱スケーリングで性能評価
- 時空間並列度のバランスについて性質を見るために，全体の並列度を固定し，空間並列度と時間並列度を変えて評価

4.1 空間方向のみ並列化による性能評価

まず始めに上記の問題に空間方向並列化を適用した．ACM-CG 法では 5 レベルを構築し，$P_s = 1, 256$ で実行した時の各レベルの未知数の数は表 3，4 となった．

表 3　並列数 $P_s = 1$

level	N	rate[%]
0	16384	100
1	5421	33.1
2	1700	10.4
3	486	2.97
4	142	0.87

表 4　並列数 $P_s = 256$

level	N	rate[%]
0	16384	100
1	5600	34.3
2	1625	9.92
3	573	3.50
4	256	1.56

表 4 によれば $P_s = 256$ は $P_s = 1$ に比べて，未知数の比率は約 2 倍近くになっているとはいえ，全体のわずか約 1.56%であることが確認できた．また ACM-CG 法の反復回数は，全てのタイムステップにおいて 3 回であり，タイムステップに応じて反復回数が変化しないことがわかった．

次に空間方向並列化による性能向上を見るために，空間並列度 P_s を $1, 4, 16, 64, 256$ に変化させ，図 17，表 5 を得た．横軸は対数スケールのタイムステップ数を表し，縦軸は対数スケールの実行時間を表し，縦軸の値が小さいほど性能が良い．

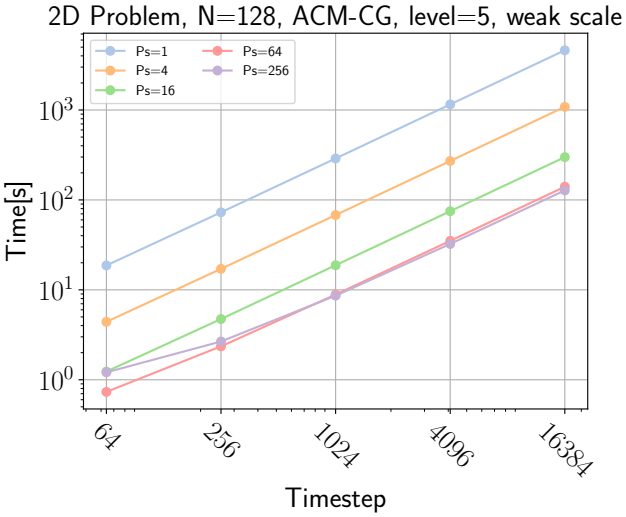

図 17　時間発展問題の空間並列

表 5　時間発展問題の空間並列

Type\Timestep	64	256	1024	4096	16384
$P_s = 1$	18.679	72.770	289.12	1154.6	4615.9
$P_s = 4$	4.407	17.115	67.908	271.08	1084.1
$P_s = 16$	1.233	4.734	18.770	74.909	299.21
$P_s = 64$	0.733	2.353	8.855	35.060	140.24
$P_s = 256$	1.209	2.663	8.611	32.375	127.61

図 17 より，全てのタイムステップにおいて反復回数が一定のために，タイムステップが増加するに従って実行時間が増加しているのがわかる．また $N = 128^2$ という小さな領域では，$P_s = 1 \sim 64$ においては着実に実行時間が減少しているが，$P_s = 256$ においては領域を過剰に分割してしまい，大きくは実行時間が減少していない．つまりより高並列な環境を想定して，これ以上の並列性を空間方向に割り振ったとしても理想的な性能向上を図れない，と推測できる．

4.2 空間並列度固定で時間方向の弱スケーリングテスト

空間並列度を固定して，MGRIT を適用し時間方向並列化を行った．1 プロセスが 64 タイムステップを持つ弱スケーリングで行ない，図 17 に重ねてプロットしたものが図 18 であり，数値データは表 6 である．図 18 の横軸はタイムステップと時間並列度を表している．実線上のラベルは it が MGRIT の反復回数を表す．グラフ中の赤のバツ印は，空間並列化と時空間並列化の交点を表している．

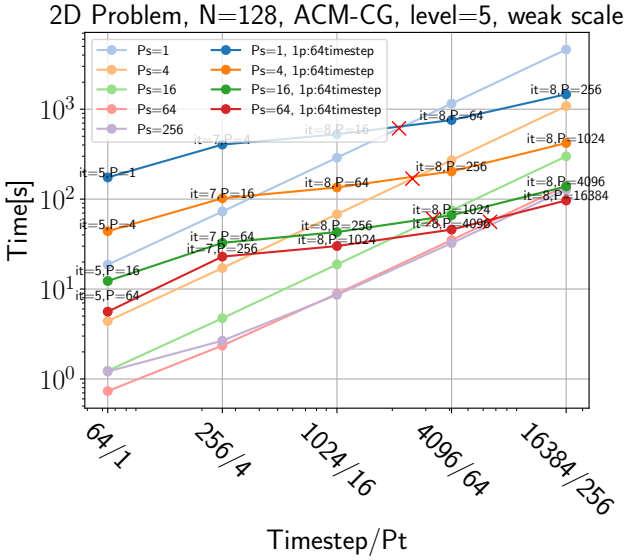

図 18　時間発展問題の時空間並列

表 6　時間発展問題の時空間並列

Type	Timestep	64	256	1024	4096	16384
	P_t	1	4	16	64	256
	iteration	5	7	8	8	8
$P_s = 1$	Time[s]	174.33	404.12	526.07	756.69	1464.9
	P	1	4	16	64	256
$P_s = 4$	Time[s]	43.998	102.151	134.91	203.48	420.42
	P	4	16	64	256	1024
$P_s = 16$	Time[s]	12.281	32.648	43.007	66.158	138.43
	P	16	64	256	1024	4096
$P_t = 64$	Time[s]	5.603	22.951	30.057	45.806	96.607
	P	64	256	1024	4096	16384

その次に時間方向並列化の効果を確認するために，図 18 から空間並列度毎に抜き出し図 19~26 を得た．

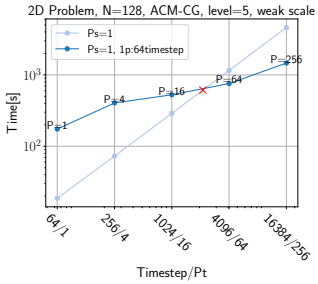

図 19　$P_s = 1$ に固定，対数軸

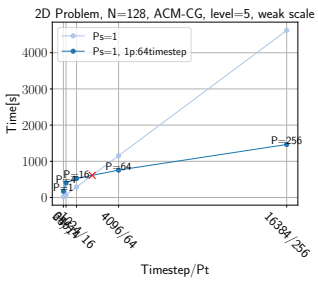

図 20　$P_s = 1$ に固定

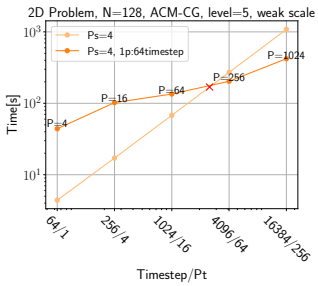

図 21　$P_s = 4$ に固定，対数軸

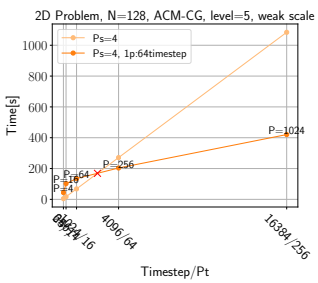

図 22　$P_s = 4$ に固定

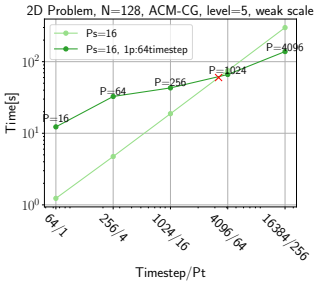

図 23　$P_s = 16$ に固定，対数軸

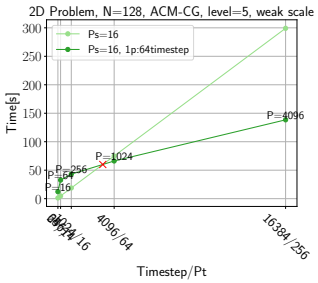

図 24　$P_s = 16$ に固定

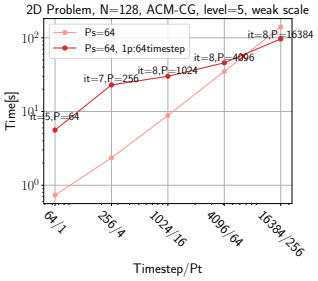

図 25　$P_s = 64$ に固定，対数軸

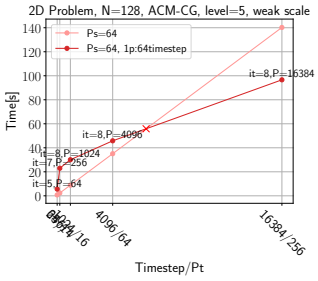

図 26　$P_s = 64$ に固定

図 19，20 は空間並列度を $P_s = 1$ で固定し，時間方向が逐次と，時間方向の弱スケーリングでの並列化を比較した図である．空間並列 $P_s = 1$ 固定で時間方向に弱スケーリングで並列化したものを比較している図である．この図の $T = 16384$ において，$P_t = 256$ の時間方向並列化の適用により，空間並列のみの実行時間に比べて約 $\frac{1}{3}$ となることが確認できた．

図 25，26 は空間並列度を $P_s = 64$ で固定し，時間方向が逐次と，時間方向の弱スケーリングでの並列化を比較した図である．この図の $T = 16384$ において，$P_t = 256$ の時間方向並列化の適用により，空間並列のみの実行時間に比べて約 $\frac{2}{3}$ となることが確認できた．この状態は 4.1 で述べた通り，これ以上空間方向に並列度を増やしたとしても性能向上を見込めない状態である．加えて時間方向の並列化を適用することでさらなる性能向上が確認でき，時間方向並列化が有効であることがわかった．

しかし図 18 から高並列度において以下の 2 つの点で，スケーラビリティが落ちていることがわかった．

(1) 弱スケーリングにおいて並列度を上げるほど，実行時間が増加している

(2) 空間並列のみと時空間並列の交点が，空間並列度を上げるにつれて右側に移動している

まずは 1 つめについて述べる．MGRIT は時間方向に Multigrid 法を適用したものであり，時間方向にスケーラブルの $\mathcal{O}(n)$ 解法である．つまりタイムステップを増やしても解く時間は一定である．しかし，図 19~ 図 26 によれば，タイムステップを増やすと実行時間も増えていることがわかる．これは最も粗いレベルのタイムステップ数が時

間方向並列度未満になれない，という本研究の実装の制約からきている．つまり，時間並列度が高くなればなるほど，最も粗いレベルのタイムステップ数が大きくなり，時間方向に逐次で解く部分が増えることを意味している．一例として $P_s = 4$ に固定した時の実行時間の内訳を図 27 にプロットした．全てが並列に動いている部分を青色，空間方向は並列に動き時間方向は逐次に動いている最下層部分を黄色としている．

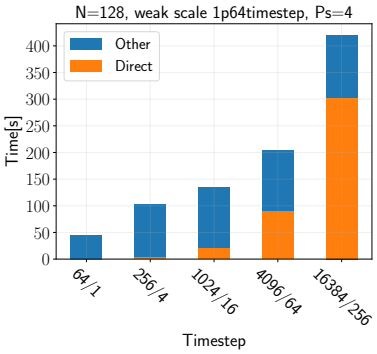

図 **27**　1p64timestep, 時間内訳

図 27 では並列度を上げるに従って，最下層部分が増大していることがわかる．逆にこの制約を無くすように時間並列度を効率的に使う手法に改善することで，時間方向にスケーラブルな解法に近づく，と予想される．

次に 2 つめについて述べる．図 18 によれば，空間並列のみの線と時空間並列の線の交点は，並列度を増やすにつれて右側に移動している．また図 19~26 の $T = 16384$ では，固定する並列度を高くするに従って，徐々に性能向上比率が悪くなっていることも確認できる．

$P_s = 1 \sim 64$ による性能向上は時間方向の並列化とは独立に考えられるため，理想的にはどの空間並列度で固定しても，同じ x 座標，タイムステップで交わるはずである．グローバルコミュニケータを分割し，空間ソルバは空間コミュニケータ内で通信を行ない，時間ソルバは時間コミュニケータ内で通信を行っているために，最大 $P = 16384$ 並列の場合においても性能向上している．とはいえ 1 万プロセス以上に及ぶ高並列度になると，MPI の高並列によるオーバーヘッドが生まれ，実行時間もより小さくなっているためにオーバーヘッドが隠蔽できずに，時間並列による性能向上率が小さくなり，交点が右側に移動していると考えた．

4.3　全体並列度固定で時空間並列度のバランス

全体並列度 P を固定し，$P = P_s \times P_t$ を満たすように，空間並列度と時間並列度を変化させ，MGRIT 適用時の性能に関する影響の評価を行った．扱う問題は上記の実験と同様の空間サイズ/空間自由度を $N^2 = 128 \times 128 = 16384$，タイムステップ数/時間自由度を 16384 とした．この問題では，空間方向の問題に対する係数行列は図 28 のような 5 点差分による 16384 行の帯行列となり，時間方向の問題に対する係数行列は空間を 1 自由度として捉えれば図 29 のような 16384 行の下三角行列となる．図 28，29 共に非ゼロ要素を「*」で表現している．つまり空間方向と時間方向では次元数が同じで同等の問題を扱っている．

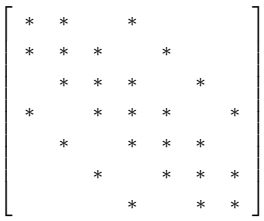

図 **28**　空間方向の係数行列概形

図 **29**　時間方向の係数行列概形

この問題に対して固定する全体並列度 P を 256, 1024, 4096 と変更して実験を行ない，図 30~32 を得た．横軸の上付きラベルは空間並列度 P_s を，下付きラベルは時間並列度 P_t を表す．また全てが並列に動いている部分を青色，空間方向は並列で動き，時間方向は逐次で動いている部分を黄色で表現している．

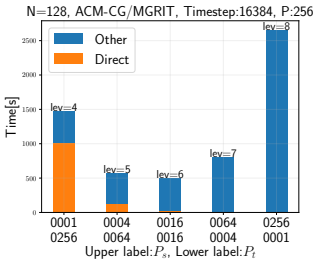

図 **30**　$P = 256$ に固定

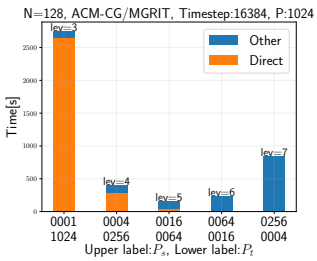

図 **31**　$P = 1024$ に固定

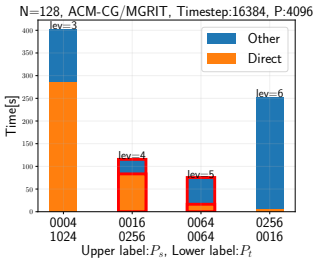

図 **32**　$P = 4096$ に固定

以下の考察において空間並列度と時間並列度の割り当てのペアを (P_s, P_t) という記述をする．全体の並列度 P は $P_s \times P_t$ である．この記述は空間並列度をペアの前者に，時間並列度をペアの後者にしたものである．

空間方向のみの並列化を考えた場合，図 17 から $P_s = 64$ が良いことがことがわかる．これを用いて時空間並列において一番実行時間が短い所を考えると，$P_s = 64$ を含む時空間並列度の割り当てが最適であると予想できる．しかし

図 30 では (16, 16) が最適であり，図 31 では (16, 64) が最適であり，図 32 では (64, 64) が最適であり，その予想に反している．図 30 において，(16, 16) と (64, 4) を比較して，(16, 16) の割り当ての方が実行時間が短いことが示すのは，空間並列度を 16 → 64 にする性能向上率よりも，時間並列度を 4 → 16 にする性能向上率の方が上回っていることである．つまり，空間方向と時間方向ともに扱っている次元数は同じために，時間方向並列化による性能向上比率は空間方向並列化の性能向上比率と同等であること，時間方向においても空間方向と同等の並列性の抽出ができていることが言える．

またこの実験においても前章の通り，時間方向の最も粗い部分の逐次性を改善することによってさらなる性能向上が見込める．図 32 において赤枠で囲まれた (16, 256) と，(64, 64) に注目する．もし時間方向に余っている並列度を効率的に使えたと仮定すれば，逐次部分が減少し (16, 256) の割り当て方が最適なポイントになる可能性がある．そうした場合は，空間方向に比べて時間方向の方が性能向上率が大きいことを示している．空間方向と時間方向共に同じ次元数とはいえ，時間方向においては空間方向を 1 自由度として捉えているために，より時間方向の方が並列性を抽出できる可能性がある．

最後に最適な割り当て方のポイントは図 30∼32 の最適な場所が異なっていることから，実行する最大並列数によって変化するのがわかる．またもし空間並列の性能向上比率と時間並列の性能向上比率がずれていると仮定すると，性能向上比率が高い方に並列度を割り当てた方が良いはずなので，最適な割り当て方のポイントでは，性能向上比率が同等になると予想される．

5. おわりに

本研究では，時間方向並列化の有効性の確認と，時空間並列度の割り当てによる性能への影響の考察を行った．時間方向並列化の有効性については，空間方向並列化による性能向上が頭打ちになっている状況に対して，時間方向並列化の弱スケーリングの実験を行った．この実験より，頭打ちになっている性能を時間方向並列化を検討することによって，さらなる性能向上が確認できた．また 1 万プロセスを超える高並列度では Flat MPI によるオーバーヘッドが隠蔽できていないと考えられた．これは空間方向ソルバの Hybrid 並列化によって，全体の並列度は変えずにオーバーヘッドを抑えることができると想定される．

時空間並列度の割り当てについては，空間方向と時間方向ともに同じ自由度の問題に対して全体の並列度を固定し，空間並列度と時間並列度を変化させる実験を行った．この実験より時間方向並列化は空間並列化と同等の性能向上率を有すること，全体の並列度によって最適な割り当て方が変化することが分かった．また空間並列性能向上比率と時間並列性能向上比率が同程度の値になるところが最適な割り当てになると想定される．

また現在の並列化手法では，時間方向の Multigrid の直接解法において，最下位レベルのタイムステップ数が時間方向の並列度以上になることが問題となっている．これを改善することで，さらなる性能向上を図ることができる．例えば時間並列度を段階的に減らし，1 プロセスの負荷を均等にする方法が考えられる．

参考文献

[1] William L. Briggs, Van Emden Henson, Steve F. McCormick, *A multigrid Tutorial Second Edition*, SIAM, (2000)

[2] I. Yavneh, *Why Multigrid Methods Are So Efficient*, Computing in Science and Engineering,8 (2006), pp. 1222

[3] R.D. Falgout, *Introduction to Algebraic Multigrid*, Computing in Science and Engineering, 8 (2006), pp. 2433

[4] B. R. Hutchinson, G. D. Raithby, *A Multigrid Method based on the Additive Correction Strategy*, Numerical Heat Transfer, (1986), Vol. 9, pp: 511-537

[5] Thor Gjesdal, *A NOTE ON THE ADDITIVE CORRECTION MULTIGRID METHOD*, International Communications in Heat and Mass Transfer, (1996)

[6] Soria Guerrero M, *Parallel multigrid algorithms for computational fluid dynamics and heat transfer*, TDX (Tesis Doctorals en Xarxa), (2005)

[7] Susie C. Keller, Clovis R. Maliska, *AGGLOMERATION SCHEMES STUDIES IN THE ADDITIVE CORRECTION MULTIGRID METHOD APPLIED TO NUMERICAL SOLUTION OF NAVIER-STOKES EQUATIONS WITH UNSTRUCTURED GRIDS*, Proceedings of ENCIT, (2012)

[8] 室田一雄, 杉原正顕, 線形計算の数理, 岩波書店

[9] K. Nakajima, *Parallel Iterative Linear Solvers with Preconditioning for Large Scale*, (2002)

[10] Tatebe O, *The Multigrid Preconditioned Conjugate Gradient Method*, 6th Copper Mountain Conference on Multigrid Methods, pp: 621-634, (1993)

[11] Tatebe. Osamu, *MGCG METHOD : A ROBUST AND HIGHLY PARALLEL ITERATIVE METHOD*, The Graduate School of The University of Tokyo, (1996)

[12] R. D. FALGOUT, S. FRIEDHOFF, TZ. V. KOLEV, S. P. MACLACHLAN, J. B. SCHRODER, *Parallel Time Integration with Multigrid*, SIAM Journal on Scientific Computing, (2014)

[13] R.D. Falgout, A. Katz, Tz.V. Kolev, J.B. Schroder ,A.M. Wissink, U.M. Yan: *Parallel Time Integration with Multigrid Reduction for a Compressible Fluid Dynamics Application*, Journal of Computational Physics, (2014)

[14] R. D. FALGOUT, S. FRIEDHOFF, TZ. V. KOLEV, S. P. MACLACHLAN, J. B. SCHRODER, S. VANDEWALLE, *Multigrid methods with spacetime concurrency*, Computing and Visualization in Science, pp: 1-21, (2017)